TOWARD MORE SCALABLE STRUCTURED MODELS

BY

SAFA MESSAOUD

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Assistant Professor Alexander G. Schwing, Chair
Professor David Forsyth
Professor Minh N. Do
Associate Professor Paris Smaragdis

ABSTRACT

While deep learning has achieved a huge success across different disciplines from computer vision and natural language processing to computational biology and physical sciences, training such models is known to require significant amounts of data. One possible reason is that structural properties of the data and problem are not modeled explicitly. Effectively exploiting the structure can help build more efficient and performing models. The complexity of the structure requires models with enough representation capabilities. However, increased structured model complexity usually leads to increased inference complexity and trickier learning procedures. Also, making progress on real-world applications requires learning paradigms that circumvent the limitation of evaluating the partition function and scale to high-dimensional datasets.

In this dissertation, we develop more *scalable* structured models, *i.e.*, models with inference procedures that can handle complex dependencies between variables efficiently, and learning algorithms that operate in high-dimensional spaces. First, we extend Gaussian conditional random fields, traditionally unimodal and only capturing pairwise variables interactions, to model multi-modal distributions with high-order dependencies between the output space variables, while enabling exact inference and incorporating external constraints at runtime. We show compelling results on the task of diverse gray-image colorization. Then, we introduce a reinforcement learning-based method for solving inference in models with general higher-order potentials, that are intractable with traditional techniques. We show promising results on semantic segmentation. Finally, we propose a new loss, max-sliced score matching (MSSM), for learning structured models at scale. We assess our model on estimation of densities and scores for implicit distributions in Variational and Wasserstein auto-encoders.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor Professor Alex Schwing for the rigorous training and continuous support throughout my time as his Ph.D. student. Alex has always been a source of inspiration because of his vast knowledge, phlegmatic attitude and constant excitement about research and new ideas. My skills and attitude wouldn't have been at their current shape without his guidance and encouragements. My sincere and heartfelt gratitude to him for making my Ph.D. journey fulfilling and worthwhile.

I am honored to have a thesis committee members that I hugely admire for their scientific contribution and high standards for the quality of research. Thank you, Professor David A. Forsyth, Professor Minh N. Do and Professor Paris Smaragdis.

I am also very grateful for the amazing environment in my research group. I thank my labmates, which I want to mention in alphabetic order subsequently: Colin Graber, Harsh Agrawal, Iou-Jen Liu, Ishan Deshpande, Jyoti Aneja, Maghav Kumar, Medhini Narasimhan, Moitreya Chatterjee, Raymond Yeh, Unnat Jain, Xiaoyang Bai, Yuan-Ting Hu and Zhongzheng Ren. I am indebted to more senior Ph.D. students from the Vision and Systems groups, notably, Tanmay Gupta, Jason Rock and Mohammad Babaeizadeh for discussions and advice at different stages of my Ph.D. Special thanks to my two mentees Shreya Jagarlamudi and Alan Q. Wang that I had the pleasure to advise on their senior thesis.

I am lucky to have had Dr. Murthy Devarakonda and Dr. Morioshi Ohara as managers, and Dr. Ching-Huei Tsou and Dr. Takeshi Ogasawara as mentors during my internships at IBM research. The exposure I had in their teams has significantly shaped the direction of my Ph.D.

My journey wouldn't have been the same without the support of many friends. In particular, I want to thank Homa Alemzadeh who was my closest friend, role model and mentor in the first phase of my Ph.D. I will forever

miss the late night gyming followed by the frozen yoghurt stop at Cocomero. Another hero of my jouney is Farah Fariri. She was the caring friend that anybody wishes to have moving to a new place. Farah introduced me to almost everybody I knew outside of my lab and classes, in my first two years. Never have the grocery rides been so fun since Farah graduated. Also, I am very indebted to Huda Ibeid. Huda's funny and positive personality tremendously helped me take my Ph.D hard moments less seriously. Last but not least, I would have never survived the long quarantine without Assma Boughoula, Asma'a Albakri and Farzi Rahman Shoptorshi. These three amazing ladies became my family, fitness buddies and studymates throughout the last year.

I am very grateful to many other amazing friends that made classes, projects and my entire UIUC experience more enjoyable: Aya Sellami, Celeste Lü, Faria Kalim, Gowthami Manikandan, Ismini Lourentzou, Jackson Wang, Kajori Banerjee, Laila Al-Bardan, Mariam Saadah, Mona Zehni, Oyuna Angatkina, Pavan Kumar Reddy, Wei Yang and Wei Zuo. It was also very refreshing to meet many bright and fun visiting scholars and students throughout the years. Many thanks to Ahmed Mzid, Dina Hadžiosmanović, Fabiana Lanotte, Lulú Labastida, Rachid Ellouze, Sabrine Mcharek and Sarah Omrane. I am also grateful to the friends that I met during my summer internships and with whom I enjoyed discovering new cities as well as the industry experience. In partricular, I thank Ayako Ohara, Haris Marata, Kaoutar El Maghraoui, Medha Shrivastav and Nazneen Rajani.

This acknowledgement would have been incomplete without thanking the CSL Student conference committee that I have been part of for two years, full of rich experiences and enjoyable moments. Thank you, Ameya Patil, AmirEmad Ghassami, Amirhossein Taghvaei, Amish Goel, Ashok Vardhan, Bihan Wen, Charbel Sakr, Chuanzheng Li, Gizem Tabak, Ivan Abraham, James Schmidt, Jonathon Hoff, Joseph Lubars, Philip Paré, Massi Amrouche, Nicole Chan, Raphael Stern, Ravi Kiran Raman, Shahzad Bhatti, Shripad Gade, Siddhartha Satpathi, Sungmin Lim and Tarek Sakakini.

Finally, no words suffice to express my gratitude toward my mother Fahima, father Younes, my sister Mouna and my brother Achraf for their love, support and encouragement.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The focus of this dissertation is building *structured models* with more scalable inference and learning procedures. *Structure* refers to statistical dependence between either the input entities in a dataset, *e.g.*, correlations between neighboring pixels in images or consecutive words in sentences, or the output entities in a given problem, *e.g.*, co-occurrence patterns of labels in a multi-label classification task. *Structured models* explicitly take these inter-dependencies into account to make globally consistent decisions. This is achieved by operating on structured objects, where the relationships between the variables are represented as sequences, grids or general graphs.

Although structured models have demonstrated success across a wide range of applications, including natural language processing, information extraction, computer vision and computational biology in the previous few decades, inference is only tractable for relatively simple structured objects and learning is only feasible for low-dimensional datasets.

In this dissertation, we leverage recent progress in deep learning research to build more scalable structured models, *i.e.*, we design inference procedures that handle complex structures efficiently, and learning algorithms that scale to high-dimensional datasets.

1.1 Applications of Structured Models

To illustrate the need for more scalable structured models, we present examples of the complex structure in data and tasks across different machine learning application fields.

Natural Language Processing (NLP):
In NLP tasks, structure arises from the syntax and semantics rules of the

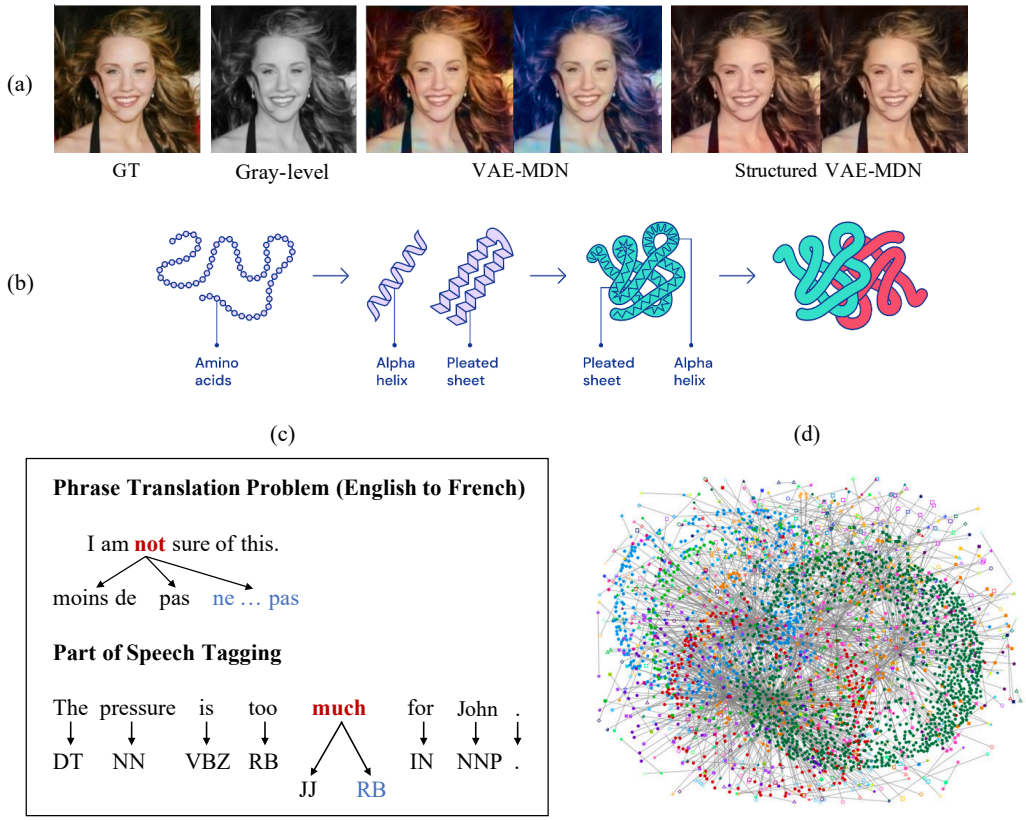


Figure 1.1: (a) Diverse colorization of a gray-level image. More globally consistent results are obtained when using a structured model [1]. (b) Prediction of 3D protein structure from the amino-acid sequence. Every protein is made-up of a sequence of amino-acids bounded together. These amino-acids interact locally to form shapes like helices and sheets. The shapes fold-up on large scale to form three-dimensional protein structures. Proteins can interact with other proteins performing functions such as signalling and transcribing DNA [2]. (c) Examples of two structured prediction tasks in natural language processing: part of speech tagging and phrase translation. The target phrase or POS tag in blue is the correct output. (d) Spammer detection in social networks using patterns of communication. The dots on the graph represent assigned or inferred IP addresses, which have been mined from Microsoft's Hotmail servers [3].

language. A classical application where the model's success depends on its ability to capture the complex linguistic structures is part-of-speech (POS) tagging: given a sentence of words, the model outputs a sequence of POS tags, one for each word in the input sentence. Without a proper modeling of the context, assigning the right tag to words that can have different functions in the sentence (*e.g.*, the word "much" in Figure 1.1(c)) is not obvious. The same requirements are valid for tasks like machine translation (Figure 1.1

(c)), question answering and co-reference resolution. Other examples include vision and language tasks, reasoning about images and text simultaneously, *e.g.*, image captioning, visual question answering, image retrieval using complex text queries or image generation from textual descriptions. Such problems require capturing relationships between visual and linguistic concepts.

Computer Vision

In computer vision, structure arises from the spacial arrangement of pixels and/or objects. A classical example is semantic segmentation, which aims at predicting a semantic label (*e.g.*, cat, dog) for every pixel in the image. Spacial constraints include label consistency between neighboring pixels and co-occurrence patterns of labels (*e.g.*, boat and sea labels are more likely to occur jointly than boat and grass labels). Another example is gray-level image colorization. Enforcing pixels with similar intensities to have similar colors leads to more consistent colorization as illustrated in Figure 1.1 (a). Beyond pixel-level patterns, capturing object similarities across different frames in a video enables identifying and tracking objects across the video in the context of *multi-object tracking*.

Computational Biology

The amino-acid sequence in a protein determines its 3D structure. Protein structure prediction (PSP) is one of the most important unsolved problems in biophysics and computational biology today (Figure 1.1 (b)). To give another example, precision medicine, based on tailoring the medical treatment to the patient's genomics make-up, relies on capturing the correlation between a wide array of variables including genomic mutations, blood tests, lifestyle and nutrition, medical history and demographics.

Recommender Systems and Social Networks

In recommender systems, structure consists in the patterns in past interactions between users and items, *e.g.*, products frequently bought jointly or users with similar purchase history. Recommender systems leverages these relationships to produce new recommendations. For example, an e-commerce platform like Amazon would suggest to buyers articles that are usually jointly purchased with other items that they have in their cart, and a social network one like Facebook would suggest friends or group that have common friends

or subscriptions. Also, leveraging communication patterns social network graphs can reveal spammers (Figure 1.1), or most influential user in a particular network [4]. Companies are interested in this information in order to decide who they may hire as marketing influencer.

Generative Modeling
Capturing structure in the data leads to more accurate object generation and diversity. For example, in image generation, explicitly learning the spatial layout and semantic relations between objects would lead to generating images of complicated scenes.

1.2 Problem Statement

Problems and data in spirit similar to the aforementioned examples require models with high expressivity to laverage the inherent complex structure. Deepnets, although capable of implicitly capturing such complexities, they suffer from sample efficiency. Structure is a compact way of representing large amounts of data. Therefore, it is beneficial to integrate the advances in deep learning with structured prediction frameworks. However, the time complexity of exact inference depends on the complexity (expressivity) of model capturing the dependency structure between the variables. Efficient solutions exist only when this structure is very simple, *e.g.*, a tree with small width. In the general case, inference is NP hard and learning is $\#P$-hard due to the computation of partition function. Therefore, the core research challenge in structured prediction is (1) scaling inference procedures to efficiently handle expressive models and (2) scaling the learning to leverage high-dimensional datasets.

In the first two chapters of the dissertation, we are attempting to push state-of-the-art on the first challenge, *i.e.*, *devising efficient inference algorithms for expressive models with higher-order potentials, i.e., terms modeling complex interactions between more than two variables?* First, we were wondering if we can extend models with efficient inference to have more expressive power without increasing the inference complexity. In particular, we are interested in the class of Gaussian conditional random fields (G-CRF), characterized by exact inference that is reduced to solving a linear system

of equations, but uni-modal and limited to capturing pairwise correlations between the variables. Furthermore, we were wondering, if we can learn inference procedures for general graphical models types and orders using reinforcement learning. Intuitively, instances of similar problems are often solved repeatedly. While humans have uncovered impressive heuristics, data driven techniques are likely to uncover even more compelling mechanisms. *What would be the best way to define the different components of a Markov decision process (MDP) to learn heuristics for solving inference in graphical models?* In particular, *how should the reward function be designed so that maximizing the expected future reward translates into solving the inference problem*?

To address the second challenge, *i.e.*, scaling the learning to leverage high-dimensional datasets, we propose a new loss, *i.e.*, max-sliced score matching (MSSM), that improves upon score matching losses. Specifically, the score matching loss (SM) circumvents the computation of the partition function but introduces a new challenge, *i.e.*, the evaluation of the trace of the Hessian of the log-likelihood. Sliced score matching (SSM) avoids the computation of the trace via projecting the scores into random directions before comparing them. We are interested in improving upon SSM by *finding the most informative projection directions*.

1.3 Contributions

In this dissertation, we present a collection of methods for scalable inference and learning in structured models. We summarize the contributions as follows:

- **Deep Gaussian conditional random field based variational autoencoders:** (Chapter 3) We extend Gaussian conditional random field, traditionally uni-modal and only limited to modeling pairwise interactions, to (1) model multi-modal distributions for the tasks of diverse gray-level image colorization via endowing them with variational autoencoders, (2) capture high-order potentials and (3) incorporate external constraints at runtime, *e.g.*, user edits in form of color strokes, while (4) keeping the inference exact. We demonstrate that our method

obtains more diverse and globally consistent colorizations on the LFW, LSUN-Church and ILSVRC-2015 datasets.

- **Learning heuristics to solve inference in general graphical models using reinforcement learning** (Chapter 4). We show that we can learn program heuristics for solving inference in higher-order conditional random fields for the task of semantic segmentation, using reinforcement learning. We show compelling results on the Pascal VOC and MOTS datasets, while scaling linearly with potential orders and number of variables.

- **Max-sliced score matching for learning structured models at scale** (Chapter 5). We propose a new loss for learning scalable structured models, *i.e.*, max-sliced score matching (MSSM), which improves upon sliced score matching (SSM). We prove that MSSM is consistent, asymptotically normal and has lower variance than SSM. We assess these models on estimation of densities and scores for implicit distributions in Variational and Wasserstein auto-encoders.

1.4 Relationship to Published Work

Following is a list of publications related to this proposal:

1. Chapter 3: **Safa Messaoud**, David Forsyth, Alexander Schwing. Structural Consistency and Controllability for Diverse Colorization. **ECCV 2018.**

2. Chapter 3: **Safa Messaoud**, Alexander Schwing. G-CRF VAE: A Structured Output Space Variational Auto-Encoder with Deep Gaussian Conditional Random Fields for Diverse and Globally Consistent Colorization. **WiML Workshop at NeurIPS 2017.**

3. Chapter 4: **Safa Messaoud**, Maghav Kumar, Alexander Schwing. Can We Learn Heuristics For Graphical Model Inference Using Reinforcement Learning? **CVPR 2020 (Oral), WiCV at CVPR 2020 (Oral), Deep Vision Workshop at CVPR 2020.**

4. Chapter 5: **Safa Messaoud**, Kuan-Chieh Wang, Alexander Schwing. Max-Sliced Score Matching. **Submitted to ICML 2021**

During this Ph.D., we also worked on few other directions that are not directly related to the dissertation topic, and therefore not presented in the dissertation, including: multi-video summarization, electronic medical records analysis [5, 6], genomics data analysis for demystifying the cause of Alzheimer disease [7] and designing customized processors for accelerating genomics pipelines [8, 9].

1.5 Outline of the Dissertation

After reviewing related work in Chapter 2, we subsequently present the aforementioned contributions in Chapters 3-5. In Chapter 3, we scale Gaussian conditional random field to model multi-modal distributions with higher-order potentials, while ensuring exact inference and enabling the incorporation of external constraints at runtime. We apply the proposed model to the task of diverse gray-level image colorization. In Chapter 4, we learn policies to solve inference in conditional random fields with arbitrary potential types and orders using reinforcement learning. We set the reward function to be the negative of the energy function. We explore different formulations for the Markov decision process parameters. Specifically, we study different reward functions and learning algorithms. On the task of semantic segmentation, we show that we scale to setups with thousands of variables and a label space of size 21 (Pascal VOC dataset). Beyond scalable inference, we propose a new loss, max-sliced score matching (MSSM) for scalable learning of structured models in high-dimensional spaces, in Chapter 5. We show that our loss outperforms other score matching losses on estimation of densities and scores for implicit distributions in Variational and Wasserstein auto-encoders.

CHAPTER 2

BACKGROUND AND RELATED WORK

Having highlighted the need for more scalable structured models, in the following, we formally define structured models (Section 2.1), then review the literature on inference (Section 2.2), learning (Section 2.3), structured deep learning frameworks (Section 2.4) as well as on learning combinatorial optimization (Section 2.5).

2.1 Structured Models

Structured models can either be discriminative, *i.e.*, mapping structured inputs to a structured output, or generative, *i.e.*, capturing the structure in the input space distribution. In the following, we will describe the notation for the discriminative case. The generative case follows the same formulation. We denote by $\boldsymbol{\mathcal{X}}$ the space of structured inputs and by $\boldsymbol{\mathcal{Y}}$ the space of structured outputs. We assume that each structured output $\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}$ is represented by N discrete or continuous variables, *i.e.*, $\boldsymbol{y} = (y_1, \cdots, y_N)$, and each variable y_i can take candidate values from a set or a range $\boldsymbol{\mathcal{L}}(y_i)$. Also, we assume the availability of a joint feature function $\boldsymbol{\phi} : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{Y}} \rightarrow \mathbb{R}^d$, that computes d-dimensional feature representations for any pair $(\boldsymbol{x}, \boldsymbol{y})$. Thinking of Part of Speech Tagging (POS), a structured input $\boldsymbol{x}$ is a sequence of words, the structured output $\boldsymbol{y}$ is a sequence of POS tags each corresponding to a word in the input sequence, $\boldsymbol{\mathcal{L}}(y_i)$ is the list of all candidate POS tags for a word x_i, and $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ can represent unary features defined on single variables, *e.g.*, word and tag embeddings, and structural features defined on a set of variables, *e.g.*, tags co-occurence statistics.

Frequently, an *energy function* $E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ is used to score a candidate structured output $\boldsymbol{y}$, given a structured input $\boldsymbol{x}$. Here, $\boldsymbol{\theta} \in \mathbb{R}^d$

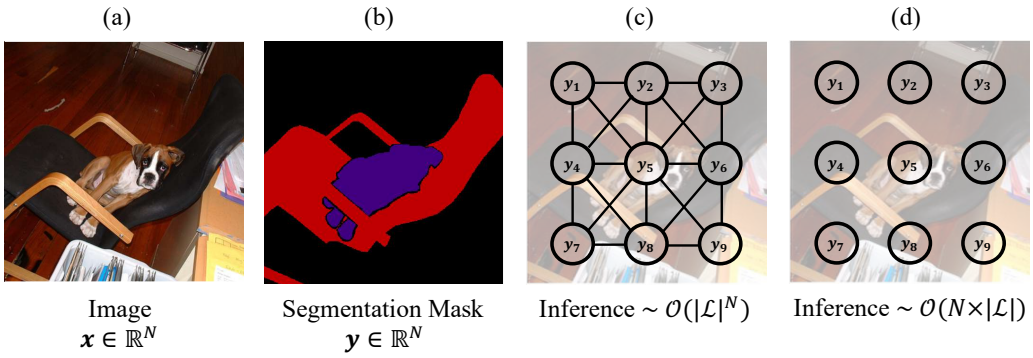


Figure 2.1: Semantic segmentation structured models with increasing expressivity and inference complexity. (a) Original image. (b) Segmentation mask. (c) Energy function with independent output variable (unary potentials). (c) Energy with inter-dependant output variables (higher-order potentials).

is a parameter. The *order* of the energy function corresponds to the maximum number of interdependent variables described by the features $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$. Usually, the energy is factorized into potentials $f_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_{\boldsymbol{c}}) = \boldsymbol{\theta}_c^T \boldsymbol{\phi}_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y})$ based on the order. Here, $\boldsymbol{c} \in \boldsymbol{\mathcal{C}}$ denotes the set of indices of inter-dependant variables, *i.e.*, a clique. $\boldsymbol{\mathcal{C}}$ is the set of all cliques. For example, the following energy is composed of unary (order 1) and pairwise (order 2) potentials:

$$E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \underbrace{\sum_{i \in \boldsymbol{\mathcal{V}}} \underbrace{\boldsymbol{\theta}_u^T \boldsymbol{\phi}_i(y_i, \boldsymbol{x})}_{f_i(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_u)}}_{\text{Unary potentials}} + \underbrace{\sum_{(i,j) \in \boldsymbol{\mathcal{E}}} \underbrace{\boldsymbol{\theta}_p^T \boldsymbol{\phi}_{i,j}(y_i, y_j, \boldsymbol{x})}_{f_{i,j}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_p)}}_{\text{Pairwise potentials}}, \tag{2.1}$$

where $\boldsymbol{\mathcal{V}} \subseteq \boldsymbol{\mathcal{C}}$ is the set of indices i of all variables y_i and $\boldsymbol{\mathcal{E}} \subseteq \boldsymbol{\mathcal{C}}$ is the set of indices (i, j) off all pairs of correlated variables y_i and y_j. A prediction is obtained by solving the *inference* problem:

$$\boldsymbol{y}^* = \underset{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}}{\operatorname{argmin}} E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}). \tag{2.2}$$

Learning consists of estimating the parameters $\boldsymbol{\theta}$ and possibly also the structure $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ from the data. In the following, we will drop the dependency on $\boldsymbol{\theta}$ for simplicity of the notation.

