UNIVERSITY OF CALIFORNIA

Los Angeles

Constrained Inference and Decoding for Controlling Natural Language Processing Models

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Tao Meng

2024

ABSTRACT OF THE DISSERTATION

Constrained Inference and Decoding for Controlling Natural Language Processing Models

by

Tao Meng
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2024
Professor Kai-Wei Chang, Chair

With the rapid development of neural models in natural language processing (NLP), large and deep models achieve state-of-the-art across NLP tasks, and are deployed in real-world applications. Models become black-box to our human. Therefore, effective approaches controlling NLP moedls are demanding. Controlling helps model solve particular tasks. For example, when we ask the model to generate a recipe, we have a constraint about what ingredients we want the recipe to contain. In addition, as NLP researchers, we are responsible for preventing models from generating offensive or other unpredictable outputs, otherwise deploying them in real-world applications may cause society issues. To control the NLP models, my research focus on injecting constraints, a set of rules that the model must follow, to control the model behaviour via constrained inference and decoding. My research goal is to develop techniques leveraging different kinds of constraints in various scenarios for structure prediction models and large language models. Generally, constraints represent human knowledge and expectation to the model outputs, and constrained inference is the bridge between human beings and the neural models.

The dissertation of Tao Meng is approved.

Yizhou Sun

Cho-Jui Hsieh

Adnan Youssef Darwiche

Kai-Wei Chang, Committee Chair

University of California, Los Angeles

2024

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to nominate my supervisor Kai-Wei Chang, as the BEST SUPERVISOR in the world. In March 2018 I got the his offer but I missed the UCLA visit day. He invited me to have a campus tour and introduced the group members to me. At the beginning of my Ph.D., I did not have much research experience, my English is poor, and my understanding to NLP was more than naive. He taught me a lot of things. Instead of how to publish, he spent much efforts teaching me how to be a good independent researcher, how to think critically, how to learn from others and how to express my thinking. I have tricky taste in research topic, and may focus on topics far from his funding projects, but Kai-Wei is always supportive. In my whole Ph.D. life I never do research on some topic I am not interested in. I am also a lazy student but I never feel pressure from him. Sometimes I lose motivation in research and don't have progress for weeks. I never got blamed by Kai-Wei and he just chatted casually with me about some research questions in our regular meeting. Sometimes I stuck in research, or got bad reviews, he always supports me by convincing me my research is good and I am a good researcher. The harshest words he said to me is 'the figure is ugly'. Before joining his group I never imagine the Ph.D. life can be such smooth and I could be so lucky to have a nice supervisor. Thank you Kai-Wei and hope you become the greatest NLP researcher!

I would also thank all the professors in my committee, Prof. Adnan Darwiche, Prof. Cho-jui Hsieh and Prof. Yizhou Sun, for providing valuable feedback during my Ph.D. Your comments and questions inspire me to think deep and think big.

It is my pleasure to have so many great collaborators from both academia and industry in my Ph.D. life. I learn to consider research problems in different perspectives.

I feel more than fortunate to spend over five years in UCLANLP group with my amazing friends in Room 368. I love the time we spend together discussing our recent research, interesting topic, good paper and so on. We drive to hotpot every weekend, sharing our lifes. I love the kind and pure atmosphere in 368.

Lastly, I want to thank to my family. They always stand behind supporting me, letting me know I am cared and protected. I love you all!

Honestly, the five-year Ph.D. journey is the best time in my life so far. I appreciate you all making my life so colorful. I will always remember my life here!

VITA

2014–2018 B.E. (Institute for Interdisciplinary Information Sciences), Tsinghua University.

PUBLICATIONS

Tao Meng, Nanyun Peng and Kai-Wei Chang. Target Language-Aware Constrained Inference for Cross-lingual Dependency Parsing. EMNLP. 2019.

Shengyu Jia*, **Tao Meng***, Jieyu Zhao, and Kai-Wei Chang. Mitigating Gender Bias Amplification in Distribution by Posterior Regularization. ACL (Short). 2020.

Fan Yin, Quanyu Long, **Tao Meng*** and Kai-Wei Chang. On the Robustness of Language Encoders against Grammatical Errors. ACL. 2020.

Da Yin, **Tao Meng*** and Kai-Wei Chang. SentiBERT: An Effective, Transferable and Interpretable Architecture for Compositional Sentiment Semantics. ACL. 2020.

Tao Meng, Kai-Wei Chang. An Integer Linear Programming Framework for Mining Constraints from Data. ICML. 2021.

Sidi Lu, **Tao Meng**, and Nanyun Peng. InsNet: An Efficient, Flexible, and Performant Insertion-based Text Generation Model. NeurIPS 2022.

Tao Meng*, Sidi Lu, Nanyun Peng and Kai-Wei Chang. Controllable Text Generation with Neurally-Decomposed Oracle. NeurIPS (Oral) 2022.

Honghua Zhang, Liunian Harold Li, **Tao Meng**, Kai-Wei Chang and Guy Van den Broeck. InsNet: On the Paradox of Learning to Reason from Data. IJCAI 2023.

CHAPTER 1

Introduction

In recent years, deep neural networks have significantly advanced natural language processing (NLP), achieving state-of-the-art results across various tasks. Models are deployed in many real-world applications, which helps reduce human efforts in many area. However, neural models are still a black-box for human. As models grow larger and more complex, they become increasingly difficult for humans to understand and control.

It is necessary to explore how to control a neural model. One reason is that in some tasks we need to achieve control on specific attributes such as lexically constrained generation, where we are asked to generate a sentence including particular tokens. Furthermore, models have to be controlled. We observe that models may generate unpredictable outputs such as offensive generation [WFK19, GGS20] or bias amplification [ZWY17]. This has created a significant challenge for NLP development and applications, highlighting the need for better model control.

To address this challenge, I propose leveraging constraints - specific rules that machine learning models must follow for certain instances or distributions - to achieve better control over neural models. My research aims to improve model control through constraints by incorporating human knowledge into constraints and injecting them into models by constrained inference and decoding.

We have different kinds of constraints. Considering the way we define the constraints, we have model-based constraints and rule-based constraints. Categorized by the scope of constraints, we have instance-level, corpus-level and distribution-level constraints. My research is to develop techniques to handle constraints in different scenarios.

In the first half of my Ph.D. life, I have been specifically focused on developing techniques in structural prediction models in NLP, injecting corpus-level and distribution-level pre-defined constraints. By leveraging constraints, we can achieve better model performance by bridging the gap between the model prediction and ground truth in corpus-level or distribution. An integer linear programming framework for automatically mining constraints is proposed to reduce human efforts in figuring out the constraints in different scenarios.

In the second half, large language models (LLMs) develop rapidly and dominate NLP community. They achieve remarkable performance on all benchmarks and refresh the state-of-the-art. I focus more on controlling LLM, including developing a constrained decoding algorithm by leveraging an auxiliary model, and propose a novel attribute controlled fine-tuning approach.

My research goal is to leverage constraints to control the black-box neural NLP models. Such controlling can not only prevent models from generating unpredictable outputs, but also improve the model performance. Results show that constrained inference is effective in controlling the models, without hurting the model performance on downstream tasks irreverent from the controlling attributes. The controlled models are closer to our expectation. As the rules defined by human, constraints represent human knowledge and expectation, while constrained inference plays a role as a bridge between human beings and neural models.

In Chapter 2 we cover some background knowledge used in my research including fundamental NLP application and NLP models.

In Chapter 3 we introduce two constrained inference algorithm that are widely used in machine learning. My research develops various of algorithms for different application and models based on them.

In Chapter 4 we show a bias amplification problem in a vision-and-language task. Specifically, a machine learning model failed to mimic the biases in the training dataset but further amplifies that. We inject distributional constraints to successfully control the amplification.

In Chapter 5 we consider the cross-lingual dependency parsing task. For low-resource langauges, we do not have enough data to train a good parser. Researchers usually leverage an English parser to parse the text in the low-resource language. However, they may have different word order, which cause a poor performance in the transfer. We utilize the knowledge about word order from human experts, compile them into constraints and inject constraints into the parser to significantly improve the performance.

In Chapter 6 we explore an integer linear programming framework to automatically mine constraints from data. In previous work all the constraints are pre-defined or given by human experts, which takes a lot of human efforts. With this framework, we are able to mine constraints in linear form from data, including structural constraints or hard constraints in puzzle games.

In Chapter 7 we adapt the constrained inference method to controlling large language models. We regard the constraints as a black-box function, and the controlling as an optimization problem. We first decompose the sentence-level constraints into token-level guidance. Considering the intractability of the decomposition, we leverage an auxiliary model to approximate the token-level guidance trained by data sampled from the model. As a decoding-time control approach, our decoding time and space complexity is similar as the original decoding.

In Chapter **??** we apply the proposed constrained decoding algorithm in large language model detoxification. We demonstarte that our algorithm is efficient and effective in controlling a large langauge model. We conduct ablation study about different factors in constrained decoding algorithm and show the empirical results.

In Chapter 9 we introduce a large language model fine-tuning algorithm with attribute control based on constrained decoding. This novel fine-tuning algorithms control pre-defined attribute during fine-tuning on corpus, achieving same level performance on bench-mark and as efficient as original fine-tuning.

We summarize this thesis in Chapter 10.

CHAPTER 2

Background and Related Work

In this chapter, we will talk about the background knowledge of the methods we analyzed and purposed in this prospectus. It will cover three parts: the first part is about integer linear programming (ILP), which is widely used in inference problems, and we will leverage it our constrained inference algorithms as well as the constraint mining framework. The second part is about cross-lingual dependency parsing, which is a fundamental NLP task and we will use it as an application to test our purposed methods. The third part is about the basic notation and formulation about large language models. The last part is about the related work about the applications using constrained inference.

2.1 Integer Linear Programming (ILP)

An ILP is basically a linear programming (LP) with restriction that all the variables can only take integer values. In LP, the objective function and the (in)equality constraints are linear with respect to the variables. Formally, an ILP can be written as:

$$\max_{\mathbf{y}\in\mathbb{Z}^d} \mathbf{w}\cdot\mathbf{y} \quad \text{s.t. } A\mathbf{y} \le \mathbf{b}. \tag{2.1}$$

where $d \in \mathbb{Z}$ is the dimension, $\mathbf{w} \in \mathbb{R}^d$ is the weights in the objective function, and $A \in \mathbb{R}^{n\times d}, \mathbf{b} \in \mathbb{R}^n$ define the linear constraints.

ILP is a general and flexible framework supporting various of constraints including propositional logic [RY04]. In practice, ILP is widely used in formulating constrained inference in machine learning tasks, including semantic role labeling [PRY04], entity-relation extraction [RY05], sentence compression [CL08], dependency parsing [MSX09], multi-lingual transfer [MPC19], corefernece resolution [CSK13], relation extraction [YFL20] and reducing bias amplification [ZWY17]. In these tasks, we get the weights from a neural model, and guarantee that without constraints, the ILP problem is same as the original neural methods. The constraints are pre-defined and are compiled from some external knowledge.

ILPs are in general NP-hard. However, in practice, these inference problems often can be solved efficiently using a commercial ILP solver. Besides, approximation inference techniques (e.g., LP-relaxation [FG09], loopy belief proprogation [MWJ13]) can be used to obtain approximate solutions. See discussion in [FJ08].

2.2 Cross-Lingual Dependency Parsing

Dependency parsing is a task of extracting the grammatical structure of a sentence. The grammatical structure is organized as a directed tree, and the arcs between two words show their dependency, labeled by the type of their relations.

Cross-lingual dependency parsing is a task that train a parser on the source language and apply it on the target language. The cross-lingual transferability is given by the multilingual embedding, such as FastText [BGJ17] or multilingual BERT [DCL18].

There are mainly two kinds of parser in dependency parsing: graph-based parser and transition-based parser. In this prospectus we will only use graph-based parser, and the inference problem can be formulated as a directed minimal spanning tree task, which will be introduced in Sec. 5.1

2.3 Large Language Models (LLMs)

LLMs are one of the most popular models in NLP community and it shows amazing capability in NLP tasks. Formally, LLM is an auto-regressive model, predicting next token distribution based on the input and generation prefix. We usually denote $\mathbf{x}$ as the input to an LLM, $\mathbf{y}$ as the generation and y_i as the token at position i. We use $\mathbf{y}_{<i}$ to denote the

generation prefix to position i. The probability of generating a particular sentence $\mathbf{y}$ given input $\mathbf{x}$ is given by

$$p(\mathbf{y}|\mathbf{x}) := \prod_i p(y_i|\mathbf{x}, \mathbf{y}_{<i}).$$

2.4 Applications of Constrained Inference

Constraints are widely incorporated in variety of tasks. To name a few, [RY04] propose to formulate constrained inferences in NLP as integer linear programming problems. To solve the intractable structure, [RC12] decompose the structure and incorporate constraints on some composite tasks. To improve the performance of a model, [CC11, PCE15] incorporate constraints on exact decoding tasks and inference tasks on graphical models, and [CSK13, Dal15, Mar15] incorporate corpus-level constraints on semi-supervised multilabel classification and coreference resolution. [ZWY17] incorporate corpus-level constraints to avoid amplifying gender bias on visual semantic role labeling and multilabel classification.

CHAPTER 3

Constrained Inference with Pre-defined Constraints on Distribution Gap

Our research goal is to design an efficient inference algorithm working on pre-defined constraints for bridging the distribution gap and given machine learning models. The constraints can either help the model perform better within domain, or improve the transferability in some cross-domain tasks. In this chapter, we introduce various of constraints and their corresponding inference algorithms.

3.1 Distribution Gap and Hard Constraints

Distribution gap means some meaningful features have different distributions between two domains, or between the model predictions and the ground truth. Formally, we denote the input space as $\mathcal{X}$ and output space as $\mathcal{Y}$. On the space $\mathcal{X} \times \mathcal{Y}$ we define feature function $h : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. The distribution gap means for two distribution p and q, the marginal distribution of h is quite different. Here p can be the real-world distribution and q can be the model predictions.

Hard constraints are a set of deterministic constraints, such as "a determiner is *always* on the left of the word it depends on" in dependency parsing task, or "an image labeled as a flower *must* be labeled as a plant, and *must not* be labeled as a car" in hierarchical multilabel classification task. Formally, the hard constraint can be formalized as equality or inequality about feature function h, such as $h(x, y) = 1$ or $h(x, y) > 0.5$.

Models usually give scores or potentials for each possible output and the inference procedure is find the optimal one. Thus, in inference the most efficient and straightforward way to incorporate the hard constraints is to filter out the invalid outputs. For example, in dependency parsing when we decide the dependent of a word, we can ignore the invalid possibilities and pick the optimal one from the rest.

However, in some tasks like structure predictions, the output space can be exponentially large and it can be hard to enumerate all the possible outputs. A general solution can be ILP discussed in Sec. 2.1 if we can compile the task and constraints into linear form, which is pretty common in practice. For some special hard constraints, there exists some ad-hoc efficient inference algorithms based on dynamic programming or decomposition, e.g., Chu-Liu algorithm [CL65] for the projective constraint in dependency parsing.

Leveraging hard constraints benefit the machine learning models since all the instances satisfy the constraints. However, meaningful hard constraints are not always exist since exceptions are very common in real-world applications. For exmaple, in part-of-speech (POS) tagging task, a constraint "every sentence should contain at least one verb" forces each predicted tag sequence has at least one verb. Formally, **x** is the input sequence and **y** is the predicted tag sequence. We define the feature function as

$$h(\mathbf{x}, \mathbf{y}) = \begin{cases} 0 & \forall i \quad y_i \neq \text{verb} \\ 1 & \exists i \quad y_i = \text{verb} \end{cases}$$

Although this constraint $h(x, y) = 1$ makes sense, there might be short phrase without verb in the corpus. Thus it's necessary to relax the hard constraints into soft constraints.

3.2 Corpus-Level Constraints and Inference

To deal with the exceptions, it is necessary to leverage soft constraints. Intuitively, soft constraints are a set of constraints that in most of the time the data should satisfy, but exceptions are also allowed, just like the verb constraint in POS tagging task. To formulate the soft constraints, basically two kinds of approaches are popular: 1) convert constraints into penalty terms in the objective function, e.g., semantic loss [XZF18]. 2) add slack variables to each of the constraints and the objective function, e.g., soft SVM. 3) formulate the task as a maximum satisfiability problem (MAXSAT), e.g., SATNet [WDW19].

These approaches are good solutions in instance level. However, they are still not hard to bridge the distribution gap. If we take the view of a branch of instances, i.e., a corpus, drawn independently from the distribution, the constraints can be better formulated and easier to control. Thus, we leverage corpus-level constraints [CSK13, Dal15, Mar15].

3.2.1 Corpus-Level Constraints

Corpus-level constraints are constraints about the proportion of instances satisfying instance-level constraints. Formally, given a corpus $\{x_i\}_{i=1}^N$ drawn from input space $\mathcal{X}$, where N is the number of instances, the model can make the corresponding predictions $\{\hat{y}_i\}_{i=1}^N$. We define $D = \{x_i, \hat{y}_i\}_{i=1}^N$ and the instance-level constraint $h(x, y) = 1$ we define the ratio function

$$R(h, D) = \frac{\sum_{(x_i,y_i)\in D} h(x_i, y_i)}{|D|} \tag{3.1}$$

$R(h, D)$ denotes the ratio of instances in D that satisfy the instance level constraint C. Based on this ratio we can formulate corpus-level constraints. We can restrict that most of the instances should satisfy the constraint by setting $R(h, D) \geq r$, or vice versa. We can also calibrate the ratio by constraint $r_1 \leq R(h, D) \leq r_2$. This kind of calibration constraints can be treated as two separate constraints.

The ratio r, r_1, r_2 can be pre-defined from external knowledge, or it can be treated as hyper-parameters, like the weights for penalty terms in soft constraints.

Actually binary constraints function h can be replaced by any real value feature function to make the constraints more general. The ratio function $R(h, D)$ represents the average feature in the predictions in the corpus D. For example, the verb constraint in POS tagging task, we can set $h(\mathbf{x}, \mathbf{y})$ as a verb counter, and formulate constraint "on average each sentence should contain at least one verb in the corpus" by setting $r = 1$. Also, we can inverse function h to make all the constraints have consistent signs ($\geq$ or $\leq$). Without the loss of generality, in the following we set each constraint has form $R(h, D) - r \leq 0$, and use the pair $C = (h, D)$ to represent the constraint.

3.2.2 Lagrangian Relaxation

To address the inference with corpus-level constraints, we leverage Lagrangian Relaxation (LR). LR has been applied in various NLP applications [RC12, RC11]. Basically, the LR algorithm introduces Lagrangian multipliers to relax the constraint optimization problem to an unconstrained optimization problem, and estimates the Lagrangian multipliers with gradient-based methods.

We denote for instance-level unconstrained inference as

$$\hat{y} = \arg\max_{\hat{y} \in \mathcal{Y}} f(x, \hat{y}), \tag{3.2}$$

the original inference algorithm can solve it. Now we want to do the global inference on the corpus, satisfying our corpus-level constraints $\mathcal{C} = \{(h_i, r_i, \theta_i\}_{i=1}^{M}$, where M is the number of constraints. We denote $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N)$ as the global predictions. In following we assume each instance has the same output space $\mathcal{Y}$ for simplicity. Actually, for different output spaces $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_N$, $\mathbf{y}$ is drawn from space $\mathcal{Y}_1 \times \mathcal{Y}_2 \times \cdots \times \mathcal{Y}_N$ and the formulation is similar. Our inference problem can be formulated as

$$\begin{aligned} \hat{\mathbf{y}} &= \arg\max_{\mathbf{y} \in \mathcal{Y}^N} \sum_{i=1}^{N} f(x_i, y_i), \\ \text{s.t.} \quad & R(h_i, D) - r_i \leq 0. \quad i \in [N], \end{aligned} \tag{3.3}$$

We plug Eq.(3.1) and rewrite the constraint $R(h, D) = r$ as

$$\sum_{i=1}^{N} h(x_i, y_i) - rN \leq 0. \tag{3.4}$$

Now we add Lagrangian multipiers $\lambda = \{\lambda_i\}_{i=1}^{M}$ to the constraints in form of Eq.(3.4), and the Lagrangian function is

$$L(D, \lambda; \mathcal{C}) = \sum_{i=1}^{N} \left(f(x_i, y_i) + \sum_{j=1}^{M} \lambda_j (h_j(x_i, y_i) - r_j N) \right). \tag{3.5}$$

We know that when the feasible solution exists, solving the min-max problem

$$\min_{\lambda \geq 0} \max_{\mathbf{y} \in \mathcal{Y}^N} L(D, \lambda; \mathcal{C})$$

is same as solving the original constrained optimization problem, which is also equivalent to solve the dual form

$$\max_{\mathbf{y} \in \mathcal{Y}^N} \min_{\lambda \geq 0} L(D, \lambda; \mathcal{C}).$$

To solve the dual form, we initialize λ_i to be 0. At iteration t, we firstly conduct an constraint-augmented inference with a fixed $\lambda^{(t)}$:

$$\hat{\mathbf{y}}^{(t)} = \arg\max_{\mathbf{y} \in \mathcal{Y}^N} L(D, \lambda^{(t)}; \mathcal{C}). \tag{3.6}$$

Although Eq.(3.6) is a global inference across the whole corpus, we notice that the Lagrangian function in Eq.(3.5) is decomposed into instance level, which means we can do inference for each instance individually. If we can take the original inference algorithm into consideration, and carefully design the constraint functions $\mathbf{h}$ to make them have the similar form as f, we can apply the original inference algorithm on the new objective function. E.g., in Eq.(3.2) the original inference algorithm works when f is a linear function in terms of y, we can design h as linear functions, too. The Lagrangian function is still linear and the original inference algorithm works.

After solving the constraint-augmented inference, we compute the ratio of every constraint $\hat{r_i}^{(t)} = R(h_i, D)$, and use sub-gradient descent algorithm to update the Lagrangian multipliers

$$\lambda_i^{(t+1)} = \max\{0, \lambda_i^{(t)} - \alpha^{(t)}(r_i - \hat{r_i}^{(t)})\}.$$

Here $\alpha^{(t)}$ denote the step size at iteration t. The algorithm is shown in Algorithm 1.

Algorithm 1 Lagrangian Relaxation for Constraint Inference

Input: Constraints $\mathcal{C} = \{(h_i, r_i, \theta_i)\}_{i=1}^{N}$, corresponding ratio learning rate decay η, initial learning rate α_0

Output: Prediction $\hat{\mathbf{y}}$, Lagrangian multipiers λ

```
1: $\alpha \leftarrow \alpha_0$
2: $\lambda_i \leftarrow 0,\ i \in [N]$
3: repeat
4:     $\hat{\mathbf{y}} \leftarrow \arg\max L(D, \lambda; \mathcal{C})$
5:     $\hat{r_i} \leftarrow R(h_i, D),\ i \in [N]$
6:     if $\forall i, abs(r_i - \hat{r_i}) \leq \theta$ then
7:         return $\hat{\mathbf{y}}$
8:     $\lambda_i \leftarrow \max\{0, \alpha(\hat{r_i} - r_i)\},\ i \in [N]$
9:     $\alpha \leftarrow \eta\alpha$
10: until MAX_ITER times
11: return $\hat{\mathbf{y}}, \lambda$
```

Actually, what we get from LR is more than a branch of inference results. These Lagrangian multipliers are quite general and we can define the Lagrangian function in instance level as

$$L(\lambda, x, y) = f(x, y) + \sum_{j=1}^{M} \lambda_j h_j(x, y), \tag{3.7}$$

here we drop the constant terms.

We can replace the objective function $f(x, y)$ in Eq.(3.2) by $L(\lambda, x, y)$. With the original inference algorithm with the new objective function, the solution is guided by these corpus-level constraints.

3.3 Distributional Constraints and Inference

The probabilistic models are able to compute the distribution over all possible outputs. Based on the distribution, we make predictions by picking the output with highest posterior probability, i.e., maximum-a-posteriori (MAP) inference. For those models, directly adding constraints on distribution is more fundamental and smooth compared to adding constraints on predictions, and also easier to bridge the gap in distribution.

3.3.1 Distributional Constraints

Constraints in distribution are usually soft constraints. We can think of a probabilistic model for POS tagging that output a distribution over POS tags for each token. With the constraint "each sentence should contain at least one verb *in expectation*", the model is still able to output a sentence without a verb after inference in some extreme cases, while, in the same time, the constraint provide meaningful supervision to the model.

Different from the instance-level constraint definition in 3.1, the distribution constraints are defined on a distribution p_θ given by a probabilistic model instead of the predictions. Formally, given input x, instance-level constraints function $h : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and the probabilistic model modeling the conditional probability $p_\theta(y|x)$, we define the expected rate about the constraints satisfaction

$$\mathbb{E}_{y \sim p_\theta(\cdot|x)} [h(x, y)] = \sum_{y \in \mathcal{Y}} p_\theta(y|x) h(x, y). \tag{3.8}$$

Based on this expectation, we can define distribution constraints. Taking the verb constraint in POS tagging task as an example again, we denote x is the token sequence and y is the predicted tag sequence. $h(x, y)$ Similar to the corpus-level constraints, we can formulate some calibration constraints as well as the relaxed hard constraints.

This definition can also be extended to corpus-level. For corpus $\mathbf{x} = \{x_i\}_{i=1}^N$ and corresponding inference instances $\mathbf{y} = \{y_i\}_{i=1}^N$, according to the independency, we extend the model into corpus-level as

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p_\theta(y_i|x_i).$$

We then extend the constraint function h to corpus-level as

$$h(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} h(x_i, y_i),$$

which represents a counter for satisfactory. The expectation formula in Eq.(3.9) in corpus-level can be written as

$$\mathbb{E}_{\mathbf{y}\sim p_\theta(\cdot|\mathbf{x})}\left[h(\mathbf{x}, \mathbf{y})\right] = \sum_{\mathbf{y}\in\mathcal{Y}^N} p_\theta(\mathbf{y}|\mathbf{x})h(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N}\sum_{y_i\in\mathcal{Y}} p_\theta(y_i|x_i)h(y_i|x_i), \tag{3.9}$$

which means this form of corpus-level distribution constraints can be decomposed into instance-level, which makes the inference tractable. Similar to the corpus-level constraint, here we can also extend function h from binary to any real value function, and we can use a consistent form to formulate the distribution constraints as

$$\mathbb{E}_{\mathbf{y}\sim p(\cdot|\mathbf{x})}\left[h(\mathbf{x}, \mathbf{y})\right] - b \leq 0. \tag{3.10}$$

3.3.2 Posterior Regularization Inference in Model Transfer

Posterior Regularization (PR) is firstly purposed by [GGG10], where the distribution constraints are incorporated in an iterative training procedure. Based on this, we purpose PR in terms of inference, i.e., without retraining the model, we leverage the constraints in inference time.

Similar to LR inference, our PR inference is also an algorithm not only gives the constrained inference results, but also a transferred model with respect to constraints. The only different is that in PR our constraints are on the distribution. We know that the prediction space $\mathcal{Y}$ can be exponentially large, while the distribution over prediction space is even much larger, which makes the constrained inference inefficient and even intractable. Luckily, the distribution constraints and corpus-level distribution constraints, as we defined in Sec. 3.3.1, are tractable based on PR.

The PR inference uses the distribution constraints to define a feasible distribution set, and find a feasible distribution that is closest to the distribution given by the model by minimizing the KL-divergence. The constrained inference problem can then be converted into an MAP inference problem on the best feasible distribution.

Formally, given a set of distribution constraints about distribution q:

$$\mathbb{E}_{y\sim q(y)}\left[\mathbf{h}(x,y)\right] - \mathbf{b} \leq 0,$$

here $\mathbf{h}(x,y)$ and $\mathbf{b}$ are vectors. We use Q to denote the feasible set

$$Q = \{q \mid \mathbb{E}_{y\sim q(y)}\left[\mathbf{h}(x,y)\right] - \mathbf{b} \leq 0\}, \tag{3.11}$$

here (x, y) is a instance but we can also use corpus-level constraints by using a branch of instances $(\mathbf{x}, \mathbf{y})$.

We have the distribution $p_\theta(y|x)$ learned by model and find the closest feasible distribution $q^*(y)$ by solving the optimization problem

$$q^*(y) = \arg\min_{q\in Q} KL(q(y)\|p_\theta(y|x)). \tag{3.12}$$

This optimization problem, considering the form of the feasible set in Eq.(3.11), according to [GGG10], has close form solution:

$$q^*(y) = \frac{p_\theta(y|x)\exp(-\lambda^*\cdot\mathbf{h}(x,y))}{Z(\lambda^*)} \tag{3.13}$$

where λ^* is the solution of

$$\begin{aligned}\lambda^* &= \arg\max_{\lambda\geq 0} -\mathbf{b}\cdot\lambda - \log Z(\lambda),\\ Z(\lambda) &= \sum_{y\in\mathcal{Y}} p_\theta(y|x)\exp(-\lambda^*\cdot\mathbf{h}(x,y)).\end{aligned} \tag{3.14}$$

The Eq.(3.14) is also an optimization problem, but compared to Eq.(3.12), we have same number of variables (λ) as constraints rather than a distribution, which is much more tractable. When the output space $\mathcal{Y}$ is not so large and we can enumerate it, we can apply sub-gradient ascent to solve the optimization problem in Eq.(3.14). After we get a feasible distribution q^* satisfying our constraints, we can apply the original inference algorithm on distribution q^* instead of p_θ to get the predictions.

When the output space $\mathcal{Y}$ is not enumerable, like $\mathcal{Y}$ is a structure or it is a global inference on corpus-level, we cannot directly apply the sub-gradient ascent and we need to do decomposition to normalization term $Z(\lambda)$ first. In the following, we take the corpus-level decomposition as an example.

In corpus-level extension of distribution constraints, we have

$$\begin{aligned} p_\theta(\mathbf{y}|\mathbf{x}) &= \prod_{i=1}^{N} p_\theta(y_i|x_i), \\ \mathbf{h}(\mathbf{x},\mathbf{y}) &= \sum_{i=1}^{N} \mathbf{h}(x_i, y_i), \\ \text{and define} \qquad Z_i(\lambda) &= \sum_{y_i \in \mathcal{Y}} p_\theta(y_i|x_i) \exp(-\lambda^* \cdot \mathbf{h}(x_i, y_i)). \end{aligned} \tag{3.15}$$

Thus,

$$\begin{aligned} Z(\lambda) &= \sum_{\mathbf{y}\in\mathcal{Y}^N} p_\theta(\mathbf{y}|\mathbf{x}) \exp(-\lambda^* \cdot \mathbf{h}(\mathbf{x},\mathbf{y})) \\ &= \sum_{\mathbf{y}\in\mathcal{Y}^N} \prod_{i=1}^{N} p_\theta(y_i|x_i) \exp(-\lambda^* \cdot \mathbf{h}(x_i, y_i)) \\ &= \prod_{i=1}^{N} \sum_{y_i\in\mathcal{Y}} p_\theta(y_i|x_i) \exp(-\lambda^* \cdot \mathbf{h}(x_i, y_i)) \\ \log Z(\lambda) &= \sum_{i=1}^{N} \log Z_i(\lambda). \end{aligned} \tag{3.16}$$

This decomposition makes the sub-gradient methods on Eq.(3.14) tractable. Also, this shows that with the corpus-level constraints, although the solution q^* is a distribution over a branch of instances, it can be decomposed into instance level distribution as

$$q^*(y_i) = \frac{p_\theta(y_i|x_i) \exp(-\lambda^* \cdot \mathbf{h}(x_i, y_i))}{Z_i(\lambda^*)}.$$

For other large space $\mathcal{Y}$, if the function $\mathbf{h}$ and the probability p_θ can be decomposed like Eq.(3.15), the decomposition on $Z(\lambda)$ also works.

CHAPTER 4

Application: Mitigating Gender Bias Amplification with Constraints Bridging the Bias Gap

Advanced machine learning techniques have boosted the performance of natural language processing. Nevertheless, recent studies, e.g., [ZWY17] show that these techniques inadvertently capture the societal bias hidden in the corpus and further amplify it. However, their analysis is conducted only on models' top predictions. In this paper, we investigate the gender bias amplification issue from the distribution perspective and demonstrate that the bias is amplified in the view of predicted probability distribution over labels. We further propose a bias mitigation approach based on posterior regularization. With little performance loss, our method can almost remove the bias amplification in the distribution. Our study sheds the light on understanding the bias amplification.

4.1 Background and Related Work

We follow the settings in [ZWY17] to focus on the imSitu vSRL dataset [YZF16], in which we are supposed to predict the activities and roles in given images and this can be regraded as a structure prediction task (see Fig. 4.1).

We apply the Conditional Random Field (CRF) model for the structure prediction task. We denote $\mathbf{y}$ as a joint prediction result for all instances, and $\mathbf{y}^i$ as a prediction result for instance i. We use $\mathbf{y}_v$ to denote the predicted activity, and $\mathbf{y}_r$ to denote the predicted role. An activity can have multiple roles and usually one of them conveys the gender information. For an instance i, the CRF model predicts the scores for every activity and role, and

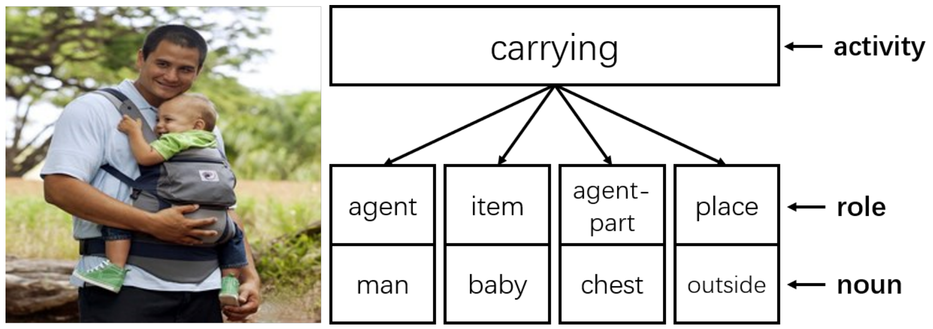


Figure 4.1: An instance from the imSitu dataset. Given an input image, the task it to identify the activity depicted in the image as well as the objects (noun) and their semantic role.

the score for a prediction is the summation of all these scores. Formally,

$$f_\theta(\mathbf{y}^i, i) = s_\theta(\mathbf{y}^i_v, i) + \sum\nolimits_{e \in \mathbf{y}^i_r} s_\theta(\mathbf{y}^i_v, e, i),$$

where $s_\theta(\mathbf{y}^i_v, i)$ and $s_\theta(\mathbf{y}^i_v, e, i)$ are the scores for activity $\mathbf{y}^i_v$ of instance i, and the score for role e of instance i with activity $\mathbf{y}^i_v$, respectively. We can infer the top structure for instance i by:

$$\arg\max\nolimits_{\mathbf{y}^i \in \mathcal{Y}^i} f_\theta(\mathbf{y}^i, i),$$

where $\mathcal{Y}^i$ refers to all the possible assignments to the instance.

[ZWY17] demonstrate bias amplification in the top prediction and present a bias mitigation technique by inference with corpus-level constraints. In the following, we extend their study to analyze the bias amplification in the posterior distribution by the CRF model and define the corresponding corpus-level distribution constraints.

4.2 Constraints and Inference Formulations

Formally, the probability of prediction $\mathbf{y}^i$ for instance i and the joint prediction $\mathbf{y}$ defined by CRF model with parameters θ are given by

$$\begin{aligned} p_\theta(\mathbf{y}^i, i) &\propto \exp(f_\theta(\mathbf{y}^i, i)), \\ p_\theta(\mathbf{y}) &= \prod\nolimits_i p_\theta(\mathbf{y}^i, i), \end{aligned} \tag{4.1}$$

since instances are mutually independent.

In this section, we will define how to quantify the bias and the bias amplification in the distribution, and introduce the corpus-level constraints towards restricting the bias in the distribution.

We focus on the gender bias on activities in the vSRL task. To quantify the gender bias given a particular activity v^*, [ZWY17] uses the percentage that v^* is predicted together with male agents among all prediction with genders. This evaluation focuses on the top prediction. In the contrast, we define bias function $B(p, v^*, D)$ w.r.t distribution p and activity v^*, evaluating the bias toward male in dataset D based on the conditional probability $P(X|Y)$, where $event\ Y$: given an instance, its activity is predicted to be v^* and its role is predicted to have a gender; $event\ X$: this instance is predicted to have gender male. Formally,

$$\begin{aligned} & B(p, v^*, D) \\ =& \mathbb{P}_{i\sim D, \mathbf{y}\sim p}(\mathbf{y}_r^i \in M | \mathbf{y}_v^i = v^* \wedge \mathbf{y}_r^i \in M \cup W) \\ =& \frac{\sum_{i\in D}\sum_{\mathbf{y}^i:\mathbf{y}_v^i=v^*, \mathbf{y}_r^i\in M} p(\mathbf{y}^i, i)}{\sum_{i\in D}\sum_{\mathbf{y}^i:\mathbf{y}_v^i=v^*, \mathbf{y}_r^i\in M\cup W} p(\mathbf{y}^i, i)}. \end{aligned} \tag{4.2}$$

This bias can come from the training set D_{tr}. Here we use $b^*(v^*, male)$ to denote the "dataset bias" toward male in the training set, measured by the ratio of between male and female from the labels:

$$b^* = \frac{\sum_{i\in D_{tr}} \mathbf{1}[\hat{\mathbf{y}}_v^i = v^*, \hat{\mathbf{y}}_r^i \in M]}{\sum_{i\in D_{tr}} \mathbf{1}[\hat{\mathbf{y}}_v^i = v^*, \hat{\mathbf{y}}_r^i \in M\cup W]},$$

where $\hat{\mathbf{y}}^i$ denotes the label of instance i.

Ideally, the bias in the distribution given by CRF model should be consistent with the bias in the training set, since CRF model is trained by maximum likelihood. However, the amplification exists in practice. Here we use the difference between the bias in the posterior distribution and in training set to quantify the bias amplification, and average it over all activities to quantify the amplification in the whole dataset:

$$A(p, v^*, D) = sgn(b^* - 0.5)[B(p, v^*, D) - b^*],$$
$$\bar{A}(p, D) = \frac{1}{|V|} \sum_{v^* \in V} A(p, v^*, D).$$

Note that if we use the top prediction indicator function to replace p in $A, \bar{A}$, it is the same as the definition of the bias amplification in top prediction in [ZWY17].

The corpus-level distribution constraints aim at mitigating the bias amplification in test set D_{ts} within a pre-defined margin γ,

$$\forall v^*,\ |A(p, v^*, D_{ts})| \leq \gamma, \tag{4.3}$$

and this corpus-level distribution constraints can be written into the form of Eq.(3.10). For example, the constaint $A(p, v^*, D_{ts}) < \gamma$ is equivalent as setting

$$h(x, y) = \begin{cases} 1 & y_r \in M \wedge y_v = v^* \\ 0 & y_r \in W \wedge y_v = v^* \\ b^* + \gamma & y_r \notin M \cup W \vee y_v \neq v^* \end{cases}$$
$$b = b^* + \gamma.$$

4.3 Experiments Setup

We conduct experiments on the vSRL task to analyze the bias amplification issue in the posterior distribution and demonstrate the effectiveness of the proposed bias mitigation technique.

Dataset Our experiment settings follow [ZWY17]. We evaluate on imSitu [YZF16] that activities are selected from verbs, roles are from FrameNet [BFL98] and nouns from WordNet [Fel98]. We filter out the non-human oriented verbs and images with labels that do not indicate the genders.

Model We analyze the model purposed together with the dataset. The score functions we describe in Sec. 5.1 are modeled by VGG [SZ15] with a feedforward layer on the top of it. The scores are fed to CRF for inference.

4.4 Bias Amplification in Distribution

Figures 4.2a and 4.2c demonstrate the bias amplification in both posterior distribution p_θ and the top predictions $\mathbf{y}$, respectively. For most activities with the bias toward male (i.e., higher bias score) in the training set, both the top prediction and posterior distribution are even more biased toward male, vise versa. If the bias is not amplified, the dots should be scattered around the reference line. However, most dots are on the top-right or bottom-left, showing the bias is amplified. The black regression line with $slope > 1$ also indicates the amplification. Quantitatively, 109 and 173 constraints are violated when analyzing the bias in distribution an in top predictions. Most recent models are trained by minimizing the cross-entropy loss which aims at fitting the model's predicted distribution with observed distribution on the training data. In the inference time, the model outputs the top predictions based on the underlying prediction distribution. Besides, in practice, the distribution has been used as an indicator of confidence in the prediction. Therefore, understanding bias amplification in distribution provides a better view about this issue.

To analyze the cause of bias amplification, we further show the degree of amplification along with the learning curve of the model (see Fig. 4.3). We observed that when the model is overfitted, the distribution of the model prediction becomes more peaky[1]. We suspect this is one of the key reasons causes the bias amplification.

4.5 Bias Amplification Mitigation

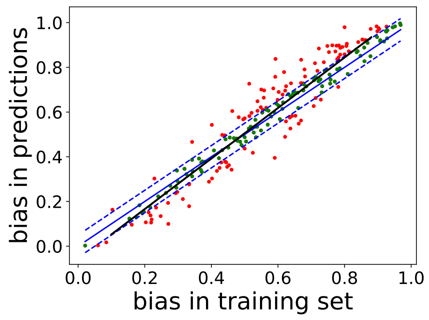
We set the margin $\gamma = 0.05$ for every constraint in evaluation. However, we employ a stricter margin ($\gamma = 0.001$) in performing posterior regularization to encourage the model

[1]This effect, called overconfident, has been also discussed in the literature [GPS17].

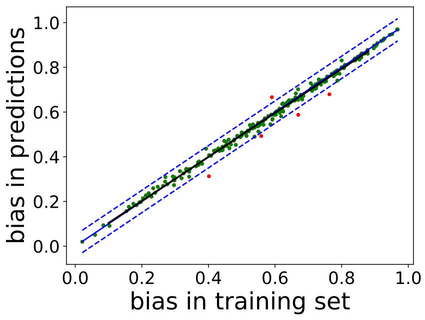
to achieve a better feasible solution. We use mini-batch to estimate the gradient w.r.t λ with Adam optimizer [KB15] when solving Eq. (3.14). We set the batchsize to be 39 and train for 10 epochs. The learning rate is initialized as 0.1 and decays after every mini-batch with the decay factor 0.998.

Results We then apply the posterior regularization technique to mitigate the bias amplification in distribution. Results are demonstrated in Figures 4.2b (distribution) and 4.2d (top predictions). The posterior regularization effectively calibrates the bias in distribution and only 5 constraints are violated after the calibration. The average bias amplification is close to 0 ($\bar{A}$: 0.032 to -0.005). By reducing the amplification of bias in distribution, the bias amplification in top predictions also reduced by 30.9% ($\bar{A}$: 0.097 to 0.067). At the same time, the model's performance is kept (accuracy: 23.2% to 23.1%).

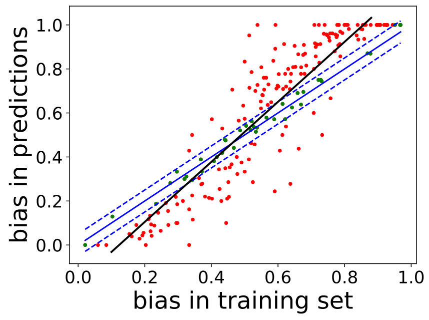
Note that calibrating the bias in distribution cannot remove all bias amplification in the top predictions. We posit that the requirement of making hard predictions (i.e., maximum a posteriori estimation) also amplifies the bias when evaluating the top predictions.



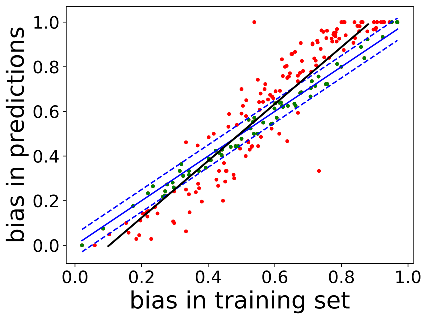
(a) bias in distribution before bias mitigation.



(b) bias in distribution after bias mitigation.



(c) bias in top predictions before bias mitigation.



(d) bias in top predictions after bias mitigation.

Figure 4.2: x-axis and y-axis are the bias toward male in the training corpus and the predictions, respectively. Each dot stands for an activity. The blue reference lines indicate the bias score in training is equal to that in test and the dash lines indicate the margin $(= 0.05)$. The dots in red stand for being out of margin and violating the constraints. The black lines are linear regressions of the dots. Results show that we can almost remove the bias amplification in distributions (see 4.2a and 4.2b), and reduce 30.9% amplification in top predictions (see 4.2c and 4.2d) after applying posterior regularization.

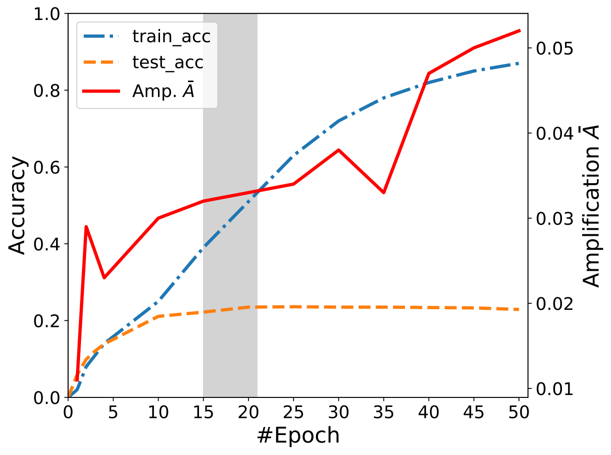


Figure 4.3: The curve of training and test accuracy, and bias amplification with the number of training epochs. The optimal model evaluated on the development set is found in the grey shade area.

CHAPTER 5

Application: Cross-lingual Dependency Parsing with Constraints Bridging the Word Order Gap between Languages

Prior work on cross-lingual dependency parsing often focuses on capturing the commonalities between source and target languages and overlooks the potential of leveraging linguistic properties of the languages to facilitate the transfer. In this research, we show that weak supervisions of linguistic knowledge for the target languages can improve a cross-lingual graph-based dependency parser substantially. Specifically, we explore several types of linguistic statistics and compile them into constraints to guide the inference process during the test time. In this application, we adapt the above two types of constraints and inference method, respectively.

5.1 Background:Graph-Based Parser

A graph-based parser learns a scoring function for every pair of words in a sentence and conducts inference to derive a directed spanning tree with the highest accumulated score. Formally, given the k-th sentence $\mathbf{x}_k = (x_{k1}, \ldots, x_{kL(k)})$ where $L(k)$ denotes the length of the k-th sentence, a graph-based parser learns a score matrix $S^{(k)}$, where $S^{(k)}_{ij}$ denotes the score to form an arc from word x_{ki} to word x_{kj}. Let y_k be an indicator function that $y_k(i,j) \in \{0,1\}$ denotes the arc from x_{ki} to x_{kj}. The maximum directed spanning tree inference can be formulated as an integer linear programming (ILP) problem:

$$y_k^* = \arg\max_{y_k \in \mathcal{Y}_k} \sum_{i,j} S_{ij}^{(k)} y_k(i,j), \tag{5.1}$$

where $\mathcal{Y}_k$ is the set of legal dependency trees of sentence k. In recent years, neural network approaches [KG16, WC16, KBK16, DM17] have been applied to modeling the scoring matrix $S^{(k)}$ and have achieved great performance in dependency parsing.

From the probabilistic point of view, if we assume for different i, j, the edge probabilities $P(y_k(i,j) = 1|\mathbf{x}_k)$ are mutually conditional independent, the probability of a whole parse tree can be written as

$$P(y_k|\mathbf{x}_k) = \prod_{i,j} P(y_k(i,j) = 1|\mathbf{x}_k)^{y_k(i,j)}. \tag{5.2}$$

If we set $S_{ij}^{(k)} = \log P(y_k(i,j) = 1|\mathbf{x}_k) + Z'_j$, where Z'_j is a constant term, then Eq. (6.3) can be regarded as the following maximum a posteriori (MAP) inference problem:

$$\begin{aligned} y_k^* &= \arg\max_{y_k \in \mathcal{Y}_k} P(y_k|\mathbf{x}_k) \\ &= \arg\max_{y_k \in \mathcal{Y}_k} \sum_{i,j} \log P(y_k(i,j) = 1|\mathbf{x}_k) y_k(i,j). \end{aligned} \tag{5.3}$$

5.2 Corpus-Level Constraints about Word Order

Given the inference problems as in equations (6.3) and (5.3), additional constraints can be imposed to incorporate expert knowledge about the languages to help yield a better parser. Instance-level constraints have been explored in the literature of dependency parsing, both in the monolingual [Dry07] and cross-lingual transfer [TMN13] settings. However, most word order features for a language are non-deterministic and cannot be compiled into instance-wise constraints.

In this work, we introduce corpus-wise constraints to leverage the non-deterministic features for cross-lingual parser. We compile the following two types of corpus-wise constraints based on corpus linguistics statistics:

- *Unary* constraints consider statistics regarding a particular POS tag (POS).
- *Binary* constraints consider statistics regarding a pair of POS tags (POS_1, POS_2).

Specifically, a unary constraint specifies the ratio r of the heads of a particular POS appears on the left of that POS.[1] Similarly, a binary constraint specifies the ratio r of POS_1 being on the left of POS_2 when there is an arc between POS_1 and POS_2.

The ratios r for the constraints are called corpus statistics, which can be estimated in one of the following ways: a) leveraging existing linguistics resources or consulting linguists; b) leveraging a higher-resource language that is similar to the target language (e.g., Finnish and Estonian) to collect the statistics. In this paper, we explore the first option and leverage the WALS features, which provide a reference for word order typology, to estimate the ratios.

Compile Constraints From WALS Features. For a particular language, once we collect the corpus-statistics of a pair of POS tags, we can formulate a binary constraint. There are different ways to estimate the corpus-statistics. For example, [Ost15] utilizes a small amount of parallel data to estimate the dominant word orders. In this paper, we simply utilize a small subset of WALS features that show the dominant order of some POS pairs (e.g. adjective and noun) in a language. They can be directly compiled into binary constraints.

Similarly, we can estimate the ratio for unary constraints based on WALS features. For a particular POS tag, we choose all WALS features related to it to formulate a feature vector f. The mapping from the vector f to the unary constraint ratio r is learnable: for each language with annotated data, we can get a WALS feature vector f_{lang} and a ratio r_{lang} from the annotation. We only need a small amount of data to estimate r_{lang} well. Given a set of languages with feature vectors and estimated ratios, we can learn the mapping by a simple linear regression, and apply it to estimate the ratio of any target language to

[1]The ratio for the head being on the right of that POS is thereby $1 - r$.

compile a unary constraint.

5.3 Formulations

Our general idea is that we use the word order corpus-level constraints on the *target language*, to transfer the model from *source language*. In this section, we formulate the constraints and inference into the forms in Sec.3.2,3.3 so that we can then apply LR and PR, respectively.

5.3.1 Model and Inference

The model we use is the graph-based parser introduced in Sec.5.1. This model can output a scoring matrix or distribution over instances. Output is a tree structure, which is formed as a set of arcs, and it is exponentially large.

The inference can be formulated as an ILP problem as Eq.(6.3), or in probabilistic view, an MAP inference as Eq.(5.3). In both of the cases the objective can be decomposed into arc-level.