The key *challenge* in building structured models is the time complexity of the inference procedure. Exact inference has exponential complexity in size of the output. Efficient methods only exist for special structures (Sec-

tion 2.2). Generally, the higher the order, the more expressive is the model and the harder is the inference. To give a concrete example, consider the semantic segmentation problem illustrated in Figure 2.1. A common energy function for the task decomposes into unary and pairwise potentials as in Equation 2.1. The structure is defined by an undirected graph $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}})$ (Figure 2.1 (c)), where each node $i \in \boldsymbol{\mathcal{V}}$ in the graph corresponds to an output variable y_i, which in turn refers to the label of pixel x_i. Edges $(i, j) \in \boldsymbol{\mathcal{E}}$ connect neighboring nodes to enforce local smoothness in the labeling, *i.e.*, labels should vary *smoothly* in the spatial domain as defined by the connectivity of the graph. The complexity of exact inference is $\mathcal{O}(|\boldsymbol{\mathcal{L}}|^N)$ as it envolves a greedy search over the entire space of output configurations $\boldsymbol{\mathcal{Y}}$. However, for the structure in Figure 2.1 (d), with only unary potentials, *i.e.*, independent output variable, the complexity drops to $\mathcal{O}(N|\boldsymbol{\mathcal{L}}|)$, as inference can be run in parallel, for every pixel separately. In general, this difficulty in inference also causes challenges in learning, as learning a structured model usually has the inference engine as a subroutine.

The introduced energy-based model can be interpreted *probabilistically*, as characterizing a Gibbs distribution:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp\left(-E(\boldsymbol{x}, \boldsymbol{y})\right)}{\sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} \exp\left(-E(\boldsymbol{x}, \boldsymbol{y})\right)}. \tag{2.3}$$

Note that computing the most likely configuration $\boldsymbol{y}$ under $p(\boldsymbol{y}|\boldsymbol{x})$, *i.e.*,

$$\boldsymbol{y}^* = \underset{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x}), \tag{2.4}$$

is equivalent to finding the configuration with the minimum energy $E(\boldsymbol{x}, \boldsymbol{y})$ as the denominator is independent of $\boldsymbol{y}$. A different interpretation comes from the area of probabilistic graphical models [10, 11], where relationships between the variables in the distributions are specified through graphs. Undirected models are called Markov random fields (MRFs). When used to model conditional distributions they are called conditional random fields (CRFs). As illustrated in Figure 2.1, nodes represent the variables and edges the sta-

tistical dependencies. A CRF induced distribution factorizes as follows:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{\prod_{\boldsymbol{c}\in\boldsymbol{\mathcal{C}}} \psi_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{c}})}{\sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} \prod_{\boldsymbol{c}\in\boldsymbol{\mathcal{C}}} \psi_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{c}})}. \tag{2.5}$$

Here, $\psi_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{c}})$ are potential functions modeling statistical dependence between the nodes, *i.e.*, variables $\boldsymbol{y}_{\boldsymbol{c}} = (y_i)_{i\in\boldsymbol{c}}$ within a clique $\boldsymbol{c}$. The CRF formulation can be easily converted into the Gibbs distribution from Equation 2.3 by setting $\psi_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{c}}) = \exp(f_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{c}}))$.

2.2 Inference in Structured Models

Maximum-a-posteriori (MAP) inference is widely used in structured prediction tasks for finding the optimal configuration $\boldsymbol{y}^*$ with the highest posterior probability $p(\boldsymbol{y}|\boldsymbol{x})$, *i.e.*, minimum energy $E(\boldsymbol{x}, \boldsymbol{y})$, via solving the program:

$$\boldsymbol{y}^* = \operatorname*{argmin}_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} E(\boldsymbol{x}, \boldsymbol{y}). \tag{2.6}$$

In the following, we review inference techniques for both discrete and continuous output variables $\boldsymbol{y}$.

2.2.1 Structured Models with Discrete Variables

In the discrete case, the program in Equation 2.6 is a combinatorial optimization problem of NP hard complexity, as it requires the enumeration and evaluation of an exponential number of configurations. However, in special cases with simple structure, inference can be run efficiently. In the following, we go through the most popular models with tractable inference.

Tree Structure

The output variables y_i are connected to each others in a tree structure. Pairwise potentials $f_{i,j}$ are defined over edges of the tree and unary potentials

f_i are defined over the nodes, resulting in the energy function:

$$E(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{N} f_i(\boldsymbol{x}, y_i) + \sum_{j \in \text{Child}(i)} f_{i,j}(\boldsymbol{x}, y_i, y_j). \tag{2.7}$$

The optimum solution $\boldsymbol{y}^*$ can be computed exactly and efficiently using dynamic programming [12] via breaking the max-operator recursively across the sub-trees. Starting from the leaves upwards until reaching the root y_1, at every sub-tree root, we compute and store the maximum over all potentials in the sub-tree. At the tree root y_1, computing the maximum over the sub-trees values results in determining $\max_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} E(\boldsymbol{x}, \boldsymbol{y})$. The optimum solution $\boldsymbol{y}^*$ can then be retrieved through back-tracing from root to leaves. Formally, the recursion formula is:

$$m(y_t) = f_t(y_t) + \sum_{k \in \text{Child}(t)} \max_{y_k} f_{t,k}(\boldsymbol{x}, y_t, y_k) + m(y_k), \quad \forall t \in [1, N]. \tag{2.8}$$

Note that, $\max\limits_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} E(\boldsymbol{x}, \boldsymbol{y}) = \max\limits_{y_1} m(y_1)$. The described algorithm is of polynomial complexity, *i.e.*, $\mathcal{O}(N|\boldsymbol{\mathcal{L}}|)$. The auxiliary variables are called *messages* as dynamic programming falls under the umbrella of *message passing* algorithms [13]. Applications with tree-structured energy function include pose estimation, sequence alignment in bio-informatics and temporal pattern recognition such as speech or handwriting recognition.

Sub-Modular Potentials

Energies with pairwise sub-modular potentials are of the form:

$$\begin{aligned} E(\boldsymbol{x}, \boldsymbol{y}) &= \sum_{i=1}^{N} f_i(\boldsymbol{x}, y_i) + \sum_{(i,j) \in \boldsymbol{\mathcal{E}}} f_{i,j}(\boldsymbol{x}, y_i, y_j) \\ \text{s.t.} \quad & f_{i,j}(\boldsymbol{x}, y_i, y_j) \leq f_i(\boldsymbol{x}, y_i) + f_j(\boldsymbol{x}, y_i), \quad \forall (i,j) \in \boldsymbol{\mathcal{E}}. \end{aligned} \tag{2.9}$$

Under the sub-modularity constraint, a graph can be constructed out of the potentials. In the binary case, the graph construction is given by [14]. In the multi-label case, the construction is introduced in [15]. Minimizing such energies can be solved exactly and efficiently via max-flow/min-cut algorithms [16] with a runtime complexity $\mathcal{O}(|\boldsymbol{\mathcal{E}}|N \log(N))$. Approximate solutions exist for pairwise potentials satisfying the semi-metric property (swap-move algo-

rithms, *e.g.*, α-β swap) or the metric property (expansion algorithms, *e.g.*, α expansion) [16]. Applications leveraging sub-modular energy functions include for-ground extraction [17] and video cutout [18].

Sparse Patterns Potentials

Sparse patterns potentials are higher-order potentials with tractable inference. For a given potential $f_{\boldsymbol{c}}(\boldsymbol{y_c})$, only a small number of assignments is specified, the remaining assignments have exactly the same score, frequently 0, *i.e.*,

$$f_{\boldsymbol{c}}(\boldsymbol{y_c}) = \begin{cases} \text{Score}(\boldsymbol{y_c}) & \text{if } \boldsymbol{y_c} = \boldsymbol{l_c}, \quad \forall \boldsymbol{l_c} \in \bar{\boldsymbol{\mathcal{L}}}(\boldsymbol{y_c}) \\ 0 & \text{Otherwise} \end{cases}, \tag{2.10}$$

with $\bar{\boldsymbol{\mathcal{L}}}(\boldsymbol{y_c}) \subset \boldsymbol{\mathcal{L}}(\boldsymbol{y_c})$ and $|\bar{\boldsymbol{\mathcal{L}}}(\boldsymbol{y_c})| \ll |\boldsymbol{\mathcal{L}}(\boldsymbol{y_c})|$. This family of potentials is surprisingly useful in practice, as we are in general only interested in exploring configurations that occur in real data. For example, in a handwriting recognition task, we are only interested in letter combinations that occur in a dictionary. That is a small number, relative to 26^N combinations for an English word with N letters.

Cardinality Potentials

Cardinality potentials on a set of binary variables $\boldsymbol{y} \in \{0,1\}^N$ are higher-order functions of the number of turned-on variables, *i.e.*, $f_{\boldsymbol{c}}(\sum_{i \in \boldsymbol{c}} y_i)$. These potentials are sparse as well, as there are only N possible values that $f_{\boldsymbol{c}}$ can take in total. It turns out, that these potentials can be useful in quite a number of different application [19, 20, 21]. For example, in message decoding, parody constraints check the number of y_i's that are on. In image segmentation, it might be useful to have a prior on the number of pixels in a given category, *e.g.*, the pixels labeled sky should be between 30% and 70%.

Decomposition Methods

Decomposition methods are usually used when the energy function is hard to optimize, but can be decomposed into potentials with special structure (*e.g.*, tree, sub-modular, sparse) that are easy to optimize separately. Assume that the energy function decomposes into two cliques $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ that are easy to

optimize, *i.e.*,

$$\max_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} E(\boldsymbol{x},\boldsymbol{y}) = \max_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} f(\boldsymbol{x},\boldsymbol{y}_{\boldsymbol{c}_1}) + f(\boldsymbol{x},\boldsymbol{y}_{\boldsymbol{c}_2}). \tag{2.11}$$

If the potentials are coupled, *i.e.*, there exist some variables $\boldsymbol{y}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2}$ that appear in both cliques $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, optimizing $f(\boldsymbol{x},\boldsymbol{y}_{\boldsymbol{c}_1})$ and $f(\boldsymbol{x},\boldsymbol{y}_{\boldsymbol{c}_2})$ separately is only meaningful if an additional constraint ensuring the consistency between the versions of $\boldsymbol{y}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2}$ ($\boldsymbol{y}^{\boldsymbol{c}_1}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2}$ and $\boldsymbol{y}^{\boldsymbol{c}_2}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2}$) appearing in $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ is added to the program above, *i.e.*,

$$\begin{aligned} \max_{\boldsymbol{y}_{\boldsymbol{c}_1}} \quad & f(\boldsymbol{y}_{\boldsymbol{c}_1},\boldsymbol{x}) + \max_{\boldsymbol{y}_{\boldsymbol{c}_2}} f(\boldsymbol{y}_{\boldsymbol{c}_2},\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{y}^{\boldsymbol{c}_1}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2} = \boldsymbol{y}^{\boldsymbol{c}_2}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2} \end{aligned} . \tag{2.12}$$

This program is then solved via dual optimization. If the optimal solution to two sub-problems agree on the value of $\boldsymbol{y}_{\boldsymbol{c}_1\cap\boldsymbol{c}_2}$, the optimal $\boldsymbol{y}^*$ for the dual equals the one for the primal. Otherwise, a wide range of techniques have been developed to recover an approximate solution for $\boldsymbol{y}^*$, *e.g.*, performing a majority vote on each of the coupling variables or just evaluating all the solutions from the sub-problems on the primal objective and picking the best one. An example application where inference with dual decomposition has shown success include 3D cell reconstruction [22, 23].
Next, we discuss approximate MAP inference techniques applied to general graphs.

Integer Linear Programming (ILP)
By introducing indicator variables $b_{\boldsymbol{c}}(\boldsymbol{l}_{\boldsymbol{c}})$ for every clique $\boldsymbol{c}$ and label configuration $\boldsymbol{l}_{\boldsymbol{c}} \in \boldsymbol{\mathcal{L}}(\boldsymbol{y}_{\boldsymbol{c}})$, the MAP objective $\min_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} \sum_{\boldsymbol{c}\in\boldsymbol{\mathcal{C}}} f_{\boldsymbol{c}}(\boldsymbol{x},\boldsymbol{y}_{\boldsymbol{c}})$ can be reduced to ILP, *i.e.*,

$$\begin{aligned} \min_{b_{\boldsymbol{c}}} \quad & \sum_{\boldsymbol{c}\in\boldsymbol{\mathcal{C}}} \sum_{\boldsymbol{l}_{\boldsymbol{c}}\in\boldsymbol{\mathcal{L}}(\boldsymbol{y}_{\boldsymbol{c}})} b_{\boldsymbol{c}}(\boldsymbol{l}_{\boldsymbol{c}}) f_{\boldsymbol{c}}(\boldsymbol{l}_{\boldsymbol{c}}) \\ \text{s.t.} \quad & b_c(\boldsymbol{l}_{\boldsymbol{c}}) \in \{0,1\}, \quad \forall \boldsymbol{c}\in\boldsymbol{\mathcal{C}}, \quad \forall l_{\boldsymbol{c}} \in \boldsymbol{\mathcal{L}}(\boldsymbol{y}_{\boldsymbol{c}}) \\ & \sum_{\boldsymbol{l}_{\boldsymbol{c}}\in\boldsymbol{\mathcal{L}}(\boldsymbol{y}_{\boldsymbol{c}})} b_{\boldsymbol{c}}(\boldsymbol{l}_{\boldsymbol{c}}) = 1, \quad \forall \boldsymbol{c}\in\boldsymbol{\mathcal{C}} \\ & \sum_{l_j, j\in\boldsymbol{c}\setminus i} b_c(\boldsymbol{l}_{\boldsymbol{c}}) = b_{\boldsymbol{c}}(l_i), \quad \forall \boldsymbol{c}\in\boldsymbol{\mathcal{C}} \end{aligned} . \tag{2.13}$$

This ILP program is NP-hard. However, there has been recently work on improving ILP search mechanism by learning heuristics to efficiently search in the solution space [24, 25, 26].

Linear Programming (LP)

One approximate technique for solving the ILP program from above is rounding, *i.e.*, relaxing the binary constraints such that $b_{\boldsymbol{c}}(\boldsymbol{l_c}) \in [0, 1]$ [27, 28, 29]. This results in a relaxed program of polynomial complexity. Convergence and tightness of LP relaxations for special classes of problems have been studied extensively in recent years and bounds have been derived for some structures [30, 31, 32, 33, 34, 35]. Convergence to the global optimum is only guaranteed when the resulting program is augmented to be convex, *e.g.*, by smoothing or augmented Lagrangian methods [36, 37, 38, 39, 40, 41, 42]. Also, sub-gradient [43] or bundle [42] methods directly consider the original function and converge to the global optimum by constructing polyhedral approximations based on sub-gradients.

Amortized Inference

Amortized inference [44, 45] is a technique that leverages the structure across data instances. Instead of running inference solver independently on each input example, a set of solutions of previously solved problems are stored and reused for new inference problems, without calling the inference solver (*e.g.*, ILP) when certain theoretical conditions are met. Amortized inference has shown to yield significant savings in diverse NLP applications like POS tag sequences, parse trees and semantic parses, due to the fact that many sentences in the data corpus have identical structured outputs.

Probabilistic MAP Inference

Probabilistic MAP Inference is another approach to MAP inference that operates on probabilities to derive the optimal configuration, *i.e.*,

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x}) = \underset{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}}{\operatorname{argmax}} \frac{\exp(-E(\boldsymbol{x},\boldsymbol{y}))}{\sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}}\exp(-E(\boldsymbol{x},\boldsymbol{y}))} = \underset{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}}{\operatorname{argmax}} \exp(-E(\boldsymbol{x},\boldsymbol{y})). \tag{2.14}$$

This adds a notion of uncertainty to the MAP inference procedure. *Max-product algorithm* solves the program above by computing the max-marginals

of each node in the graph, *i.e.*,

$$p(y_i|\boldsymbol{x})^* = \max_{y_j, j\neq i} p(\boldsymbol{y}|\boldsymbol{x}), \tag{2.15}$$

then deriving $\boldsymbol{y}^*$ via backtracing. Assuming a pairwise energy function of the form:

$$E(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i\in\boldsymbol{\mathcal{V}}} f_i(\boldsymbol{x}, \boldsymbol{y}) + \sum_{(i,j)\in\boldsymbol{\mathcal{E}}} f_{i,j}(\boldsymbol{x}, y_i, y_j), \tag{2.16}$$

and using a serial schedule, we compute the messages $m_{i\to j}$ between nodes i and j for $(i,j) \in \boldsymbol{\mathcal{E}}$ as follows:

$$m_{i\to j}(\boldsymbol{x}, y_j) = \max_{y_i\in\boldsymbol{\mathcal{Y}}} e^{f_i(\boldsymbol{x},y_i)} e^{f_{i,j}(\boldsymbol{x},y_i,y_j)} \prod_{k\in\boldsymbol{\mathcal{N}}(i)\setminus j} m_{k\to i}(\boldsymbol{x}, y_i). \tag{2.17}$$

Once all messages have been computed, we derive the max-marginals at every node:

$$p(y_i|\boldsymbol{x})^* = e^{f_i(\boldsymbol{x},y_i)} \prod_{k\in\boldsymbol{\mathcal{N}}(i)} m_{k\to i}(\boldsymbol{x}, y_i). \tag{2.18}$$

For recovering the MAP assignment, we also store the argmax for every message, *i.e.*, $\delta_{i\to j}(\boldsymbol{x}, y_j) = \text{argmax}_{y_i}\, m_{i\to j}(\boldsymbol{x}, y_j)$. We start backtracing by computing $y_i^* = \text{argmax}_{y_i}\, p(y_i|\boldsymbol{x})$ for an arbitrary node i. Then, for every node $j \in \boldsymbol{\mathcal{N}}(i)$, we compute $y_j^* = \delta_{i\to j}(\boldsymbol{x}, y_i^*)$. Intuitively, this simulates computing $y_j^* = \text{argmax}_{y_j}\, p(y_j|\boldsymbol{x}, y_i^*)$.

Variational inference is another methods that converts the probabilistic inference problem from Equation 2.14 into an optimization one. The main idea is to approximate $p(\boldsymbol{y}|\boldsymbol{x})$ with another distribution $q(\boldsymbol{y}|\boldsymbol{x})$ that has a simpler structure. KL-divergence, $\text{KLD}(q(\boldsymbol{y}|\boldsymbol{x})||p(\boldsymbol{y}|\boldsymbol{x})) = \sum_{\boldsymbol{y}} q(\boldsymbol{y}|\boldsymbol{x}) \log \frac{q(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{y}|\boldsymbol{x})}$, is minimized to optimize the parameters of $q(\boldsymbol{y}|\boldsymbol{x})$. Mean-field inference is a special case, where we assume that the variational distribution $q(\boldsymbol{y}|\boldsymbol{x})$ factorizes over the variable y_i, *i.e.*, $q(\boldsymbol{y}|\boldsymbol{x}) = \prod_i q_i(y_i|\boldsymbol{x})$. Considering a distribution $p(\boldsymbol{y}|\boldsymbol{x})$ characterized by an energy $E(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{c}} f_{\boldsymbol{c}}(\boldsymbol{x}, \boldsymbol{y})$, we set the derivative of the KL-divergence $\text{KLD}(q(\boldsymbol{y}|\boldsymbol{x})||p(\boldsymbol{y}|\boldsymbol{x}))$ to 0, and obtain the marginals

$$q_i(y_i|\boldsymbol{x}) \propto \exp \sum_{\boldsymbol{c}\in\boldsymbol{\mathcal{C}}_i} \sum_{\boldsymbol{y}_{\boldsymbol{c}}} f_{\boldsymbol{c}}(\boldsymbol{y}_{\boldsymbol{c}}) \prod_{j\in\boldsymbol{c}, j\neq i} q_j(y_j). \tag{2.19}$$

Here, $\boldsymbol{\mathcal{C}}_i$ is the set of cliques that include node y_i. MAP inference under this setup is reduced to computing the best assignment for every y_i variable separately as $y_i^* = \text{argmax}_{y_i} q_i(y_i|\boldsymbol{x})$.

To **summarize**, exact MAP inference in the discrete case is an NP hard combinatorial optimization problem that is only tractable for special simple structures. Approximate methods are efficient but only come with guaranteed in special cases. In Chapter 4, we propose learning inference engines for general energy functions with discrete variables and higher-order potentials using reinforcement learning. We show that we scale linearly with the potential order and number of nodes.

2.2.2 Structured Models with Continuous Variables

Energy-based models with continuous variables (*e.g.*, temperature, location or distance) arise in many applications like stereo-estimation [46], body and/or hand tracking [47, 48]. However, inference in structured models with continuous variables is trickier than in the discrete case. In fact, using discrete optimization techniques for the continuous case is not obvious, *e.g.*, defining factors over continuous variables and replacing summation with integration, in the sum-product algorithm is not guaranteed to lead to a correct answer. This is mainly due to two challenges: First, unlike in the discrete case, there is no universal representation of a factor over continuous variables. Picking parametric factors for each factor does not solve the problem as multiplying or marginalizing factors would change the distribution. Second, integration introduces new subtleties as the integral might be infinite or ill defined or requires the use of numerical integration methods, which might lead to approximation errors [11].