5.3.2 Constraints

Each constraints C is about the ratio of particular two sets of arcs. We denote set C^+ is the arc set that it has the same direction defined in C, while C^- is the set of arcs in the opposite direction. Note that C^+ and C^- is mutual exclusive, and some arcs are not included in either set. The overall idea is that we want in prediction the arcs in C^+ has a ratio r among $C^+ \cup C^-$. Therefore, we define the corresponding arc-level constraint function on arc $x_i \to x_j$ as

$$h_{arc}(i,j) = \begin{cases} \frac{1}{L} & (i,j) \in C^+ \\ 0 & (i,j) \in C^- \\ \frac{r}{L} & (i,j) \notin C^+ \cup C^- \end{cases}, \tag{5.4}$$

where L is the length of the input sequence, and the instance-level constraint function is defined as

$$h(x, y) = \sum_{(i,j) \in y} h_{arc}(i, j).$$

The corpus-level constraint can be written as calibration form

$$|R(h, D) - r| \leq \theta,$$

where θ is the tolerance margin. We then can follow Sec.3.2 to transfer the model and do the constrained inference.

In probabilistic view, the distribution p_θ is trained on source language and we want to find a closest distribution satisfying constraints on the target language from p_θ measured by KL-divergence. The corpus-level distribution constraint can be written as

$$|\mathbb{E}_{\mathbf{y} \sim q}[h(\mathbf{x}, \mathbf{y})] - r| \leq \theta.$$

We then can follow Sec.3.3 to transfer the probabilistic model and do the inference.

5.4 Experimental Results

In this section, we evaluate the proposed algorithms by transferring an English dependency parser to 19 target languages covering 13 language families of real low-resource languages. We first introduce the experimental setup including data selection and constraint details and then discuss the results as well as in-depth analysis.

5.5 Setup

Model and Data We train the best performing Att-Graph parser proposed in [AZM19] on English and transfer it to 19 target languages in UD Tree Bank v2.2 [NAA18].[2] The

[2]We make the selection to prioritize the coverage of language families and low resource languages. The language family information can be found in Table 5.1.

Family	Lang.	Features	Baseline	Lagrangian Relaxation			Posterior Regularization		
				Oracle	WALS	ΔWALS	Oracle	WALS	ΔWALS
IE.Germanic	en	1,1,1	90.5	90.3	90.4	-0.1	90.4	90.6	+0.1
IE.Indic	ur	-1,-1,1	18.3	35.2	34.0	+15.7	35.0	33.7	+15.4
IE.Indic	hi	-1,-1,1	34.3	52.4	53.4	+19.1	51.3	49.1	+14.8
Dravidian	ta	-1,-1,1	36.1	42.8	43.4	+7.3	43.1	43.0	+6.9
Turkic	tr	-1,-1,1	31.2	35.2	37.1	+5.9	35.1	36.3	+5.1
Afro-Asiatic	ar	1, 1, -1	38.5	47.3	45.3	+6.8	45.8	43.7	+5.2
Afro-Asiatic	he	1, 1, -1	55.7	58.8	57.6	+1.9	58.3	57.6	+1.9
Austronesian	id	1, 1, -1	49.3	53.1	52.3	+3.0	52.3	51.9	+2.6
Korean	ko	-1,-1,1	34.0	37.1	37.2	+3.2	36.3	36.4	+2.4
IE.Celtic	cy	1, 1, -1	47.3	54.2	51.7	+4.4	53.8	50.0	+2.7
IE.Romance	ca	1, 1, -1	73.9	74.9	73.8	-0.1	74.9	74.7	+0.8
IE.Romance	fr	1, 1, -1	77.8	79.1	78.7	+0.9	79.0	79.0	+1.2
Uralic	et	1, -1,1	65.3	65.5	65.8	+0.5	65.7	66.0	+0.7
Uralic	fi	1, -1,1	66.7	67.1	67.0	+0.3	66.9	67.1	+0.4
IE.Slavic	hr	1, 1, 1	62.2	63.7	63.2	+1.0	63.6	63.4	+1.2
IE.Slavic	bg	1, 1, 1	79.6	79.7	79.2	+0.0	79.7	79.7	+0.1
IE.Baltic	lv	1, 1, 1	70.3	70.7	69.5	-0.8	70.5	69.9	-0.4
IE.Latin	la	?, ?, ?	47.4	48.0	45.6	-1.8	48.1	47.3	-0.1
IE.Germanic	da	1, 1, 1	76.6	76.6	76.5	-0.1	76.6	76.6	+0.0
IE.Germanic	nl	0, 1, 1	67.5	67.6	67.5	+0.0	67.9	67.9	+0.4
Average Performance			54.3	58.4	57.8	+3.5	58.1	57.5	+3.1

Table 5.1: Cross-lingual transfer performances for dependency parsing on 19 languages from 13 different families, with the performance on the source language (English) as a reference. Performances are reported per UAS (we observe similar trends for LAS). We compare the baseline model [AZM19] with our two algorithms (Lagrangian relaxation and posterior regularization) considering the oracle constraints, and the corpus-statistics constraints compiled from WALS. Columns △WALS denote the improvements bring by leveraging WALS feature as constraints. We also create a Features column to show three WALS features [83A,85A,87A] for each language. The values {1, -1, 0, ?} stand for the same as English, opposite to English, no dominant order, and feature missing, respectively.

model takes words and predicted POS tags[3] as input, and achieve transfer by leverag-

[3]We use predicted POS tags provided in UD v2.2.

ing pre-trained multi-lingual FastText [BGJ17] embeddings that project the word embeddings from different languages into the same space using an offline transformation method [STH17, LCR18]. The SelfAtt-Graph model uses a Transformer [VSP17] with relative position embedding as the encoder and a deep biaffine scorer [DM17] as the decoder. We follow the setting in [AZM19] to train and tune only on the source language (English) and directly transfer to all the target languages. We modify their decoder to incorporate constraints with the proposed constrained inference algorithms during the transfer phase without retraining the model.

Constraints We consider two types of constraints: 1) instance-level projective constraints for avoiding creating crossing arcs in the dependency trees, 2) corpus-statistics constraints constructed by the process described in Section 5.2. We consider the following three corpus-statistics constraints: $C1 = (NOUN)$, $C2 = (NOUN, ADP)$, $C3 = (NOUN, ADJ)$; intuitively, $C1$ concerns about the ratio of nouns being on the right of their heads; $C2$ concerns about the ratio of nouns being on the left of adpositions among all noun-adposition arcs; $C3$ concerns about the ratio of nouns being on the left of adjectives among all noun-adjective arcs.

For binary constraints, $C2$ and $C3$ can be directly compiled from WALS feature $85A$ and $87A$ respectively. We encode "dominant order" specified in WALS as the ratio being always greater than 0.75 (i.e., $r = 0.875$ and $\theta = 0.125$). If there is no dominant order or the feature is missing, we set $r = 0.5$ and $\theta = 0.25$. Some WALS features like $82A, 83A$ are also about word order, but we need to specify the arc types to utilize them. For simplicity, we only consider forming constraints from the POS tags in this paper. To estimate the ratio for unary constraint $C1$, we use the WALS features $82A, 83A, 85A, 86A, 87A, 88A, 89A$ that are related to $NOUN$ to form feature vectors, and do regression on languages in the test set except the target language to predict the constraint ratio. The process guarantees the target language remain unseen during the ratio estimation process. The ratios on the regression training languages are estimated by sampling 100 sentences in the training set per language.

We also consider an oracle setting where we collect a "ground-truth" ratio of each constraint for the target language to estimate an upper bound of our inference algorithms. In the oracle setting, we estimate the ratio on the whole training corpus of the target language and set the margin to $\theta = 0.01$.

5.6 Parsing Performances

We first compare the performances of the cross-lingual dependency parser with or without constraints. Table 5.1 illustrates the results for the 19 target languages we selected, along with the performance on the source language (English). The performance on English is not as high as the dependency parsers specialized for English, because to achieve transfer, we have to freeze the pre-trained multi-lingual word embeddings. Yet this parser achieved the best single-source transfer performances according to [AZM19].

As is shown in Table 5.1, the improvements by our constrained inference algorithms are dramatic in a few languages that have very distinct word order features from the source language. For example, the parsing performance of Hindi (hi) improves about 15% in UAS with WALS features via both Lagrangian relaxation and posterior regularization inference. The improvements are less obvious for languages that are in the same family as English such as Danish(da) and Dutch(nl). This is expected as the corpus linguistic statistics of these languages are similar to English thus the constraints are mostly satisfied with the baseline parser. Comparing Lagrangian relaxation and posterior regularization, we find posterior regularization being more robust and less sensitive to the errors in the corpus-statistics estimation, while Lagrangian relaxation gives a higher improvement on average. Overall, the two proposed constrained inference algorithms improved the transfer performance by 3.5% and 3.1% per UAS on average on 19 target languages.

For languages like Finnish (fi) and Estonian (et), the WALS setting works even better than the oracle. We suspect the reason being the large margin we set in the WALS setting.

When the estimated corpus-statistics is different from the real ratio in the test set, the large margin relaxes the constraints, thus could result in better performances.

Discussion. Despite the major experiments and analysis are conducted using English as the only source language, our approach is general and does not have restriction on the choice of the source language(s). To verify this claim, we run experiments with Hebrew as the source language. Under the oracle setting, Lagrangian relaxation and posterior regularization improve the baseline by 4.4% and 4.1%, respectively.

We observed that if we compile WALS features into hard constraints (i.e., set r = 0 or 1), the constraint inference framework only improves performance on half of the languages. For example, in Estonian (et), the performance *drops* about 3%. This is because WALS only provides the dominant order. Therefore, treating WALS as hard constraints introduces error to the inference.

Finally, we assume if we can access to native speakers, the corpus-statistics can be estimated by a few partial annotations of parse trees. In our simulation, using less than 300 arcs, we can achieve the same performance as using the oracle.

5.7 Contributions of Individual Constraints

We analyze the contribution of each constraint demonstrated in Table 5.2. Here we use the oracle setting to reduce the noise introduced by corpus-statistics estimation errors. The results are based on Lagrangian relaxation inference. As shown in Table 5.2, Despite some languages have non-projective dependencies, we observed performance improvements on almost all the languages when the projective constraint is enforced. All the constraints we formulated have positive contributions to the performance improvements. $C1 = (NOUN)$ brings the largest gain probably because its widest coverage.

Table 5.1 shows that the performance of Hindi improves from 34% to over 51% per UAS for both inference algorithms. To better understand where the improvements come from,

Model	UAS	coverage	Δ
baseline	54.3	N/A	N/A
+Proj.	54.6	N/A	+0.3
+Proj.+C1	57.0	0.24	+2.4
+Proj.+C2	55.7	0.08	+1.1
+Proj.+C3	55.0	0.07	+0.4
oracle	58.4	N/A	+4.1

Table 5.2: Ablation study: average UAS of baseline model with different sets of constraints. Proj. represent projective constraints. C1-C3 and oracle are introduced in 5.5. The improvements for projective constraint and oracle are compared to baseline. For the other three constraint sets the improvement is compared to model with projective constraint.

we conduct an analysis to breakdown the contribution of each individual constraint for Hindi. Table 5.3 shows the results. We can see that since the corpus linguistic statistics between Hindi and English are distinct, the baseline model only achieves low performance. With the constrained inference, especially the postposition constraint (C2), the proposed inference algorithm bring significant improvement.

To verify the effectiveness of the constraints, we analyze the relation between the performance improvements and corpus statistics ratio gaps between the source and the target languages. To quantify the ratio gap, we weight constraints by their coverage rate and compute the weighted average of the ratio difference between source and target languages. Results show that the performance improvement is highly related to the ratio gap. The Pearson Correlation Coefficient is 0.938. The figure is shown in Fig. 5.1

Const.	statistics	improvement
+Proj.	N/A	+0.1
C1	0.30/0.36/0.94	+6.9
C2	0.00/0.06/1.00	+11.3
C3	0.14/0.27/0.12	+0.5
All	N/A	+18.1

Table 5.3: Contribution of individual constraints and their statistics in Hindi. The second column lists the ratios estimated from oracle in English/ baseline in Hindi/ oracle in Hindi, respectively. The improvement is measured in UAS. The improvement of constraints is computed same as Table 5.2

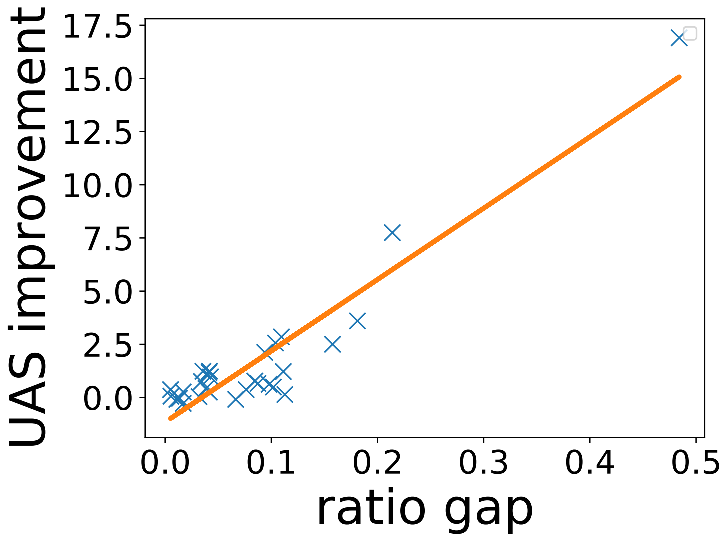


Figure 5.1: Ratio gap v.s. Δ perf.

Figure 5.2: The performance improvement is highly correlated to the difference in corpus linguistic statistics (estimated by weighted average ratio gaps in constraints) between target and source languages. (The Pearson Correlation Coefficient is 0.938.)

CHAPTER 6

An ILP Framework for Mining Constraints from Data

In Chap. 3 we discuss how to incorporate pre-defined constraints with neural models in inference. However, for some applications, manually identifying constraints is tedious. Besides, some constraints are obscure and cannot be easily identified by humans. For example, if we shuffle columns of all sudoku puzzles with the same order, the puzzles still follow a set of constraints, while it is hard for humans to recognize the underlying rules. Inspired by representation learning methods automate feature extraction, we envision that *an artificial intelligence system that could automatically recognize underlying constraints among output labels from data and incorporate them in the prediction time.*

Inspired by the great success of ILP in constrained output structure predictions, we propose a novel framework to formulate the constraint learning based on ILP. In particular, we estimate the feasible set defined by the constraints and explore three techniques: 1) mining inequality constraints to form a superset of the feasible set by constructing an outer polytope based on seen data; 2) mining equality constraints by dimension reduction of the superset; and 3) latent variable method to deal with complex constraints or leverage prior knowledge. We also propose an algorithm to induce the subset of the feasible set to help evaluate the quality of the constraints. Note that despite the constraint mining algorithm is designed under the ILP framework, our algorithm does not involve solving ILP when mining the constraints.

6.1 Mining Constraints with Integer Linear Programming

We first review the constraints mining framework based on ILP. We then propose an approach to mine constraints by estimating the outer and inner polytopes of the feasible set. Finally, we discuss how to extend the framework to capture complex constraints.

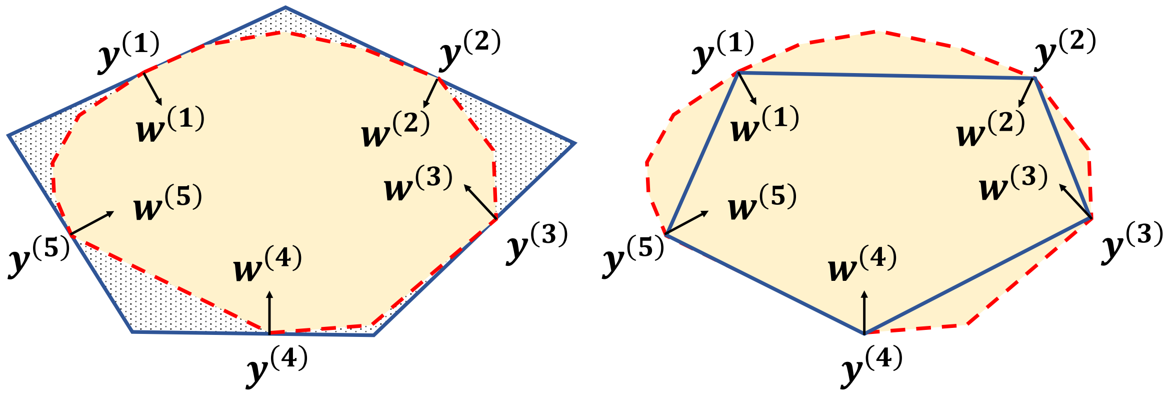
ILP is a linear optimization problem with linear constraints and the values of variables are restricted to integers. Formally, the ILP problem can be formulated as

$$\max_{\mathbf{y}\in\mathbb{Z}^d} \quad \mathbf{w}\cdot\mathbf{y} \qquad \text{s.t.} \quad A\mathbf{y} \leq \mathbf{b}, \tag{6.1}$$

where $\mathbf{w} \in \mathbb{R}^d$ is the coefficients of the objective function (a.k.a. weights) and $\mathbf{y}$ is an integer vector that encodes the output label.[1] The matrix A and vector $\mathbf{b}$ specify the constraints. We use S^* to denote the feasible set defined by the constraints. Various structure prediction problems can be casted into the ILP formulation. For example, dependency parsing can be formulated as finding the maximum spanning tree in a directed graph [MPR05], where each node represents a word and the edge w_{ij} represents how likely the word i is the dependent of the word j predicted by a model. $\mathbf{y} = \{\mathbf{y}_{ij}\}, \mathbf{y}_{ij} \in \{0,1\}$ is the indicator of the edges in the resulting tree. The objective in Eq. (6.1) then can be interpreted as the total score of edges in $\mathbf{y}$, and the constraints, described by $(A, \mathbf{b})$, restrict $\mathbf{y}$ to be a tree [MSX09].

Prior works (see, e.g., [MSX09]) mostly assume the constraints $(A, \mathbf{b})$ are given. However, in this paper, we assume $(A, \mathbf{b})$ are unknown and our goal is to identify the underlying feasible set S^* spanned by $(A, \mathbf{b})$ using a set of objective-solution pairs $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{k}$ that satisfy constraints defined by $(A, \mathbf{b})$. For example, in MST, giving a set of weights $\mathbf{w}^{(i)}$ (adjacency matrix) with the corresponding optimal solution $\mathbf{y}^{(i)}$, our algorithm identifies the structure of the output $\mathbf{y}$ form a tree structure.

[1] In structure output prediction, usually each element of $\mathbf{y}$ takes value 1 or 0, indicating if a specific value is assign to a specific output variable or not. $\mathbf{w}$ are the scores of sub-components of output assigned by a model.



(a) Outer polytope (S_O) (b) Inner polytope (S_I)

Figure 6.1: The pentagons (blue solid line) in Fig. 6.1a and 6.1b show the outer and the inner polytopes of 5 training samples $\{\mathbf{w}_i, \mathbf{y}_i\}_{i=1}^5$ (see Sec. 6.1.1). The dashed red line shows the boundary of the feasible set (yellow region). We also show w_i as the normal vector of the outer line.

In the following, we introduce algorithms to estimate the feasible set for mining the underlying constraints. These mined constraints define an superset of the feasible set. We also design an algorithm to get the subset of the feasible set to evaluate the estimation.

6.1.1 Mining Inequality Constraints

In the following, we discuss how to estimate the underlying feasible set S^* associated with inequality constraints. Our approach finds a convex hall S_O defined by a set of learned inequality constraints that is an outer polytope (i.e., superset) of the feasible set S^*. We also purpose a method to get an inner polytope S_I that is a subset of S^* and use the gap between the S_O and S_i to estimate the quality of approximation. Figure 6.1 shows an example about S_I, S_O defined by 5 training samples in a 2-dimensional space. We denote $S_I^{(i)}, S_O^{(i)}$ as the inner and outer polytopes after considering the first i samples.

Outer polytope We first introduce how to identify S_O. Assume that we already know part of the constraints $A', \mathbf{b}'$. We initialize the outer polytope as $S_O^{(0)} = \{\mathbf{y} \in \mathbb{Z}^d \mid A'\mathbf{y} \leq \mathbf{b}'\}$

(if $A', \mathbf{b}'$ are empty, $S_O^{(0)} = \mathbb{Z}^d$). For every training sample $(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$, we consider adding the following constraint to the outer polytope

$$\mathbf{w}^{(i)} \cdot \mathbf{y} \leq \mathbf{w}^{(i)} \cdot \mathbf{y}^{(i)}. \tag{6.2}$$

Since $\mathbf{y}^{(i)}$ is the optimal solution under weight $\mathbf{w}^{(i)}$, all the points in the feasible set must sit in the half-space defined by Eq. (6.2), otherwise $\mathbf{y}^{(i)}$ is not the optimal solution. We have $S_O^{(i)} = \{\mathbf{y} \in S_O^{(0)} \mid \mathbf{w}^{(j)} \cdot \mathbf{y} \leq \mathbf{w}^{(j)} \cdot \mathbf{y}^{(j)},\ j = 1, 2, \ldots, i\}$, and $S^* \subseteq S_O = S_O^{(k)} \subseteq \cdots \subseteq S_O^{(1)} \subseteq S_O^{(0)}$.

The outer polytope $S_O^{(i)}$ (intuitively, the upper bound of the feasible set S^*) is tight when we only observe the first i samples. That is, if we query $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(i)}$, we will find $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(i)}$ are (one of) the optimal solutions. Therefore, $S_O^{(i)}$ is a possible feasible set. Since $S^* \subseteq S_O^{(i)}$, this bound is tight. This shows that without any further assumption, we cannot do better than S_O for estimating the outer polytope of the feasible set S^*.

In the test time, we are requested to conduct inference with unseen input weight $\mathbf{w}^{(q)}$. Since all constraints in S_O are linear, we solve the following ILP problem

$$\begin{aligned} \max\nolimits_{\mathbf{y}\in\mathbb{Z}^d}\ \mathbf{w}^{(q)} \cdot \mathbf{y} \text{ s.t. } & \left[A'^T\ \ \mathbf{w}^{(1)}\ \ \ldots\ \ \mathbf{w}^{(k)}\right]^T \mathbf{y} \\ & \leq \left[\mathbf{b}'^T\ \ \mathbf{w}^{(1)} \cdot \mathbf{y}^{(1)}\ \ \ldots\ \ \mathbf{w}^{(k)} \cdot \mathbf{y}^{(k)}\right]^T. \end{aligned} \tag{6.3}$$

The objective value of the solution of Eq. (6.3) might be higher than the optimum as the solution might not satisfy all the underlying constraints. We will show that empirically the outer polytope can approximate the feasible set effectively in Sec. 6.2. Although the number of constraints grows linearly with the number of training samples, we find that empirically the inference time does not grow much.[2]

[2] In structure output prediction, constraints are often associated with only the problem structure. Therefore, all the inference instances share the same constraint set, and the overhead in solving ILPs is amortized [SKR12, KSR13, CUK15].

Inner polytope To understand the quality of S_O, we also construct the inner polytope S_I, then we can use the gap between S_O and S_I to estimate the quality of the approximation. We first initialize $S_I^{(0)} = \emptyset$. For every training sample i: $(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$, we set $S_I^{(i)} = convex\_hull(\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(i)}\})$, and then $S_I^{(i-1)} \subseteq S_I^{(i)}$. Since all $\{\mathbf{y}^{(i)}\}$ are in the feasible set that is convex, all the convex hulls must be subsets of the feasible set. Therefore, we have $S_I^{(0)} \subseteq S_I^{(1)} \subseteq \cdots \subseteq S_I^{(k)} = S_I \subseteq S^*$.

Similarly, we can prove that $S_I^{(i)}$ is a possible feasible set after observing the first i samples, which means as a lower bound, $S_I^{(i)}$ is also tight.

When we conduct inference with S_I, we examine every vertex of the convex hull and choose the one with the optimal objective. Since it is an inner polytope of the feasible set, the solution is guaranteed to satisfy all constraints, and the objective value can be lower than the optimum. Although inner polytope and outer polytope methods are two separate algorithms, the gap between their objective function value and the size of the feasible set provide an estimation of the tightness of the bound.

In Sec. 6.3.1 we will show that, empirically, this approach converges with reasonable number of training samples and running time is discussed in Sec. 6.3.2.

Dealing with predicted weights When we incorporate the proposed approach with a structured prediction model, the weights $\mathbf{w}$ are predicted by a base model. In this situation, the predicted weights $\mathbf{w}$ can be noisy and the corresponding label $\mathbf{y}$ may not be the optimal solution to Eq. (6.3). As the result, the outer polytope may not contain some feasible solutions as they are filtered out later by the algorithm. To handle the noise, we adapt Eq. (6.2) to

$$\mathbf{w}^{(i)} \cdot \mathbf{y} \leq \mathbf{w}^{(i)} \cdot \mathbf{y}^{(i)} + \xi_i, \; i \in [k], \tag{6.4}$$

where ξ_i is a slack variable to ensure every training point $\mathbf{y}_i \in S^*$ satisfies Eq. (6.4)

$$\xi_i = \min_{j \in [k]} \{\mathbf{w}^{(i)} \cdot \mathbf{y}^{(j)} - \mathbf{w}^{(i)} \cdot \mathbf{y}^{(i)}\}.$$

6.1.2 Mining Equality Constraints

When there are equality constraints in the output label space. Effectively, the dimension of the output space is reduced. However, the dimension of the set S_O is the same as that of $\mathbf{w}$ and $\mathbf{y}$. Therefore, this inspires us to find the sub-space of S_O to further tighten the feasible set.

For example, in the MST problem the number of edges we select is exact $N-1$ where N is the number of nodes. Formally, the linear constraint $\mathbf{1} \cdot \mathbf{y} = N-1$ holds for every feasible point $\mathbf{y}$.

We denote this $d'-$dimensional sub-space as $S_D = \{\mathbf{y} \mid W_{eq} \cdot \mathbf{y} = \mathbf{c}\}$. We can obtain $W_{eq}, \mathbf{c}$ by solving the kernel of $[\mathbf{y}^T, \mathbf{1}]$, which is

$$\begin{bmatrix} \mathbf{y}_1^T & \mathbf{y}_2^T & \dots & \mathbf{y}_n^T \\ 1 & 1 & \dots & 1 \end{bmatrix}^T \begin{bmatrix} W_{eq}^T \\ -\mathbf{c}^T \end{bmatrix} = \mathbf{0}. \tag{6.5}$$

With the sub-space S_D, the intersection of S_O and S_D is used to replace S_O as the outer polytope of the feasible set: $S_I \subseteq S^* \subseteq S_O \cap S_D = \{\mathbf{y} \in S_O \mid W_{eq}\mathbf{y} = \mathbf{c}\}$. For the reliability of this algorithm, we give two lemmas:

Lemma 1: If there is an underlying equality constraint $W_{eq} \cdot \mathbf{y} = c$, our algorithm can find it.

Proof Sketch: For any underlying equality constraint $W_{eq} \cdot \mathbf{y} = c$, all the labels of training points $\mathbf{y}_i$ should satisfy it and it must be in the kernel of Eq. (6.5).

Lemma 2: For an equality constraint given by this algorithm, the probability that this constraint does not hold for the optimal solution of a random query is less than $\frac{1}{eM}$, where M is the number of training points.

Proof Sketch: We use D_w to denote the domain of the query (weights), and $f^*(\mathbf{w})$ as the ground truth solution for the query $\mathbf{w}$. We denote the equality constraint learned by the algorithm is $g(\mathbf{y}) = 0$. Therefore, all the data $\mathbf{y_i}$ in the training data satisfy $g(\mathbf{y}_i) = 0$. We let $p = Pr_{\mathbf{w} \sim D_w}(g(f^*(\mathbf{w}) = 0)$. The probability of all the training data satisfying $g(\mathbf{y}_i) = 0$ is p^M. Thus, the probability that this constraint does not hold for a random query solution is

$$p^M(1-p) \leq \frac{M^M}{(M+1)^{M+1}} < \frac{1}{eM}.$$

6.1.3 Latent Variables

Some prediction problems involve constraints with complex logics and require auxiliary variables to model the problem structure. Thanks to the flexibility of the ILP framework, we can introduce latent variables to extend the expressiveness of the constraint mining framework:

$$\max_{\mathbf{y} \in \mathbb{Z}^d} \quad \mathbf{w} \cdot \mathbf{y} \qquad \text{s.t.} \quad A_{pre} \begin{bmatrix} \mathbf{y} \\ \mathbf{h} \end{bmatrix} \leq \mathbf{b}_{pre}, \quad A \begin{bmatrix} \mathbf{y} \\ \mathbf{h} \end{bmatrix} \leq \mathbf{b}, \tag{6.6}$$

where $\mathbf{h}$ are latent variables, and they appear in the constraints but not in the objective function in Eq. (6.6). Despite that $\mathbf{h}$ is not part of the output, it facilitates to formulate the ILP problem. In general, a set of pre-defined constraints $(A_{pre}, \mathbf{b}_{pre})$ are given to describe the relations between $\mathbf{y}$ and $\mathbf{h}$. Then, given a set of $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^k$, our goal is to learn the constraints $(A, \mathbf{b})$.

We then adapt the method in Sec 6.1.2 to solve the kernel of the matrix $[\mathbf{y}^T, \mathbf{h}^T, \mathbf{1}]^T$. In this way, we can mine equality constraints with respect to $\mathbf{h}$ and then derive the outer polytope S_O. Since $\mathbf{h}$ is determined by the variables $\mathbf{y}$, adding constraints on $\mathbf{h}$ also reduces the size of S_O.

For example, consider multi-label classification with output $\mathbf{y}$, where $y_i, i = 1 \ldots m$ is a binary indicator of class i. If we would like to identify the constraints between pairs of labels from $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})\}$, we can introduce a set of latent variables $\{h_{i,j,b_1,b_2}\}_{i,j=1\ldots m; b_1,b_2 \in \{0,1\}}\}$ with pre-defined equality constraints $h_{i,j,b_1,b_2} = (y_i = b_1) \wedge (y_j = b_2), \forall i, j, b_1, b_2$. They can be further formulated as

$$\forall i, j, \begin{cases} h_{i,j,1,0} + h_{i,j,1,1} = y_i \\ h_{i,j,0,1} + h_{i,j,1,1} = y_j \\ \sum_{b_1,b_2 \in \{0,1\}} h_{i,j,b_1,b_2} = 1. \end{cases}$$

By introducing h_{i,j,b_1,b_2}, we are able to capture some correlations between labels. For example, label i and label j cannot be positive at the same time can be represented by a constraint $h_{i,j,1,1} = 0$. Therefore, the latent variables make the constraint learning framework more expressive.

6.2 Experiments

We experiment on two synthetic problems, 9×9 Sudoku and minimal spanning tree (MST), to show that the proposed methods can capture different kinds of constraints. We then incorporate the proposed technique with a feed-forward neural network in a hierarchical multi-label classification problem. For all the experiments, we use the Gurobi v8.1.1 [Gur16] as the ILP solver.[3]

6.2.1 Sudoku

In Sudoku, given a 9×9 grid with numbers partially filled in, the player is requested to fill in the remaining of the grid with constraints that each 3×3 sub-grid, each column and each row must contain all the numbers of $1, 2, \ldots, 9$. The size of the feasible set is 6.67×10^{21} [FJ05] which is extremely large. Our goal is to use the proposed method to solve Sudoku puzzles without telling the model the rules.

We follow the experiment setting in [WDW19] to represent the solution of Sudoku as a vector $\mathbf{y} \in \{0, 1\}^{729}$, where y_{ijk} denotes the $i-$th row $j-$th column is the number k or not.

[3]We configure the ILP solver such that it outputs optimal solution (i.e., set $MIPGap = 0$). The relaxed LP will be discussed in Sec. 6.3.2.

Table 6.1: Sudoku results. EQ stands for equality constraint mining. Performance is reported in entry-level accuracy. Our approaches can successfully identify the underlying Sudoku constraints.

(a) Original Sudoku

Model	Train	Test
ConvNet [Par18]	72.6%	0.04%
ConvNetMask [Par18]	91.4%	15.1%
SATNet [WDW19]	99.8%	98.3%
Outer + EQ (ours)	100%	100%

(b) Permuted Sudoku

Model	Train	Test
ConvNet [Par18]	0%	0%
ConvNetMask [Par18]	0.01%	0%
SATNet [WDW19]	99.7%	98.3%
Outer + EQ (ours)	100%	100%

The partially filled entries $(r_i, c_i) = n_i$ (i.e., row r_i column c_i is number n_i) are encoded in $\mathbf{w} \in \{0, 1\}^{729}$, where we set the corresponding weight for $w_{r_i c_i n_i}$ to be 1 and the rest to be 0. In this way, maximizing the objective function $\mathbf{w} \cdot \mathbf{y}$ guarantees $y_{ijk} = 1$ if $w_{ijk} = 1$. Then given pairs of $\{\mathbf{w}, \mathbf{y}\}$, our method mind the underlying constraints A and $\mathbf{b}$ in Eq. (6.3).

We experiment on the dataset introduced in [WDW19]. The dataset contains $9,000$ training and $1,000$ test samples, each of which has a unique solution. We also conduct experiments in the permuted setting [WDW19], where a pre-defined permutation function is used to shuffle the 9×9 grid. In the permuted setting, it is almost impossible for humans to identify the underlying rules, despite the puzzle is still filled in a certain order. We follow the configuration in [WDW19] to compare our approach with a convolution neural network for Sudoku [Par18] (ConvNet) and SATNet [WDW19] and report our results along with their published results in Table 6.1.

As shown in the table, by using the equality constraints mining technique, our framework can realize the Sudoku rules and achieve 100% accuracy. The constraints we mine reduce the size of candidate solution space from 2^{729} to 6.67×10^{21} (i.e., the number of feasible Sudoku puzzles). Note that our approaches, as well as SATNet, do not utilize the position clues in the data. Therefore, it is not affected by permutations.

In fact, the equality constraints we learn are exactly the rules of Sudoku. The Sudoku rules can be represented linearly as

$$\begin{gathered}\sum_{i=1}^{9} y_{kij} = 1,\ \sum_{i=1}^{9} y_{jki} = 1,\ \forall j, k, \\ \sum_{i=1}^{3}\sum_{j=1}^{3} y_{(x+i)(y+j)k} = 1, \forall x, y \in \{0, 3, 6\} \forall k.\end{gathered} \tag{6.7}$$

We use S^* to denote the space defined by Eq. (6.7) and $\hat{S}$ as the space of our constraints. We verify the $S^* = \hat{S}$ by

1. Verifying that each constraint in S^* is indicated by the constraints we learn. This property guarantee that $S^* \subseteq \hat{S}$.
2. Comparing the dimension of S^* and S. We find that $d(S^*) = d(\hat{S}) = 249.$

In this way, we confirm that our framework can mine the underlying Sudoku rules successfully.

6.2.2 Minimal Spanning Tree

As discussed in Sec. 2.2, inference problems in many structured output prediction applications (e.g., dependency parsing) can be modeled as searching the minimal (or maximal) spanning tree (MST). In the following, we verify if the proposed approach can identify the structure of solution is a tree by merely providing pairs of adjacency matrices and the corresponding MST. We encode the MST problem as described in Sec. 6.1.

Table 6.2: MST results in exact match (EM) accuracy in Train and Test, the average edge accuracy (Edge) and the ratio of solutions that are feasible (Feasibility). Here, the feasibility means the solution forms a tree.

(a) $10,000$ training samples

Model	Train EM	Test EM	Test Edge	Test Feasibility
NN	8.3%	6.9%	89.3%	12.8%
Inner	100%	41.6%	93.7%	100%
Outer	100%	71.1%	96.1%	71.1%
Outer+EQ	100%	72.9%	96.1%	72.9%

(b) $20,000$ training samples

Model	Train EM	Test EM	Test Edge	Test Feasibility
NN	9.5%	10.4%	91.1%	13.4%
Inner	100%	69.2%	97.0%	100 %
Outer	100%	87.2%	98.0%	87.2%
Outer+EQ	100%	91.8%	98.7%	91.8%

We generate a dataset with 7 nodes. The dataset contains $20,000$ training and 500 test data. Every data point consists of an adjacency matrix serialized in a vector $\mathbf{w}$, where $\mathbf{w}_{i,j}$ represents the distance between node i and node j, and its corresponding MST. The entry in the adjacency matrix is independently sampled from a uniform distribution in $[-1, 1]$. We filtered out the adjacency matrix with identical values to ensure every sample has a unique optimal solution.

We test the inner polytope (Inner), outer polytope (Outer) and outer polytope with equality constraint mining (Outer+EQ) methods. We compare our approaches with fully connected feed-forward neural networks (NN) with $1, 2$ or 3 layers, which directly learn the association between $\mathbf{w}$ and $\mathbf{y}$ and the hyper-parameters are given in the Appendix. We set hidden dimension to be 50. Despite that our methods do not have hyper-parameters, to

Table 6.3: Detailed results (grey zone in Fig. 6.2) evaluated by exact match accuracy (EM), the ratio of solutions that are feasible (Feasibility), and average accuracy of label assignments (Label Acc.).

Model	Test EM	Test Feasibility	Test Label Acc.
Baseline	62.6%	73.8%	98.92%
Inner	79.8%	100%	98.98%
Outer	71.3%	89.7%	98.85%
Outer+Latent	79.8%	100%	98.98%

tune the neural network, we generate another 500 dev data points.

Table 6.2 shows the results in exact match (i.e., correct MST) and edge accuracies. The results show that Baseline-NN is unable to learn the tree structure from the given examples. We find that NN can learn reasonably well in each individual edge but is terrible to capture the output is a tree. In particular, in 87.2% and 86.6% of cases for 10,000, 20,000 training samples, respectively, the output by NN is not feasible (not a tree). Therefore, its exact match accuracy is low. In comparison, the proposed approaches Outer and Outer+EQ mostly produce feasible solutions[4], resulting in much higher exact match accuracy. Comparing the results for $10,000$ and $20,000$ data points, we find that Outer+EQ is more effective than NN when doubling the training data as it improves 20% exact match accuracy.

6.2.3 Hierarchical Multi-label Classification

Finally, we apply the proposed approaches to a real-world problem and demonstrate its ability to cooperate with machine learning models. We conduct experiments on ImCLEF07A [DKL11], which contains $10,000$ training samples and $1,006$ test samples. Each sample has

[4]Note that for Outer and Outer+EQ methods, the results of feasibility are the same as test EM because all feasible trees are in the outer polytope. Therefore, if a model outputs a feasible tree, the tree is guaranteed to be optimal.

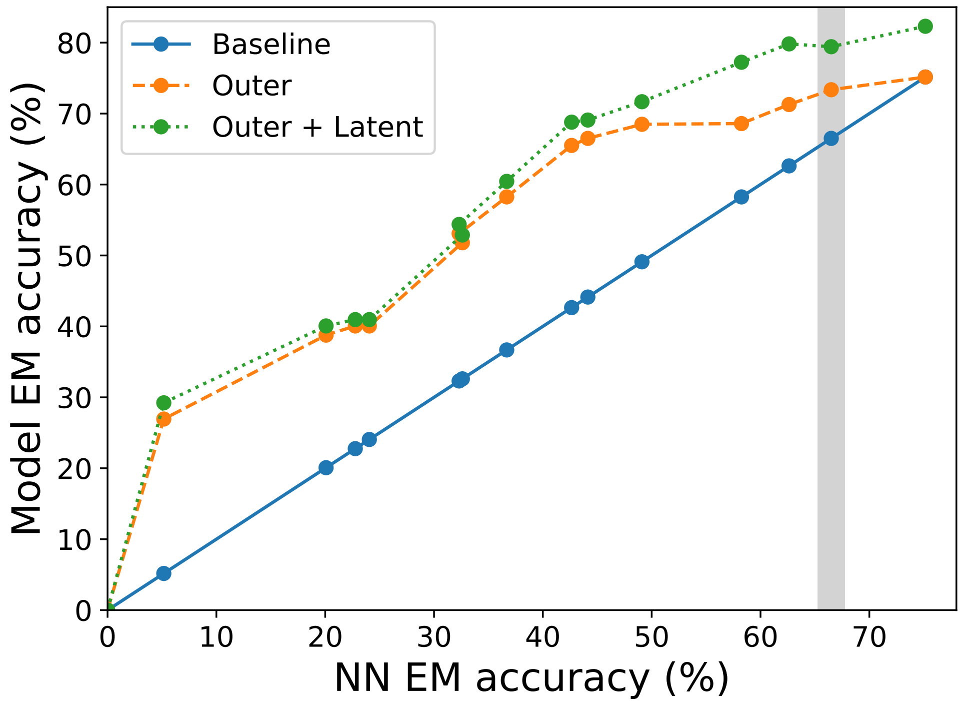


Figure 6.2: Results on ImCLEF07A in EM accuracy using base models with different performance levels.

80 features and a set of labels selected from 96 classes. There is a hierarchy among the labels and the depth of the hierarchical structure is 4. A feasible label set forms a path from the root to a leaf node.

The base model is a 3-layer fully connected feed-forward neural network with hidden dimension 80. This model outputs a vector $\mathbf{c}$, where each component $c_i \in [0, 1]$ is predicted independently to the input instance. For the baseline model, if $c_i > 0.5$, then the label i is positive.

We take the base model as a sub-routine and use it to assign weight $\mathbf{w}$ in Eq. (6.3). Specifically, $\mathbf{w} = \mathbf{c} - 0.5 \times \mathbf{1}$. Without constraints, solving the ILP in Eq. (6.3) is equivalent to make predicton by the baseline model. We evaluate 1) the inner polytope method (Inner), 2) the outer polytope method (Outer) and 3) the outer polytope method with latent variables (Outer+Latent). In Outer, as $\mathbf{w}$ is generated by a predicted model, we use Eq. (6.4) to allow noise. In Outer+Latent, we use the label pairwise latent variables defined in Sec 6.1.3. To reduce the label spaces, we follow the convention to consider only induce latent variables to label pairs that occur in the training set.

Different from Sec. 6.2.1 and 6.2.2, the weight $\mathbf{w}$ is a score vector predicted by the base model. To understand how the constraint mining approaches incorporate with base models with different performance levels, we train multiple version of base models with different number of layers and training epochs then demonstrate the performance of our approach with these base models. The hyper-parameters for these models are given in the Appendix. The results are shown in Fig. 6.2. Results show that Outer and Outer+Latent improve the base models in all cases. Even with a weak base model with only 10% in exact match accuracy, Outer and Outer+Latent are able to learn underlying constraints and improve the performance by more than 20%. The different between Outer and Outer+Latent is not apparent when the base model is inaccurate. However, when the base model performance increases, Outer+Latent is capable of capturing more fine-grained constraints than Outer and achieves better performance. When the baseline achieves 0 loss in training data (the right-most column points), the constraints learned by Outer can not filter out any point in the space. Therefore, Outer achieves the same performance as *Baseline*. However, Outer+Latent can still mine constraints related to the latent variable and improve the performance.

Table 6.3 highlights the detailed results with one base model.[5] Outer improves Baseline about 12% in exact match accuracy and 16% in feasibility. This demonstrates Outer can successfully filter out many infeasible solutions and guide the model to find the correct ones. Inner learns exactly the feasible set as all pairs of labels appear in the test set also appear in the training. Similarly, Outer+Latent is able to identify all the dependencies be-

[5]We choose the second best baseline model since the best one gets 100% accuracy in training set, which causes the inequality constraints learned by Outer method filter out nothing.

tween labels and achieves high performance. The classes in this task have a tree structure and it has depth 4 including the root (root is a virtual concept that it is not a real class). We use $p(x)$ to denote the parent class of class x, and L_i to denote the set of classes on layer i, $i \in \{1, 2, 3\}$. We verify the constraints with the same method in Sec. 6.2.1. The mined equality constraints are the linear transformation of the following constraints:

$$\begin{aligned} &\sum\nolimits_{j\in L_i} y_j = 1, \forall i \in \{1,2,3\}, \\ &h_{parent(x),x,0,1} = 0, \forall x : parent(x) \neq root. \end{aligned} \tag{6.8}$$

6.3 Analysis and Discussion

6.3.1 Feasible Set Size Analysis

We provide a theoretical analysis about the convergence speed of the proposed approaches by estimating the cardinality of the outer polytope and inner polytopes. Our methods in Sec. 6.1 estimate the feasible set by squeezing the outer polytope and enlarging the inner polytope. We analyze how the sizes of outer polytope and inner polytope change with respect to the number of training samples.

We denote the size of ground truth feasible set S^* as M, the size of universal label space is N. The weights $\mathbf{w}$ are drawn from the distribution $D_{\mathbf{w}}$. For each point i, we use p_i to denote the probability that given a randomly sampled weight $\mathbf{w} \sim D_{\mathbf{w}}$, $\mathbf{y}^{(i)}$ get higher score than all the feasible points. Formally, $p_i = P_{\mathbf{w}\sim D_{\mathbf{w}}}\{\mathbf{w}\cdot\mathbf{y}^{(i)} \geq \mathbf{w}\cdot\mathbf{y}^{(j)}, \forall j \in S^*\}$. The following lemma bounds the expectation sizes of the outer and inner polygons. Full proof is in the Appendix.

Lemma 3: The expectation of the sizes of outer and inner polygon is given by

$$\begin{aligned} \mathbb{E}[|S_I|] &= M - \sum\nolimits_{i\in S^*} (1-p_i)^k ; \\ \mathbb{E}[|S_O|] &= M + \sum\nolimits_{j\notin S^*} (1-p_j)^k, \end{aligned} \tag{6.9}$$

Proof sketch: We first consider the outer polygon. The point j out of the feasible set appears in the outer polygon if and only if it is not filtered out by any constraints, which is $(1-p_j)^k$.

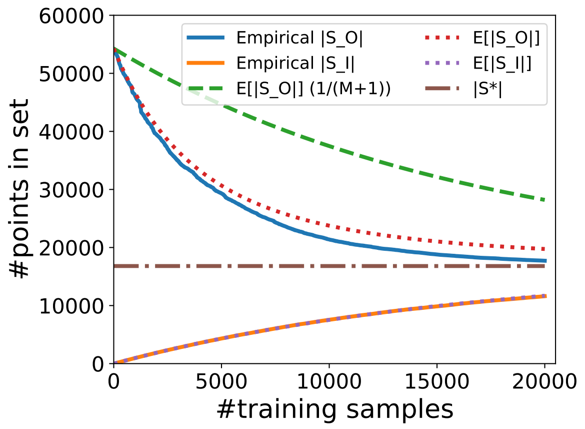
We then consider the inner polygon. The point i appears in the inner polygon if and only if at least one training sample takes it as the optimal solution, which is $1-(1-p_i)^k$.

Case study: MST We take MST discussed in Sec.6.2.2 as an example. According to Matrix-Tree Theorem [CK78], we know that the number of spanning trees is $M = 16,807$. The universal set size before mining equality constraints is $N_O = 2^{21} = 2,097,152$. However, with the equality constraints mining method, we can identify the constraint $\sum_{i=1}^{21} y_i = 6$, (i.e., number of edges is 6). With this constraint the size of the space is reduced to $N = \binom{21}{6} = 54,264$.

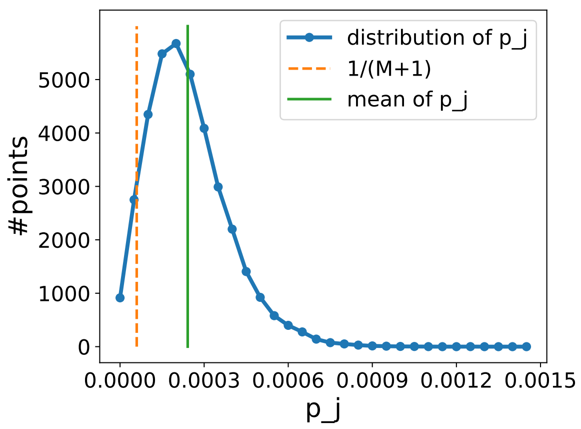
p_i in Eq. (6.9) is difficult to estimate directly; therefore, we approximate it by $p_i = \frac{1}{M}$ for $i \in S^*$ and $p_j = \frac{1}{M+1}$ for $j \notin S^*$. The approximation is exact if the following assumption hold (see details and proof in Appendix). *Data symmetric*: for k different points $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(k)}$, $P_{\mathbf{w}\sim D_{\mathbf{w}}}\{\mathbf{w}\cdot\mathbf{y}^{(1)} \geq \mathbf{w}\cdot\mathbf{y}^{(i)}, i \in [k]\} = \frac{1}{k}$. MST only satisfies the part of the assumption, therefore, for $i \in S^*$, the approximation $p_i = \frac{1}{M}$ is close, and there is a gap between empirical $p_j (j \notin S^*)$ compared with $\frac{1}{M+1}$ (see Fig. 6.3b), this causes the gap between the empirical result and the estimated expectation about the outer polytope. Fig. 6.3a shows the empirical sizes of outer and inner polytope $|S_O|, |S_I|$, with their theoretical expectation in Eq. (6.9) and the ground truth $|S^*|$. The expectation of Inner perfectly fits the empirical results and the two curves are almost coincide. Both of the Outer and Inner method eventually converge to the ground truth, and the Outer method is closer.

6.3.2 Discussion about Running Time

Table 6.4 shows the training and test running time of our approaches. For our approach, the training time refers to the time required for identifying the feasible set. As shown in Sec 6.1, our approach only needs to pass all samples once, and the complexity is linear



(a) Set size



(b) The dist. of $p_j, j \notin S^*$

Figure 6.3: On the left Fig. 6.3a shows the empirical and expectation sizes of outer and inner polytope, comparing with ground truth. $E[|S_O|](1/(M+1))$ is the expectation of outer polytope size estimated by $p_j = \frac{1}{M+1}$ while $E[|S_O|]$ is estimated with empirical p_j. Note that the empirical inner polytope and its theoretical curve almost coincide. On the right Fig. 6.3b shows the distribution of p_j estimated from $20,000$ training data comparing with $1/(M+1)$.

in terms of the number of constraints and samples. Therefore, our approach is more efficient than the baseline neural network in the Sudoku and MST experiments. For the HMC example, the training time of our approach includes updating the model parameters of the underlying neural networks. Therefore, the training time is longer compared with the Sudoku and MST cases.

For the inference time, we report the average time on solving one test sample. Despite ILP is NP-hard, a commercial solver (e.g., Gurobi) is capable of solving the problem within a reasonable time. Therefore, without carefully engineering to optimize the running time, the ILP solver can produce solutions within a few seconds.

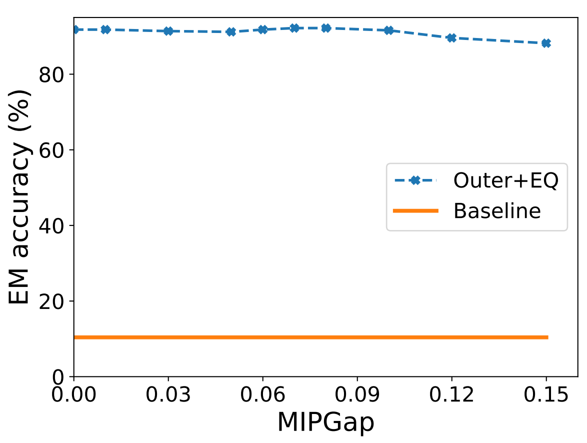
Note that despite the constraints are mined using the ILP framework, it does not mean that the inference has to be solved by an ILP solver. Once the constraints are identified,

Table 6.4: Running time in seconds for experiments. The time is computed by averaging in 3 runs.