Discretization

The most straightforward way to cope with continuous variables is to discritize them and use techniques described in Section 2.2.1. As the variables are interdependent, it has been shown that joint discretization works better than discretizing every variable independently [49, 50, 51]. Some approaches define an energy function that scores both the discritization and the struc-

ture [49, 51]. However, this results in additional complexity to the model. Also, choosing the criterion to optimize to determine the best discretization is not obvious. Besides, discritization results in a loss of information, *e.g.*, dependent continuous random variables may become independent when discretized. To mitigate this issue, Margaritis *et al.* [52] propose to compute independence tests at several different discretization resolutions.

Parametric Models

An important subset of continuous MRFs that has been well studied is Gaussian MRFs [53]. Inference in this setup corresponds to determining the variance and mean of the distribution. Exact inference is possible in case of Gaussian networks through a simple modification of the sum-product algorithm. Under some constraints, loopy belief propagation is guaranteed to return the correct means for the variables [11]. An extension to the purely Gaussian case is non-Gaussian continuous densities: intermediate factors in the computation are approximated as Gaussians (approximate marginalization) [11]. MAP assignment can be determined either via sampling or via coordinate/gradient descent. Also, mean-field inference for Gaussian conditional random field has been proposed in [54]. Gaussian Markov random fields have been used in the past for different computer vision applications including semantic segmentation [55, 56], human part segmentation and saliency estimation [56], image labeling [57] and image denoising [58, 59]. A sparse Gaussian conditional random field trained with a LEARCH framework has been proposed for colorization in [60].

Other classes of parametric densities include polynomial densities, where inference has been solved via augmented Lagrangian, dual decomposition or semi-definite programming, for applications like non-rigid image registration, deformable surface 3D reconstruction, shape from shading, optical flow estimation and image denoising [61, 62].

Particle-Based Methods

Particle-based method is an approximation approach that makes no parametric assumptions about the distribution. Instead, it approximates the distribution as a set of instantiations to all or some variables, referred to as particles. For instance, to estimate $p(y = c)$, the probability of a random variable y being equal to some value $c \in \mathbb{R}$, we generate particles, *i.e.*, samples,

$\{y^{(i)}\}_{i=1}^{M}$ from $p(y)$ and compute $p(y=c) = \mathbb{E}[f(y)] = \frac{1}{M}\sum_{m=1}^{M} f(y^{(i)})$. For sampling, usually MCMC methods or importance sampling are used. Various message passing algorithms have been suggested [63, 64] based on discrete approximations in the form of particles. Extensions include non-parametric belief propagation [65], max-product versions [66] and Stein variational message passing [67]. The primary disadvantage of this approach is the fact that a very large number of particles might be required for a good approximation.

Fusion Moves
Fusion moves [68, 69] are inference methods for pairwise random fields that generalize move making algorithms like α-expansion or α-β swap. The main idea is to fuse various sub-optimal solutions, *i.e.*, proposals, into a better one. Hence, the process relies on the generation of good proposals, a task that is often difficult in practice.

In **summary**, inference with continuous variables is mostly approximate and only exact in special cases of simple parametric distributions. This limits the models expressivity. In Chapter 3, we extend Gaussian conditional random field to model higher-order inter-dependencies as well as multi-modality for the task of gray-image colorization, while enabling exact inference as well as the integration of external constraints in form of user color strokes at run time.

2.3 Learning Structured Models

Learning consists in estimating the parameters $\boldsymbol{\theta}$ of the energy $E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$ function given the training data $\boldsymbol{\mathcal{D}} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{|\boldsymbol{\mathcal{D}}|}$. Different objectives have been studied.

2.3.1 Structured SVM Loss (SSVM)

SSMV or the max-margin loss [70, 71] is used in discriminative models. It aims at finding the parameters $\boldsymbol{\theta}$ such that the difference between the energy $E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta})$ of the desired ground-truth solution $\boldsymbol{y}^{(i)}$, $\forall i \in [1, \cdots, |\boldsymbol{\mathcal{D}}|]$ and the energy $E(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta})$ of any other solution $\boldsymbol{y} \in \boldsymbol{\mathcal{Y}} \backslash \boldsymbol{y}^{(i)}$ is at most $\triangle(\boldsymbol{y}, \boldsymbol{y}^{(i)})$

via solving

$$E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta}) \leq E(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta}) + \triangle(\boldsymbol{y}^{(i)}, \boldsymbol{y}). \tag{2.20}$$

Here, $\triangle(\boldsymbol{y}^{(i)}, \boldsymbol{y})$ is the task loss measuring the discrepancy between the prediction $\boldsymbol{y}$ and the ground-truth $\boldsymbol{y}^{(i)}$. The Hamming distance is a frequent choice for the task loss, *i.e.*, $\triangle(\boldsymbol{y}^{(i)}, \boldsymbol{y}) = \mathbb{1}_{y \neq y^{(i)}}$ [72, 73] as it can be decomposed into a sum of unary terms and integrated easily in the MRF energy without increasing the order of the model. The overall SSVM program is:

$$\min_{\boldsymbol{\theta}} \frac{1}{|\boldsymbol{\mathcal{D}}|} \sum_{i=1}^{|\boldsymbol{\mathcal{D}}|} \Big[\max_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} [E(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta}) + \triangle(\boldsymbol{y}, \boldsymbol{y}^{(i)})] - E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta}) \Big]. \tag{2.21}$$

Gradient descent can be used to solve the objective above. At every training iteration, inference is performed, *i.e.*,

$$(\boldsymbol{y}^{*})^{(i)} = \operatorname*{argmax}_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} (E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta}) + \triangle(\boldsymbol{y}, \boldsymbol{y}^{(i)})), \tag{2.22}$$

then, the weights are updated by computing the gradients:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{|\boldsymbol{\mathcal{D}}|} \sum_{i=1}^{|\boldsymbol{\mathcal{D}}|} \frac{\partial E(\boldsymbol{x}^{(i)}, (\boldsymbol{y}^{*})^{(i)}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \frac{\partial E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{2.23}$$

Here, $\alpha \in \mathbb{R}$ is the learning rate.

2.3.2 Log-Likelihood Loss (LL)

Another option to learn the parameters $\boldsymbol{\theta}$ of the energy $E(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$ is to optimize the log-likelihood objective

$$\min_{\boldsymbol{\theta}} -\frac{1}{|\boldsymbol{\mathcal{D}}|} \sum_{i=1}^{|\boldsymbol{D}|} \log p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{-1}{|\boldsymbol{\mathcal{D}}|} \sum_{i=1}^{|\boldsymbol{\mathcal{D}}|} \log \frac{\exp -E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta})}{\sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} \exp -E(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta})}, \tag{2.24}$$

via gradient descent, resulting in the update rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \frac{1}{|\boldsymbol{\mathcal{D}}|} \sum_{i=1}^{|\boldsymbol{\mathcal{D}}|} \Big[\frac{-\partial E(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \sum_{\boldsymbol{y} \in \mathcal{Y}} \frac{\partial E(\boldsymbol{x}^{(i)}, \boldsymbol{y}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big]. \tag{2.25}$$

Computing the partition function $\sum_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} \frac{\partial E(\boldsymbol{x}^{(i)},\boldsymbol{y};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}$ is intractable as it involves the summation over a very large number of configurations. In the following, we cover methods that have been developed to overcome this issue.

Pseudo-Likelihood

Pseudo-likelihood estimators [74] factorize the joint distribution into product of conditional distributions, *i.e.*, $p(\boldsymbol{x};\boldsymbol{\theta}) = \prod_i p(x_i|x_{\boldsymbol{\mathcal{N}}(i)};\boldsymbol{\theta})$. The log-likelihood objective becomes:

$$\min_{\boldsymbol{\theta}} \frac{-1}{|\boldsymbol{\mathcal{D}}|}\sum_{d=1}^{|\boldsymbol{\mathcal{D}}|}\sum_{i=1}^{N}\log p(x_i^{(d)}|\boldsymbol{x}_{\boldsymbol{\mathcal{N}}(i)}^{(d)}) = \min_{\boldsymbol{\theta}} \frac{-1}{|\boldsymbol{\mathcal{D}}|}\sum_{d=1}^{|\boldsymbol{\mathcal{D}}|}\sum_{i=1}^{N}\log \frac{e^{-\sum_{j\in\boldsymbol{\mathcal{N}}(i)} f_{i,j}(x_i^{(d)},x_j^{(d)})}}{\sum_{x_i'} e^{-\sum_{j\in\boldsymbol{\mathcal{N}}(i)} f_{i,j}(x_i',x_j)}}, \tag{2.26}$$

with $\boldsymbol{\mathcal{N}}(i)$ denoting the neighbor of node i in the MRF graph. Although the partition functions of the single factors are less computationally expensive than the original one, the complexity still scales with the potentials order and the size of the label space.

Sampling Methods

Sampling techniques rely on MCMC to estimate the likelihood and its gradients. A prominant example is *Contrastive divergence* [75, 76, 77], which approximates the gradient of the log-likelihood by a stochastic estimator that uses samples generated from few Markov chain Monte Carlo (MCMC) [78] steps, *i.e.*,

$$\frac{\partial \log p(\boldsymbol{x};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = \mathbb{E}_{x'\sim p(\boldsymbol{x'};\boldsymbol{\theta})}\left[\frac{\partial E(\boldsymbol{x'};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right] - \frac{\partial E(\boldsymbol{x};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}. \tag{2.27}$$

However, this process is still challenging as it results in biased gradients. This is mainly attributed to the infinite number of steps required to draw samples from an unormalized density. In practice, when a finite number of steps is used, the distribution of the sampler can be arbitrarily far away from the distribution parametrized by the model. Also, the process is sensitive to the sampler parameters like the step size, number of steps and the initialization. Recent approaches propose learning the sampler distribution end-to-end [79, 80, 81].

Noise-Contrastive Estimation (NCE)

Noise-contrastive estimation [82] recasts learning as a binary classification proxy problem, that uses the same parameters but requires statistics that are easier to compute. Specifically, NCE tries to distinguish samples from the data distribution, from samples $\boldsymbol{x}_n$ generated by a noisy easy to sample from distribution $p_n(\boldsymbol{x}_n)$, *i.e.*,

$$\min_{\boldsymbol{\theta}} \frac{1}{2|\boldsymbol{\mathcal{D}}|} \sum_{i=1}^{|\boldsymbol{\mathcal{D}}|} \log(h(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})) + \log(1 - h(\boldsymbol{x}_n^{(i)}; \boldsymbol{\theta})), \tag{2.28}$$

with $h(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{p(\boldsymbol{x}; \boldsymbol{\theta})}{p(\boldsymbol{x}; \boldsymbol{\theta}) + p_n(\boldsymbol{x})}$, and $p(\boldsymbol{x}; \boldsymbol{\theta})$ being the model distribution. The choice of the noise distribution is tricky as $p_n(\boldsymbol{x}_n)$ needs to have an analytically tractable density, be easy to draw samples from while being close to the data distribution. In high-dimensional space, it is hard to satisfy all three requirements. Mostly, the noise distribution is far from the data one which results in an easy classification problem and $p(\boldsymbol{x}; \boldsymbol{\theta})$ not learning much.

Score Methods

Score methods use surrogate objectives which have similar optima to the maximum likelihood objective. Notably, *score matching* (SM) [83] minimizes the Fisher divergence between the empirical distribution and the model one, *i.e.*,

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \mathbb{E}_{\boldsymbol{x} \sim p_d(\boldsymbol{x})}[(\boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{s}_d(\boldsymbol{x}))^2], \tag{2.29}$$

with $\boldsymbol{s}_d(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log(p_d(\boldsymbol{x}))$ is the score of the unknown data distribution and $\boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta}) = \nabla_{\boldsymbol{x}} \log(p(\boldsymbol{x}; \boldsymbol{\theta}))$ is the score of the model distribution. By applying integration by parts, Hyvarinen *et al.* [83] show that the program in Equation 2.29 is equivalent to

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim p_d}[\mathrm{tr}(\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})) + \frac{1}{2} \|\boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})\|^2], \tag{2.30}$$

with $\mathrm{tr}(\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta}))$ being the trace of the Hessian $\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})$. While this formulation avoids the computation of the partition function, it introduces a new challenge, *i.e.*, computing the trace of the Hessian, which scales quadratically with the input dimensions. This renders the application of score matching slow and unpractical for real applications with high-dimensional data. Different methods have been proposed to approximate the trace. Notably,

Approximate Backpropagation [84] only limits the backpropagation to the diagonal of the Hessian instead of computing the full matrix. The method has no guarantees on the approximation errors and no direct support in modern automatic differentiation frameworks. Another approach is *Curvature propagation* (CP) by Martens *et al.* [85]. The proposed estimator introduces noise for each node in the network which leads to high variance. Besides, it requires manually modifying the backpropagation code to handle complex numbers in neural networks.

Denoising score matching (DSM) [86, 87] is a different method for scaling score matching, that completely circumvents the Hessian. DSM applies the original score matching to noise-corrupted data, *i.e.*,

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \mathbb{E}_{q_\sigma(\widetilde{\boldsymbol{x}}|\boldsymbol{x})p_d(\boldsymbol{x})}[\|s_m(\widetilde{\boldsymbol{x}};\boldsymbol{\theta}) - \nabla_{\widetilde{\boldsymbol{x}}} \log q_\sigma(\widetilde{\boldsymbol{x}}|\boldsymbol{x})\|^2]. \tag{2.31}$$

Here, $q_\sigma(\widetilde{\boldsymbol{x}}|\boldsymbol{x})$ is a noise distribution. Song *et al.* [87] choose $q_\sigma(\widetilde{\boldsymbol{x}}|\boldsymbol{x})$ to be a Gaussian distribution $\mathcal{N}(\widetilde{\boldsymbol{x}}|\boldsymbol{x}, \sigma^2 I)$. The noisy samples in this case are generated by injecting Gaussian noise to the original samples, *i.e.*, $\widetilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\epsilon}\sigma^2$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$. DSM has the following limitations: (1) it can only recover the noise corrupted data distribution, (2) its performance is very sensitive to the choice of σ and which is non-trivial and based on heuristics.

Sliced Score Matching (SSM) [88] is another method aiming at scaling Equation 2.30 via projecting $s_d(\boldsymbol{x})$ and $s_m(\boldsymbol{x};\boldsymbol{\theta})$ onto some random direction $\boldsymbol{v}$ and comparing their average difference along those random directions, *i.e.*,

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \mathbb{E}_{\boldsymbol{v}\sim p_{\boldsymbol{v}}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[(\boldsymbol{v}^T s_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{v}^T s_d(\boldsymbol{x}))^2], \tag{2.32}$$

with p_v satisfying $\mathbb{E}_{p_{\boldsymbol{v}}}[\boldsymbol{v}\boldsymbol{v}^T] = \boldsymbol{I}$. Song *et al.* [88] show, via integration by parts, that the program above is equivalent to

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{v}}} \mathbb{E}_{p_d} \left[\frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \boldsymbol{v}^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}\right]. \tag{2.33}$$

Intuitively, the first term in the loss is at its lowest if $\boldsymbol{x}$ a stationary point of $\log p(\boldsymbol{x};\boldsymbol{\theta})$. The second term pushes the Hessian $\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})$ toward negative definiteness, to ensure that $\log p(\boldsymbol{x};\boldsymbol{\theta})$ is concave in $\boldsymbol{x}$, *i.e.*, $\boldsymbol{x}$ is a maximum

of $\log p(\boldsymbol{x};\boldsymbol{\theta})$. Note that $\boldsymbol{v}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}$ with $\mathbb{E}_{p_{\boldsymbol{v}}}[\boldsymbol{v}\boldsymbol{v}^T]$ is the Hutchinson estimator [89] of the trace of the Hessian $\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})$. The second derivative order derivative term $\boldsymbol{v}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x},\boldsymbol{w})\boldsymbol{v}$ can be efficiently computed using two gradient operations for a single v, as illustrated in Algorithm 1. If we average over M directions, $M+1$ gradient operations are required. In contrast, estimating the trace requires N differentiations. Hence, $M < N$ should be ensured to achieve the advantage of the method. SSM's main limitation is high variance due to the small number of the projection directions. In Chapter 5, we propose a new score matching objective, max-sliced score matching (MSSM), that replaces the averaging over random projection directions with computing the most informative one.

Algorithm 1: Slice Score Matching

1: **input:** Energy $E(\cdot;\boldsymbol{\theta})$, $\boldsymbol{x}$, $\boldsymbol{v}$
2: $\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \leftarrow -\nabla_{\boldsymbol{x}}E(\boldsymbol{x};\boldsymbol{\theta})$
3: $\boldsymbol{v}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \leftarrow \nabla_{\boldsymbol{x}}(\boldsymbol{v}^T\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))$
4: $J(\boldsymbol{\theta}) \leftarrow \boldsymbol{v}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v} + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2$
5: **return** $J(\boldsymbol{\theta})$

Other score based surrogate losses include *Stein discrepancy* [90] and its generalized form:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\theta})}[\boldsymbol{h}(\boldsymbol{x})\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x};\boldsymbol{\theta})^T + \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x})], \tag{2.34}$$

where $p(\boldsymbol{x};\boldsymbol{\theta})$ is a continuously differentiable density and $\boldsymbol{h}(\boldsymbol{x})$ is a function satisfying some regularity conditions. Different choices of $\boldsymbol{h}(\boldsymbol{x})$ led to different versions of Stein discrepancy. For instance, Li *et al.* [91] propose setting $\boldsymbol{h}(\boldsymbol{x})$ as the feature map of some kernel in the Stein class [92] of $q(\boldsymbol{x})$, and solving a finite-sample version of Equation 2.34 to obtain estimates of $\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x})$ at the sample points. Shi *et al.* [93] adopt a different approach: they build their estimator by expanding $\nabla_{\boldsymbol{x}}\log q(\boldsymbol{x})$ as a spectral series of eigenfunctions and solve for the coefficients using the generalized Stein discrepancy's form above.

To **summarize**, learning structured models at scale is as challenging as learning scalable inference techniques. Additional complications arise from using approximate inference algorithms as subroutines in the learning procedure. Recently introduced deep learning platforms alleviate this issue by

jointly reasoning about inference and learning in an end to end fashion.

2.4 Deep Structured Models

Structured models were initially used in a post-processing stage [94, 95]. In a first stage, deepnets were trained to produce local evidence for every output y_i. In a second step, local evidence was fixed and correlations were learned [96, 97, 98, 99, 100, 101, 102]. While leading to impressive results, a two-step training procedure seemed counterintuitive. Subsequently unified frameworks for learning and inference have been proposed. Different components were parametrized as deepnets.

Deep Potentials

Given an explicitly specified energy function, potentials are parametrized via deepnets. Classical inference techniques are then leveraged to solve the inference task. For instance, Tompson *et al.* [103] learn unaries and pairwise terms using convolutional nets (CNN) for the task of pose estimation. Jaderberg *et al.* [104] propose an energy-based model for text recognition and learn the unaries using a CNN that predicts characters at each position of the output and higher-order potentials via another CNN that detects the presence of N-grams. To give another example, Chandra *et al.* [105] learn the covariance matrix and mean of a Gaussian conditional random field using CNNs, for the task of semantic segmentation.

Deep Energy Function

Different frameworks learning an energy function parametrized by a deepnet have been recenly proposed [106, 107, 108, 109]. For instance, *Structured Prediction Networks* (SPEN) [106] relaxes all output variables to be continuous, performs inference via gradient descent and uses the SSVM loss for learning. SPEN is however prone to overfitting and has no guarantees learning the desired structure. To enforce structural constraints, Graber *et al.* [107, 108], include graphical model inference inside the deepnet. Intuitively, this increases the expressivity of the model without incurring the cost of including higher-order potentials within the explicit structure. Another framework operating on deep energies is Deep Value Networks (SVN) [109].

Instead of the SSVM loss, they learn to estimate the task loss on different output configurations for a given input. For examples, when applied to image segmentation, the value network takes an image and a segmentation mask as inputs and predicts a scalar estimating the intersection over union (IoU) metric between the input and ground truth masks.

Deep Inference Engine
Deepnets were used to approximate parts of classical inference algorithms or output the inference solution $\boldsymbol{y}^*$. Chen *et al.* [110] propose a platform where model parameters $\boldsymbol{\theta}$ are learned such that a single message passing iteration is enough for learning. Zheng *et al.* [111] formulate inference with Gaussian pairwise potentials as Recurrent Neural Networks [111]. Liu *et al.* [112] propose learning to approximate mean-field inference using convolutional and pooling operations only, and do not rely on iterative refinement of the solution. Another model is InfNet [113], it uses the same objective as SPEN but replaces the gradient descent inference with a network that produces the output of inference problem.

Autoregressive Models represent another class of deep structured frameworks. Specifically, they define an order over the output variables and predict one variable at a time conditioned on the previous ones. This mimicks a chain rule on a distribution. RNNs are notable examples, that have been successfully applied to a range of sequence prediction problems, including speech recognition [114, 115], language modeling [116, 117], machine translation [118, 119]. A few variants of RNNs also exist that enhances its ability to handle longer sequences [120, 121, 122, 123].

2.5 Learning-Based Combinatorial Optimization

Decades of research on combinatorial optimization, often also referred to as discrete optimization, uncovered a large amount of valuable exact, approximate and heuristic algorithms. Already in the early 2000s, but more prominently recently [124, 125, 126, 127, 128, 129], learning based algorithms have been suggested for combinatorial optimization. They are based on the intuition that instances of similar problems are often solved repeatedly. While

humans have uncovered impressive heuristics, data driven techniques are likely to uncover even more compelling mechanisms. It is beyond the scope of this dissertation to review the vast literature on combinatorial optimization. Instead, we subsequently focus on learning based methods. Among the first, is work by Boyan and Moore [125], discussing how to learn to predict the outcome of a local search algorithm in order to bias future search trajectories. Around the same time, reinforcement learning techniques were used to solve resource-constrained scheduling tasks [124]. Reinforcement learning is also the technique of choice for recent approaches addressing NP-hard tasks [127, 128, 129, 130] like the traveling salesman, knapsack, maximum cut, and minimum vertex cover problems. Similarly, promising results exist for structured prediction problems like dialog generation [131, 132, 133], program synthesis [134, 135, 136], semantic parsing [137], architecture search [138], chunking and parsing [139], machine translation [140, 141, 142], summarization [143], image captioning [144], knowledge graph reasoning [145], query rewriting [146, 147] and information extraction [148, 149]. Instead of directly learning to solve a given program, machine learning techniques have also been applied to parts of combinatorial solvers, *e.g.*, to speed up branch-and-bound rules [150, 151, 152, 153]. We also want to highlight recent work on learning to optimize for continuous problems [154, 155].