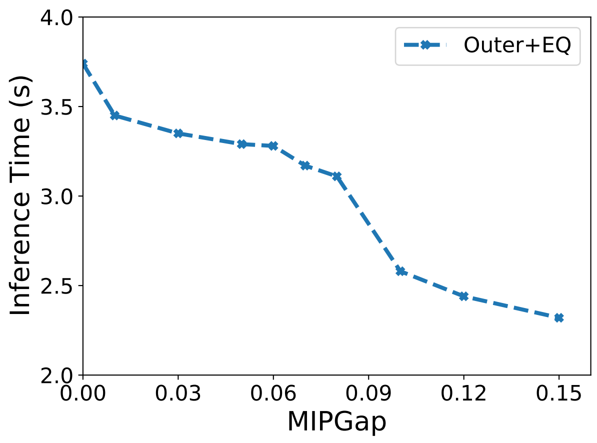
Experiment	Model	Training	Inference
MST	Baseline NN	190	5e-4
	Outer	24.1	7.9
	Outer+EQ	26.5	3.8
Sudoku	ConvNet	636.4	5e-4
	Outer+EQ	5.2	1.2
HMC	Baseline NN	44.9	4e-4
	Outer	560.3	1.1
	Outer+Latent	599.3	7.1

one can design a specific constraint solver to speed up the inference. Besides, the ILP inference can be accelerated by amortizing the computations when solving a batch test instances [SKR12, CUK15] or by applying approximate inference algorithms for solving ILP, e.g., LP relaxation methods [KP07, MFA15].

To demonstrate how the performance of our approach is affected by the approximate ILP solver. We show a trade-off curve in MST 20,000 training experiments in Fig. 6.4 by solving inference using Gurobi with different MIPGap, a parameter of the Gurobi solver controlling the quality of solutions. Specifically, MIPGap specifies the maximum gap of the objective function values between the returned solution and the optimum solution. We vary MIPGap from 0 (exact solutions are returned) to 0.15. The experimental results demonstrate that the inference can be accelerated using an approximate inference solver with a trade-off of moderate performance loss.



(a) EM accuracy



(b) Inference time

Figure 6.4: The trade-off between inference time and model accuracy when an approximate inference solver is used. Results are on MST. The x-axis is the allowed maximum relative gap between the returned solution and the optimum solution. On the left figure, the performance of our approach drops when the MIPGap is large, but our approach still significantly outperforms the neural network baseline (10.4%). The right figure shows that the inference time significantly reduces when MIPGap gets large.

CHAPTER 7

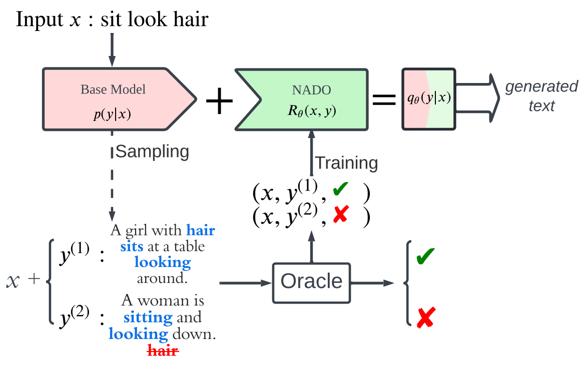
Controllable Text Generation with Neurally-Decomposed Oracle

7.1 Introduction

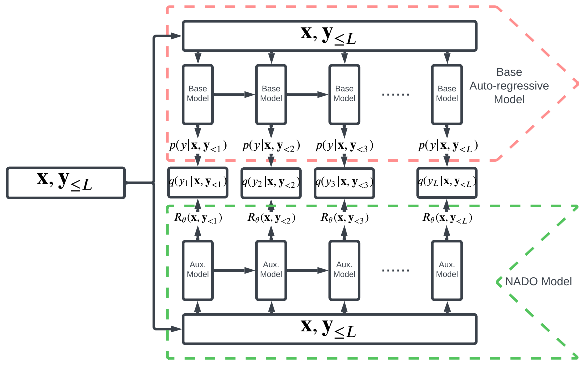
Auto-regressive language models have been widely used for text generation. With the recent development of large-scale pre-trained language models [RWC19, BMR20, RSR20a, LLG20], they have achieved state-of-the-art performances in applications such as machine translation [BCB15, LPM15], image captioning [AHB18, YJW16] and open domain text generation [ZL14, YPW19, VL15, SLL15, LZZ18]. However, many applications such as open-domain creative generation [YPW19, GCW20, TP22, HCT22, CHW22, SMH22] require to control model output with specific sequence-level attributes. The attributes can be specified by a set of rules[1] or by an abstract concept (*e.g.*, the generated text follows a particular writing style). How to control auto-regressive language models to satisfy these attributes is an open challenge.

In this paper, we propose a general and flexible framework for controllable text generation. Given a base pre-trained language model and a sequence-level oracle function indicating whether an attribute is satisfied, our goal is to guide the text generation to satisfy certain attributes using the oracle. To this end, we propose to decompose the sequence-level oracle into token-level guidance, such that when generating the $i-$th token in the output sequence given the prefix, instead of sampling from the base model, we modify the

[1] For example, lexical constraints require certain words to appear in the generated text [HL17, LZS20]


Input x : sit look hair
Base Model $p(y|x)$
NADO $R_\theta(x,y)$
$q_\theta(y|x)$
generated text
Sampling
Training
$(x, y^{(1)}, ✓)$
$(x, y^{(2)}, ✗)$
$x +$
$y^{(1)}$: A girl with hair sits at a table looking around.
$y^{(2)}$: A woman is sitting and looking down. ~~hair~~
Oracle
✓
✗

(a) Take lexically constrained generation as an example, where the oracle checks whether all keywords in the input $\mathbf{x}$ are incorporated in generated text $\mathbf{y}$. With proper training using samples from the base model p (dashed arrow) labeled by the oracle, we decompose the oracle into token-level guidance and parameterize it by an auxiliary model R_θ (NADO). We use R_θ to provide guidance when generating text with the base model (see details in Fig. 1(b)).


$\mathbf{x}, \mathbf{y}_{\leq L}$
Base Model
Base Auto-regressive Model
$p(y|\mathbf{x}, \mathbf{y}_{<1})$
$p(y|\mathbf{x}, \mathbf{y}_{<2})$
$p(y|\mathbf{x}, \mathbf{y}_{<3})$
$p(y|\mathbf{x}, \mathbf{y}_{<L})$
$q(y_1|\mathbf{x}, \mathbf{y}_{<1})$
$q(y_2|\mathbf{x}, \mathbf{y}_{<2})$
$q(y_3|\mathbf{x}, \mathbf{y}_{<3})$
$q(y_L|\mathbf{x}, \mathbf{y}_{<L})$
$R_\theta(\mathbf{x}, \mathbf{y}_{<1})$
$R_\theta(\mathbf{x}, \mathbf{y}_{<2})$
$R_\theta(\mathbf{x}, \mathbf{y}_{<3})$
$R_\theta(\mathbf{x}, \mathbf{y}_{<L})$
Aux. Model
NADO Model
......

(b) Illustration of the controlled generation process. Both the base model and the auxiliary model (NADO) take input $\mathbf{x}$ and the generated sequence (prefix) $\mathbf{y}_{<L}$ as input. The base model, in each step, outputs a token distribution $p(y_i|\mathbf{x}, \mathbf{y}_{<i})$. Guided by NADO R_θ, we obtain the distribution q (See Sec. 7.2.2), based on which we generate the output token.

Figure 7.1: Illustration of pipeline incorporating NADO (left) and model architecture (right).

probability distribution of the output token based on the token-level guidance. Specifically, we formulate the control as an optimization problem based on posterior regularization [GGG10] and solve the closed-form optimal solution to incorporate the token-level guidance for text generation. The decomposition is approximated by an auxiliary neural network model, called NeurAlly-Decomposed Oracle (NADO), which is trained on data sampled from the base model and supervised by the sequence-level oracle (see the illustration Fig. 7.1a). We further provide theoretical analysis on how NADO's approximation quality affects the controllable generation results. Note that in the entire process, we treat the base model and the sequence-level oracle as black-box functions, without the need for any refactoring or fine-tuning.

A few existing controllable generation works (*e.g.*, [LWZ21, LWW22a]) design search algorithms for generating texts with *lexical constraints*. However, their approaches cannot generally be applied to constraints such as style. Another line of work such as PPLM [DML20], GeDI [KGM21], and FUDGE [YK21] also aim to guide the base model with an auxiliary model. However, they either shift the base model distribution in a post-hoc manner without theoretical guarantee, or/and require external labeled data to train the auxiliary model. [KED21, KEK22] propose a generation with distributional control approach. Our control objective derived through posterior regularization resembles their energy-based model representation. However, they approximate the energy-based model using a KL-adaptive distributional policy, while we propose to decompose the sequence-level oracles into token-level approximated by NADO. With the decomposition, base models receive explicit controlling signal in generating every token from the oracle. Furthermore, since NADO is trained on the data sampled from the base models, it aligns better with the base model's distribution and thus can achieve better control.

We conduct experiments on lexically constrained generation (LCG) tasks and a machine translation (MT) formality change task. In LCG tasks, the oracle is a rule-based keyword checker. We achieve almost perfect keyword incorporation with significantly boosted BLEU scores compared to previous approaches that design specific decoding algorithms [LWZ21]. In the formality-controlled MT task, we are provided with a formality oracle predicting whether a sentence is formal or not, and the goal is to guide the model to generate formal translations. Compared with recent work [YK21], we improve the BLEU score by 3 points as well as improve the formality rate, demonstrating NADO's superior ability to incorporate external oracle supervision. Both experiments demonstrate the effectiveness of our framework in dealing with various types of control while maintaining high-quality generation results.[2]

7.2 Methodology

We approach the sequence-level controllable text generation problem by decomposing the sentence-level oracle into token-level guidance. We formulate this as an optimization problem. Since the token-level guidance is intractable, we propose to train an auxiliary model, called NeurAlly-Decomposed Oracle (NADO), to approximate it. During the inference time, NADO guides the base model to generate sequences that satisfy the oracle constraints.

In the rest of this section, we discuss 1) the formulation to decompose the sequence-level oracle function into token-level guidance; 2) the formulation to incorporate the token-level guidance into the base model to achieve control; 3) the approximation of the token-level guidance using NADO; 4) a theoretical analysis of the impact of NADO approximation to the controllable generation results; and 5) the training of NADO.

7.2.1 Setup: Notations and Problem Formulation

We use $\mathbf{x} \in \mathcal{X}$ to denote the input and $\mathbf{y} \in \mathcal{Y}$ to denote the generated sequence. y_i is the $i-$th token in $\mathbf{y}$ and $\mathbf{y}_{<i}$ is the sequence prefix from the beginning to the $(i-1)-$th token. We denote the base auto-regressive generation model as $p(y_i|\mathbf{x}, \mathbf{y}_{<i})$, hence the sequence-level distribution is given by $p(\mathbf{y}|\mathbf{x}) = \prod_i p(y_i|\mathbf{x}, \mathbf{y}_{<i})$. A sequence-level oracle is defined as a boolean function $C : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$. We formalize the optimization objective based on posterior regularization [GGG10]. Basically, we explore a token-level distribution $q^*(y_i|\mathbf{x}, \mathbf{y}_{<i})$ and its corresponding sequence-level distribution $q^*(\mathbf{y}|\mathbf{x})$, satisfying

1. $q^*(\mathbf{y}|\mathbf{x}) = \prod_i q^*(y_i|\mathbf{x}, \mathbf{y}_{<i})$, *i.e.*, q^* can be treated as an auto-regressive model.

2. $q^*(\mathbf{y}|\mathbf{x}) = 0$ if $C(\mathbf{x}, \mathbf{y}) = 0$, *i.e.*, q^* only generates sequences satisfying the oracle C.

[2]Our code can be found at `https://github.com/MtSomeThree/constrDecoding`.

3. Given an input $\mathbf{x}$, $KL(p(\mathbf{y}|\mathbf{x})\|q^*(\mathbf{y}|\mathbf{x}))$ is minimized, *i.e.*, q^* should be as similar to the base model as possible.

[KED21, KEK22] derive a similar optimization formulation as property 2, 3, to represent constraints through energy-based models and approximate it with distributional policy gradient. In this work, we propose to decompose oracle to token-level guidance to steer the generation. We discuss our approach in the following.

7.2.2 Token-level Guidance and Closed-Form Solution For q^*

Before we compute the solution for q^*, given the base model p and oracle C, we first define the token-level guidance as a success rate prediction function $R_p^C(\mathbf{x})$, which defines the probability of the sequence generated by p satisfies the oracle C given the input $\mathbf{x}$. We similarly define $R_p^C(\mathbf{x}, \mathbf{y}_{\le i})$ as the probability of success given input $\mathbf{x}$ and prefix $\mathbf{y}_{<i}$. By definition, we have

$$\begin{aligned} R_p^C(\mathbf{x}) &= \mathrm{Pr}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})}\left[C(\mathbf{x},\mathbf{y})=1\right] = \sum\nolimits_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}|\mathbf{x})C(\mathbf{x},\mathbf{y}) \\ R_p^C(\mathbf{x},\mathbf{y}_{\le i}) &= \mathrm{Pr}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})}\left[C(\mathbf{x},\mathbf{y})=1|\mathbf{y}_{<i}\right] = \sum\nolimits_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}|\mathbf{x},\mathbf{y}_{<i})C(\mathbf{x},\mathbf{y}). \end{aligned} \tag{7.1}$$

With the function R_p^C, we now derive the closed-form solution of q^* considering conditions 2 and 3 defined in Sec. 7.2.1. Given input $\mathbf{x}$, we define the feasible sequence-level distribution set Q as

$$Q := \{q| \sum\nolimits_{\mathbf{y}:\ C(\mathbf{x},\mathbf{y})=0} q(\mathbf{y}|\mathbf{x}) = 0\}, \tag{7.2}$$

then the sequence-level closed-form solution for q^* is given by

$$q^*(\mathbf{y}|\mathbf{x}) = \arg\min_{q\in Q} KL(p(\mathbf{y}|\mathbf{x})\|q(\mathbf{y}|\mathbf{x})) = \frac{p(\mathbf{y}|\mathbf{x})C(\mathbf{x},\mathbf{y})}{R_p^C(\mathbf{x})}. \tag{7.3}$$

Considering condition 1 in Sec. 7.2.1 to make q^* tractable, we decompose $q^*(\mathbf{y}|\mathbf{x})$ into token-level. The closed-form solution is given by

$$q^*(y_i|\mathbf{x}, \mathbf{y}_{<i}) = \frac{R_p^C(\mathbf{x}, \mathbf{y}_{\leq i})}{R_p^C(\mathbf{x}, \mathbf{y}_{\leq i-1})} p(y_i|\mathbf{x}, \mathbf{y}_{<i}). \tag{7.4}$$

The decomposition is unique. The proof and detailed derivation can be found in the appendix.

Control with Soft Constraints. In Eq. (7.2) we define the feasible distribution set as distribution that the possibility of a sequence violate the oracle function is 0. However, in some applications, we expect to control the generation with soft constraints. For example, we want the model to generate sentence about sports with probability $r = 0.8$. Our framework also supports controlling the generation with soft constraints. To achieve this, with a pre-defined ratio $r \in [0, 1]$, we alternatively define a general feasible set Q as

$$Q := \{q | \sum\nolimits_{\mathbf{y}\colon C(\mathbf{x},\mathbf{y})=1} q(\mathbf{y}|\mathbf{x}) = r\},$$

where Eq. (7.2) is the special case when $r = 1$. The general token-level closed-form solution is

$$q^*(y_i|\mathbf{x}, \mathbf{y}_{<i}) = \frac{\alpha R_p^C(\mathbf{x}, \mathbf{y}_{\leq i}) + \beta(1 - R_p^C(\mathbf{x}, \mathbf{y}_{\leq i}))}{\alpha R_p^C(\mathbf{x}, \mathbf{y}_{\leq i-1}) + \beta(1 - R_p^C(\mathbf{x}, \mathbf{y}_{\leq i-1}))} p(y_i|\mathbf{x}, \mathbf{y}_{<i}),$$

where $\alpha = \frac{r}{R_p^C(\mathbf{x})}, \beta = \frac{1-r}{1-R_p^C(\mathbf{x})}$.

Similar to Eq. (7.4), once we have access to R_p^C, we can directly compute the closed-form solution even though the form is much more complicated. In this paper we only focus on hard constraints ($r = 1$), however, here we demonstrate that our framework is capable of handling soft constraints as well.

7.2.3 Approximating R_p^C by NADO and Theoretical Analysis

Unfortunately, function R_p^C defined in Eq. (9.5) is intractable. We cannot enumerate all possible sequences **y** since the space is exponentially large and essentially infinite. Hence, we train a neural model NADO to approximate this well-defined function. We use R_θ^C to denote NADO parameterized by θ. In this section, we derive bounds to provide a theoretical analysis about the correlation between errors in approximation and errors in corresponding sequence-level distribution. Generally, when R_θ^C approximates R_p^C precisely enough, we have an upper bound for the sequence-level distribution discrepancy. The following lemma provides the formal definition.

Lemma 1 We define distribution

$$q(y_i|\mathbf{x}, \mathbf{y}_{<i}) \propto \frac{R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i})}{R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i-1})} p(y_i|\mathbf{x}, \mathbf{y}_{<i}). \tag{7.5}$$

If there exists $\delta > 1$ such that given input $\mathbf{x}$, $\forall \mathbf{y}_{<i}$, $\frac{1}{\delta} < \frac{R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i})}{R_p^C(\mathbf{x}, \mathbf{y}_{\le i})} < \delta$, we have

$$KL(q^*(\mathbf{y}|\mathbf{x}) \| q(\mathbf{y}|\mathbf{x})) < (2L+2) \ln \delta,$$

where L is the length of the sequence $\mathbf{y}$.

We also notice that by definition, R_p^C satisfies the following equation:

$$\sum_{y_i} R_p^C(\mathbf{x}, \mathbf{y}_{\le i}) p(y_i|\mathbf{x}, \mathbf{y}_{<i}) = R_p^C(\mathbf{x}, \mathbf{y}_{\le i-1}). \tag{7.6}$$

If R also satisfies Eq. (7.6), we can tighten this bound. Formally,

Lemma 2 Given the condition in Lemma 1, if q is naturally a valid distribution without normalization (*i.e.*, $\sum_{y_i} \frac{R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i})}{R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i-1})} p(y_i|\mathbf{x}, \mathbf{y}_{<i}) = 1$), we have

$$\forall x, KL(q^*(\mathbf{y}|\mathbf{x}) \| q(\mathbf{y}|\mathbf{x})) < 2 \ln \delta.$$

This lemma shows that with the auto-regressive property, the error does not accumulate along with the sequence. The proof is in the appendix. These two bounds indicate that when training the model R_θ^C, we should push it to satisfy Eq. (7.6) while approximating R_p^C.

7.2.4 Training NADO

In Fig. 7.1b we show the architecture of NADO. In general, NADO can be any seq2seq model. During training, it takes $\mathbf{x}, \mathbf{y}$ as input and predicts from $R_\theta^C(\mathbf{x}, \mathbf{y}_{\le 0})$ to $R_\theta^C(\mathbf{x}, \mathbf{y}_{\le T})$.

During the inference time, there are two parallel forward pass[3] to compute the token distribution q. Considering the size of the NADO is usually much smaller than the base model, the whole forward pass takes no more than 2x base model forward pass time.

Now we discuss the training objective. In training, with some predefined input distribution $\mathcal{X}$, we sample $\mathbf{x} \sim \mathcal{X}, \mathbf{y} \sim p(\mathbf{y}|\mathbf{x})$. We take these sampled $(\mathbf{x}, \mathbf{y})$ pairs as training examples, and use the boolean value $C(\mathbf{x}, \mathbf{y})$ as their labels for all steps. We use cross entropy (denoted as $CE(\cdot, \cdot)$) as the loss function, formally, $L_{CE}(\mathbf{x}, \mathbf{y}, R_\theta^C) = \sum_{i=0}^{T} CE(R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i}), C(\mathbf{x}, \mathbf{y}))$. Given a particular input $\mathbf{x}$, in expectation, we have

$$\begin{aligned}\mathbb{E}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})} L_{CE}(\mathbf{x}, \mathbf{y}, R_\theta^C) &= \sum\nolimits_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}|\mathbf{x}) L_{CE}(\mathbf{x}, \mathbf{y}, R_\theta^C) \\ &= \sum_{i=0}^{T} R_p^C(\mathbf{x}, \mathbf{y}_{\le i}) \log R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i}) + (1 - R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i})) \log(1 - R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i})) \\ &= \sum_{i=0}^{T} CE(R_p^C(\mathbf{x}, \mathbf{y}_{\le i}), R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i}))\end{aligned} \tag{7.7}$$

Therefore, L_{CE} empirically estimates the cross entropy loss between R_θ^C and the ground truth R_p^C which is intractable.

As we analyze above, we also regularize R_θ^C for satisfying Eq. (7.6) based on KL-divergence:

$$L_{reg}(\mathbf{x}, \mathbf{y}, R_\theta^C) = f_{KL}\left(\sum_{y_i} R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i}) p(y_i|\mathbf{x}, \mathbf{y}_{<i}), R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i-1})\right).$$

$f_{KL}(p, q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$ is KL-divergence regarding p and q as two Bernoulli distributions. We use a hyper-parameter $\lambda > 0$ to balance these losses. The final training loss is

$$L(\mathbf{x}, \mathbf{y}, R_\theta^C) = L_{CE}(\mathbf{x}, \mathbf{y}, R_\theta^C) + \lambda L_{reg}(\mathbf{x}, \mathbf{y}, R_\theta^C). \tag{7.8}$$

[3]In practice, to avoid enumerating the vocabulary, R_θ^C outputs a vector over vocabulary (*i.e.*, $R_\theta^C(\mathbf{x}, \mathbf{y}_{\le i-1} \oplus y)$ for all possible y, $\oplus$ is the concatenation operation), then we can directly do element-wise multiplication between R_θ^C and p.

7.2.5 Sampling

In Sec. 7.2.4 we describe that we train NADO by sampled data from base model p. One advantage is that we are able to leverage different sampling strategies to better adapt to different application scenarios. It is also possible to leverage reinforcement learning to train R_θ^C, and we discuss our connection to reinforcement learning in the appendix. In this section, we introduce two sampling strategies and their corresponding properties.

Sampling with Temperature Control. In some task, the output sequences are not diverse much, in other words, the token distribution in each step is very peaky. Since our NADO is trained on the sampled examples, we expect those examples to cover as much tokens combination as possible to avoid overfitting. Therefore, we add temperature factor T to smooth the distribution [AHS85]. Specifically, we sample $\mathbf{y}$ from distribution $p(\mathbf{y}|\mathbf{x})^{\frac{1}{T}}$, and add coefficient $p(\mathbf{y}|\mathbf{x})^{1-\frac{1}{T}}$ when computing the cross-entropy loss. Formally, the expected loss is

$$\mathbb{E}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})^{\frac{1}{T}}}\left[p(\mathbf{y}|\mathbf{x})^{1-\frac{1}{T}} L_{CE}(\mathbf{x},\mathbf{y},R_\theta^C)\right] = \sum\nolimits_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}|\mathbf{x}) L_{CE}(\mathbf{x},\mathbf{y},R_\theta^C),$$

which is same as the original expected loss in Eq. (7.7).

Importance Sampling. In practice, the training process of NADO can be extraordinarily difficult when samples generated by the base model p hardly satisfy C. *i.e.* $\mathbb{E}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})}[p(C|\mathbf{x},\mathbf{y})] \simeq 0$. Hence, we introduce the importance sampling [HM54] to tackle this issue. Basically, we leverage existing partially trained $\hat{R}_\theta$ to form distribution $\hat{q}$. Although $\hat{R}_\theta$ is not well-trained, it is still able to provide positive guidance to produce samples satisfying C. Note that $\hat{q}$ does not have to be updated in each training epoch. With coefficient $\frac{p(\mathbf{y}|\mathbf{x})}{\hat{q}(\mathbf{y}|\mathbf{x})}$, the expected loss is same as the original expected loss:

$$\mathbb{E}_{\mathbf{y}\sim \hat{q}(\mathbf{y}|\mathbf{x})}\left[\frac{p(\mathbf{y}|\mathbf{x})}{\hat{q}(\mathbf{y}|\mathbf{x})} L_{CE}(\mathbf{x},\mathbf{y},R_\theta^C)\right] = \sum\nolimits_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}|\mathbf{x}) L_{CE}(\mathbf{x},\mathbf{y},R_\theta^C).$$

7.3 Experiments

We conduct experiments on two tasks: lexically constrained generation (LCG) and machine translation (MT) with formality change. For the former, we use GPT-2 [RWC19] as the base model and for the latter, we use a sequence-to-sequence model, MarianMT [JGD18]. We demonstrate our framework is generally effective in both scenarios. The boolean oracle is a rule-based function checking whether all lexical constraints are satisfied in LCG task, while in MT it is a classifier trained on an external dataset identifying the formality of the text. We put all details about hyper-parameter settings in the appendix.

7.3.1 Text Generation with Lexical Constraints

We evaluate our model on two general classes of LCG problems:

- Unsupervised LCG: annotation for lexical constraints are not available during training, but are expected to be in their exact order and lexical form during inference.
- Supervised LCG: annotation for lexical constraints are available, yet the words may appear in a different lexical form (e.g., "look" can appear in the past tense "looked") or a different order in the generated text.

In both cases, we define oracle C as a boolean function indicating whether the generated sequence *satisfies all of the lexical constraints*. We do not naturally have negative samples (*i.e.* the sequences that do not satisfy all constraints) to train the auxiliary model in both settings, thus, it is non-trivial to compare against methods requiring both positive and negative labeled data for training the auxiliary model like FUDGE and GeDi.

Data Setup For unsupervised LCG, we follow the settings in POINTER [ZWL20] and conduct our experiments on Yelp! Review and News dataset. Each of the unsupervised LCG dataset contains a great number of un-annotated, raw sequences for training (160K for Yelp! Review and 268,586 for News). During inference, the model is expected to generate text lexically constrained in the exact order and form by a specific number of keywords (7 for Yelp! Review and 4 for News). For supervised LCG, we evaluate the proposed method on CommonGen [LZS20]. CommonGen is a supervised LCG task that aims to examine the commonsense of neural text generation models. For training, it contains 32,651 unique key *concepts* (*i.e.* the constraints) with 67,389 completed sequences in total. It also contains a validation set with 993 *concepts* and 4018 reference sequences. For a more robust evaluation, the dataset maintains an open leaderboard that benchmarks different approaches on a withheld test set. We follow most of the data configurations specified in the original paper that first introduced the datasets.

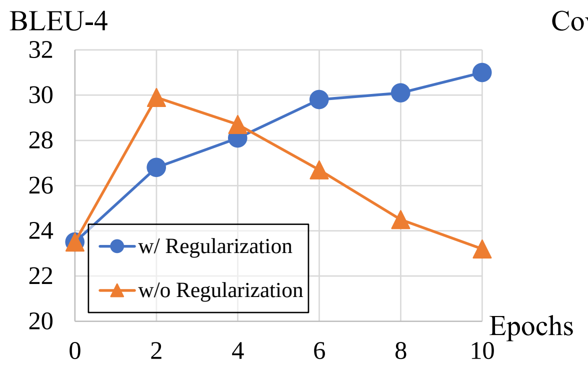
General Model Setup We investigate the effectiveness of different factors in our framework by enumerating different combinations of them. We implement two types of base model:

- (Seq2seq base model) A sequence-to-sequence model $p(\mathbf{y}|\mathbf{x})$ that takes into account the lexical constraints as condition sequence input;
- (DA base model) A language model that is only domain-adapted to $p(\mathbf{y})$ but unconditioned on anything. This is a challenging setting, since we impose the lexical constraints only with NADO. This setting is to better verify the effectiveness and efficiency of the proposed method and control irrelevant factors.

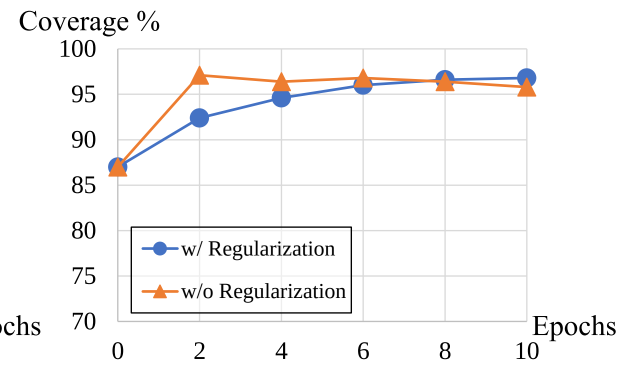
Under both $p(\mathbf{y}|\mathbf{x})$ and $p(\mathbf{y})$ settings, we fine-tune the base model from the pre-trained GPT2-Large.

During training, NADO is trained as a Seq2seq-like model[4], which takes in the keys (for unsupervised LCGs, they are generated by randomly sampling a specific number of natural words in the original sentence) and generates the token-level guidance $R_\theta^C(\mathbf{x}, \mathbf{y}_{\leq i})$. For each pseudo key, we sample 32 target text with top-p ($p = 0.8$) random sampling from

[4]In this experiment, the input $\mathbf{x}$ is only describing the lexical constraint C. However, our framework also supports general inputs in other Seq2seq tasks with constraints. For example, machine translation with lexical constraints where the constraint C is different from the input $\mathbf{x}$.



(a) BLEU-4 comparison of NADO training with/without using Eq. 7.6



(b) Coverage comparison of NADO training with/without using Eq. (7.6).

Figure 7.2: Comparative study of the effectiveness of regularization in NADO training.

base model p. We conduct experiments to test different training setups for NADO:

- (NADO training) The proposed training process described in Sec. 7.2.4.
- (Warmup) We warm up NADO by maximizing the likelihood of positive samples, but only backpropagating the gradient to the parameters of R_θ. The warm-up R_θ^C is used for importance sampling described in Sec. 7.2.5. With DA base models, however, the warmup process is always incorporated for practical success of training (see the results for DA pretrained w/o warmup).

We also consider the setting with warmup only, which can be treated as a stronger baseline to verify that the major improvement of our framework is not coming from the extended capacity in NADO.

Results and Analysis We compare the performance under different setups of our model to previous state-of-the-art methods on LCG tasks, including insertion-based models (Levenshtein Transformer [GWZ19] with Lexical Constraints [SCT20], InsNet [LMP22], etc.) and decoding-based algorithms. We also compare the results with a simple baseline which address the problems with a standard Seq2seq pipeline. The results are as shown in Table 7.1.

NADO consistently improves the BLEU score and coverage in different setups. Furthermore, under the best setting of each task (see bolded items in the table), NADO performs significantly better than most baselines in generation quality and can achieve very good lexical constraints coverage rate. Compared to InsNet, it is much easier for an autoregressive model with NADO to handle flexible reordering/transformation of lexical constraints. This is reflected in the performance comparison of InsNet and NADO on CommonGen dataset. Under most settings, a Seq2seq base model makes it easier for the framework to perform well, as it guarantees a reasonable level of lexical constraint coverage in even the initial state of the model.

Using a DA pretrained base model is a even challenging setup since the lexical constraints are only imposed with NADO. Therefore, the base model distribution is much distinct from the one filtered by the oracle, which is shown by poor performances on both metrics. However, with warmup and NADO under importance sampling, we show that it is still possible to obtain a powerful model with the proposed method.

To further study the correlation between the base model quality and the improvement of NADO, we conduct experiments on GPT-2 base model. The GPT-2 base model has lower scores with and without NADO compared with GPT-2 large, while the coverage improvements are similar. It shows NADO is capable to push the base model distribution towards the oracle if the base model has decent quality.

We also do human evaluation on base model (GPT-2 Large fine-tune) and the best NADO system, together with the gold reference for comparison. The results are shown in Tab. 7.2. The evaluation metrics are detailed described in the Appendix. Some qualitative are shown in Tab. 7.3.

To study the importance of the regularization term, we conduct an ablative study under the optimal setting on the CommonGen dataset (Seq2seq base model with NADO only). The results are shown in Figure 7.2. While the success of achieving lexical control does not degenerate when NADO w/o regularization overfits, adding regularization can significantly improve the robustness of NADO generation quality when training NADO for more epochs.

7.3.2 Machine Translation with Formality Change

Datasets and Setup We follow the experimental setting in FUDGE [YK21] to formalize the results of machine translation. Given an informal source sentence, our goal is to translate it into formal sentence written in the target language. We conduct our experiments on Fisher and CALLHOME Spanish-English Speech Translation Corpus [PKL13], where both of the Spanish source and English reference are informal and casual. Instead of evaluating the translation on original references, we use the formal and fluent rewritten version of references [SSW19] to evaluate the translation quality by BLEU scores. In the training process, the formal version reference is unseen to the models. We also evaluate the formality scores by a discriminator trained on GYAFC formality dataset [RT18] as what FUDGE paper does. In this experiment, pre-trained Marian MT model [JGD18] is used as the base model.

In FUDGE, the authors train an auxiliary model also on GYAFC modeling token-level guidance $P(\text{formal}|\mathbf{y}_{<i})$, and leverage it to guide the base model by Bayesian rule

$$P(y_i|\mathbf{y}_{<i}, \text{formal}) \propto P(y_i|\mathbf{y}_{<i})P(\text{formal}|\mathbf{y}_{\leq i}). \tag{7.9}$$

For the formality supervision, FUDGE leverages an external token-level oracle. In NADO, we load the same oracle but exclusively leverage sequence-level binary supervision as oracle C. We randomly choose 10,000 (7.2%) source texts from the training set as input examples, and sample 8 target texts by sampling with temperature T from base model p for each source text. We use those sampled examples to train NADO. In total, we have $80,000$ training samples, which is similar to the number of training data (105k) for the token-level oracle in FUDGE. All the methods are using greedy decoding.

Results and Discussion The experimental results are shown in Table 7.4. Compared to FUDGE, although only the sequence-level supervision is leveraged, we are consistently better in both metrics, especially in BLEU score we boost about 3 points. We conjecture that the improvement is because our formulation is more principle and correct. In methods using auxiliary model to guide the base model, including FUDGE, their formulation is based on Eq. 7.9. However, the auxiliary model is trained on a distribution different from where the base model is pretrained on, which leads to a distributional discrepancy issue. In other words, directly multiplying these two terms is not rigorous, since they are estimated on two different distributions. On the contrary, NADO is trained specifically to the base model. This avoids the discrepancy issue and provides an accurate guidance. Considering we are using the same oracle function and similar number of training samples, the higher generation quality reflected by BLEU scores supports our conjecture.

In sampling, for each input we sample 8 examples to train R_θ^C, which are usually identical in this task. Applying temperature in sampling allows NADO to be trained with more diverse data. Results show that with a properly set temperature, we can further improve the generation quality.

It is still possible that the neural oracle leverages some superficial or even spurious features and NADO is catering those features in order to improve the formality scores. For example, some informal little words like "hmm" "uh", and some abbreviations like " 'cause " "gonna" could make the formality score lower. We find that NADO tends to fix them. However, how to get an good oracle is orthogonal to our contributions.

7.4 Conclusion

We purpose a general and efficient framework for controllable generation. We leverage an auxiliary neural model, NADO, to approximate the decomposed oracle guidance, and incorporate it with a fixed base model. By training with sampled data from the base model, NADO aligns better with the base model, and our framework is more flexible dealing with various application scenarios provided by different sampling methods. As NADO is a general framework, in the future, we plan to apply it in boarder application scenarios. For example, reducing societal bias [SCN19] (e.g., gender or racial bias) in generation by providing corresponding oracle.

Table 7.1: Unsupervised/Supervised Lexically Constrained Generation results on Yelp Review (unsupervised), News (unsupervised) and CommonGen (supervised) dataset. CVRG stands for constraints coverage. For insertion-based models, on CommonGen dataset we directly use the keyword as initial context with no further permutation. p, q denote the base model and the combined model in our framework, respectively. The domain adaptation pretrained model produces samples unconditioned on the constraints, and thus results in worse results than other setups. Results with * mark are from the open leader board on the test set instead of development set.

Dataset	Yelp Review (test)		News (test)		CommonGen (dev)	
Metrics	BLEU-2/4	CVRG	BLEU-2/4	CVRG	BLEU-3/4	CVRG
Insertion-based Baselines						
InsNet-Sequential [LMP22]	19.4/5.8	100%	16.3/5.0	100%	26.2/18.7	100%
ConstLevT [SCT20]	14.8/4.0	100%	11.8/1.9	100%	21.3/12.3*	96.9%*
Algorithmic Baselines						
GPT-2-Large Finetune + Sampling	16.4/5.3	94.5%	13.2/4.2	81.8%	34.2/24.7*	82.2%*
Neural Logic [LMP22]	-	-	-	-	36.7/26.7*	97.7%*
A*esque Decoding [LWW22a]	-	-	-	-	-/28.2*	97.6%*
Model Setups (Ours)						
p (Domain Adaptation pretrain)	5.3/0.4	5.4%	4.0/0.8	0.9%	9.3/3.9	8.5%
p (Seq2seq pretrain)	16.6/4.8	91.2%	13.0/3.4	74.0%	34.2/23.5	87.0%
q (DA pretrained p + warmup)	16.2/4.3	75.4%	12.6/2.8	66.7%	32.7/20.9	79.7%
q (DA pretrained p + warmup + NADO)	16.9/5.4	95.6%	**15.4/4.7**	**92.3%**	37.8/26.2	96.1%
q (Seq2seq pretrained p + warmup)	16.8/5.7	94.2%	13.6/4.2	85.0%	35.2/24.8	90.2%
q (Seq2seq pretrained p + NADO)	**17.4/6.0**	**96.7%**	15.0/4.5	91.9%	**40.9/30.8**	**97.1%**
q (Seq2seq pretrained p + warmup + NADO)	16.7/4.7	92.8%	14.4/4.4	86.1%	40.2/30.3	95.9%
GPT-2 Base Reference						
q (Seq2seq pretrained p)	-	-	-	-	32.17/22.98	76.8%
q (Seq2seq pretrained p + NADO)	-	-	-	-	33.61/24.01	85.5%

Table 7.2: Human evaluation of generated texts in CommonGen test set. The detailed description for the four metrics (scale: from 1 to 3) and the evaluation setups can be found in the Appendix. Baseline stnads for GPT-2 Large fine-tune setting, and NADO stands for the best system, Seq2seq pretrained + NADO. We also evaluate the first gold reference provided in the dataset for comparison. NADO outperforms base model in all four metrics. (The difference is statistical significant tested by Wilcoxon signed ranks one-sided test, p-value < 0.02)

Model	Quality	Plausibility	Concepts	Overall
Baseline	2.39	2.46	2.40	2.37
NADO (Ours)	2.51	2.52	2.52	2.47
Gold Ref.	2.53	2.58	2.59	2.56

Table 7.3: Some more qualitative generation results with randomly selected concepts about NeurIPS.

Constraint:	The generated texts should contain all the given concepts in arbitrary order
Concepts	**look forward discuss NeurIPS**
Base Model Sample #1	Players **discuss** the **look** of **forward** NeurrIPS. (**NeurIPS**)
Base Model Sample #2	Football player and **forward discuss** a **look** at the move. (**NeurIPS**)
NADO Sample #1	People **look forward** to **discussing** the future of **NeurIPS**.
NADO Sample #2	We **look forward** to meeting and **discussing** the future of **NeurIPS**.
Concepts	**excite paper accept NeurIPS**
Base Model Sample #1	Researchers are **excited** after **acceptance** of their **paper** at IPS. (**NeurIPS**)
Base Model Sample #2	Scientists **excited** to **accept paper accepted** at **NeurIPS**.
NADO Sample #1	**NeurIPS** is **excited** to **accept** the **paper** of researcher.
NADO Sample #2	**NeurIPS** is **excited** to announce that it has **accepted papers**.

Table 7.4: Formal Machine Translation results. We follow [YK21] setting to choose BLEU score and average formality scores as the metric. We slightly improve the formality score compared to FUDGE, while significantly boost the BLEU score.

Method	BLEU	Avg. Formality
MarianMT [JGD18]	16.98	0.45
FUDGE [YK21]	17.96	0.51
NADO + Random Sampling	20.84	0.54
NADO + Sampling with $T = 5/4$	21.04	0.53
NADO + Sampling with $T = 5/3$	20.77	0.52

CHAPTER 8

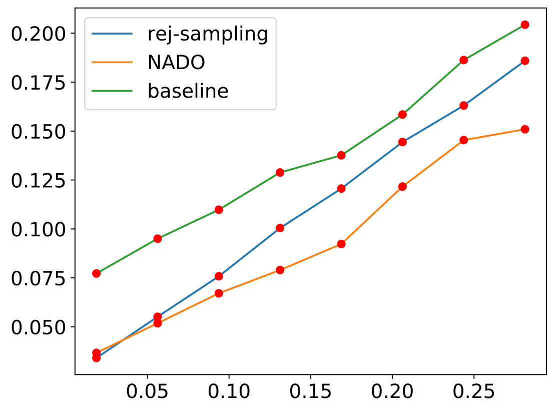
Application: Controllable Generation with Large Language Models in Detoxification

8.1 Introduction

Large language models (LLMs) have demonstrated impressive performance across various tasks, benefiting from their extensive pretraining on vast amounts of data. However, recent studies have unveiled a concerning issue: LLMs can generate toxic texts [WFK19, SCN19], raising the need to evaluate and address toxicity to prevent the generation of disrespectful or offensive content in practical applications. [GGS20] collect prompts that trigger LLMs to generate toxic outputs, revealing the existence of toxicity problems in popular models like GPT-3 [BMR20], PaLM [CND22], and OPT [ZRG22]. Consequently, it is crucial to explore systematic techniques for detoxifying large language models.

Previous work has attempted to prevent LLMs from generating toxic texts by fine-tuning models on filtered corpora [RSR20b, WPX22, OWJ22]. However, given the massive number of parameters in current models, we focus on decoding-time detoxification methods in this study, as they offer higher efficiency. Since toxicity is an abstract attribute of an entire sentence, detoxifying individual sentences by rejecting specific lexical features proves ineffective. The challenge lies in bridging the gap between the token-by-token generation process of auto-regressive models and the sentence-level attribute of toxicity.

Controllable generation via plug-and-play [DML20] is an efficient technique that leverages auxiliary models to exert control over the generation process at decoding time, with-



(a) Comparative study of the effectiveness between NADO and rejection sampling evaluated by the toxicity trend. x-axis is the toxicity of the prompts. x, y-axis are the toxicity score of the prompts and generated texts, respectively.

Model (Size)	Method	Max. tox.	PPL
GPT-3 XL (1.3b)	Baseline*	0.57	10.18
	Rej. Samp.*	0.45	10.18
	DExpert*	0.31	19.87
	DAPT*	0.47	10.40
	NADO	0.33	12.11
OPT-30b (30b)	Baseline	0.55	9.27
	Rej. Samp.	0.47	9.18
	NADO	0.32	11.65

(b) Comparative study of the effectiveness between NADO and rejection sampling, DExpert and DAPT evaluated by the expected maximum toxicity. The results labeled by * are from [WPX22].

Table 8.1: Comparative study of detoxification methods evaluated in toxicity trend (8.1a) and expected maximum toxicity (8.1b).

out the need for retraining or fine-tuning the generation model. While recent work such as FUDGE [YK21], GeDi [KGM21], DExpert [LSL21] and NADO [MLP22] has achieved remarkable performance in various controllable generation tasks, these techniques have primarily been tested on relatively small models like GPT-2 with 1.5 billion parameters. The effectiveness of these methods for detoxifying large language models remains unclear, and their adoption could significantly impact inference time and the scale of auxiliary models.

In this study, we investigate the detoxification of LLMs with up to 30 billion parameters using controllable generation. We compare the detoxification performance, as measured by toxicity metrics and generation quality, with existing detoxification methods such as lexically controlled techniques and domain adaptation. Furthermore, we analyze significant factors, including base model sizes, auxiliary model sizes, and architecture, within the context of controllable generation. Our empirical results demonstrate that the controllable generation approach exhibits decent control performance, which is not heavily influenced by the size of the base generation model. Notably, we find that a parameter-efficient auxiliary model, with only 125 million parameters compared to the 30 billion of the controlled model, is sufficient to effectively control the toxicity of an LLM.

8.2 Experiments

We conduct experiments focusing on detoxifying OPT [ZRG22] with NADO [MLP22]. We compare the performance with existing methods, and do ablation studies about the effect of architectures and model sizes in controllable generation.

Setup We follow the setting in [ZRG22, CND22] to setup the experiments with Real-Toxicity-Prompts (RTP) dataset [GGS20] and the Perspective API [1]. The RTP dataset contains 100k prompts, based on which language models might be triggered to generate some toxic sentences. PerspectiveAPI is a widely used toxicity detection tool, and we use the TOXICITY score to evaluate the toxicity the sentence[2]. In NADO training, we sample 20,000 prompts from RTP dataset and sample 200 sentences with length 50 from the base model for each prompt. The sampled sentences are labeled by PerspectiveAPI. The NADO is trained on the 4 million sampled sentences. The detailed hyper-parameters are shown in the Appendix. In testing, we sample another 10,000 prompts from RTP dataset, and sample 25 sentences per prompt. For each sentence we evaluate the toxicity score by PerspectiveAPI and the perplexity given by the base language models.

[1]https://github.com/conversationai/perspectiveapi

[2]In PerspectiveAPI, TOXICITY means a rude, disrespectful, or unreasonable comment; likely to make people leave a discussion. The API is regularly updated. Our results are evaluated by October 2022.

8.2.1 Comparsion of Detoxification Methods

In our experiments, we begin by comparing different methods for controlling the generation of toxic texts, namely lexically control, domain adaptation, and auxiliary models. The lexically control approach involves down-weighting specific tokens in beam search to avoid generating toxic n-grams [SCN21, LWZ21, LWW22b]. Following the methodology of [LJH18, SCN21], we employ salient scores to select toxic n-grams from the RTP dataset. We use rejection sampling to generate sentences without these toxic n-grams, simulating the lexically controlled beam search methods. To evaluate toxicity, we employ two metrics: 1) toxicity trend analysis, where prompts are grouped into bins based on their toxicity score, and we calculate the average generated toxicity for each bin, and 2) average maximum toxicity score, where we sample 25 sentences for each prompt and calculate the average maximum toxicity score. The results are presented in Table 8.1.

Compared to rejection sampling, our proposed NADO method consistently achieves lower toxicity scores across prompts with different toxicity levels while maintaining good generation quality. Compared to DExpert [LSL21] which also leverages auxiliary model in decoding, NADO is much better in keeping the generation quality. Domain adaptive training (DAPT) [GGS20] performs well in preserving generation quality but falls short in toxicity control compared to controllable generation.

8.2.2 The Effect of Coupling Models

We conduct a comparative study on the NADO architecture proposed in [MLP22]. We compare the coupling NADO, which shares the encoder of the OPT-30b model and uses a 350m decoder, with the independent NADO, where the NADO is an independent neural model without encoder sharing. In the independent NADO, we fine-tune a pretrained OPT-350m model as the architecture. We find that for prompts with low toxicity levels, the two architectures perform similarly. However, for prompts with higher toxicity levels, the independent NADO outperforms the coupling NADO. The independent NADO leverages

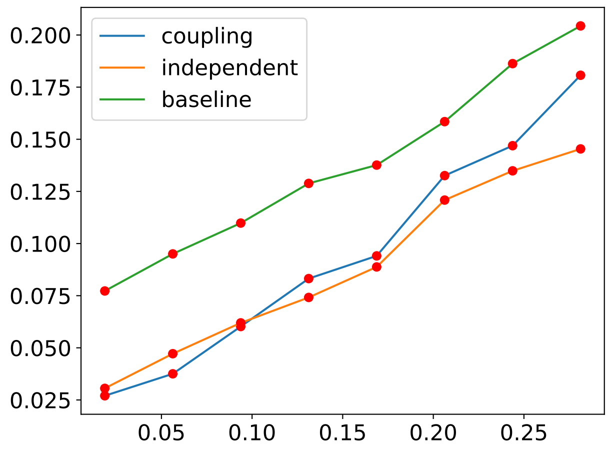


Figure 8.1: Comparative study of the effect of coupling in NADO architecture. x, y-axis are the toxicity score of the prompts and generated texts, respectively.

the pretrained weights more effectively, leading to better performance despite comparable perplexity scores. The results are shown in Figure 8.1.

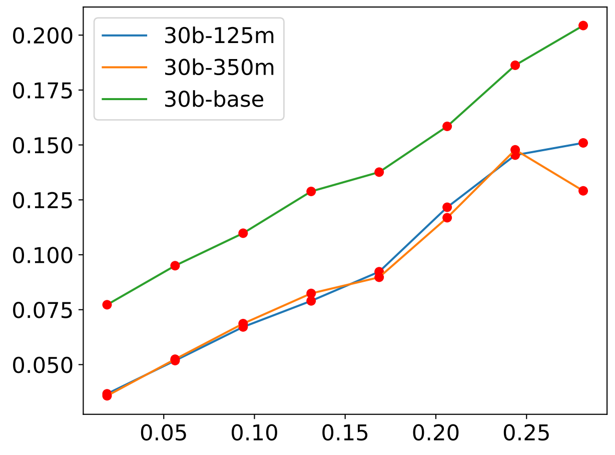
8.2.3 The Effect of Base Model Size

Next, we analyze the impact of the base model size on the controllable generation. A larger base model tends to provide a better next token distribution, but it can also result in a more complex and challenging control task. In our study, we select three base model sizes: 125m, 350m, and 30b. We use opt-350m as the NADO model for control. The performance results are presented in Table 8.2a.

Our findings reveal that the base model size does not significantly affect the toxicity scores before and after control. This demonstrates that the performance of controllable generation via auxiliary models is not highly sensitive to the base model size. Although we observe a slightly larger perplexity increase in the larger controlled models compared to the base models, the differences remain relatively narrow.

Base size	Method	Max. tox.	PPL
125m	base	0.54	14.73
	NADO	0.33	14.90
350m	base	0.55	12.97
	NADO	0.33	13.06
30b	base	0.55	9.27
	NADO	0.32	11.65

(a) Ablation study of base model size in detoxification evaluated by maximum toxicity. The auxiliary model (NADO) size is fixed to be 350m.



(b) Ablation study of auxiliary model (NADO) size in detoxification.

Table 8.2: Ablation study of the base model and auxiliary model sizes in detoxification.

8.2.4 The Effect of Auxiliary Model Size

Lastly, we analyze the impact of the auxiliary model size on controlling the base model. In this analysis, we use OPT-30b as the base model and investigate the following question: How many parameters does the auxiliary model require to effectively control the base model? We consider auxiliary model sizes of 125m and 350m, and the results are depicted in Figure 8.2b.

The two curves for the 125m and 350m NADO models exhibit remarkable similarity. This indicates that a 125m NADO model, which is only 1/240 the size of the 30b base model, possesses sufficient capacity to learn the toxicity criteria and effectively control the base model. Based on this observation, we infer that even for larger models such as GPT-3 or OPT-175b, an auxiliary model with a considerably smaller size is still capable of effectively controlling toxicity.