Given those impressive results on challenging real-world problems, we wonder: Can we learn programs for solving higher-order CRFs? Since CRF inference is typically formulated as a combinatorial optimization problem, we want to know how recent advances in learning based combinatorial optimization can be leveraged. We present a technique for learning heuristics for solving CRF inference using reinforcement learning in Chapter 4.

CHAPTER 3

GAUSSIAN CONDITIONAL RANDOM FIELDS BASED VARIATIONAL AUTO-ENCODERS

In this chapter, we present a conditional random field (CRF) based variational auto-encoder (VAE) formulation which is able to model multi-modal distributions while efficiently taking into account *structural consistency*, via higher-order potentials with exact inference procedure. Moreover, we introduce a *controllability* mechanism that enables incorporating external constraints from diverse sources, at runtime. We apply our model to gray-level image colorization. We illustrate visually appealing results on the Labeled Faces in the Wild (LFW) [156], LSUN-Church [157] and ILSVRC-2015 [158] datasets and assess the photo-realism aspect with a user study.

3.1 Related Work

Before discussing the details of our proposed model, we briefly review the areas of colorization and variational auto-encoders subsequently.

Colorization: Colorizing a given gray-level image is an important task in the media and advertising industry. It requires to predict the two missing channels of a provided gray-level input. Similar to other computer vision tasks like monocular depth-prediction or semantic segmentation, colorization is ill-posed. However, unlike the aforementioned tasks, colorization is also ambiguous, *i.e.*, many different colorizations are perfectly plausible. For instance, differently colored shirts or cars are very reasonable, while there is certainly less diversity in shades of façades. Capturing these subtleties is a non-trivial problem. Early colorization methods rely on user-interaction in the form of a reference image or scribbles [164, 165, 166, 167, 168, 169]. First attempts to automate the colorization process [170] rely on classifiers trained

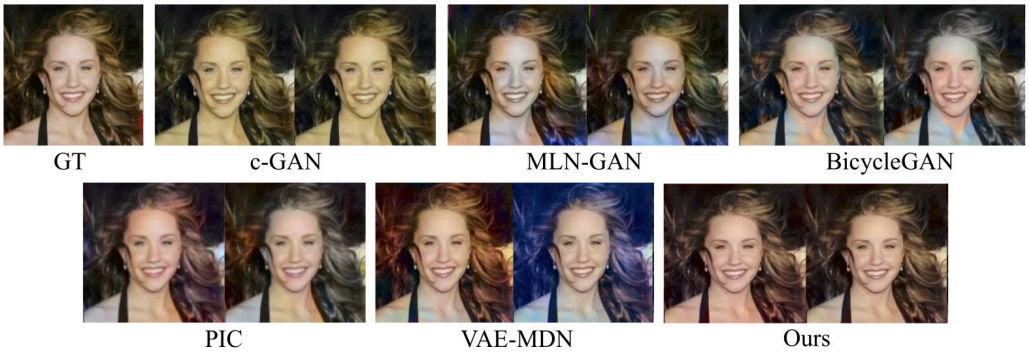


Figure 3.1: Diverse colorizations of the ground truth (GT) generated by c-GAN [159], MLN-GAN [160], BicycleGAN [161], PIC [162], VAE-MDN [163] and our approach.

on datasets containing a few tens to a few thousands of images. Naturally, recent deepnet based methods scaled to much larger datasets containing millions of images [171, 172, 173, 174, 175, 176]. All these methods operate on a provided intensity field and produce a single color image which does not embrace the ambiguity of the task. To address ambiguity, Royer *et al.* [162] use a PixelCNN [177] to learn a conditional model $p(\boldsymbol{x}|\boldsymbol{g})$ of the color field $\boldsymbol{x}$ given the gray-level image $\boldsymbol{g}$, and draw multiple samples from this distribution to obtain different colorizations. In addition to compelling results, failure modes are reported due to ignored complex long-range pixel interactions, *e.g.*, if an object is split due to occlusion. Similarly, Guadarrama *et al.* [178] use PixelCNNs to learn multiple embeddings $\boldsymbol{z}$ of the gray-level image, before a convolutional refinement network is trained to obtain the final image. Note that in this case, instead of learning $p(\boldsymbol{x}|\boldsymbol{g})$ directly, the color field $\boldsymbol{x}$ is represented by a low-dimensional embedding $\boldsymbol{z}$. Although, the aforementioned PixelCNN based approaches yield diverse colorization, they lack large-scale spatial coherence and are prohibitively slow due to the auto-regressive, *i.e.*, sequential, nature of the model. Another conditional latent variable approach for diverse colorization was proposed by Deshpande *et al.* [163]. The authors train a variational auto-encoder to produce a low-dimensional embedding of the color field. Then, a Mixture Density Network (MDN) [179] is used to learn a multi-modal distribution $p(\boldsymbol{z}|\boldsymbol{g})$ over the latent codes. Latent samples are afterwards converted to multiple color fields using a decoder. This approach offers an efficient sampling mechanism. However, the output is often speckled because colors are sampled independently

for each pixel. Beyond the aforementioned probabilistic formulations, conditional generative adversarial networks [159] have been used to produce diverse colorizations. However, mode collapse, which results in the model producing one color version of the gray-level image, is a frequent concern in addition to consistency. This is mainly due to the generator learning to largely ignore the random noise vector when conditioned on a relevant context. Cao *et al.* [160] address the former issue by concatenating the input noise channel with several convolutional layers of the generator. A second solution is proposed by Zhu *et al.* [161], where the connection between the output and latent code is encouraged to be invertible to avoid many to one mappings. These models show compelling results when tested on datasets with strong alignment between the samples, *e.g.*, the LSUN bedroom dataset [157] in [160] and image-to-image translation datasets [180, 181, 159, 182, 183] in [161]. We will demonstrate in Section 3.3 that they lack global consistency on more complex datasets. The above mentioned generative techniques often lack structural consistency, *e.g.*, parts of a shirt differ in color or the car is speckled. Inconsistencies are due to the fact that structural coherence is only encouraged implicitly when using deepnet based generative methods. For example, in results obtained from [163, 159, 162, 160, 161] illustrated in Figure 3.1, the color of the shoulder and neck differ as these models are sensitive to occlusion. In addition, existing diverse colorization techniques often lack a form of controllability permitting to interfere while maintaining structural consistency. Our proposed model addressed all the above mentioned challenges, *i.e.*, diversity, structural consistency and controllability. To this end we formulate the colorization task by augmenting variational auto-encoder models with Gaussian Conditional Random Fields (G-CRFs).

Variational Auto-Encoders: Variational auto-encoders (VAEs) [184] and conditional variants [185], *i.e.*, conditional VAEs (CVAEs), have been used to model ambiguity in a variety of tasks [186, 187]. They are based on the manifold assumption stating that a high-dimensional data point $\boldsymbol{x}$, such as a color image, can be modeled based on a low-dimensional embedding $\boldsymbol{z}$ and some auxiliary data $\boldsymbol{g}$, such as a gray-level image. Formally, existence of a low-dimensional embedding space and a transformation via the conditional $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{g})$ is assumed. Given a dataset $\boldsymbol{\mathcal{D}}$ containing pairs of conditioning information $\boldsymbol{g}$ and desired output $\boldsymbol{x}$, *i.e.*, given $\boldsymbol{\mathcal{D}} = \{(\boldsymbol{g}^{(i)}, \boldsymbol{x}^{(i)})\}_{i=1}^{|\boldsymbol{\mathcal{D}}|}$, CVAEs formulate maximization of the conditional log-likelihood $\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g})$, param-

eterized by $\boldsymbol{\theta}$, by considering the following identity:

$$\begin{aligned} &\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g}) - D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g}), p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})) = \\ &-D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g}), p(\boldsymbol{z}|\boldsymbol{g})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})}[\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g},\boldsymbol{z})]. \end{aligned} \tag{3.1}$$

Hereby, $D_{\mathrm{KL}}(\cdot,\cdot)$ denotes the Kullback-Leibler (KL) divergence between two distributions, and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$ is used to approximate the intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$ of a deepnet which models the conditional $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g},\boldsymbol{z})$. The approximation of the posterior, *i.e.*, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$, is referred to as the encoder, while the deepnet used for reconstruction, *i.e.*, for modeling the conditional $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g},\boldsymbol{z})$, is typically called the decoder. Since the KL-divergence is non-negative, we obtain a lower bound on the data log-likelihood $\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g})$ when considering the right-hand side of the identity given in Eq. 3.1. CVAEs minimize the negated version of this lower bound, *i.e.*,

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g}), p(\boldsymbol{z}|\boldsymbol{g})) - \frac{1}{N}\sum_{i=1}^{N} \ln p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g},\boldsymbol{z}^{(i)}), \tag{3.2}$$

where $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})}$ is approximated via N samples $\boldsymbol{z}^{(i)} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$. For simplicity of the exposition, we ignored the summation over the samples in the dataset $\boldsymbol{\mathcal{D}}$, and provide the objective for training of a single pair $(\boldsymbol{x},\boldsymbol{g})$.

We next discuss how we combine those ingredients for diverse, controllable yet structurally coherent colorization.

3.2 Approach

Our proposed colorization model has several appealing properties: (1) *diversity*, *i.e.*, it generates diverse and realistic colorizations for a single gray-level image; (2) *global coherence*, enforced by explicitly modeling the output-space distribution of the generated color field using a fully connected Gaussian conditional random field (G-CRF); (3) *controllability*, *i.e.*, our model can consider external constraints at runtime efficiently. For example, the user can enforce a given object to have a specific color or two separated regions to have the same colorization.

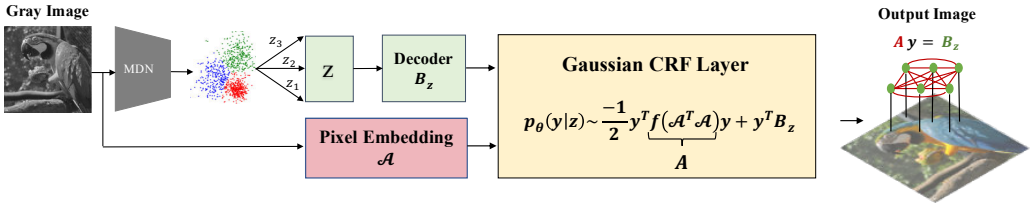


Figure 3.2: A fully connected Gaussian conditional random field (G-CRF) based VAE for diverse and globally coherent colorization. To generate diverse colorizations, we use a mixture density network (MDN) to represent the multi-modal distribution of the color field embedding $\boldsymbol{z}$ given the gray-level image $\boldsymbol{g}$. At test time, we sample multiple embeddings that are subsequently decoded to generate different colorizations. To ensure global consistency, we model the output space distribution of the decoder using a G-CRF.

3.2.1 Overview

We provide an overview of our approach in Figure 3.2. Given a gray-level image $\boldsymbol{g}$ with P pixels, our goal is to produce different color fields $\boldsymbol{y} \in \mathbb{R}^{2P}$ consisting of two channels $\boldsymbol{y}_a \in \mathbb{R}^P$ and $\boldsymbol{y}_b \in \mathbb{R}^P$ in the *Lab* color space. In addition, we enforce spatial coherence at a global scale and enable controllability using a Gaussian conditional random field which models the output space distribution. To produce a diverse colorization, given a dataset $\boldsymbol{\mathcal{D}} = \{\boldsymbol{x}^{(i)}, \boldsymbol{g}^{(i)}\}_{i=1}^{|\boldsymbol{\mathcal{D}}|}$, we want to learn a multi-modal conditional distribution $p(\boldsymbol{x}|\boldsymbol{g})$ of the color field $\boldsymbol{x}$ given the gray-level image $\boldsymbol{g}$. However, learning this conditional is challenging since the color field $\boldsymbol{x}$ and the intensity field $\boldsymbol{g}$ are high dimensional. Hence, training samples for learning $p(\boldsymbol{x}|\boldsymbol{g})$ are sparsely scattered and the distribution is difficult to capture, even when using large datasets. Therefore, we assume the manifold hypothesis to hold, and we choose to learn a conditional $p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{g})$ based on low-dimensional embeddings $\boldsymbol{z}$ captured from $\boldsymbol{x}$ and $\boldsymbol{g}$, by using a variational auto-encoder which approximates the intractable posterior $p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{g})$ via an encoder. Deshpande *et al.* [163] demonstrated that sampling from the approximation of the posterior results in low variance of the generated images. Following Deshpande *et al.* [163], we opt for a multi-stage training procedure to directly sample from $p(\boldsymbol{z}|\boldsymbol{g})$ as follows. To capture the low-dimensional embedding, in a *first training stage*, we use a variational auto-encoder to learn a parametric unimodal Gaussian encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{g}) \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}, \sigma^2_{\boldsymbol{\phi}}\boldsymbol{I})$ of the color field embedding $\boldsymbol{z}$ given both the gray-level image $\boldsymbol{g}$ and the color image $\boldsymbol{x}$ (Figure 3.3 (a)). At the same time, we learn the parameters $\boldsymbol{\theta}$ of the decoder

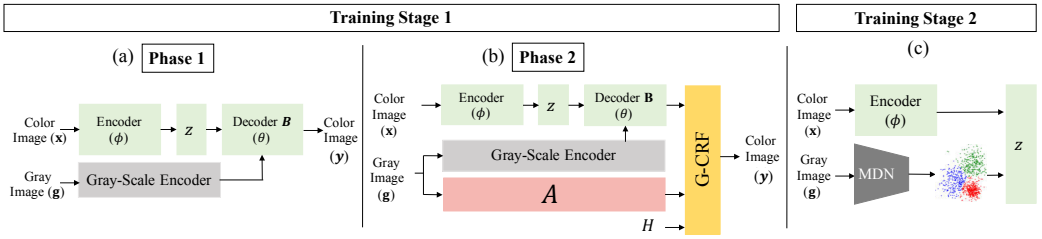


Figure 3.3: Overview of the model architecture and the training procedure. In the first training stage, we learn a low-dimensional embedding $\boldsymbol{z}$ of the *color field* $\boldsymbol{x}$ conditioned on the gray-level image $\boldsymbol{g}$ using a VAE. To disentangle color from structure, we first learn the unary term $\boldsymbol{B}$ in *phase 1*, then in *phase 2*, learn a precision matrix that encodes the *structure* of the image by imposing the constraint that pixels with similar intensities should have similar colorizations. To enable controllability, we use a training schedule specified in the matrix $\boldsymbol{H}$ to incrementally mask the decoded pixel colors in the unary term $\boldsymbol{B}$ and hence gradually rely on the $\boldsymbol{A}$ matrix to restore the colorization from the unary term. In the second training stage, we use an MDN to learn a multi-modal distribution of the latent embedding given the gray-level image.

$p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{g})$. Importantly, we note that the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$ takes advantage of both the color image $\boldsymbol{x}$ and the gray-level intensities $\boldsymbol{g}$ when mapping to the latent representation $\boldsymbol{z}$. Due to the use of the color image, we expect that this mapping can be captured to a reasonable degree using a uni-modal distribution, *i.e.*, we use a Gaussian.

However, multiple colorizations can be obtained from a gray-scale image $\boldsymbol{g}$ during inference. Hence, following Deshpande *et al.* [163], we do not expect a uni-modal distribution $p(\boldsymbol{z}|\boldsymbol{g})$ to be accurate during testing, when only conditioning on the gray-level image $\boldsymbol{g}$.

To address this issue, in a *second training stage*, we train a mixture density network (MDN) $p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$ to maximize the log-likelihood of embeddings $\boldsymbol{z}$ sampled from $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$ (Figure 3.3 (b)). Intuitively, for a gray-level image, the MDN predicts the parameters of M Gaussian components each corresponding to a different colorization. The embedding $\boldsymbol{z}$ that was learned in the first stage is then tied to one of these components. The remaining components are optimized by close-by gray-level image embeddings.

At test time, N different embeddings $\{\boldsymbol{z}\}_{k=1}^{N}$ are sampled from the MDN $p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$ and transformed by the decoder into diverse colorizations, as we show in Figure 3.2.

To encourage globally coherent colorizations and to ensure controllability, we use a fully connected G-CRF layer to model the output space distribution.

The negative log-posterior of the G-CRF has the form of a quadratic energy function:

$$E(\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^T \boldsymbol{A_g} \boldsymbol{y} - \boldsymbol{B_{z,g}} \boldsymbol{y}. \tag{3.3}$$

It captures unary and higher-order correlations (HOCs) between the pixels' colors for the a and b channels. Intuitively, the joint G-CRF enables the model to capture more global image statistics which turn out to yield more spatially coherent colorizations as we will show in the results section. The unary term $\boldsymbol{B_{z,g}}$ is obtained from the VAE decoder and encodes the color per pixel. The HOC term $\boldsymbol{A_g} = f(\boldsymbol{\mathcal{A}_g^T} \boldsymbol{\mathcal{A}_g})$ is responsible for encoding the structure of the input image. It is a function of the inner product of low-rank pixel embeddings $\boldsymbol{\mathcal{A}_g}$, learned from the gray-level image and measuring the pairwise similarity between the pixels' intensities. The intuition is that pixels with similar intensities should have similar colorizations. The HOC term is shared between the different colorizations obtained at test time. Beyond global consistency, it also enables controllability by propagating user edits encoded in the unary term properly. Due to the symmetry of the HOC term, the quadratic energy function has a unique global minimum that can be obtained by solving the system of linear equations:

$$\boldsymbol{A_g} \boldsymbol{y} = \boldsymbol{B_{z,g}}. \tag{3.4}$$

Subsequently, we drop the dependency of $\boldsymbol{A}$ and $\boldsymbol{B}$ on $\boldsymbol{g}$ and $\boldsymbol{z}$ for notational simplicity.

We now discuss how to perform inference in our model and how to learn the model parameters such that colorization and structure are disentangled and controllability is enabled by propagating user strokes.

3.2.2 Inference

In order to ensure a globally consistent colorization, we take advantage of the structure in the image. To this end, we encourage two pixels to have similar colors if their intensities are similar. Thus, we want to minimize the difference between the color field $\boldsymbol{y}$ for the a and b channels and the weighted average of the colors at similar pixels. More formally, we want to encourage the equalities $\boldsymbol{y}_a = \hat{\boldsymbol{S}} \boldsymbol{y}_a$ and $\boldsymbol{y}_b = \hat{\boldsymbol{S}} \boldsymbol{y}_b$, where $\hat{\boldsymbol{S}} = \text{softmax}(\boldsymbol{\mathcal{A}^T} \boldsymbol{\mathcal{A}})$ is a

similarity matrix obtained from applying a softmax function to every row of the matrix resulting from $\boldsymbol{\mathcal{A}^T\mathcal{A}}$. To simplify, we use the block-structured matrix $\boldsymbol{S} = \text{diag}(\hat{\boldsymbol{S}}, \hat{\boldsymbol{S}})$.

In addition to capturing the structure, we obtain the color prior and controllability by encoding the user input in the computed unary term $\boldsymbol{B}$. Hence, we add the constraint $\boldsymbol{Hy} = \boldsymbol{\alpha}$, where $\boldsymbol{H}$ is a diagonal matrix with 0 and 1 entries corresponding to whether the pixel's value is not or is specified by the user, and $\boldsymbol{\alpha}$ a vector encoding the color each pixel should be set to.

With the aforementioned intuition at hand we obtain the quadratic energy function to be minimized as: $E_{\boldsymbol{\theta},\boldsymbol{g},\boldsymbol{z}}(\boldsymbol{y}) = \frac{1}{2}\|(\boldsymbol{I}-\boldsymbol{S})\boldsymbol{y}\|^2 + \frac{1}{2}\beta\|\boldsymbol{Hy}-\boldsymbol{\alpha}\|^2$, with β being a hyper-parameter. This corresponds to a quadratic energy function of the form $\frac{1}{2}\boldsymbol{y}^T\boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{y} + C$, where $\boldsymbol{A} = (\boldsymbol{S}-\boldsymbol{I})^T(\boldsymbol{S}-\boldsymbol{I}) + \beta\boldsymbol{H}^T\boldsymbol{H}$, $\boldsymbol{B} = -2\beta\boldsymbol{\alpha}^T\boldsymbol{H}$ and $C = \beta\boldsymbol{\alpha}^T\boldsymbol{\alpha}$. It is immediately apparent that the unary term only encodes color statistics while the HOC term is only responsible for structural consistency. Intuitively, the conditional $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{g},\boldsymbol{z})$ is interpreted as a Gaussian multi-variate density:

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{g}) \propto \exp(-E_{\boldsymbol{\theta},\boldsymbol{g},\boldsymbol{z}}(\boldsymbol{y})), \tag{3.5}$$

parametrized by the above defined energy function $E_{\boldsymbol{\theta},\boldsymbol{g},\boldsymbol{z}}$. It can be easily checked that $\boldsymbol{A}$ is a positive definite full rank matrix. Hence, for a strictly positive definite matrix, inference is reduced to solving a linear system of equations:

$$((\boldsymbol{I}-\boldsymbol{S})^T(\boldsymbol{I}-\boldsymbol{S}) + \beta\boldsymbol{H}^T\boldsymbol{H})\boldsymbol{y} = \beta\boldsymbol{H}^T\boldsymbol{\alpha}. \tag{3.6}$$

We solve the linear system above using the LU decomposition of the $\boldsymbol{A}$ matrix. Empirically, we observe that the linear system above is well-conditioned and led to stable learning. How to learn the terms $\boldsymbol{\alpha}$ and $\boldsymbol{S}$ will be explained in the following.

3.2.3 Learning

We now present the two training stages illustrated in Figure 3.3 to ensure color and structure disentanglement and to produce diverse colorizations. We also discuss the modifications to the loss given in Equation 3.2 during each stage.