CHAPTER 9

Attribute Controlled Fine-tuning for Large Language Models

9.1 Introduction

Large language models (LLMs) have demonstrated impressive performance across a variety of tasks which has led to their widespread adoption for a multitude of generative AI applications. However, they carry the risk of producing inappropriate, unsafe, unfair outputs [WFK19, SCN19, GGS20, HSW24]

Ideally, LLMs should learn to comply with constraints and policies specified by users. For example, in a user-facing application like a chatbot, LLMs should never generate toxic or offensive responses, nor to divulge sensitive information. While there are several post hoc methods to moderate LLMs' outputs [MZA23], it lacks an efficient and principal approach to training LLMs to adhere to constraints.

Formally, we define a sequence-level oracle as a function that takes an LLM's output and returns whether it satisfies a predefined set of attribute constraints. In practice, the oracle can be a rule-based, model-based, or mixed system (e.g., a classifier that decides whether a sentence is toxic). Given a pre-trained LLM and the oracle, we aim to fine-tune an LLM to achieve the following: 1) **Attribute control:** The LLM output passes the oracle with a high probability. 2) **Utility preservation:** The LLM maintains performance comparable to the original LLM on utility benchmarks. 3) **Training efficiency:** The cost of fine-tuning with attribute control is similar to that of the original fine-tuning process.

While existing approaches can meet some of these criteria, achieving all of them is challenging. For example, filtering training data with the oracle function before fine-tuning is a simple and efficient method. However, this approach often results in utility drops due to potentially excessive data volumes being removed and discarding data containing information that was required for utility improvement. Another promising approach is reinforcement learning (RL) considering controlling criteria in reward function [SKS23, MLG23]. However, RL setups tend to be inefficient and require significant overhead in generating preference data compared to generic fine-tuning.

In this work, we propose a novel solution to training LLM with constraints. Inspired by the classic idea of constraint-driven learning [CRR07] and posterior regularization [GGG10], we incorporate constraints as a regularizer during the fine-tuning process. Specifically, we estimate the closest distribution from the current model that satisfies the constraints, and penalize the gap from current model distribution to this estimated distribution to regularize the LLM during fine-tuning. We iterate through this process to push the LLM closer to the feasible region of generations, making the estimation progressively more accurate.

This iterative fine-tuning process updates the base LM and regularizer sequentially, causing the running time significantly longer than original fine-tuning approach. Thus, we parallelize our algorithm by updating the base LM and regularizer simultaneously based on their status in last iteration. Empirically, the parallelization achieves same level performance compared to sequential, and the time complexity is same as original fine-tuning approach.

To verify the effectiveness of our proposed approach, we utilize toxicity as a control attribute and conduct experiments on detoxification, multi-task scenario and toxicity classification. Generally, our approach achieve the best toxicity control and maintain the same level of performance compared to the original model. Compared to related methods, we can achieve the best trade-off between the model utilities and toxicity control.

We also conduct experiments on toxicity classification task. In this task, we have two goals that are conflict in original fine-tuning approach: improving the model capability in understanding toxicity, and decrease the toxicity in model generation. However, with our approach, we successfully control the toxicity generation, while achieve same level classification performance compared to original fine-tuning approach.

We summarize our contributions as follows:

- We provide an efficient and effective solution to the attribute controlled fine-tuning problem.
- Empirically, we achieve the current best trade-off between attribute control (measured using toxicity) and utility performance against a suite of baselines.
- We show that our approach enables the model to retain knowledge of the concept of a given attribute (tested via attribute classification tasks) and yet selectively choose to avoid generating it. This can not be achieved via generic fine-tuning.

9.2 Methodology

In this section, we first formally define the problem of attribute controlled fine-tuning. We then discuss our posterior regularization method and its approximation version with an auxiliary model. Finally, we introduce an adaptive regularizer method to allow the model to apply different constraints on different fine-tuning corpus.

9.2.1 Formalization and Algorithm Overview

We use p_θ to denote the LLM and θ is its trainable weights, $\mathbf{x}$ is the input (e.g. prompts), and $\mathbf{y}$ is the generation of the model. We denote $\mathbf{y}_{<i}$ as the generation prefix $(y_0, y_1, \ldots, y_{i-1})$. $C\mathbf{x}, \mathbf{y} : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ denotes a black box oracle function which takes prompt $\mathbf{x}$ and model output $\mathbf{y}$ as input, and predicts whether the generation satisfies the constraints.

For example, the oracle takes a user input $\mathbf{x}$ and the model response $\mathbf{y}$ as input, predicting 0 when the response is offensive, indicating that the response is not acceptable.

Given an LLM p_θ, a black box oracle $C\mathbf{x}, \mathbf{y}$, and a training dataset $D = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, our goal is to fine-tune the LLM as $p_{\tilde{\theta}}$ so that in some evaluation input space $\mathcal{X}$, the model satisfies the constraint in expectation:

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{X},\mathbf{y}\sim p_{\tilde{\theta}}(\mathbf{y}|\mathbf{x})}[C\mathbf{x}, \mathbf{y}] \geq \delta. \tag{9.1}$$

Post fine-tuning, we also require that the fine-tuned model $p_{\tilde{\theta}}$ preserves the utility of the original LLM p_θ.

Illustrated in Fig. 9.1, our algorithm starts from the base model p_θ and a feasible distribution region Q. From the base model p_θ, we estimate the closest distribution q^* in Q (Sec. 9.2.3, 9.2.4). We leverage q^* as a reference distribution to regularize the fine-tuning (Sec. 9.2.2). This process can be done iteratively (Sec. 9.2.5). To speed up the fine-tuning process, we propose parallel fine-tuning (Sec. 9.2.6).

9.2.2 Fine-tuning LLM with Posterior Regularization

Given a training data $\mathbf{x}, \mathbf{y}$, typically the objective we fine-tune the LLM p_θ is defined as

$$L_{LM}(p_\theta; \mathbf{x}, \mathbf{y}) = \sum\nolimits_i L_{CE}(p_\theta(y_i|\mathbf{x}, \mathbf{y}_{<i}), 1),$$

where L_{CE} is the cross-entropy loss. To achieve attribute control, we propose to add a regularization term that penalizes the violation of constraints.

The general idea of our approach is to fine-tune LM with a regularizer penalize the Following posterior regularization [GGG10], we define

$$Q := \{q \mid \mathbb{E}_{\mathbf{x}\sim\mathcal{X},\mathbf{y}\sim q(\mathbf{y}|\mathbf{x})}[C\mathbf{x}, \mathbf{y}] \geq \delta\}$$
$$D_{KL}(p_\theta \| Q) := \min\nolimits_{q\in Q} D_{KL}(p_\theta \| q).$$

The feasible region Q is the set of distributions that satisfy the constraint in Eq. (9.1). Illustrated by Fig. 9.1(a), the regularization term D_{KL} is defined as the smallest distance from

Q measured by Kullback–Leibler (KL) divergence. The overall objective of fine-tuning is

$$L(p_\theta; \mathbf{x}, \mathbf{y}, Q) := L_{LM}(p_\theta; \mathbf{x}, \mathbf{y}) + \lambda D_{KL}(p_\theta \| Q), \tag{9.2}$$

where λ is the hyper-parameter balancing the two terms.

9.2.3 Optimal Distribution in Feasible Region

To compute the regularizer term in fine-tuning, we need to find the optimal distribution q^* as the reference distribution by solving the optimization problem

$$q^* = \arg\min_{q: E_{\mathbf{y}\sim q(\mathbf{y}|\mathbf{x})}[C\mathbf{x},\mathbf{y}]=1} D_{KL}(q \| p). \tag{9.3}$$

[MLP22] shows the close-form solution can be derived as

$$q^*(y_i | \mathbf{x}, \mathbf{y}_{<i}) \propto p_\theta(y_i | \mathbf{x}, \mathbf{y}_{<i}) R_C^p(\mathbf{x}, \mathbf{y}_{<i} \oplus y_i), \tag{9.4}$$

where $\oplus$ represents concatenation operation. $R_C^p(\mathbf{x}, \mathbf{y}_{<i})$ is the probability that the LLM will pass the oracle when the generation finishes given input $\mathbf{x}$ and prefix $\mathbf{y}_{<i}$, and is given by

$$R_C^p(\mathbf{x}, \mathbf{y}_{<i}) = \Pr_{\mathbf{y}\sim p_\theta(\mathbf{y}|\mathbf{x},\mathbf{y}_{<i})}[C\mathbf{x}, \mathbf{y} = 1] \tag{9.5}$$

Unfortunately, although the function R_C^p is well-defined, it is not tractable. To achieve the optimal solution in Eq. (9.3), in this work, we estimate R_C^p from the training data and the LLM, and update the two terms in Eq. (9.3) iteratively. In sections 9.2.4 and 9.2.5, we describe how we estimate R_C^p from the data and the current model p_θ, and how we update the model p_θ with the help of the estimated R_C^p.

Note that in fine-tuning objective Eq. (9.2), the reference distribution q is fixed and we update p_θ, so the regularizer is $D_{KL}(p_\theta \| q)$. However, here the model p is fixed and we seek for optimal q, so we minimize $D_{KL}(q \| p_\theta)$.

9.2.4 Estimating R_C^p from Training Data and LLM

To estimate R_C^p, we train an auxiliary model R_ϕ from the training data weighted by the base LLM p_θ. Formally, we first remove repetition in D to get $\tilde{D}$, and set the objective function as

$$L(R_\phi; \mathbf{x}, \mathbf{y}) = p_\theta(\mathbf{y}|\mathbf{x}) \sum_i L_{CE}(R_\phi(\mathbf{x}, \mathbf{y}_{<i}), C(\mathbf{x}, \mathbf{y}_{<i})). \tag{9.6}$$

Considering the expectation on the empirical distribution $\tilde{D}$, we have

$$\begin{aligned}&\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\tilde{D}}[p_\theta(\mathbf{y}|\mathbf{x}) L_{CE}(R_\phi(\mathbf{x}, \mathbf{y}_{<i}), C\mathbf{x}, \mathbf{y})]\\ =&\mathbb{E}_{\mathbf{x}\sim\tilde{D},\mathbf{y}\sim p_\theta(\mathbf{y}|\mathbf{x})}[L_{CE}(R_\phi(\mathbf{x}, \mathbf{y}_{<i}), R_C^p(\mathbf{x}, \mathbf{y}_{<i}))],\end{aligned}$$

Therefore, the global minimum of the objective function is $R_\phi(\mathbf{x}, \mathbf{y}_{<i}) = R_C^p(\mathbf{x}, \mathbf{y}_{<i})$.

In [MLP22] the auxiliary mode is trained by the data sampled from p_θ without weighting the data. The expected loss is the same. In our experiments we apply sampling to train the auxiliary model, when there is no available training data. Hereafter in this work, we follow [MLP22] and refer to this auxiliary model as the neurally-decomposed oracle (NADO).

9.2.5 Iteratively Updating p_θ by Regularized Fine-tuning

Once we estimate R_p^C by NADO R_ϕ, we are able to get the estimated optimal distribution q from Eq. (9.3) by replacing R_p^C with R_ϕ. We then plug in the estimated optimal distribution to the fine-tuning objective in Eq. (9.2) as

$$\begin{aligned}&L(p_\theta; \mathbf{x}, \mathbf{y}, q)\\ =&L_{LM}(p_\theta; \mathbf{x}, \mathbf{y}) + \lambda D_{KL}(p_\theta(\mathbf{y}|\mathbf{x}) \| q(\mathbf{y}|\mathbf{x}))\\ =&\sum_i \log p_\theta(y_i|\mathbf{x}, \mathbf{y}_{<i}) + \lambda D_{KL}(p_\theta(y_i|\mathbf{x}, \mathbf{y}_{<i}) \| q(y_i|\mathbf{x}, \mathbf{y}_{<i}))\end{aligned} \tag{9.7}$$

Intuitively, a model fine-tuned with the objective in Eq. (9.7) exhibits a trade-off between the model quality and the amount of control. Fine-tuned on this objective, the model converges at some mid point between p_θ and q.

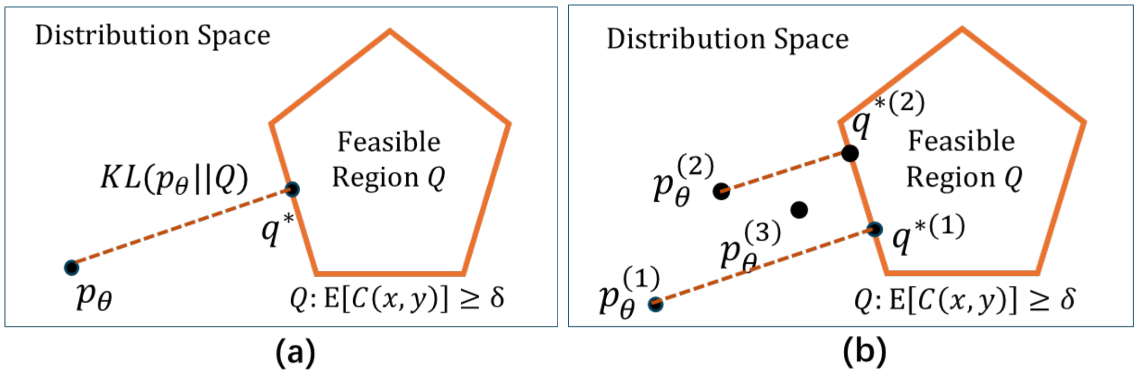


Figure 9.1: A conceptually visualization of base LLM distribution p_θ and optimal distribution q^* in fine-tuning. The polygon is representing the feasible region Q where the constraints are satisfied. On (a) it shows the regularizer term is defined as the closest distance from p_θ to Q. Regularized by KL-divergence from q, on (b) we show the LLM distribution p_θ is gradually pushed towards the feasible region.

Now we are able to estimate R_p^C by R_ϕ from the training data and model p_θ (Sec. 9.2.4), and fine-tune p_θ with estimated optimal distribution q derived from R_ϕ. It is straightforward to update both models iteratively, and we call this process as sequential fine-tuning. Compared to a single round updating, e.g., decoding-time control, we gradually push the base model distribution towards the feasible region, and the estimated optimal distribution can be more accurate. As shown in Fig. 9.2(a) and described in Sec. 9.2.1, we iteratively run the following three steps:

- Based on current LLM $p_\theta^{(i)}$, sample or weight data $D^{(i)}$ labeled by the oracle.
- Train NADO $R_\phi^{(i)}$ using the data $D^{(i)}$ initialized with $R_\phi^{(i-1)}$.
- Fine-tune the LLM $p_\theta^{(i)}$ with the KL-divergence between $p_\theta^{(i)}$ and $q^{(i)}$ given by Eq. (9.4).

The distribution of the base model can be conceptually visualized in Fig. 9.1(b) during fine-tuning. As the base model $p_\theta^{(i)}$ getting closer to the feasible region, the estimated optimal distribution $q^{(i)}$ will be more accurate compared to the estimation from the original base model distribution $q^{(1)}$.

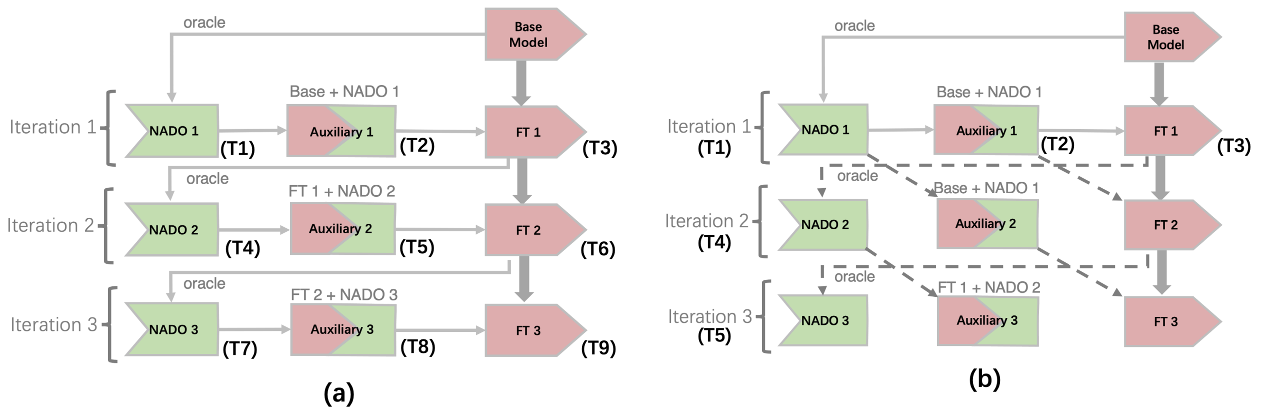


Figure 9.2: An illustration of sequential and parallel fine tuning for three iterations. We use T (time step) to indicate the time. Oracle, symbolizes the process of sampling data from an LLM, labeling with an oracle and training the NADO model. On the left, we show sequential execution with the grey arrows showing the direction of flow. On the right, we show the parallelized execution. Note that in this case, all components (left to right) of each iteration is run at the same time step (except in iteration 1). Note also, that the grey dashed arrows (from iteration 2 onwards) do not flow across components within the same iteration level, indicating the independence of each component from other components in the same level. This allows them to be executed in parallel.

9.2.6 Parallel Fine-tuning

The iterative fine-tuning process outlined in Section 9.2.5 executes its steps sequentially solving the optimization problem in Eq. (9.3) in each round. However, while accurate, it is also inefficient. In this section, we propose a parallel fine-tuning method which works to improve efficiency.

In parallel fine-tuning, we propose a set up that works to process the three steps outlined in Section 9.2.5 in parallel (see Fig. 9.2(b)). Given $p_\theta^{(i)}$, $D^{(i)}$ and $q^{(i)}$, the following three steps are processed simultaneously:

- Based on current LLM $p_\theta^{(i)}$, sample or weight data $D^{(i+1)}$ labeled by the oracle.
- Train NADO $R_\phi^{(i+1)}$ using data $D^{(i)}$ initialized with $R_\phi^{(i)}$.
- Fine-tune the LLM $p_\theta^{(i+1)}$ with the KL-divergence between $p_\theta^{(i)}$ and $q^{(i)}$ given by Eq. (9.4).

After one round, we get $p_\theta^{(i+1)}$, $D^{(i+1)}$ and $q^{(i+1)}$. The LLM keeps fine-tuning on the dataset with a regularizer, and the regularizer is updated at every checkpoint. In sequential fine-tuning, the fine-tune process will terminate at each checkpoint, waiting for the regularizer to update with the data sampled or weighted by the LLM. Compared to a baseline which fine-tunes without control, the extra time cost in our method is only the extra computation on the regularizer and the time cost in dumping checkpoints. The additional memory cost for NADO is not significant, since it is relatively small compared to the base LLM.

9.2.7 Adaptive Regularizer

The data for fine-tuning a LLM usually consists of a mix of multiple sources. In practice, we may need to apply different constraints on different fine-tuning datasets. For example, when fine-tuning on datasets that might inadvertently contain toxic content, we would want the model to learn not to generate toxic content. However, even with this control in place, we would like the model to not vary too much from the origin distribution on a general dataset. In this case, we can apply a corresponding regularizer for different constraints.

Formally, we denote the training dataset as $D = \bigcup_{i=1}^{N} D_i$. For each subset D_i, it has a corresponding constraint oracle C_i. We use the base LLM to weight the subset, and C_i to label them. According to Eq. (9.6), we train NADO R_{ϕ_i} for the constraint oracle C_i, and compute the estimated optimal distribution q_i. We use q_i as the reference distribution in the KL-divergence. Therefore, in different subsets, we use different regularizers. Specifically, when we set q_i as the original distribution, the regularizer is used to preserve the distribution. We refer to this regularizer as the *preserving regularizer*.

In this work, we demonstrate performance in the single task of controlling the toxicity of the LLM while preserve the same performance level as the original model. We apply the toxicity regularizer when fine-tuning on toxicity-related datasets, while using preserving regularizer to preserve the distribution on other datasets. Formally, we denote p_0 as the original LLM, q as the estimated optimal distribution under toxicity constraint oracle, and $D_T \subset D$ as the toxicity-related training set. We adapt the fine-tuning objective in Eq. (9.2) to

$$\begin{aligned} L(p_\theta; D, q) = & \sum\nolimits_{\mathbf{x},\mathbf{y}\in D} L_{LM}(p_\theta; \mathbf{x}, \mathbf{y}) \\ & + \lambda \sum\nolimits_{\mathbf{x},\mathbf{y}\in D_T} D_{KL}(p_\theta(\mathbf{y}|\mathbf{x}) \| q(\mathbf{y}|\mathbf{x})) + \lambda \sum\nolimits_{\mathbf{x},\mathbf{y}\notin D_T} D_{KL}(p_\theta(\mathbf{y}|\mathbf{x}) \| p_0(\mathbf{y}|\mathbf{x})) \end{aligned} \tag{9.8}$$

9.3 Experiments

For our experiments, we choose toxicity as the controlled attribute. Toxicity, as discussed in Section 9.1, is of significant importance as a metric for LLM evaluation [BMR20, TLI23, CND22, TMS23]. In this context, we apply our fine-tuning schema in three different scenarios; (1) **detoxification:** testing the effectiveness of our proposed approach in attribute control, (2) **multi-task scenario:** testing that the controlled model preserves the same level performance on other tasks, and (3) **toxicity classification:** testing whether the control affects the model performance on attributes related tasks. Detailed notes on data

Model	API Toxicity	ToxiGen
Llama baseline	0.315	23.0
Reinforcement Learning	0.269	12.3
NADO Decoding Control	0.289	14.4
Ours (sequential)	**0.259**	11.0
Ours (parallel)	0.261	**10.9**

Table 9.1: Toxicity scores of Llama-7B model with different detoxification methods.

pre-processing, hyper-parameter choice for model training and the architecture of auxiliary models can be found in the Appendix.

9.3.1 Detoxification

We first test the performance of our method on the detoxification task.

Setup We use Llama-7B [TLI23] as the base model. NADO has a similar architecture but with only 8 layers. For our experiments, we use RealToxicPrompts (RTP) [GGS20] and ToxiGen [HGP22] datasets. For each dataset, we sample 50k prompts for fine-tuning and another 5k for evaluation. During the evaluation, we prompt the model with each data point from the evaluation set and generate 32 tokens. For the RTP dataset, we measure the average toxicity across the generations by using PerspectiveAPI. For ToxiGen, we use the pre-trained Toxigen (RoBERTa) classifier, which was released with the dataset, to calculate the percentage of generated sentences that are toxic. We test three detoxification methods, in addition to the Llama baseline:

- **Reinforcement Learning:** For each prompt in the evaluation set, we sample 32 generations. We utilize the PerspectiveAPI and ToxiGen classifier confidence scores as reward for the two test sets, respectively. We then use the policy gradient [SMS99] to update the base language model.

- **NADO controlled decoding:** For each prompt in the two test sets, we sample 32 sentences and obtain binary labels from PerspectiveAPI and the ToxiGen classifier, respectively. When using PerspectiveAPI we set a threshold of toxicity score > 0.1 as toxic.
- **Ours:** We follow the NADO controlled decoding oracle setup. We split the 50k fine-tuning set into 5 groups. We separately run iterative sequential fine-tuning and parallel fine-tuning for 5 rounds using these groups.

Results The results are shown in Tab. 9.1. We observe that on both datasets our method achieves the best detoxification (given the same amount of training data). We observe that there is a significant performance improvement brought on by iterative fine-tuning when compared to NADO controlled decoding, which shows that directly estimated the optimal distribution is not optimal. The iterative process enables the gradual push of the base model distribution towards the feasible region (Fig 9.1), and the estimated optimal distribution improves in its accuracy. The sequential and parallel fine-tuning results show comparable performance. Since parallel fine-tuning is more efficient, we focus on this method from this point onward.

9.3.2 Multi-task Scenario

Here we explore how our proposed control method applied on the toxicity attribute, affects model utility and performance on other tasks.

Setup We use Llama-7B [TLI23] and Falcon-7B [AAA23] as base models, and fine-tune each of them on a mixture of ToxiGen andWikitext [MXB16] data in equal proportions. We evaluate model performance on toxicity and utility by using the following metrics:

- **ToxiGen (toxicity):** Same set up as the detoxification experiment in Section 9.3.1.
- **MMLU (utility):** We do 5-shot evaluation on the MMLU benchmark [HBB21] and report the average score.

Model		ToxiGen	MMLU(5-shot)	Com. Reasoning (0-shot)
Llama-7B	Baseline	23.0	35.1	75.6
	Filtering	21.9	34.6	75.1
	RL	15.2	33.6	73.2
	NADO decoding	16.8	31.1	71.4
	Ours w/o Adaptive	13.6	30.4	71.9
	Ours w/ Adaptive	14.2	33.9	73.6
Falcon-7B	Baseline	14.0	27.2	76.1
	Filtering	13.6	26.4	74.9
	RL	9.8	25.4	74.4
	NADO decoding	7.3	23.6	72.5
	Ours w/o Adaptive	7.1	23.1	71.8
	Ours w/ Adaptive	7.3	26.1	74.5

Table 9.2: Benchmark performance of Llama-7B and Falcon-7B with toxicity control.

- **Commonsense Reasoning (utility):** We do 0-shot evaluation on 4 commonsense reasoning benchmarks, BoolQ [CLC19], PIQA [BZB20], HellaSwag [ZHB19] and WinoGrande [SBB20], and report the average score.

We test 5 different methods:

- **Filtering:** We filter out all the data labeled as toxic by the ToxiGen classifier.
- **Reinforcement Learning:** We take the confidence score provided by the ToxiGen classifier as reward, and apply policy-gradient to minimize the toxicity.
- **NADO controlled decoding:** We train the auxiliary model on ToxiGen sampled data, and control the model generation at decoding time.
- **Ours (without Adaptive):** We apply parallel fine-tuning on both datasets with the auxiliary model trained on ToxiGen sampled data.

Win rate(top row)	Base	Filter	RL	Ours
Base	N/A	44.3	45.1	51.4
Filter	55.7	N/A	53.4	61.6
RL	54.9	46.6	N/A	61.3
Ours	48.6	38.4	38.7	N/A

Table 9.3: Pairwise comparison by OPT-30B on ToxiGen sampling data.