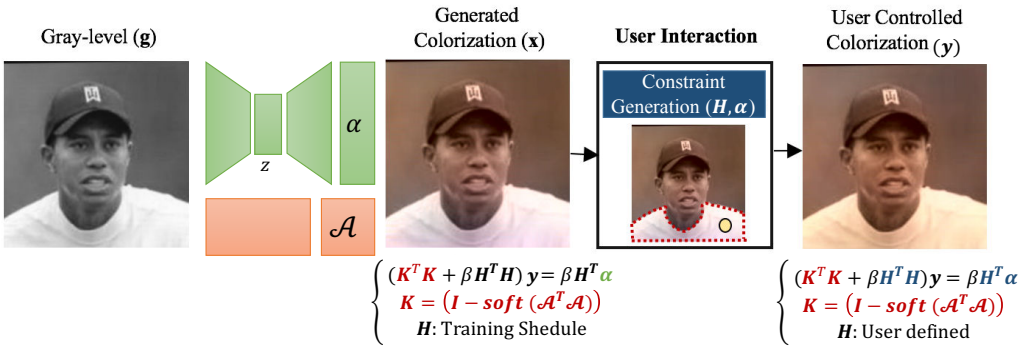


Figure 3.4: Controllability: Given a gray-level image, we learn to disentangle structure from colorization. The HOC term is used to propagate sparse user edits encoded in the $\boldsymbol{H}$ and $\boldsymbol{\alpha}$ terms.

Stage 1: Training a Structured Output Space Variational Auto-Encoder: During the first training stage, we use the variational auto-encoder formulation to learn a low-dimensional embedding for a given color field. This stage is divided into two phases to ensure color and structure disentanglement. In a first phase, we learn the unary term produced by the VAE decoder. In the second phase, we fix the weights of the VAE apart from the decoder's two top-most layers and learn a D-dimensional embedding matrix $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{D \times P}$ for the P pixels from the gray-level image. The matrix $\hat{\boldsymbol{S}}$ obtained from applying a softmax to every row of $\boldsymbol{\mathcal{A}}^T\boldsymbol{\mathcal{A}}$ is used to encourage a smoothness prior $\boldsymbol{y} = \boldsymbol{S}\boldsymbol{y}$ for the a and b channels. In order to ensure that the $\boldsymbol{S}$ matrix learns the structure required for the controllability stage, where sparse user edits need to be propagated, we follow a training schedule where the unary terms are masked gradually using the $\boldsymbol{H}$ matrix. The input image is reconstructed from the sparse unary entries using the learned structure. When colorization from sparse user edits is desired, we solve the linear system from Equation 3.6 for the learned HOC term, an $\boldsymbol{H}$ matrix and $\boldsymbol{\alpha}$ term encoding the user edits, as illustrated in Figure 3.4. We explain the details of the training schedule in the experimental section. Given the new formulation of the G-CRF posterior, the program for the first training stage reads as follows:

$$\begin{aligned} \min_{\boldsymbol{\phi},\boldsymbol{\theta}} \quad & D_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}, \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}\boldsymbol{I})), \mathcal{N}(0, \boldsymbol{I})) - \frac{1}{N}\sum_{i=1}^{N} \ln p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{z}^{(i)}, \boldsymbol{g}) \\ \text{s.t.} \quad & \boldsymbol{z}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}, \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}\boldsymbol{I}) \end{aligned} \tag{3.7}$$

Subsequently we use the term L to refer to the objective function of this program.

Stage 2: Training a Mixture Density Network (MDN): Since a color image $\boldsymbol{x}$ is not available during testing, in the second training stage, we capture the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$, a Gaussian which was learned in the first training stage, using a parametric distribution $p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$. Due to the dependence on the color image $\boldsymbol{x}$ we expect the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$ to be easier to model than $p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$. Therefore, we let $p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$ be a Gaussian mixture model (GMM) with M components. Its means, variances, and component weights are parameterized via a mixture density network (MDN) with parameters $\boldsymbol{\psi}$. Intuitively, for a given gray-level image, we expect the M components to correspond to different colorizations. The colorfield embedding $\boldsymbol{z}$ learned from the first training stage is mapped to one of the components by minimizing the negative conditional log-likelihood, *i.e.*, by minimizing:

$$-\ln p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g}) = -\ln \sum_{i=1}^{M} \pi_{\boldsymbol{g},\boldsymbol{\psi}}^{(i)} \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{g,\boldsymbol{\psi}}^{(i)}, \sigma). \tag{3.8}$$

Hereby, $\pi_{\boldsymbol{g},\boldsymbol{\psi}}^{(i)}$, $\boldsymbol{\mu}_{\boldsymbol{g},\psi}^{(i)}$ and σ refer to, respectively, the mixture coefficients, the means and a fixed co-variance of the GMM learned by an MDN network parametrized by $\boldsymbol{\psi}$. However, minimizing $-\ln p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$ is hard as it involves the computation of the logarithm of a summation over the different exponential components. To avoid this, we explicitly assign the code $\boldsymbol{z}$ to that Gaussian component m, which has its mean closest to $\boldsymbol{z}$, *i.e.*, $m = \underset{i}{\operatorname{argmin}} \|\boldsymbol{z} - \boldsymbol{\mu}_{\boldsymbol{g},\boldsymbol{\psi}}^{(i)}\|$. Hence, the negative log-likelihood loss $-\ln p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{g})$ is reduced to solving the following program:

$$\min_{\boldsymbol{\psi}} -\ln \pi_{\boldsymbol{g},\boldsymbol{\psi}}^{(m)} + \frac{\|\boldsymbol{z} - \boldsymbol{\mu}_{\boldsymbol{g},\boldsymbol{\psi}}^{(m)}\|^2}{2\sigma^2} \qquad \text{s.t.} \begin{cases} \boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}, \boldsymbol{\sigma}_{\boldsymbol{\phi}}^2 \boldsymbol{I}) \\ m = \underset{i \in \{1,\ldots,M\}}{\operatorname{argmin}} \|\boldsymbol{z} - \boldsymbol{\mu}_{g,\boldsymbol{\psi}}^{(i)}\| \end{cases} \tag{3.9}$$

Note that the latent samples $\boldsymbol{z}$ are obtained from the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{g})$ learned in the first stage. Next, we show that the gradient of the loss with respect of A and B can be computed efficiently via evaluating closed-form expressions.

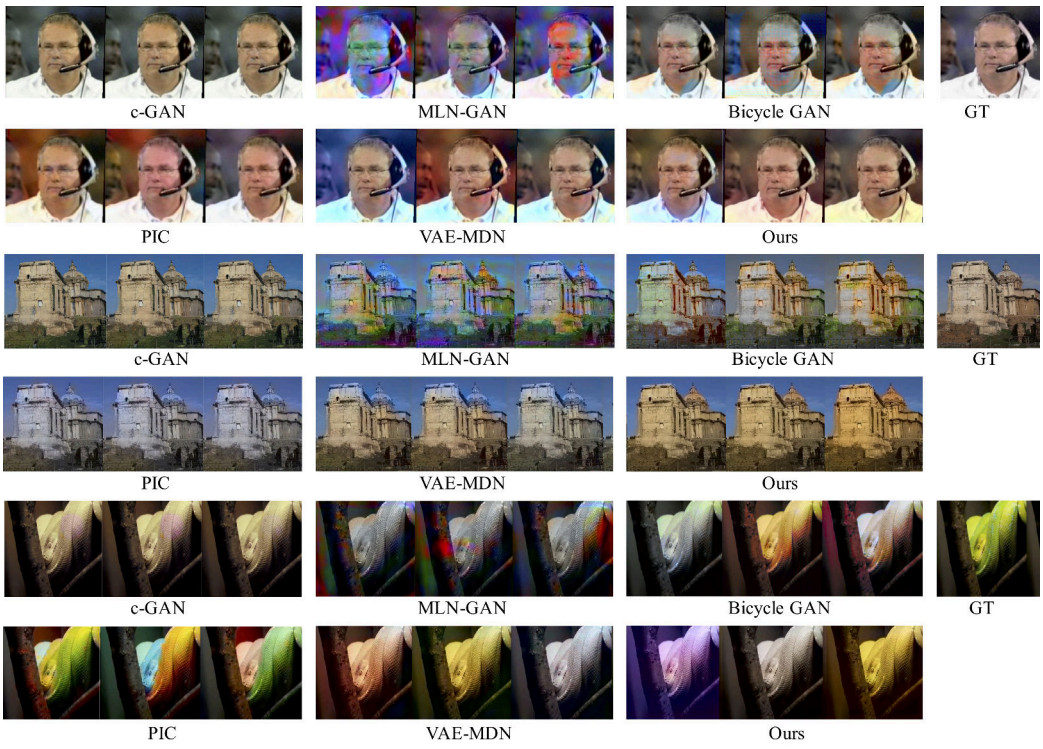


Figure 3.5: Qualitative comparison of diverse colorizations obtained from c-GAN [159], MLN-GAN [160], BicycleGAN [161], PIC [162], VAE-MDN [163] and our approach.

Gradients of the G-CRF Parameters: During the first training phase's forward pass, the G-CRF receives the unary term $\boldsymbol{B}$ and the HOC term $\boldsymbol{A}$, and outputs the reconstructed color field after solving the linear system $\boldsymbol{Ay} = \boldsymbol{B}$. In the backward pass, the G-CRF layer receives the gradient of the objective function L of the program given in Equation 3.7 with respect to $\boldsymbol{x}$ and computes closed-form expressions for the gradient of the loss with respect to $\boldsymbol{A}$ and $\boldsymbol{B}$. Using the chain rule, the gradients of the remaining parameters can be expressed in terms of $\frac{\partial L}{\partial \boldsymbol{A}}$ and $\frac{\partial L}{\partial \boldsymbol{B}}$.

Note that the gradient of the unary term decomposes as $\frac{\partial L}{\partial \boldsymbol{y}} = \frac{\partial L}{\partial \boldsymbol{B}} \frac{\partial \boldsymbol{B}}{\partial \boldsymbol{y}}$. By taking Equation 3.4 into account, we compute $\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{y}} = \boldsymbol{A}$. Hence, it can be easily verified that a closed-form expression of the gradient of the loss with respect to the unary term $\boldsymbol{B}$ corresponds to solving the following system of linear equations:

$$\boldsymbol{A} \frac{\partial L}{\partial \boldsymbol{B}} = \frac{\partial L}{\partial \boldsymbol{y}}. \tag{3.10}$$

Table 3.1: Results of the user study (% of the model in **bold** winning).

	Ours vs VAE-MDN	***Ours*** *vs PIC*	***VAE-MDN*** *vs PIC*
LFW	61.12 %	59.04 %	57.17 %
LSUN-Church	66.89 %	71.61 %	54.46 %
ILSVRC-2015	54.79 %	66.98 %	62.88%

Similarly, we derive the gradient of the pairwise term by application of the chain rule: $\frac{\partial L}{\partial \boldsymbol{A}_{i,j}} = \frac{\partial L}{\partial \boldsymbol{B}_i} \frac{\partial \boldsymbol{B}_i}{\partial \boldsymbol{A}_{i,j}} = \frac{\partial L}{\partial \boldsymbol{B}_i} \boldsymbol{y}_j$. Hence:

$$\frac{\partial L}{\partial \boldsymbol{A}} = \frac{\partial L}{\partial \boldsymbol{B}} \cdot \boldsymbol{y}^T = \frac{\partial L}{\partial \boldsymbol{B}} \otimes \boldsymbol{y}. \tag{3.11}$$

3.3 Experiments

Next, we present quantitative and qualitative results on three datasets of increasing color field complexity: (1) the Labeled Faces in the Wild dataset (LFW) [156], which consists of 13,233 face images aligned by deep funneling [188]; (2) the LSUN-Church dataset [157] containing 126,227 images and (3) the validation set of ILSVRC-2015 (ImageNet-Val) [158] with 50,000 images. We compare the diverse colorizations obtained by our model with three baselines representing three different generative models: (1) the conditional generative adversarial network [159, 160, 161]; (2) the variational auto-encoder with MDN [163]; and (3) the probabilistic image colorization model [162] based on PixelCNN. Note that Deshpande *et al.* [163] presents a comparison between VAE-MDN and a conditional VAE, demonstrating the benefits of the VAE-MDN approach.

3.3.1 Baselines

Conditional Generative Adversarial Network: We compare our approach with three GAN models: the c-GAN architecture proposed by Isola *et al.* [159], the GAN with multi-layer noise by Cao *et al.* [160] and the Bicycle GAN by Zhu *et al.* [161].

Variational Auto-Encoder with Mixture Density Network (VAE-MDN): The architecture by Deshpande *et al.* [163] trains an MDN based

Table 3.2: Quantitaive comparison with baselines. We use the error-of-best per pixel (Eob), the variance (Var), the mean structural similarity SSIM across all pairs of colorizations generated for one image (SSIM) and the training time (Train.) as performance metrics.

	LFW				LSUN-Church				ILSVRC-2015			
Method	eob.	Var.	SSIM.	Train.	eob.	Var.	SSIM.	Train.	eob.	Var.	SSIM.	Train.
c-GAN[**?**]	.047	$8.40e^{-6}$	.92	∼**4**h	.048	$6.20e^{-6}$	.94	∼**39**h	.048	$8.88e^{-6}$	.91	∼**18**h
MLN-GAN[160]	.057	$2.83e^{-2}$	**.12**	∼**4**h	.051	$2.48e^{-2}$	**.34**	∼**39**h	.063	$1.73e^{-2}$	.38	∼**18**h
BicycleGAN[161]	.045	$6.50e^{-3}$	.51	∼**4**h	.048	$2.20e^{-2}$	.38	∼**39**h	.042	$2.20e^{-2}$	**.15**	∼**18**h
VAE-MDN[163]	.035	$1.81e^{-2}$	.49	∼**4**h	.028	$\mathbf{1.05e^{-2}}$	.77	∼**39**h	.033	$7.17e^{-3}$	.48	∼**18**h
PIC[162]	.043	$\mathbf{5.32e^{-2}}$	.36	∼48h	.047	$7.40e^{-5}$	.91	∼144h	.035	$\mathbf{6.74e^{-2}}$	.19	∼96h
Ours	$\mathbf{11e^{-5}}$	$8.86e^{-3}$	.61	∼**4**h	$\mathbf{93e^{-6}}$	$1.17e^{-2}$	.83	∼**39**h	$\mathbf{12e^{-5}}$	$8.80e^{-3}$	.52	∼**18**h

auto-encoder to generate different colorizations. It is the basis for our method.

Probabilistic Image Colorization (PIC): The PIC model proposed by Royer *et al.* [162] uses a CNN network to learn an embedding of a gray-level images, which is then used as input for a PixelCNN network.

3.3.2 Architecture and Implementation Details

In this section, we discuss the architecture of the encoder, the MDN, the decoder, the structure encoder and the G-CRF layer. We introduce the following notation for describing the different architectures. We denote by $C_a(k, s, n)$ a convolutional layer with kernel size k, stride s, n output channels and activation a. We write B for a batch normalization layer, $U(f)$ for bilinear up-sampling with scale factor f and $FC(n)$ for fully connected layer with n output channels. To regularize, we use dropout for fully connected layers.

Encoder: The *encoder* network learns an approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{g})$ conditioned on the gray-level image representation and the color image. Its input is a colorfield of size $64 \times 64 \times 2$ and it generates an embedding $\boldsymbol{z}$ of size 64. The architecture is as follows: $64 \times 64 \times 2 \rightarrow C_{ReLU}(5, 2, 128) \rightarrow B \rightarrow C_{ReLU}(5, 2, 256) \rightarrow B \rightarrow C_{ReLU}(5, 2, 512) \rightarrow B \rightarrow C_{ReLU}(4, 2, 1024) \rightarrow B \rightarrow FC(64)$.

MDN: The *MDN*'s input is a $28 \times 28 \times 512$ gray-level feature embedding from [48]. It predicts the parameters of 8 Gaussian components, namely 8

Table 3.3: Average PSNR (dB) (higher is better) vs. number of revealed points ($|\boldsymbol{H}|$).

	Levin et al. [165]			Endo et al. [189]			Barron et al. [190]			Zhang et al. [175]			**Ours**		
$\|H\|$	10	50	100	10	50	100	10	50	100	10	50	100	10	50	100
PSNR	26.5	28.5	30	24.8	25.9	26	25.3	28	29	28	30.2	31.5	26.7	29.3	30.4

means of size 64, and 8 mixture weights. We use a fixed spherical variance σ equal to 0.1. The *MDN* network is constructed as follows: $28 \times 28 \times 512 \rightarrow C_{ReLU}(5, 1, 384) \rightarrow B \rightarrow C_{ReLU}(5, 1, 320) \rightarrow B \rightarrow C_{ReLU}(5, 1, 288) \rightarrow B \rightarrow C_{ReLU}(5, 2, 256) \rightarrow B \rightarrow C_{ReLU}(5, 1, 128) \rightarrow B \rightarrow FC(4096) \rightarrow FC(8 \times 64 + 8)$.

Decoder: During training, the *decoder*'s input is the embedding of size 64 generated by the *encoder*. At test time, the input is a 64-dimensional latent code sampled from one of the Gaussian components of the *MDN*. The *decoder* generates a vector $\boldsymbol{B}$ of unary terms of size $(32 \times 32 \times 2)$. It consists of operations of bilinear up-sampling and convolutions. During training, we learn four feature maps f_1, f_2, f_3 and f_4 of the gray-level image of sizes $4 \times 4 \times 1024$, $8 \times 8 \times 512$, $16 \times 16 \times 256$ and $32 \times 32 \times 128$, respectively. We concatenate these representations with the ones learned by the decoder, after every up-sampling operation, *i.e.*, f_1 is concatenated with the decoder's representation after the first up-sampling, f_2 after the second upsampling and so on. The architecture is described as follows: $1 \times 1 \times 64 \rightarrow U(4) \rightarrow C_{ReLU}(4, 1, 1024) \rightarrow B \rightarrow U(2) \rightarrow C_{ReLU}(5, 1, 512) \rightarrow B \rightarrow U(2) \rightarrow C_{ReLU}(5, 1, 256) \rightarrow B \rightarrow C_{ReLU}(5, 1, 128) \rightarrow B \rightarrow U(2) \rightarrow C_{ReLU}(5, 1, 64) \rightarrow B \rightarrow C_{ReLU}(5, 1, 2) \rightarrow (32 \times 32 \times 2)$.

Structure Encoder: The structure encoder learns a 512-dimensional embedding for every node in the down-sampled gray-level image of size $32 \times 32 \times 1$. It consists of stacked convolutional layers: $(32 \times 32 \times 1) \rightarrow C_{ReLU}(5, 1, 16) \rightarrow B \rightarrow C_{ReLU}(5, 1, 32) \rightarrow B \rightarrow C_{ReLU}(5, 1, 64) \rightarrow B \rightarrow C_{ReLU}(5, 1, 128) \rightarrow B \rightarrow C_{ReLU}(5, 1, 256) \rightarrow B \rightarrow C_{ReLU}(5, 1, 512) \rightarrow 512 \times (32 \times 32)$.

G-CRF Layer: The G-CRF is implemented as an additional layer that has as input the unary term $\boldsymbol{B}$ and the embedding matrix $\boldsymbol{\mathcal{A}}$ produced by the structure encoder. During the forward pass, it outputs a color field of size $32 \times 32 \times 2$ by solving the linear system in Equation 3.4 for the channels a and b separately. During the backward pass, it generates the gradient of the loss with respect to the unary term and the the embedding matrix respectively.

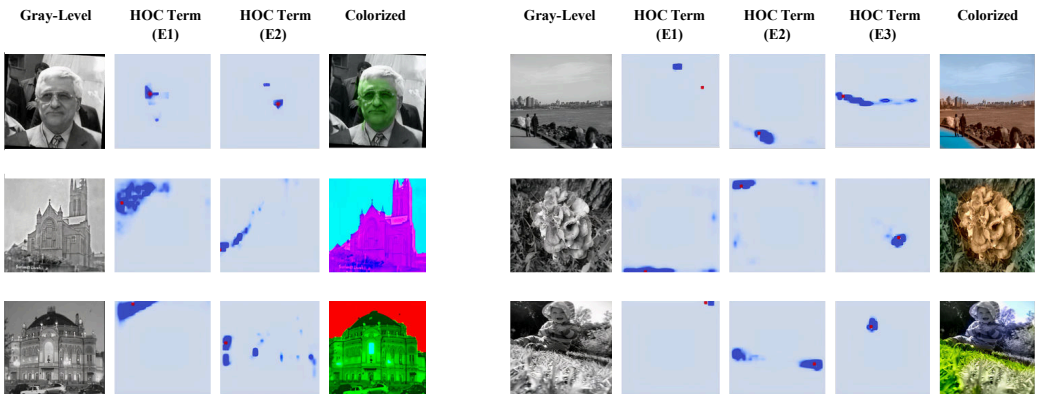


Figure 3.6: Controllability. Colorization from sparse user edits.

To compute the gradient of $\boldsymbol{B}$, we solve the linear system in Equation 3.10. The gradient of $\boldsymbol{A}$ is given in Equation 3.11.

3.3.3 Comparison with Baselines

We qualitatively compare the diversity and global spatial consistency of the colorizations obtained by our models with the ones generated by the aforementioned baselines, in Figures 3.1 and 3.5. We observe that our approach is the only one which generates a consistent colorization of the skin of the girl in Figure 3.1. We are also able to uniformly color the ground, the snake, and the actor's coat in Figure 3.5.

For global consistency evaluation, we perform a user study, presented in Table 3.1, where participants are asked to select the more realistic image from a pair of images at a time. We restrict the study to the three approaches with the overall lowest error-of-best (eob) per pixel reported in Table 3.2, namely VAE-MDN, PIC and our model. We use the clicking speed to filter out inattentive participants. Participants did neither know the paper content nor were the methods revealed to them. We gathered 5,374 votes from 271 unique users. The results show that users prefer colorizations obtained with the proposed approach.

To evaluate diversity, we use two metrics: (1) the variance of diverse colorizations and (2) the mean structural similarity *SSIM* [191] across all pairs of colorizations generated for one image. We report our results in Table 3.2.

Figure 3.7: Visualization of the unary term. The first row corresponds to the ground truth image. We visualize one possible colorization in the third row and its corresponding unary term in the second row.

3.3.4 Global Consistency

Our model noticeably outperforms all the baselines in producing spatially coherent results as demonstrated by the user study. PIC generates very diversified samples for the LFW and ILSVRC-2015 datasets but lacks long-range spatial dependencies because of the auto-regressive nature of the model. For example, the snake in the second row of Figure 3.5 has different colors for the head and the tail, and the woman's skin tone is inconsistent in Figure 3.1. The VAE-MDN, Bicycle GAN and MLN-GAN outputs are sometimes speckled and objects are not uniformly colored. For example, parts of the dome of the building in the second row of Figure 3.5 are confused to be part of the sky and the shirt in the third row is speckled. In contrast, our model is capable of capturing complex long-range dependencies. This is confirmed by the user study.

3.3.5 Diversity

Across all datasets, c-GAN suffers from mode collapse and is frequently unable to produce diverse colorizations. The PIC, MLN-GAN and Bicycle GAN models yield the most diverse results at the expense of photo-realism. Our model produces diverse results while ensuring long-range spatial consistency.

3.3.6 Controllability

For the controllably experiments, we set the β hyper-parameter to 1 during training and to 5 during testing. We opt for the following training schedule,

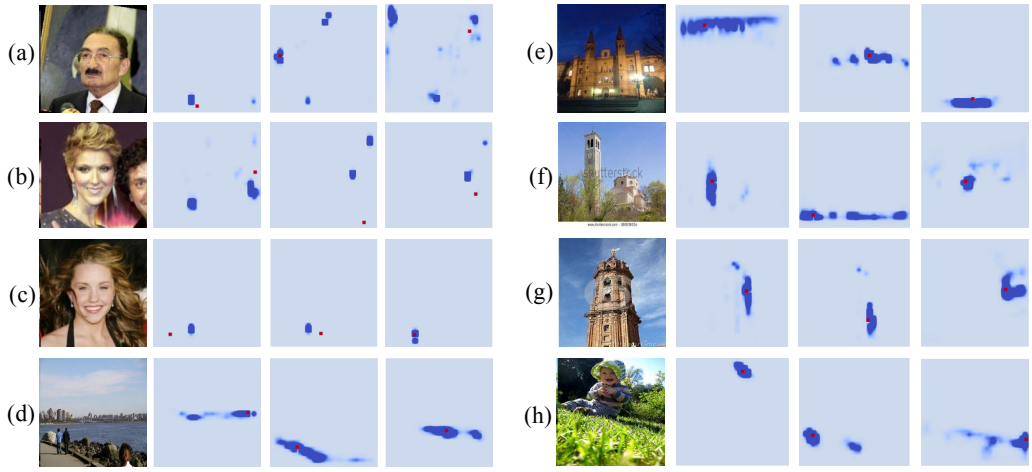


Figure 3.8: Visualization of the HOC term. For every example, we show the ground truth image and three HOC terms corresponding to three different pixels marked in red.

to force the model to encode the structure required to propagate sparse user inputs in the controllability experiments: We train the unary branch for 15 epochs (Stage1, Phase1), then train the HOC term for 15 epochs as well (Stage1, Phase2). We use the diagonal matrix $\boldsymbol{H}$ to randomly specify L pixels which colors are encoded by the unary branch $\boldsymbol{\alpha}$. We decrease L following a training schedule from 100% to 75%, 50%, 25% then 10% of the total number of pixels after respectively epochs 2, 4, 6, 8, 10 and 12. Note that additional stages could be added to the training schedule to accommodate for complex datasets where very sparse user input is desired. In Figure 3.6, we show that with two edits ($E1$ and $E2$), we colorize a face in green (Zhang *et al.* [175] use three edits) and the sky and the building in different colors. With three user edits ($E1$, $E2$ and $E3$), we show that we can colorize more complex images. We show the edits E using red markers. We visualize the attention weights per pixel, corresponding to the pixel's row in the similarity matrix $\boldsymbol{S}$, in blue, where darker shades correspond to stronger correlations.

Quantitatively, we report the average PSNR for 10, 50 and 100 edits on the ImageNet test set in Table 3.3, where edits (points) corresponding to randomly selected 7×7 patches are revealed to the algorithm. We observe that our method achieves slightly better results than the one proposed by Levin *et al.* [165] as our algorithm learns, for every pixel color, an "attention mechanism" over all the pixels in the image while Levin *et al.* impose local smoothness.

3.3.7 Visualization of the HOC and Unary Terms

In order to obtain more insights into the model's dynamics, we visualize the unary terms, $\boldsymbol{B}$, and the similarity matrix S, in respectively Figure 3.7 and Figure 3.8. As illustrated in Figure 3.8, the HOC term has learned complex long-range pixel affinities through end-to-end training. The results in Figure 3.7 further suggest that the unary term outputs a colorization with possibly some noise or inconsistencies that the HOC term fixes to ensure global coherency. For example, for the picture in the second column in Figure 3.7, the colors of the face, chest and shoulder predicted by the unary term are not consistent, and were fixed by the binary term which captured the long-range correlation as it is shown in Figure 3.8 (c).

We notice different interesting strategies for encoding the long-range correlations: On the LSUN-Church dataset, the model encourages local smoothness as every pixel seems to be strongly correlated to its neighbors. This is the case for the sky in Figure 3.8 (e). The model trained on the LFW dataset, however encoded long-range correlation. To ensure consistency over a large area, it chooses some reference pixels and correlates every pixel in the area, as can be seen in Figure 3.8 (c). We provide more results in Section B.

3.4 Conclusion

We proposed a Gaussian conditional random field based variational autoencoder formulation for colorization and illustrated its efficacy on a variety of benchmark datasets, outperforming existing methods. The developed approach goes beyond existing methods in that it does not only model the ambiguity which is inherent to the colorization task, but also takes into account structural consistency, via incorporating higher potentials. The formulation enables a controllability mechanism that allows integrating external constraints. Inference with these potentials is performed exactly, via solving a linear system of equations.

CHAPTER 4

LEARNING HEURISTICS FOR INFERENCE IN GENERAL GRAPHICAL MODELS USING REINFORCEMENT LEARNING

In Chapter 3, we introduced a Gaussian conditional random field modeling higher-order correlations between the output space variables, while ensuring exact inference. In this chapter, we use reinforcement learning to learn heuristics for inference in general energy-based models without imposing any constraints on the potentials types and orders. We assess our method on the task of semantic segmentation. We show that we can efficiently solve higher-order potentials that are untractable with traditional techniques. We present compelling results on MOTS multi-object tracking and segmentation dataset [192] (3 classes) and scale to Pascal VOC semantic segmentation dataset [193] (21 classes). We start by reviewing work on semantic segmentation.

4.1 Related Work

In early 2000, classifiers were locally applied to images to generate segmentations [197] which resulted in a noisy output. To address this concern, as early as 2004, He *et al.* [198] applied conditional random fields (CRFs) [199] and multi-layer perceptron features. For inference, Gibbs sampling was used, since MAP inference is NP-hard due to the combinatorial nature of the program. Progress in combinatorial optimization for flow-based problems in the 1990s and early 2000s [200, 201, 202, 203, 204, 205, 206, 207] showed that min-cut solvers can find the MAP solution of sub-modular energy functions of graphical models for binary segmentation. Approximation algorithms like

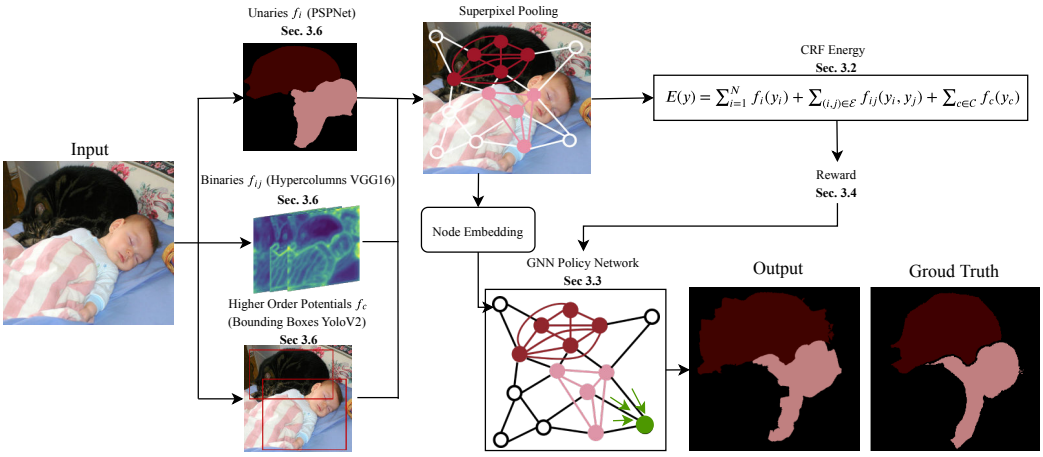


Figure 4.1: Pipeline of the proposed approach. Inference in a higher-order CRF is solved using reinforcement learning for the task of semantic segmentation. For Pascal VOC, unaries are obtained from PSPNet [194], pairwise potentials are computed using hypercolumns from VGG16 [99] and higher-order potentials are based on detection bounding boxes from YoloV2 [195]. The policy network is modeled as a graph embedding network [196] following the CRF graph structure. It sequentially produces the labeling of every node (superpixel).

swap-moves and α-expansion [206] were developed to extend applicability of min-cut solvers to more than two labels. Semantic segmentation was further popularized by combining random forests with CRFs [208]. Recently, the performance on standard semantic segmentation benchmarks like Pascal VOC 2012 [193] has been dramatically boosted by convolutional networks. Both deeper [209] and wider [210, 211, 212] network architectures have been proposed. Advances like spatial pyramid pooling [194] and atrous spatial pyramid pooling [213] emerged to remedy limited receptive fields. Other approaches jointly train deepnets with CRFs [214, 215, 216, 217, 218, 95, 219] to better capture the rich structure present in natural scenes.

4.2 Approach

We first present an overview of our approach before we discuss the individual components in greater detail.

4.2.1 Overview

Finding the optimal semantic segmentation configuration, *i.e.*, finding the minimizing argument of the energy, generally involves solving an NP-hard combinatorial optimization problem. Notable exceptions include energies with sub-modular co-occurrence terms. Instead of using classical directions, here, we assess suitability of learning based combinatorial optimization. Intuitively, we argue that CRF inference for the task of semantic segmentation exhibits an inherent similarity which can be exploited by learning based algorithms. We therefore first provide an overview of the developed approach, outlined in Figure 4.1. Just like classical approaches, we also use local evidence and co-occurrence information, obtained from deepnets. This information is consequently used to form an energy function defined over a conditional random field (CRF). An example of a CRF with variables corresponding to superpixels (circles), pairwise potentials (edges) and higher-order potentials obtained from object detections (fully connected cliques) is illustrated in Figure 4.1. However, different from classical methods, we find the minimizing configuration of the energy by repeatedly applying a learned policy network. In every iteration, the policy network selects a random variable, *i.e.*, the pixel and its label by computing a probability distribution over all currently unlabeled pixels and their labels. Specifically, the pixel and label are determined by choosing the highest scoring entry in a matrix where the number of rows and columns correspond to the currently unlabeled pixels and the available labels respectively, as illustrated in Figure 4.2. Differently, from the RL-based inference approach proposed in the previous chapter, where rewards are obtained after the entire solution (summary) is generated, we propose schemes for training with denser rewards, with the goal of improving the scalability and performance of the inference engine.

4.2.2 Problem Formulation

Formally, given an image $\boldsymbol{x}$, we are interested in predicting the semantic segmentation $\boldsymbol{y} = (y_1, \ldots, y_N) \in \boldsymbol{\mathcal{Y}}$. Hereby, N denotes the total number of pixels or superpixels, and the semantic segmentation of a superpixel $i \in \{1, \ldots, N\}$ is referred to via $y_i \in \boldsymbol{\mathcal{L}} = \{1, \ldots, |\boldsymbol{\mathcal{L}}|\}$, which can be assigned one out of $|\boldsymbol{\mathcal{L}}|$ possible discrete labels from the set of possible labels $\boldsymbol{\mathcal{L}}$.

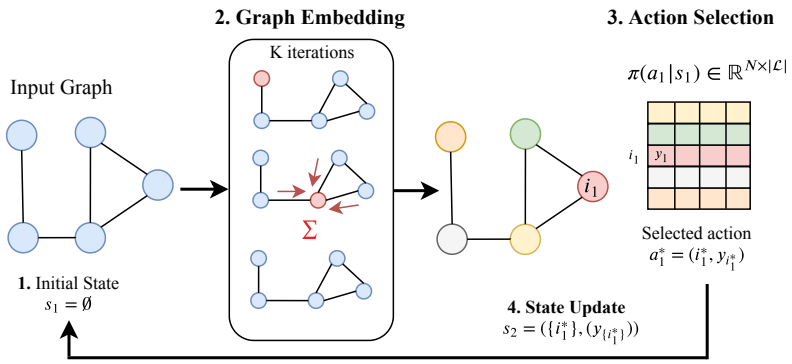


Figure 4.2: Illustration of one iteration of reinforcement learning for the inference task. The policy network samples an action $a_1 = (i_1^*, y_{i_1^*})$ from the learned distribution $\boldsymbol{\pi}(a_1|s_1) \in \mathbb{R}^{N\times|\boldsymbol{\mathcal{L}}|}$ at iteration $t = 1$.

Classical techniques obtain local evidence $f_i(y_i)$ for every pixel or superpixel, and co-occurrence information in the form of pairwise potentials $f_{ij}(y_i, y_j)$ and higher-order potentials $f_{\boldsymbol{c}}(\boldsymbol{y_c})$. The latter assigns an energy to a clique $\boldsymbol{c} \subseteq \{1, \ldots, N\}$ of variables $\boldsymbol{y_c} = (y_i)_{i\in\boldsymbol{c}}$. For readability, we drop the dependence of the energies f_i, f_{ij} and $f_{\boldsymbol{c}}$ on the image $\boldsymbol{x}$ and the parameters of the employed deepnets. The goal of energy-based semantic segmentation is to find the configuration $\boldsymbol{y}^*$ which has the lowest energy $E(\boldsymbol{y})$, *i.e.*,

$$\boldsymbol{y}^* = \arg\min_{\boldsymbol{y}\in\boldsymbol{\mathcal{Y}}} E(\boldsymbol{y}) \triangleq \sum_{i=1}^{N} f_i(y_i) + \sum_{(i,j)\in\boldsymbol{\mathcal{E}}} f_{ij}(y_i, y_j) + \sum_{\boldsymbol{c}\in\boldsymbol{\mathcal{C}}} f_{\boldsymbol{c}}(\boldsymbol{y_c}). \tag{4.1}$$

Hereby, the sets $\boldsymbol{\mathcal{E}}$ and $\boldsymbol{\mathcal{C}}$ subsume respectively the captured set of pairwise and higher-order co-occurrence patterns. Details about the potentials are presented in Section 4.2.6.

Solving the combinatorial program given in Equation 4.1, *i.e.*, inferring the optimal configuration $\boldsymbol{y}^*$ is generally NP-hard. Different from existing methods, we develop a learning based combinatorial optimization heuristic for semantic segmentation with the intention to better capture the intricacies of energy minimization than can be done by hand-crafting rules. The developed heuristic sequentially labels one variable y_i, $i \in \{1, \ldots, N\}$, at a time.

Formally, selection of one superpixel at a time can be formulated in a reinforcement learning context, as shown in Figure 4.2. Specifically, an agent operates in $t \in \{1, \ldots, N\}$ time-steps according to a policy $\boldsymbol{\pi}(a_t|\boldsymbol{s}_t)$ which

encodes a probability distribution over actions $a_t \in \boldsymbol{\mathcal{A}}_t$ given the current state $\boldsymbol{s}_t$. The current state subsumes in selection order the indices of all currently labeled variables $\boldsymbol{I}_t \subseteq \{1, \ldots, N\}$ as well as their labels $\boldsymbol{y}_{\boldsymbol{I}_t} = (y_i)_{i \in \boldsymbol{I}_t}$, *i.e.*, $\boldsymbol{s}_t \in \{(\boldsymbol{I}_t, \boldsymbol{y}_{\boldsymbol{I}_t}) : \boldsymbol{I}_t \subseteq \{1, \ldots, N\}, \boldsymbol{y}_{\boldsymbol{I}_t} \in \boldsymbol{\mathcal{L}}^{|\boldsymbol{I}_t|}\}$. We start with $\boldsymbol{s}_1 = \emptyset$. The set of possible actions $\boldsymbol{\mathcal{A}}_t$ is the concatenation of the label spaces $\boldsymbol{\mathcal{L}}$ for all currently unlabeled pixels $j \in \{1, \ldots, N\} \setminus \boldsymbol{I}_t$, *i.e.*, $\boldsymbol{\mathcal{A}}_t = \bigoplus_{j \in \{1,\ldots,N\} \setminus \boldsymbol{I}_t} \boldsymbol{\mathcal{L}}$. We emphasize the difference between the concatenation operator and the product operator used to obtain the semantic segmentation output space $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{L}}^N$, *i.e.*, the proposed approach does not operate in the product space.

As mentioned before, the policy $\boldsymbol{\pi}(a_t|\boldsymbol{s}_t)$ results in a probability distribution over actions $a_t \in \boldsymbol{\mathcal{A}}_t$ from which we greedily select the most probable action

$$a_t^* = \arg\max_{a_t \in \boldsymbol{\mathcal{A}}_t} \boldsymbol{\pi}(a_t|\boldsymbol{s}_t).$$

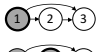
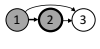
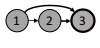
The most probable action a_t^* can be decomposed into the index for the selected variable, *i.e.*, i_t^* and its state $y_{i_t^*} \in \boldsymbol{\mathcal{L}}$. We obtain the subsequent state $\boldsymbol{s}_{t+1}$ by combining the extracted variable index i_t^* and its labeling with the previous state $\boldsymbol{s}_t$. Specifically, we obtain $\boldsymbol{s}_{t+1} = \boldsymbol{s}_t \oplus (i_t^*, y_{i_t^*})$ by slightly abusing the $\oplus$-operator to mean concatenation to a set and a list maintained within a state.

Algorithm 2: Inference Procedure

```
s_1 = ∅;
for t = 1 to N do
    a_t^* = arg max_{a_t ∈ A_t} π(a_t|s_t)
    (i_t^*, y_{i_t^*}) ← a_t^*
    s_{t+1} = s_t ⊕ (i_t^*, y_{i_t^*})
Return: ŷ ← s_{N+1}
```

Formally, we summarize the developed reinforcement learning based semantic segmentation algorithm used for inferring a labeling $\hat{\boldsymbol{y}}$ in Algorithm 6. In the following, we describe the policy function $\boldsymbol{\pi}_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t)$, which we found to work well for semantic segmentation, and different variants to learn its parameters $\boldsymbol{\theta}$.

Table 4.1: Illustration of the energy reward computation following the two proposed reward schemes on a fully connected graph with 3 nodes.

t	i_t	E_t	$r_t = -(E_t - E_{t-1})$	$r_t = \pm 1$	Graph
0	–	0	–	–	
1	1	$f_1(y_1)$	$-f_1(y_1)$	$-1+2\cdot\mathbb{1}_{\{(E_t(y_1)<E_t(\hat{y}_1))\forall\hat{y}_1\}}$	
2	2	$f_1(y_1)+f_2(y_2)$ $+f_{12}(y_1,y_2)$	$-f_2(y_2)-f_{12}(y_1,y_2)$	$-1+2\cdot\mathbb{1}_{\{(E_t(y_1,y_2)<E_t(y_1,\hat{y}_2))\forall\hat{y}_2\}}$	
3	3	$f_1(y_1)+f_2(y_2)+f_3(y_3)$ $+f_{12}(y_1,y_2)+f_{23}(y_2,y_3)$ $+f_{13}(y_1,y_3)+f_{13}(y_1,y_3)$ $+f_{\{1,2,3\}}(y_1,y_2,y_3)$	$-f_3(y_3)-f_{23}(y_2,y_3)$ $-f_{13}(y_1,y_3)$ $-f_{\{1,2,3\}}(y_1,y_2,y_3)$	$-1+2\cdot\mathbb{1}_{\{(E_t(y_1,y_2,y_3)<E_t(y_1,y_2,\hat{y}_3))\forall\hat{y}_3\}}$	

4.2.3 Policy Function

We model the policy function $\boldsymbol{\pi_\theta}(a_t|\boldsymbol{s}_t)$ using a graph embedding network [196]. The input to the network is a weighted graph $\boldsymbol{\mathcal{G}}(\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}, w)$, where nodes $\boldsymbol{\mathcal{V}} = \{1, \ldots, N\}$, correspond to variables, *i.e.*, in our case superpixels, $\boldsymbol{\mathcal{E}}$ is a set of edges connecting neighboring superpixels, as illustrated in Figure 4.1 and $w : \boldsymbol{\mathcal{E}} \to \mathbb{R}^+$ is the edge weight function. The weights $\{w(i,j)\}_{\{j:(i,j)\in\boldsymbol{\mathcal{E}}\}}$ on the edges between a given node i and its neighbors $\{j : (i,j) \in \boldsymbol{\mathcal{E}}\}$ form a distribution, obtained by normalizing the dot product between the hypercolumns [99] $\boldsymbol{g}_i$ and $\boldsymbol{g}_j$ via a softmax across neighbors. At every iteration, the state $\boldsymbol{s}_t$ is encoded in the graph $\boldsymbol{\mathcal{G}}$ by tagging node $i \in \boldsymbol{\mathcal{V}}$ with a scalar $h_i = 1$ if the node is part of the already labeled set $\boldsymbol{I}_t$, *i.e.*, if $i \in \boldsymbol{I}_t$ and 0 otherwise. Moreover, a one-hot encoding $\tilde{\boldsymbol{y}}_i \in \{0,1\}^{|\boldsymbol{\mathcal{L}}|}$ encodes the selected label of nodes $i \in \boldsymbol{I}_t$. We set $\tilde{\boldsymbol{y}}_i$ to equal the all zeros vector if node i has not been selected yet.

Every node $i \in \boldsymbol{\mathcal{V}}$ is represented by a p-dimensional embedding, where p is a hyperparameter. The embedding is composed of $\tilde{\boldsymbol{y}}_i$, h_i as well as superpixel features $\boldsymbol{b}_i \in \mathbb{R}^F$ which encode appearance and bounding box characteristics that we discuss in detail in Section 4.3.

The output of the network is a $|\boldsymbol{\mathcal{L}}|$-dimensional vector $\boldsymbol{\pi}_i$ for each node $i \in V$, representing the scores of the $|\boldsymbol{\mathcal{L}}|$ different labels for variable i.

The network iteratively generates a new representation $\boldsymbol{\mu}_i^{(k+1)}$ for every node $i \in \boldsymbol{\mathcal{V}}$ by aggregating the current embeddings $\boldsymbol{\mu}_i^{(k)}$ according to the graph structure $\boldsymbol{\mathcal{E}}$ starting from $\boldsymbol{\mu}_i^{(0)} = \boldsymbol{0}$, $\forall i \in \boldsymbol{\mathcal{V}}$. After K steps, the embedding captures long range interactions between the graph features as well as the graph properties necessary to minimize the energy function E.