- **Ours (with Adaptive):** We apply an adaptive regularizer as described in Eq. (9.8). We use the regularizer with the original distribution as reference on Wikitext data, while using the toxicity control regularizer on the ToxiGen sampled data.

Results The results are shown in Tab. 9.2. We observe that all detoxification methods cause a performance drop on our utility metrics (i.e. MMLU and commonsense reasoning). Filtering is not effective for detoxification. Our method with the adaptive regularizer achieves the best trade-off between toxicity control and model utility.

When used without the adaptive regularizer, our method achieves the best toxicity control. However, this comes at the cost of utility loss. This indicated that the toxicity regularizer trained on ToxiGen sampled data does not perform well on the Wikitext data. The adaptive regularizer helps preserve the model utility while fine-tuning on Wikitext data.

We note that Falcon-7B has much lower toxicity when compared to Llama-7B. The consistent performance trends observed in both base models, demonstrate that our method is robust to different base models independent of its levels of toxicity.

We further analyze the generation quality of a larger model, OPT-30B [ZRG22], to do pairwise comparison on model generations for ToxiGen prompts from 4 systems: (1) the base Llama-7B model, (2) filtering, (3) RL and (4) ours with the adaptive regularizer. We

Model	API Toxicity	Classify ROC
baseline	0.315	0.910
SFT(LLM loss)	0.344	**0.966**
Ours(LLM loss)	**0.288**	0.959
SFT(classification)	0.314	0.972

Table 9.4: Jigsaw dataset performance of Llama-7B model with toxicity control.

do not consider NADO controlled decoding and ours without the adaptive regularizer, as they are obviously worse in terms of model quality. The results are shown in Tab. 9.3. We show that OPT-30B prefers our system (with the adaptive regularizer) the best, with slight improvement over the base model.

9.3.3 Toxicity Classification

An interesting research question is whether the model actually understands what toxicity is, or if it simply ignores toxicity-related contents and ignores those texts in both input and generation. To answer this question, we design an experiment on learning knowledge about toxicity while detoxifying language models. If the model simply ignores toxicity related contents, it cannot perform well on the tasks about toxicity, and we will observe a consistent trend between the task performance and toxicity in generation.

Setup We fine-tune the Llama-7B on the Jigsaw toxicity classification dataset [JVI22]. We compare the performance of models fine-tuned using our controlled method to ones fine-tuned using uncontrolled fine-tuning. We use classification performance and generation toxicity (as evaluated by PerspectiveAPI) as metrics of comparison. Specifically, we compare three methods:

- **Supervised fine-tuning with LLM loss:** We concatenate each question and answer in the Jigsaw dataset, and fine-tune with a language modeling objective.

- **Ours with LLM loss:** We train an auxiliary model on RTP sampled data labeled by PerspectiveAPI, and fine-tune the language model same as above on Jigsaw dataset with the toxicity regularizer.
- **Supervised fine-tuning as classification:** We treat each question in Jigsaw as the prompt and only calculate loss on the answers. This is regarded the upper bound of performance for this task.

Results The results are shown in Tab. 9.4. We observe that if we fine-tune the LLM on the Jigsaw dataset without toxicity control, the generation toxicity increases significantly (9.2%, from 0.315 to 0.344). The reason is that Jigsaw consists of toxic content and fine-tuning on this shifts the model out distribution to be toxic. In comparison, when using our fine-tuning schema which leverages the toxicity regularizer, we achieve decreased toxicity. Notably our method also improves classification performance, achieving almost similar performance to uncontrolled fine-tuning, demonstrating our approach makes the model understand the toxicity rather than simply make model ignore the toxicity contents in training data.

9.4 Conclusion

We propose a novel approach to controlling LLM generations via fine-tuning with attribute control. Using toxicity as our attribute of choice, we showcase results on detoxification, multi-task fine-tuning and toxicity classification. With negligible extra cost in time and space compared to typical fine-tuning, our approach shows significant improvements with respect to the baselines we use.

CHAPTER 10

Conclusion

With the rapid development of NLP models, we are aware that it is critical to control the models. In this dissertation, in order to control the models, we propose to leverage constraints, and introduce multiple techniques in constrained inference and constrained decoding. The proposed methods are shown to be effective and also efficient in structure prediction models and the popular large language models.

Existing literature lacks a throughout study about how to handle different kinds of constrains in different models. Usually, prior work forcus on a specific task, a specific constraint and a specific neural model. In this dissertation, we cover a great propotion of scenarios. In Chap. 3 we introduce the constrained inference for instance-level, corpus-level and distribution-level constraints in structure prediction models. In Chap. 4 and Chap. 5 we introduce two of the applications. In Chap. 6 we introduce mining linear constraints and corresponding constrained inference. In Chap. 7 we introduce constrained decoding in language model, and in Chap. 9 we introduce a fine-tuning approach with attribute control based on constrained decoding.

Constraints encode human knowledge, and point out the difference or discrepency between the model output and human expectation. Leveraging constraints can also improve align the model outputs to human if proper constraints are provided. Furthermore, constrained inference is also an approach to query large language models for specific knowledge or generation. Thus, in my understanding, constraints encode the knowledge and expectation of we human, and constrained inference and decoding is the bridge between human and models.

Constrained inference is powerful. Its goal is to satisfy the pre-defined constraints. Therefore, defining good constraints is the precondition to achieve good results via constrained inference. However, the constraints can be complicated and abstract. With the development of NLP and large language models, the evaluation for NLP models is getting complicated. In the past researchers in NLP focus more on some basic metrics including classification accuracy, and other simple matching based metrics like BLEU. Later, we care more about some abstract evaluation like fluency, consistency and the style of the output. Human evaluation plays an important role in model or algorithm evaluation. Nowadays, the alignment with human becomes a crucial metric for large language models. Hence, the constraints we expect to inject also become complicated and abstract, which leads to demanding of advanced techniques handling such constraints.

References

[AAA23] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. "The Falcon Series of Open Language Models." *CoRR*, **abs/2311.16867**, 2023.

[AHB18] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering." In *CVPR*, 2018.

[AHS85] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines." *Cognitive science*, **9**(1):147–169, 1985.

[AZM19] Wasi Uddin Ahmad, Zhisong Zhang, Xuezhe Ma, Eduard Hovy, Kai-Wei Chang, and Nanyun Peng. "On Difficulties of Cross-Lingual Transfer with Order Differences: A Case Study on Dependency Parsing." In *NAACL*, 2019.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In *ICLR*, 2015.

[BFL98] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. "The Berkeley FrameNet Project." In *COLING-ACL*, 1998.

[BGJ17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching Word Vectors with Subword Information." *Transactions of the Association for Computational Linguistics*, **5**:135–146, 2017.

[BMR20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models are Few-Shot Learners." In *NeurIPS*, 2020.

[BZB20] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. "PIQA: Reasoning about Physical Commonsense in Natural Language." In *AAAI*, 2020.

[CC11] Yin-Wen Chang and Michael Collins. "Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation." In *EMNLP*. ACL, 2011.

[CHW22] Hong Chen, Rujun Han, Te-Lin Wu, Hideki Nakayama, and Nanyun Peng. "Character-Centric Story Visualization via Visual Planning and Token Alignment." In *EMNLP*, 2022.

[CK78] Seth Chaiken and Daniel J Kleitman. "Matrix tree theorems." *Journal of combinatorial theory, Series A*, **24**(3):377–381, 1978.

[CL65] Y. J. Chu and T. H. Liu. "On the shortest arborescence of a directed graph." *Science Sinica*, **14**, 1965.

[CL08] James Clarke and Mirella Lapata. "Global Inference for Sentence Compression: An Integer Linear Programming Approach." *J. Artif. Intell. Res.*, **31**:399–429, 2008.

[CLC19] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. "BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions." In *NAACL-HLT (1)*, 2019.

[CND22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. "PaLM: Scaling Language Modeling with Pathways." *CoRR*, **abs/2204.02311**, 2022.

[CRR07] Ming-Wei Chang, Lev-Arie Ratinov, and Dan Roth. "Guiding Semi-Supervision with Constraint-Driven Learning." In *ACL*. The Association for Computational Linguistics, 2007.

[CSK13] Kai-Wei Chang, S. Sundararajan, and S. Sathiya Keerthi. "Tractable Semi-supervised Learning of Complex Structured Prediction Models." In *ECML/PKDD*, 2013.

[CUK15] Kai-Wei Chang, Shyam Upadhyay, Gourab Kundu, and Dan Roth. "Structural Learning with Amortized Inference." In *AAAI*, 2015.

[Dal15] Bhavana Bharat Dalvi. *Constrained Semi-supervised Learning in the Presence of Unanticipated Classes*. PhD thesis, Google Research, 2015.

[DCL18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *arXiv preprint arXiv:1810.04805*, 2018.

[DKL11] Ivica Dimitrovski, Dragi Kocev, Suzana Loskovska, and Sašo Džeroski. "Hierarchical annotation of medical images." *Pattern Recognition*, **44**(10-11):2436–2449, 2011.

[DM17] Timothy Dozat and Christopher D Manning. "Deep biaffine attention for neural dependency parsing." *Internation Conference on Learning Representations*, 2017.

[DML20] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. "Plug and Play Language Models: A Simple Approach to Controlled Text Generation." In *ICLR*. OpenReview.net, 2020.

[Dry07] Matthew S Dryer. "Word order." *Language typology and syntactic description*, **1**:61–131, 2007.

[Fel98] C Fellbaum. "Wordnet: An on-line lexical database.", 1998.

[FG09] Menachem Fromer and Amir Globerson. "An LP View of the M-best MAP problem." In *NIPS*, 2009.

[FJ05] Bertram Felgenhauer and Frazer Jarvis. "Enumerating possible Sudoku grids." *Preprint available at http://www. afjarvis. staff. shef. ac. uk/sudoku/sudoku. pdf*, 2005.

[FJ08] Thomas Finley and Thorsten Joachims. "Training structural SVMs when exact inference is intractable." In *ICML*, 2008.

[GCW20] Seraphina Goldfarb-Tarrant, Tuhin Chakrabarty, Ralph Weischedel, and Nanyun Peng. "Content Planning for Neural Story Generation with Aristotelian Rescoring." In *EMNLP*, 2020.

[GGG10] Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. "Posterior Regularization for Structured Latent Variable Models." *J. Mach. Learn. Res.*, **11**:2001–2049, 2010.

[GGS20] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. "RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models." In *EMNLP (Findings)*, volume EMNLP 2020 of *Findings of ACL*, pp. 3356–3369. Association for Computational Linguistics, 2020.

[GPS17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. "On Calibration of Modern Neural Networks." In *ICML*, 2017.

[Gur16] Inc. Gurobi Optimization. "Gurobi Optimizer Reference Manual.", 2016.

[GWZ19] Jiatao Gu, Changhan Wang, and Junbo Zhao. "Levenshtein transformer." *NeurIPS*, **32**, 2019.

[HBB21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. "Measuring Massive Multitask Language Understanding." In *ICLR*, 2021.

[HCT22] Rujun Han, Hong Chen, Yufei Tian, and Nanyun Peng. "Go Back in Time: Generating Flashbacks in Stories with Event Temporal Prompts." In *NAACL*, 2022.

[HGP22] Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. "ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection." In *ACL (1)*, 2022.

[HL17] Chris Hokamp and Qun Liu. “Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search.” In *ACL (1)*, pp. 1535–1546. Association for Computational Linguistics, 2017.

[HM54] John M Hammersley and K William Morton. “Poor man’s monte carlo.” *Journal of the Royal Statistical Society: Series B (Methodological)*, 1954.

[HSW24] Yue Huang, Lichao Sun, Haoran Wang, Siyuan Wu, Qihui Zhang, Yuan Li, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, Xiner Li, Hanchi Sun, Zhengliang Liu, Yixin Liu, Yijue Wang, Zhikun Zhang, Bertie Vidgen, Bhavya Kailkhura, Caiming Xiong, Chaowei Xiao, Chunyuan Li, Eric P. Xing, Furong Huang, Hao Liu, Heng Ji, Hongyi Wang, Huan Zhang, Huaxiu Yao, Manolis Kellis, Marinka Zitnik, Meng Jiang, Mohit Bansal, James Zou, Jian Pei, Jian Liu, Jianfeng Gao, Jiawei Han, Jieyu Zhao, Jiliang Tang, Jindong Wang, Joaquin Vanschoren, John Mitchell, Kai Shu, Kaidi Xu, Kai-Wei Chang, Lifang He, Lifu Huang, Michael Backes, Neil Zhenqiang Gong, Philip S. Yu, Pin-Yu Chen, Quanquan Gu, Ran Xu, Rex Ying, Shuiwang Ji, Suman Jana, Tianlong Chen, Tianming Liu, Tianyi Zhou, William Yang Wang, Xiang Li, Xiangliang Zhang, Xiao Wang, Xing Xie, Xun Chen, Xuyu Wang, Yan Liu, Yanfang Ye, Yinzhi Cao, Yong Chen, and Yue Zhao. “TrustLLM: Trustworthiness in Large Language Models.” In *Forty-first International Conference on Machine Learning*, 2024.

[JGD18] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. “Marian: Fast Neural Machine Translation in C++.” In *ACL (4)*, pp. 116–121. Association for Computational Linguistics, 2018.

[JVI22] Naman Jain, Skanda Vaidyanath, Arun Shankar Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram K. Rajamani, and Rahul Sharma. "Jigsaw: Large Language Models meet Program Synthesis." In *ICSE*, 2022.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In *ICLR*, 2015.

[KBK16] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A Smith. "Distilling an Ensemble of Greedy Dependency Parsers into One MST Parser." In *EMNLP*, 2016.

[KED21] Muhammad Khalifa, Hady Elsahar, and Marc Dymetman. "A Distributional Approach to Controlled Text Generation." In *ICLR*, 2021.

[KEK22] Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. "Controlling Conditional Language Models without Catastrophic Forgetting." In *ICML*, 2022.

[KG16] Eliyahu Kiperwasser and Yoav Goldberg. "Simple and accurate dependency parsing using bidirectional LSTM feature representations." *Transactions of the Association for Computational Linguistics*, **4**:313–327, 2016.

[KGM21] Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq R. Joty, Richard Socher, and Nazneen Fatema Rajani. "GeDi: Generative Discriminator Guided Sequence Generation." In *EMNLP (Findings)*, pp. 4929–4952. Association for Computational Linguistics, 2021.

[KP07] Alex Kulesza and Fernando Pereira. "Structured Learning with Approximate Inference." In *NIPS*, pp. 785–792, 2007.

[KSR13] Gourab Kundu, Vivek Srikumar, and Dan Roth. "Margin-based Decomposed Amortized Inference." In *ACL*, 2013.

[LCR18] Guillaume Lample, Alexis Conneau, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. "Word translation without parallel data." In *ICLR*, 2018.

[LJH18] Juncen Li, Robin Jia, He He, and Percy Liang. "Delete, Retrieve, Generate: a Simple Approach to Sentiment and Style Transfer." In *NAACL-HLT*, pp. 1865–1874. Association for Computational Linguistics, 2018.

[LLG20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." In *ACL*, 2020.

[LMP22] Sidi Lu, Tao Meng, and Nanyun Peng. "InsNet: An Efficient, Flexible, and Performant Insertion-based Text Generation Model." In *NeurIPS*, 2022.

[LPM15] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation." In *EMNLP*, 2015.

[LSL21] Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. "DExperts: Decoding-Time Controlled Text Generation with Experts and Anti-Experts." In *ACL/IJCNLP (1)*, 2021.

[LWW22a] Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. "NeuroLogic A*esque Decoding: Constrained Text Generation with Lookahead Heuristics." In *NAACL-HLT*, 2022.

[LWW22b] Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. "NeuroLogic A*esque Decoding: Constrained Text Generation with Lookahead Heuristics." In *NAACL-HLT*, pp. 780–799. Association for Computational Linguistics, 2022.

[LWZ21] Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. "NeuroLogic Decoding: (Un)supervised Neural Text Generation with Predicate Logic Constraints." In *NAACL-HLT*, 2021.

[LZS20] Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. "CommonGen: A Constrained Text Generation Challenge for Generative Commonsense Reasoning." *Findings of EMNLP*, 2020.

[LZZ18] Sidi Lu, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu. "Neural text generation: Past, present and beyond." *arXiv preprint arXiv:1803.07133*, 2018.

[Mar15] André F. T. Martins. "Transferring Coreference Resolvers with Posterior Regularization." In *ACL*, 2015.

[MFA15] André F. T. Martins, Mário A. T. Figueiredo, Pedro M. Q. Aguiar, Noah A. Smith, and Eric P. Xing. "AD^3: alternating directions dual decomposition for MAP inference in graphical models." *J. Mach. Learn. Res.*, **16**:495–545, 2015.

[MLG23] Sidharth Mudgal, Jong Lee, Harish Ganapathy, YaGuang Li, Tao Wang, Yanping Huang, Zhifeng Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, Jilin Chen, Alex Beutel, and Ahmad Beirami. "Controlled Decoding from Language Models." *CoRR*, **abs/2310.17022**, 2023.

[MLP22] Tao Meng, Sidi Lu, Nanyun Peng, and Kai-Wei Chang. "Controllable Text Generation with Neurally-Decomposed Oracle." *CoRR*, **abs/2205.14219**, 2022.

[MPC19] Tao Meng, Nanyun Peng, and Kai-Wei Chang. "Target Language-Aware Constrained Inference for Cross-lingual Dependency Parsing." In *EMNLP*, 2019.

[MPR05] Ryan T. McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. "Non-Projective Dependency Parsing using Spanning Tree Algorithms." In *HLT/EMNLP*, 2005.

[MSX09] André F. T. Martins, Noah A. Smith, and Eric P. Xing. "Concise Integer Linear Programming Formulations for Dependency Parsing." In *ACL/IJCNLP*, 2009.

[MWJ13] Kevin Murphy, Yair Weiss, and Michael I Jordan. "Loopy belief propagation for approximate inference: An empirical study." *arXiv preprint arXiv:1301.6725*, 2013.

[MXB16] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. "Pointer Sentinel Mixture Models.", 2016.

[MZA23] Todor Markov, Chong Zhang, Sandhini Agarwal, Florentine Eloundou Nekoul, Theodore Lee, Steven Adler, Angela Jiang, and Lilian Weng. "A holistic approach to undesired content detection in the real world." In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 15009–15018, 2023.

[NAA18] Joakim Nivre, Mitchell Abrams, Željko Agić, and et al. "Universal Dependencies 2.2.", 2018. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[Ost15] Robert Östling. "Word Order Typology through Multilingual Word Alignment." In *ACL*, 2015.

[OWJ22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. "Training language models to follow instructions with human feedback." In *NeurIPS*, 2022.

[Par18] Kyubyong Park. "Can Convolutional Neural Networks Crack Sudoku Puzzles?" `https://github.com/Kyubyong/sudoku`, 2018.

[PCE15] Nanyun Peng, Ryan Cotterell, and Jason Eisner. "Dual Decomposition Inference for Graphical Models over Strings." In *EMNLP*, 2015.

[PKL13] Matt Post, Gaurav Kumar, Adam Lopez, Damianos G. Karakos, Chris Callison-Burch, and Sanjeev Khudanpur. "Improved speech-to-text translation with the Fisher and Callhome Spanish-English speech translation corpus." In *IWSLT*, 2013.

[PRY04] Vasin Punyakanok, Dan Roth, Wen-tau Yih, and Dav Zimak. "Semantic Role Labeling Via Integer Linear Programming Inference." In *COLING*, 2004.

[RC11] Alexander M. Rush and Michael Collins. "Exact Decoding of Syntactic Translation Models through Lagrangian Relaxation." In *ACL*, 2011.

[RC12] Alexander M Rush and MJ Collins. "A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing." *Journal of Artificial Intelligence Research*, **45**:305–362, 2012.

[RSR20a] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." *J. Mach. Learn. Res.*, **21**:140:1–140:67, 2020.

[RSR20b] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." *J. Mach. Learn. Res.*, **21**:140:1–140:67, 2020.

[RT18] Sudha Rao and Joel R. Tetreault. "Dear Sir or Madam, May I Introduce the GYAFC Dataset: Corpus, Benchmarks and Metrics for Formality Style Transfer." In *NAACL-HLT*, pp. 129–140. Association for Computational Linguistics, 2018.

[RWC19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. "Language models are unsupervised multitask learners." *OpenAI blog*, **1**(8):9, 2019.

[RY04] Dan Roth and Wen-tau Yih. "A Linear Programming Formulation for Global Inference in Natural Language Tasks." In *CoNLL*, 2004.

[RY05] Dan Roth and Wen-tau Yih. "Integer linear programming inference for conditional random fields." In *ICML*, 2005.

[SBB20] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. "WinoGrande: An Adversarial Winograd Schema Challenge at Scale." In *AAAI*, 2020.

[SCN19] Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. "The Woman Worked as a Babysitter: On Biases in Language Generation." In *EMNLP*, 2019.

[SCN21] Emily Sheng, Kai-Wei Chang, Prem Natarajan, and Nanyun Peng. ""Nice Try, Kiddo": Investigating Ad Hominems in Dialogue Responses." In *NAACL-HLT*, pp. 750–767. Association for Computational Linguistics, 2021.

[SCT20] Raymond Hendy Susanto, Shamil Chollampatt, and Liling Tan. "Lexically constrained neural machine translation with Levenshtein transformer." *arXiv preprint arXiv:2004.12681*, 2020.

[SKR12] Vivek Srikumar, Gourab Kundu, and Dan Roth. "On Amortizing Inference Cost for Structured Prediction." In *EMNLP-CoNLL*, 2012.

[SKS23] Charlie Snell, Ilya Kostrikov, Yi Su, Sherry Yang, and Sergey Levine. "Offline RL for Natural Language Generation with Implicit Language Q Learning." In *ICLR*, 2023.

[SLL15] Lifeng Shang, Zhengdong Lu, and Hang Li. "Neural responding machine for short-text conversation." *arXiv preprint arXiv:1503.02364*, 2015.

[SMH22] Alexander Spangher, Yao Ming, Xinyu Hua, and Nanyun Peng. "Sequentially Controlled Text Generation." In *EMNLP*, 2022.

[SMS99] Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In *NIPS*, pp. 1057–1063. The MIT Press, 1999.

[SSW19] Elizabeth Salesky, Matthias Sperber, and Alexander Waibel. "Fluent Translations from Disfluent Speech in End-to-End Speech Translation." In *NAACL-HLT (1)*, 2019.

[STH17] Samuel L. Smith, David H. P. Turban, Steven Hamblin, and Nils Y. Hammerla. "Offline bilingual word vectors, orthogonal transformations and the inverted softmax." In *ICLR*, 2017.

[SZ15] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In *ICLR*, 2015.

[TLI23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. "LLaMA: Open and Efficient Foundation Language Models." *CoRR*, **abs/2302.13971**, 2023.

[TMN13] Oscar Täckström, Ryan T. McDonald, and Joakim Nivre. "Target Language Adaptation of Discriminative Transfer Parsers." In *HLT-NAACL*, 2013.

[TMS23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. "Llama 2: Open Foundation and Fine-Tuned Chat Models." *CoRR*, **abs/2307.09288**, 2023.

[TP22] Yufei Tian and Nanyun Peng. "Zero-Shot Sonnet Generation with Discourse-Level Planning and Aesthetics Features." In *NAACL*, 2022.

[VL15] Oriol Vinyals and Quoc Le. "A neural conversational model." *arXiv preprint arXiv:1506.05869*, 2015.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is All you Need." In *NIPS*, 2017.

[WC16] Wenhui Wang and Baobao Chang. "Graph-based Dependency Parsing with Bidirectional LSTM." In *ACL*, 2016.

[WDW19] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver." In *ICML*, 2019.

[WFK19] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. "Universal Adversarial Triggers for Attacking and Analyzing NLP." In *EMNLP/IJCNLP (1)*, pp. 2153–2162. Association for Computational Linguistics, 2019.

[WPX22] Boxin Wang, Wei Ping, Chaowei Xiao, Peng Xu, Mostofa Patwary, Mohammad Shoeybi, Bo Li, Anima Anandkumar, and Bryan Catanzaro. "Exploring the Limits of Domain-Adaptive Training for Detoxifying Large-Scale Language Models." *CoRR*, **abs/2202.04173**, 2022.

[XZF18] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge." In *ICML*, 2018.

[YFL20] Yuan Ye, Yansong Feng, Bingfeng Luo, Yuxuan Lai, and Dongyan Zhao. "Integrating Relation Constraints with Neural Relation Extractors." In *AAAI*, 2020.

[YJW16] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. "Image Captioning with Semantic Attention." In *CVPR*, 2016.

[YK21] Kevin Yang and Dan Klein. "FUDGE: Controlled Text Generation With Future Discriminators." In *NAACL-HLT*. Association for Computational Linguistics, 2021.

[YPW19] Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. "Plan-and-write: Towards better automatic storytelling." In *AAAI*, 2019.

[YZF16] Mark Yatskar, Luke S. Zettlemoyer, and Ali Farhadi. "Situation Recognition: Visual Semantic Role Labeling for Image Understanding." In *CVPR*, 2016.

[ZHB19] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. "HellaSwag: Can a Machine Really Finish Your Sentence?" In *ACL (1)*, 2019.

[ZL14] Xingxing Zhang and Mirella Lapata. "Chinese poetry generation with recurrent neural networks." In *EMNLP*, 2014.

[ZRG22] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. "OPT: Open Pre-trained Transformer Language Models." *CoRR*, **abs/2205.01068**, 2022.

[ZWL20] Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. "Pointer: Constrained text generation via insertion-based generative pre-training." *arXiv preprint arXiv:2005.00558*, 2020.

[ZWY17] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. “Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints.” In *EMNLP*, 2017.