Formally, the update rule for node i is

$$\boldsymbol{\mu}_i^{(k+1)} \leftarrow \text{Relu}\left(\boldsymbol{\theta}_1^{(k)} h_i + \boldsymbol{\theta}_2^{(k)} \tilde{\boldsymbol{y}}_i + \boldsymbol{\theta}_3^{(k)} \boldsymbol{b}_i + \boldsymbol{\theta}_4^{(k)} \sum_{j:(i,j)\in\boldsymbol{\mathcal{E}}} w(i,j)\boldsymbol{\mu}_j^{(k)}\right), \tag{4.2}$$

where $\boldsymbol{\theta}_1^{(k)} \in \mathbb{R}^p$, $\boldsymbol{\theta}_2^{(k)} \in \mathbb{R}^{p\times|\mathcal{L}|}$, $\boldsymbol{\theta}_3^{(k)} \in \mathbb{R}^{p\times F}$ and $\boldsymbol{\theta}_4^{(k)} \in \mathbb{R}^{p\times p}$ are trainable parameters. After K steps, $\boldsymbol{\pi}_i$ for every unlabeled node $i \in \{1,\dots,N\} \setminus \boldsymbol{I}_t$ is obtained via

$$\boldsymbol{\pi}_i = \boldsymbol{\theta}_5 \boldsymbol{\mu}_i^{(K)} \quad \forall i \in \{1,\dots,N\} \setminus \boldsymbol{I}_t, \tag{4.3}$$

where $\boldsymbol{\theta}_5 \in \mathbb{R}^{|\mathcal{L}|\times p}$ is another trainable model parameter. We illustrate the policy function $\boldsymbol{\pi}_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t)$ and one iteration of inference in Figure 4.2.

4.2.4 Reward Function

At step t, we define the reward as the difference between the value of the negative new energy E_t and the negative energy from the previous step E_{t-1}, *i.e.*, $r_t(\boldsymbol{s}_t, a_t) = E_{t-1}(\boldsymbol{y}_{\boldsymbol{I}_{t-1}}) - E_t(\boldsymbol{y}_{\boldsymbol{I}_t})$, where $E_0 = 0$. Note that, with this design choice for the reward function $r_t(\boldsymbol{s}_t, a_t)$, the cumulative reward coincides exactly with the objective function that we aim at maximizing, *i.e.*, $\sum_{t=1}^{N} r_t(\boldsymbol{s}_t, a_t) = -E(\hat{\boldsymbol{y}})$, where $\hat{\boldsymbol{y}}$ is extracted from $\boldsymbol{s}_{N+1}$. Potentials depending on variables that are not labeled at time t are not incorporated in the evaluation of $E_t(\boldsymbol{y}_{\boldsymbol{I}_t})$.

We also study a second scheme, where the reward is truncated to $+1$ or -1, *i.e.*, $r_t(\boldsymbol{s}_t, a_t) \in \{-1, 1\}$. For every selected node i_t, with label y_{i_t}, we compare the energy function $E_t(\boldsymbol{y}_{\boldsymbol{I}_t})$ with the one obtained when using all other labels $\hat{y}_{i_t} \in \boldsymbol{\mathcal{L}} \setminus y_{i_t}$. If the chosen label y_{i_t} results in the lowest energy, the obtained reward is $+1$, otherwise it is -1.

Note that the unary potentials result in a reward for every time step. Pairwise and high-order potentials result in a sparse reward as their value is only available once all the superpixels forming the pair or clique are labeled. We illustrate the energy and reward computation on a graph with three fully connected nodes in Table 4.1.

4.2.5 Learning Policy Parameters

To learn the parameters $\boldsymbol{\theta}$ of the policy function $\boldsymbol{\pi}_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t)$, a plethora of reinforcement learning algorithms are applicable. To provide a careful assessment of the developed approach, we study two different techniques, Q-learning and Monte-Carlo Tree Search, both of which we describe next.

Q-learning: In the context of Q-learning, we interpret the $|\boldsymbol{\mathcal{L}}|$-dimensional policy network output vector corresponding to a currently unlabeled node $i \in \{1, \ldots, N\} \setminus \boldsymbol{I}_t$ as the Q-values $Q(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$ associated to the action a_t of selecting node i and assigning label $y_i \in \boldsymbol{\mathcal{L}}$. Since we only consider actions to label currently unlabeled nodes we obtain a total of $|\boldsymbol{\mathcal{A}}_t|$ different Q-values.

We perform standard Q-learning and minimize the $L2$-loss $(z - Q(\boldsymbol{s}_t, a_t; \boldsymbol{\theta}))^2$, where we use target $z = \gamma \max_{a'} Q(\boldsymbol{s}_{t+1}, a'; \boldsymbol{\theta}) + r_t(\boldsymbol{s}_t, a_t)$ for a non-terminal state. The reward is denoted r_t and detailed above. The terminal state is reached when all the nodes are labeled.

Instead of updating the Q-function based on the current sample $(\boldsymbol{s}_t, a_t, r_t(\boldsymbol{s}_t, a_t), \boldsymbol{s}_{t+1})$, we use a replay memory populated with samples (graphs) from previous episodes. In every iteration, we select a batch of samples and perform stochastic gradient descent on the squared loss.

During the exploration phase, beyond random actions, we encourage the following three different sets of actions to generate more informative rewards for training: (1) $\boldsymbol{\mathcal{M}}_1^{(t)}$: Nodes that are adjacent to the already selected ones in the graph. Otherwise, the reward will only be based on the unary terms as the pairwise ones are only evaluated if the neighbors are labeled ($t = 2$ in Table 4.1). We assign a score $M_1(\boldsymbol{s}_t, a_t)$ to every available action $a_t = (i_t, y_{i_t}) \in \boldsymbol{\mathcal{A}}_t$ to encourage the exploration of the set $\boldsymbol{\mathcal{M}}_1^{(t)}$:

$$M_1(\boldsymbol{s}_t, a_t) = \frac{|\{j : (j \notin \boldsymbol{I}_t) \text{ and } (i_t, j) \in \boldsymbol{\mathcal{E}}\}|}{|\{j : (i_t, j) \in \boldsymbol{\mathcal{E}}\}|}. \tag{4.4}$$

(2) $\boldsymbol{\mathcal{M}}_2^{(t)}$: Nodes with the lowest unary distribution entropy at iteration t. The low entropy indicates a high confidence of the unary deepnet. Hence, the labels of the corresponding nodes are more likely to be correct and provide useful information to neighbors with higher entropy in the upcoming iterations. We assign a score $M_2(\boldsymbol{s}_t, a_t)$ to every available action $a_t = (i_t, y_{i_t}) \in \boldsymbol{\mathcal{A}}_t$

to encourage the exploration of the set $\boldsymbol{\mathcal{M}}_2^{(t)}$:

$$M_2(\boldsymbol{s}_t, a_t) = \frac{\exp\left(-S_{i_t}\right)}{\sum_{j \in \{1,\ldots,N\} \setminus \boldsymbol{I}_t} \exp\left(-S_j\right)}. \tag{4.5}$$

Here, S_{i_t} denotes the entropy of the unary distribution evaluated at node i_t. (3) $\boldsymbol{\mathcal{M}}_3^{(t)}$: Nodes with the same label as neighbouring nodes belonging to the same higher-order potential at iteration t, *i.e.*,

$$M_3(\boldsymbol{s}_t, a_t) = \begin{cases} 1 & \text{if } y_{i_t} = \underset{k \in \boldsymbol{\mathcal{L}}}{\operatorname{argmax}} \sum\limits_{\{j : j \in \boldsymbol{I}_t \text{ and } (i_t, j) \in \boldsymbol{\mathcal{C}}\}} \mathbb{1}_{\{y_j = k\}} \\ 0 & \text{otherwise} \end{cases}. \tag{4.6}$$

Hence, at train time, the next action a_t is selected as follows:

$$a_t^* = \begin{cases} \underset{a_t \in \boldsymbol{\mathcal{A}}_t}{\operatorname{argmax}}\, Q(\boldsymbol{s}_t, a_t; \boldsymbol{\theta}) & \text{with probability } \epsilon \\ \underset{a_t \in \boldsymbol{\mathcal{A}}_t}{\operatorname{argmax}}\, M_1(\boldsymbol{s}_t, a_t) & \text{with probability } (1-\epsilon)/4 \\ \underset{a_t \in \boldsymbol{\mathcal{A}}_t}{\operatorname{argmax}}\, M_2(\boldsymbol{s}_t, a_t) & \text{with probability } (1-\epsilon)/4 \\ \underset{a_t \in \boldsymbol{\mathcal{A}}_t}{\operatorname{argmax}}\, M_3(\boldsymbol{s}_t, a_t) & \text{with probability } (1-\epsilon)/4 \\ \text{Random} & \text{with probability } (1-\epsilon)/4 \end{cases}. \tag{4.7}$$

Here ϵ is a fixed probability modeling the exploration-exploitation tradeoff. At test time, $a_t^* = \underset{a_t \in \boldsymbol{\mathcal{A}}_t}{\operatorname{argmax}}\, Q(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$.

Monte-Carlo Tree Search: While DQN tries to learn a policy from looking at samples representing one action at a time, MCTS has the inherent ability to update its policy after looking multiple steps ahead via a tree search procedure. At training time, through extensive simulations, MCTS builds a *statistics tree*, reflecting an empirical distribution $\boldsymbol{\pi}^{\text{MCTS}}(a_t|\boldsymbol{s}_t)$. Specifically, for a given image, a node in the search tree corresponds to the state $\boldsymbol{s}_t$ in our formulation and an edge corresponds to a possible action a_t. The root node is initialized to $\boldsymbol{s}_1 = \emptyset$. The statistics stored at every node correspond to (1) $N(\boldsymbol{s}_t)$: the number of times state $\boldsymbol{s}_t$ has been reached, (2) $N(a_t|\boldsymbol{s}_t)$: the number of times action a_t was chosen in state $\boldsymbol{s}_t$ in all previous simulations, as well as (3) $\tilde{r}_t(\boldsymbol{s}_t, a_t)$: the averaged reward across all simulations starting at $\boldsymbol{s}_t$ and taking action a_t. The MCTS policy is defined as $\boldsymbol{\pi}^{\text{MCTS}}(a_t|\boldsymbol{s}_t) = \frac{N(a_t|\boldsymbol{s}_t)}{N(\boldsymbol{s}_t)}$.

A simulation involves three steps: 1) selection, 2) expansion and 3) value backup. *Selection* corresponds to choosing the next action given the current node $\boldsymbol{s}_t$, based on four factors: 1) a variant of the probabilistic upper confidence bound (PUCB) [220] given by $U(\boldsymbol{s}_t, a_t; \boldsymbol{\theta}) = \frac{\tilde{r}(\boldsymbol{s}_t, a_t)}{N(a_t|\boldsymbol{s}_t)} + \boldsymbol{\pi}_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t) \frac{\sqrt{N(\boldsymbol{s}_t)}}{1+N(a_t|\boldsymbol{s}_t)}$, 2) $\boldsymbol{\mathcal{M}}_1^{(t)}$ 3) $\boldsymbol{\mathcal{M}}_2^{(t)}$ and 4) $\boldsymbol{\mathcal{M}}_3^{(t)}$ similarly to the DQN case. Formally,

$$a_t^* = \underset{a_t \in \boldsymbol{\mathcal{A}}_t}{\operatorname{argmax}} \begin{cases} U(\boldsymbol{s}_t, a_t; \boldsymbol{\theta}) + M_1(\boldsymbol{s}_t, a_t) & \text{with probability } \frac{1}{3} \\ U(\boldsymbol{s}_t, a_t; \boldsymbol{\theta}) + M_2(\boldsymbol{s}_t, a_t) & \text{with probability } \frac{1}{3} \\ U(\boldsymbol{s}_t, a_t; \boldsymbol{\theta}) + M_3(\boldsymbol{s}_t, a_t) & \text{with probability } \frac{1}{3} \end{cases}. \tag{4.8}$$

Expansion consists of constructing a child node for every possible action from the parent node $\boldsymbol{s}_t$. The possible actions include the nodes which have not been labeled. The child nodes' cumulative rewards and counts are initialized to 0. Note that selection and expansion are limited to a depth d_{sim} starting from the root node in a simulation. *Value backup* refers to back-propagating the reward from the current node on the path to the root of the sub-tree. The visit counts of all the nodes in the path are incremented as well. Once n_{sim} simulations are completed, we compute $\boldsymbol{\pi}^{\text{MCTS}}$ for every node. The next action a_t from the root node is decided according to $\boldsymbol{\pi}^{\text{MCTS}}(a_t|\boldsymbol{s}_t)$: $a_t \sim \boldsymbol{\pi}^{\text{MCTS}}(a_t|\boldsymbol{s}_t)$ at train time and $a_t = \text{argmax}_{a_t \in \boldsymbol{\mathcal{A}}_t} \boldsymbol{\pi}^{\text{MCTS}}(a_t|\boldsymbol{s}_t)$ at inference. The next node becomes the root of the sub-tree. The experience ($\boldsymbol{s}_t$, $\boldsymbol{\pi}^{\text{MCTS}}$) is stored in the replay buffer. The whole process is repeated until all N nodes in the graph $\mathcal{G}$ are labeled. The policy network is then trained through a cross entropy loss *i.e.*,

$$L(\boldsymbol{\theta}) = -\sum_{\boldsymbol{s}} \sum_{a} \boldsymbol{\pi}^{\text{MCTS}}(a|\boldsymbol{s}) \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(a|\boldsymbol{s}), \tag{4.9}$$

to match the empirically constructed distribution. Here, the second sum is over all valid actions from a state s sampled from the replay buffer. We summarize the MCTS training procedure below in Algorithm 3. Note that we run 10 episodes per graph during training, but for simplicity we present the training for a single episode per graph.

The replay-memory for both DQN and MCTS is divided into two blocks. The first block corresponds to the unary potential, while the second block corresponds to the overall energy function. A node is assigned to the sec-

Algorithm 3: Monte Carlo Tree Search Training

input : Head node: $\boldsymbol{s}_1$, n_{sim}: number of simulations, d_{sim} : depth of simulations

output: A labeling $\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}$ for all the nodes $\boldsymbol{\mathcal{V}}$

```
// Looping over the graphs from the dataset
for all 𝒢(𝒱, ℰ, w) do
    // Initialization
    s_1 = ∅
    r̃(s_i, a) = 0, ∀s_i, i ∈ {1, ..., N}, ∀a
    // Looping over graph nodes 𝒱
    for t = 1 to N do
        // Running simulations
        for n = 1 to n_sim do
            // Create and expand a sub-tree
            for j = t to t + d_sim do
                Select a_j according to Equation 4.8 and advance
                 temporary state in sub-tree.
            Backup rewards along the visited nodes in the simulations.
            Update node visit counts.
        Compute tree policy π^MCTS with visit counts.
        Select the next action a_t ~ π^MCTS(a_t|s_t).
        Update the root node s_{t+1} ← s_t ⊕ a_t.
        Store (s_t, π^MCTS) in Replay Buffer.
    Sample M examples from Replay Buffer to update neural
     network parameters using Equation 4.9.
```

ond block if its associated reward is higher than the one obtained from its unary labeling. This ensures positive rewards from all the potentials during training. Every block is further divided into $|\boldsymbol{\mathcal{L}}|$ categories corresponding to the $|\boldsymbol{\mathcal{L}}|$ classes of the selected node. This guarantees a balanced sampling of the label classes in every batch during training. Beyond DQN and MCTS, we experimented with policy gradients but could not get it to work as it is an on-policy algorithm. Reusing experiences for the structured replay buffer was crucial for success of the learning algorithm.

4.2.6 Energy Function

Finally we provide details on the energy function E given in Equation 4.1. The unary potentials $f_i(y_i) \in \mathbb{R}^{|\boldsymbol{\mathcal{L}}|}$ are obtained from a semantic segmentation

deepnet. The pairwise potential encodes smoothness and is computed as follows:

$$f_{i,j}(y_i, y_j) = \psi(y_i, y_j) \cdot \alpha_p \cdot \mathbb{1}_{|\boldsymbol{g}_i^T \boldsymbol{g}_j| < \beta_p}, \tag{4.10}$$

where $\psi(y_i, y_j)$ is the label compatibility function describing the co-occurrence of two classes in adjacent locations and is given by the Potts model:

$$\psi(y_i, y_j) = \begin{cases} 1 & \text{if } y_i \neq y_j \\ 0 & \text{otherwise} \end{cases}. \tag{4.11}$$

Moreover, $|\boldsymbol{g}_i^T \boldsymbol{g}_j|$ is the above defined unnormalized weight $w(i, j)$ for the edge connecting the i^{th} and j^{th} nodes, *i.e.*, superpixels. Intuitively, if the dot product between the hypercolumns $\boldsymbol{g}_i$ and $\boldsymbol{g}_j$ is smaller than a threshold β_p and the two superpixels are labeled differently, a penalty of value α_p incurs.

While the pairwise term mitigates boundary errors, we address recognition errors with two detection-based higher-order potentials [221]. For this purpose, we use the YoloV2 bounding box object detector [195] as it ensures a good trade-off between speed and accuracy. Every bounding box b is presented by a tuple $(l_b, c_b, \boldsymbol{I}_b)$, where l_b is the class label of the detected object, c_b is the confidence score of the detection and $\boldsymbol{I}_b \subseteq \{1, \ldots, N\}$ is the set of superpixels that belong to the foreground detection obtained via Grab-Cut [17].

The first higher-order potential (HOP1) encourages superpixels within a bounding box to take the bounding box label, while enabling recovery from false detections that do not agree with other energy types. For this purpose, we add an auxiliary variable z_b for every bounding box b. We use $z_b = 1$, if the bounding box is inferred to be valid, otherwise $z_b = 0$. Formally,

$$f(y_{\boldsymbol{I}_b}, z_b) = \begin{cases} w_b \cdot c_b \cdot \sum_{i \in \boldsymbol{I}_b} \mathbb{1}_{y_i = l_b} & \text{if } z_b = 0 \\ w_b \cdot c_b \cdot \sum_{i \in \boldsymbol{I}_b} \mathbb{1}_{y_i \neq l_b} & \text{if } z_b = 1 \end{cases}, \tag{4.12}$$

where, $w_b \in \mathbb{R}$ is a weight parameter. This potential can be simplified into a sum of pairwise potentials between z_b and each y_i with $i \in \boldsymbol{I}_b$, *i.e.*, $f(\boldsymbol{y}_{\boldsymbol{I}_b}, z_b) =$

Table 4.2: Performance results for the minimizing the energy function E_t under reward scheme 1 ($R_t^1 = -(E_t - E_{t-1})$) and reward scheme 2 ($R_t^2 = \pm 1$).

	Nodes	Metrics	Supervised	Unary					Unary + Pairwise								
				BP	R_t^1		R_t^2		BP	TBP	DD	L-Flip	α-Exp	R_t^1		R_t^2	
					DQN	MCTS	DQN	MCTS						DQN	MCTS	DQN	MCTS
Pascal VOC	50	IoU (sp)	85.21	**88.59**	88.04	88.19	**88.59**	**88.59**	**88.73**	**88.73**	**88.73**	**88.73**	88.72	43.31	66.51	87.91	**88.73**
		IoU (p)	69.05	**72.56**	70.77	71.99	**72.56**	**72.56**	**72.43**	**72.43**	**72.43**	**72.43**	**72.43**	38.54	38.75	72.16	**72.43**
	250	IoU (sp)	83.54	**88.01**	87.29	**88.01**	**88.01**	**88.01**	88.10	88.10	88.10	88.10	88.10	88.06	88.22	**88.56**	88.52
		IoU (p)	75.88	80.64	80.47	**80.64**	**80.64**	**80.64**	80.68	80.68	80.68	80.68	80.68	80.54	80.86	**80.84**	80.75
	500	IoU (sp)	84.91	**87.39**	87.34	**87.39**	**87.39**	**87.39**	87.55	87.55	87.56	87.55	87.55	82.23	83.67	87.80	**87.84**
		IoU (p)	77.93	**82.35**	82.20	**82.35**	**82.35**	**82.35**	82.48	82.48	82.48	82.47	82.47	77.36	79.14	82.64	**82.70**
MOTS	2000	IoU (sp)	82.49	82.64	80.98	82.64	82.64	82.64	82.64	82.64	82.64	82.64	82.64	80.39	82.64	82.65	**82.64**
		IoU (p)	79.01	79.23	73.85	79.82	79.85	79.85	79.86	79.86	79.86	79.86	79.86	78.08	78.85	79.88	**79.86**

$\sum_{i \in \boldsymbol{I}_b} f_{i,b}(y_i, z_b)$, where:

$$f_{i,b}(y_i, z_b) = \begin{cases} w_b \cdot c_b \cdot \mathbb{1}_{y_i = l_b} & \text{if } z_b = 0 \\ w_b \cdot c_b \cdot \mathbb{1}_{y_i \neq l_b} & \text{if } z_b = 1 \end{cases} . \tag{4.13}$$

This simplification enables solving the higher-order potential using traditional techniques like mean field inference [221].

To show the merit of the RL framework, we introduce another higher-order potential (HOP2) that can not be seamlessly reduced to a pairwise one:

$$f(\boldsymbol{y}_{\boldsymbol{I}_b}) = \lambda_b \cdot \mathbb{1}_{(\sum_{i \in \boldsymbol{I}_b} y_i = l) < \frac{|\boldsymbol{I}_b|}{C}}, \tag{4.14}$$

with λ_b and C being scalar parameters. This potential is evaluated for bounding boxes with special characteristics to encourage the superpixels in the bounding box to be of label l. Intuitively, if the number of superpixels $i \in \boldsymbol{I}_b$ having label l is less than a threshold $\frac{|\boldsymbol{I}_b|}{C}$, a penalty λ_b incurs. For Pascal VOC, we evaluate the potential on bounding boxes b included in larger bounding boxes, as we noticed that the unaries frequently miss small objects overlapping with other larger objects in the image ($l = l_b$). For MOTS, we evaluate this potential on bounding boxes of type "pedestrians" overlapping with bounding boxes of type "bicycle". As cyclists should not be labeled as pedestrians, we set l to be the background class. Transforming this term into a pairwise one to enable using traditional inference techniques requires an exponential number of auxiliary variables.

Table 4.3: Performance results for the minimizing the energy function E_t under reward scheme 1 ($R_t^1 = -(E_t - E_{t-1})$) and reward scheme 2 ($R_t^2 = \pm 1$).

	Nodes	Metrics	Unary + Pairwise + HOP1									Unary + Pairwise + HOP1+ HOP2			
			BP	TBP	DD	L-Flip	α-Exp	R_t^1		R_t^2		R_t^1		R_t^2	
								DQN	MCTS	DQN	MCTS	DQN	MCTS	DQN	MCTS
Pascal VOC	50	IoU (sp)	89.26	89.27	89.27	89.27	88.58	57.43	73.37	89.55	**89.66**	58.34	73.85	90.05	**90.09**
		IoU (p)	72.59	72.59	72.59	72.60	72.35	51.88	53.71	72.83	**72.85**	50.53	51.71	72.94	**72.95**
	250	IoU (sp)	88.54	88.53	88.55	88.54	88.07	60.60	64.82	**88.94**	88.91	82.19	81.89	89.30	**89.57**
		IoU (p)	80.91	80.91	80.93	80.91	80.65	57.36	59.73	**81.07**	81.05	74.94	74.77	81.23	**81.33**
	500	IoU (sp)	87.95	87.96	87.96	87.95	87.54	37.80	57.66	**88.73**	88.69	43.99	45.67	**88.43**	88.21
		IoU (p)	82.72	82.72	82.72	82.71	82.47	36.65	52.73	**83.05**	82.95	41.74	42.44	**82.79**	82.67
MOTS	2000	IoU (sp)	83.17	83.17	83.17	83.17	83.16	83.14	83.30	83.27	**83.28**	83.13	83.19	**83.29**	**83.29**
		IoU (p)	81.21	81.21	81.21	81.21	81.17	80.61	81.92	82.68	**82.69**	80.61	80.63	**82.77**	**82.77**

4.3 Experiments

In the following, we evaluate our learning based inference algorithm on Pascal VOC [193] and MOTS [193] datasets. The original Pascal VOC dataset contains 1464 training and 1449 validation images. In addition to this data, we make use of the annotations provided by [222], resulting in a total of 10582 training instances. The number of classes is 21. MOTS is a multi-object tracking and segmentation dataset for cars and pedestrians (2 classes). It consists of 12 training sequences (5027 frames) and 9 validation ones (2981 frames). In this work, we perform semantic segmentation at the level of superpixels, generated using SLIC [223]. Every superpixel corresponds to a node i in the graph as illustrated in Figure 4.1. The unary potentials at the pixel level are obtained from PSPNet [194] for Pascal VOC and TrackR-CNN [192] for MOTS. The superpixels' unaries are the average of the unaries of all the pixels that belong to that superpixel. The higher-order potential is based on the YoloV2 [195] bounding box detector.

PSPNet, TrackR-CNN and the hypercolumns from VGGNet are not fine-tuned. Only the graph policy net is trained. For MOTS, we apply our model to every frame of the video. For MCTS, we set the number of simulations during exploration to 50 and the simulation depth to 4. At test time, we run 20 simulations with a depth of 4. The models are trained for 10 epochs, equivalent to around $375,000$ training iterations for Pascal VOC and $188,000$ for MOTS. The parameters of the energy function, α_p, β_p, w_b, c_b, λ_b and C, are obtained via a grid search on a subset of 500 nodes from the training data. The number of iterations K of the graph neural network is set to

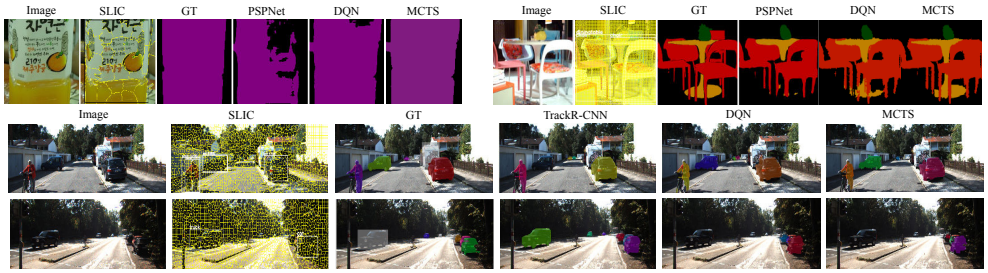


Figure 4.3: Success cases.

3. The dimension F of the node features $\boldsymbol{b}_i$ equals 85 for Pascal VOC and 30 for MOTS, consisting of the unary distribution, the unary distribution entropy, and features of the bounding box. The bounding box features are the confidence and label of the bounding box, its unary composition at the pixel level, percentage of overlap with other bounding boxes and their associated labels and confidence. The embedding dimension p is 32 for Pascal VOC and 16 for MOTS. As an optimizer, we use Adam with a learning rate of 0.001.

Evaluation Metrics: As evaluation metrics, we use intersection over union (IoU) computed at the level of both superpixels (sp) and pixels (p). IoU (p) is obtained after mapping the superpixel level labels to the corresponding set of pixels.

Baselines: We compare our results to the segmentations obtained by five different solvers from three main categories: (1) message passing algorithms, *i.e.*, Belief propagation (BP) [224] and Tree-reweighted Belief Propagation (TBP) [225], (2) a Lagrangian relaxation method, *i.e.*, Dual Decomposition Subgradient (DD) [42], and (3) move making algorithms, *i.e.*, Lazy Flipper (LFlip) [226] and α-expansion as implemented in [227]. Note that these solvers cannot optimize our HOP2 potential. Besides, we train a supervised model that predicts the node label from the provided node features.

Performance Evaluation: We show the results of solving the program given in Equation 4.1 in Table 4.2 and Table 4.3, for unary (Col. 5), unary plus pairwise (Col. 6), unary plus pairwise plus HOP1 (Col. 7) and unary plus pairwise plus HOP1 and HOP2 (Col. 8) potentials. For every potential type, we report results on graphs with superpixel numbers 50, 250 and 500 for Pascal VOC and 2000 for MOTS, obtained from DQN and MCTS, trained each with the two reward schemes discussed in Section 4.2.5. Since MOTS has small objects, we opt for a higher number of superpixels. It

is remarkable to observe that DQN and MCTS are able to learn heuristics which outperform the baselines. Interestingly, the policy has learned to produce better semantic segmentations than the ones obtained via MRF energy minimization. Guided by a reward derived just from the energy function, the graph neural network (the policy) learns characteristic node embeddings for every semantic class by leveraging the hypercolumn and bounding box features as well as the neighborhood characteristics. The supervised baseline shows low performance, which proves the merit of the learned policies. Overall MCTS performance is comparable to the DQN one. This is mainly due to the learned policies being somewhat local and focusing on object boundaries, not necessitating a large multi-step look-ahead, as we will show in the following. In Figure 4.3, we report success cases of the RL algorithms. Smoothness modeled by our proposed energy fixed the bottle segmentation in the first image. Furthermore, our model detects missing parts of the table in the second image in the first row and a car in the image in the second row, that were missed by the unaries. Also, we show that we fix a mis-labeling of a truck as a car in the image in the third row.

Flexibility of Potentials: In Figure 4.4, we show examples of improved segmentations when using the pairwise, HOP1 and HOP2 potentials respectively. The motorcycle driver segmentation improved incrementally with every potential (first image) and the cyclist is not detected anymore as a pedestrian (second image).

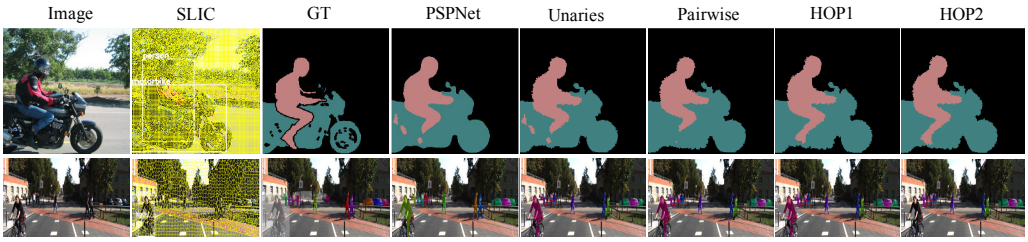


Figure 4.4: Output of our method for different potentials.

Generalization and Scalability: The graph embedding network enables training and testing on graphs with different number of nodes, since the same parameters are used. We investigate how models trained on graphs with few nodes perform on larger graphs. As shown in Table 4.4, compelling accuracy and IoU values for generalization to graphs with up to 500, 1000, 2000, and 10000 nodes are observed when using a policy trained on graphs of 250 nodes

for Pascal VOC, and to graphs with up to 5000 and 10000 nodes when using a policy trained on 2000 nodes for MOTS. Here, we consider the energy consisting of the combined potentials (unary, pairwise, HOP1 and HOP2). Note that we outperform PSPNet at the pixel level for Pascal VOC.

Table 4.4: Generalization of the learned policy.

		PSPNet	500		1000		2000		10000	
Pascal VOC			DQN	MCTS	DQN	MCTS	DQN	MCTS	DQN	MCTS
	IoU (sp)	–	**88.74**	88.73	87.58	**87.61**	86.36	**86.39**	84.66	**84.67**
	IoU (p)	82.61	**83.06**	83.01	83.71	**83.73**	83.74	**83.78**	83.80	**83.82**
		TrackR-CNN	**5000**				**10000**			
MOTS			**DQN**		**MCTS**		**DQN**		**MCTS**	
	IoU (sp)	–	**79.81**		79.80		**76.73**		76.69	
	IoU (p)	84.98	**83.49**		83.46		**84.69**		84.63	

Runtime Efficiency: In Table 4.5, we show the inference runtime for respectively the baselines, DQN and MCTS. The runtime scales *linearly* with the number of nodes and does not even depend on the potential type/order in case of DQN, as inference is reduced to a forward pass of the policy network at every iteration (Figure 4.2). DQN is faster than all the solvers apart from α-exp. However, performance-wise, α-expansion has worse results (Table 4.2 and Table 4.3). MCTS is slower as it performs multiple simulations per node and requires computation of the reward at every step.

Table 4.5: Runtime during inference in seconds for Pascal VOC dataset.

Nodes	U+P							U+P+HOP1							U+P+HOP1 +HOP2	
	BP	**TBP**	**DD**	**L-Flip**	**α-Exp**	**DQN**	**MCTS**	**BP**	**TBP**	**DD**	**L-Flip**	**α-Exp**	**DQN**	**MCTS**	**DQN**	**MCTS**
50	0.14	0.52	0.28	0.12	**0.01**	0.04	0.20	0.15	0.62	0.31	0.187	**0.01**	0.04	0.23	**0.04**	0.24
250	1.56	2.13	1.26	0.53	**0.04**	0.22	2.22	1.70	2.77	1.65	0.59	**0.07**	0.22	2.89	**0.22**	3.01
500	3.26	4.76	2.82	1.07	**0.12**	0.52	7.27	3.37	5.37	3.70	0.97	**0.22**	0.53	9.17	**0.52**	9.69
1000	6.63	9.65	6.84	1.80	**0.30**	0.78	18.5	7.22	10.4	7.47	2.25	**0.36**	0.78	21.6	**0.78**	22.8
2000	12.3	19.9	14.8	3.57	**0.70**	1.70	38.3	12.7	23.9	15.1	4.47	**0.72**	1.70	43.2	**1.72**	46.2
10000	72.8	130.9	143.7	22.6	**4.81**	8.23	202.1	88.7	140.1	106.9	23.5	**4.72**	8.25	209.7	**8.20**	210.3

Learned Policies: In Figure 4.5, we show the probability map across consecutive time steps. The selected nodes are in white. The darker the superpixel, the smaller the probability of selecting it next. The probability maps are obtained by first computing an N-dimensional score vector $\boldsymbol{\phi_\theta}((i,\cdot)|\boldsymbol{s}_t) = \sum_{y\in\boldsymbol{\mathcal{L}}} \pi_{\boldsymbol{\theta}}((i,y)|\boldsymbol{s}_t)\ \forall i \in \{1,\dots,N\}$ by summing over all the label scores per node and then normalizing $\boldsymbol{\phi}$ to a probability distribution over the non selected superpixels $i \in \{1,\dots,N\} \setminus \boldsymbol{I}_t$. We found that the heuristic learns a notion of smoothness, choosing nodes that are in close

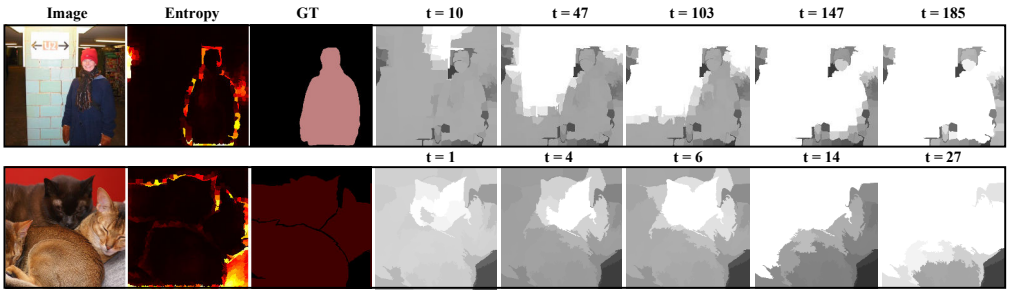


Figure 4.5: Visualization of the learned policy.

proximity and of the same label as the selected ones. Also, the policy learns to start labeling the nodes with low unary distribution entropy first, then decides on the ones with higher entropy.

Comparing Reward Schemes: In Table 4.2 and Table 4.3 , we observe that the second reward scheme ($r_t = \pm 1$) generally outperforms the first one ($r_t = -E_t + E_{t-1}$). This is due to the fact that, the rewards for wrong actions, in this scheme, can be higher than the ones for good actions. Specifically, in Figure 4.6 we plot the distribution of the rewards of good actions (in blue) and the one of wrong actions (in orange) for $50,000$ randomly chosen actions from the replay memory. To better illustrate the cause, we consider the unary energy case and visualize the class distribution of two nodes i and j. If we label node i to be of class 0 (good action), and node j to be of class 1 (wrong action), the resulting rewards are $f_i(0)$ for node i and $f_j(1)$ for node j. Note that $f_i(0) < f_j(1)$, since the distribution of the labels in case of node i is almost uniform, whereas the mass for node j is put on the first two labels.

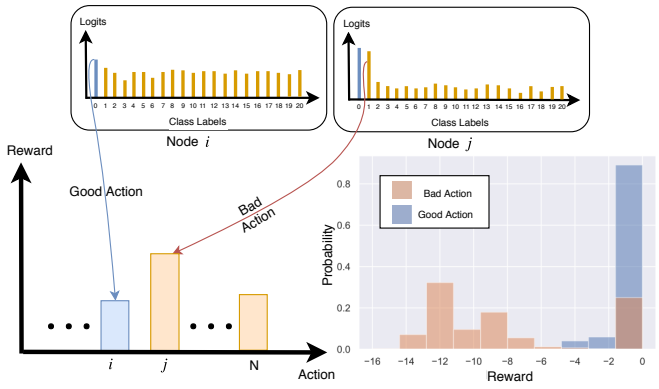


Figure 4.6: Explanation of the low performance of the first reward scheme ($r_t = -E_t + E_{t-1}$).

Visualization of Learned Embeddings: In Table 4.2 and Table 4.3, we observe that our model can produce better segmentations than the ones obtained by just optimizing energies. Guided by the reward and due to network regularization, the policy net captures contextualized embeddings of classes beyond energy minimization. Intuitively, a well calibrated energy function yields rewards that are well correlated with F1-scores for segmentation. When TSNE-projecting the policy nets node embeddings for Pascal VOC data into a 2D space, we observe that they cluster in 21 groups, as illustrated in Figure 4.7.

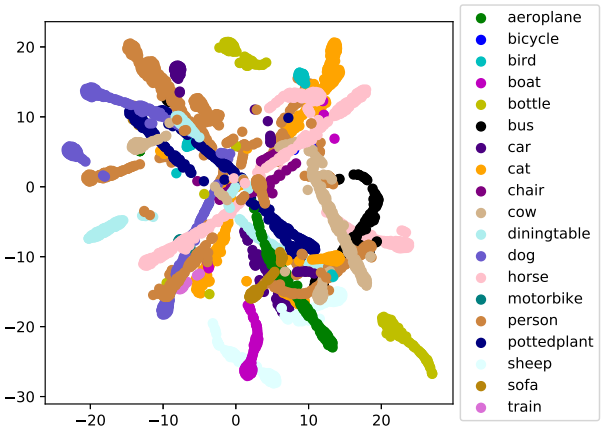


Figure 4.7: Visualization of the learned embeddings.

Limitations: Our method is based on super-pixels, hence datasets with small objects require a large number of nodes and a longer runtime (MOTS *vs* Pascal VOC). Also, our method is sensitive to bounding box class errors, as illustrated in Figure 4.8 (first example), and to the parameters calibration of the energy function, as shown in the second example of the same figure. We plan to address the latter concern in future work via end-to-end training. Furthermore, little is know about deep reinforcement learning convergence. Nevertheless, it has been successfully applied to solve combinatorial programs by leveraging the structure in the data. We show that in our case as well, it converges to reasonable policies. We provide additional results in Section B.

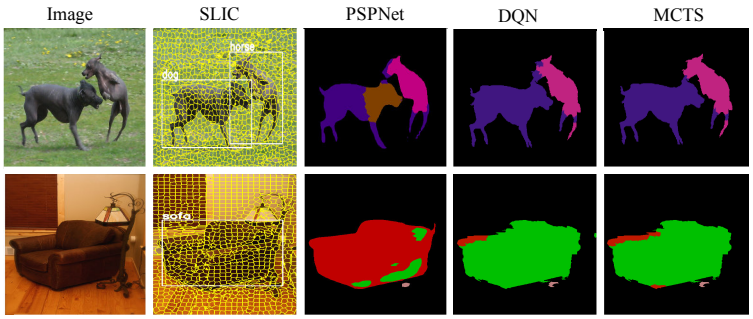


Figure 4.8: Failure cases.

4.4 Conclusion

We study how to solve higher-order CRF inference for semantic segmentation with reinforcement learning with dense rewards. The approach is able to scale to problems with a large label space, deal with potentials that are too expensive to optimize using conventional techniques and outperforms traditional approaches while being more efficient. Hence, the proposed approach offers more flexibility and scalability for energy functions while scaling linearly with the number of nodes and the potential order. To answer our question: can we learn heuristics for graphical model inference? We think we can but we also want to note that a lot of manual work is required to find suitable features and graph structures. For this reason we think more research is needed to truly automate learning of heuristics for graphical model inference. Also, we encourage further investigation of more global and dense reward schemes. We hope the research community will join us in this quest.

CHAPTER 5

MAX-SLICED SCORE MATCHING FOR LEARNING STRUCTURED MODELS AT SCALE

In the previous chapters, we proposed several techniques for scalable inference in graphical models. In this chapter, we introduce a new loss for learning structured models: max-sliced score matching (MSSM) improving upon sliced score matching (SSM) loss. Unlike SSM that projects the data and model scores into random directions and computes their average difference along those directions, MSSM only considers the most informative direction. We plan to assess our model on (1) estimating densities, specifically for the class of Deep Kernel Exponential Families (DKEF) [228] and NICE flow models [229]), as well as on (2) estimating scores for implicit distributions in variational inference applications, in particular, for image generation using variational auto-encoders (VAE) [184] and Wasserstein auto-encoders (WAE) [230] with implicit encoders. Related work on score matching is reviewed in Section 2.3.2.

5.1 Approach

Given a dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{|\mathcal{D}|}$ of samples $\boldsymbol{x} \sim p_d(\boldsymbol{x})$ drawn from an unknown data distribution $p_d(\boldsymbol{x})$, the goal of density estimation is to find the parameters $\boldsymbol{\theta}$ of a model distribution $p_m(\boldsymbol{x}; \boldsymbol{\theta})$ which best fits the samples $\mathcal{D}$ drawn from the unknown data distribution $p_d(\boldsymbol{x})$.

Energy-based models (EBMs) are frequently used for this purpose, representing the model distribution via

$$p_m(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp -E(\boldsymbol{x}; \boldsymbol{\theta})}{Z}, \quad Z = \int_{\boldsymbol{x'}} \exp -E(\boldsymbol{x'}; \boldsymbol{\theta}) d\boldsymbol{x'},$$

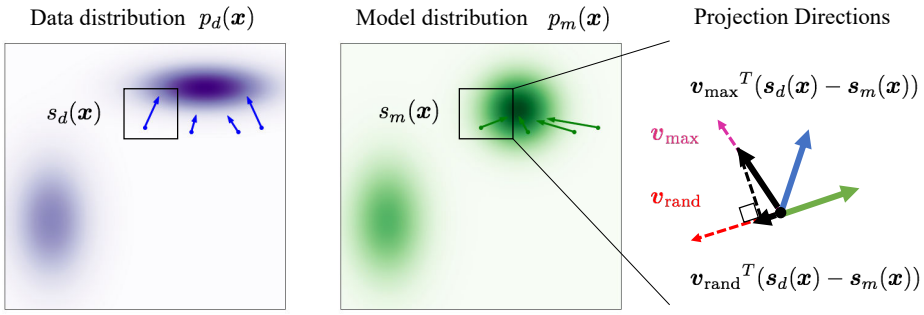


Figure 5.1: Visualization of the effect of slicing. $\boldsymbol{s}_d(\boldsymbol{x})$, and $\boldsymbol{s}_m(\boldsymbol{x})$ are the scores, *i.e.*, gradients of the log density of the data and model distributions respectively. Sliced score matching depends on projecting the difference in scores to a scalar. A random projection $\boldsymbol{v}_{\text{rand}}$ could make the difference in scores appear smaller than it is. Our work aims to select the projection that maximizes the difference in scores (see $\boldsymbol{v}_{\text{max}}$). We find this to make learning more efficient.

where Z is referred to as the normalizing constant, or partition function.

The most popular techniques for learning an EBM consist in approximating the maximum likelihood gradient (see, *e.g.*, work by Du *et al.* [231] and references therein), or in using surrogate objectives like score matching. The first approach suffers from having to approximate the intractable partition function. Score matching methods circumvent this by only looking at the scores $s(\boldsymbol{x})$, *i.e.*, the gradient of the unnormalized density $s(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$. Hyvärinen *et al.* [83] proposed the Score Matching (SM) objective (Equation 2.30) for learning EBMs, but the Hessian term scales quadratically with the input dimensions. To reduce computation, Song *et al.* [88] proposed Sliced Score Matching (SSM) which addresses

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \mathbb{E}_{\boldsymbol{v} \sim p_{\boldsymbol{v}}} \mathbb{E}_{\boldsymbol{x} \sim p_d}[(\boldsymbol{v}^T \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{v}^T \boldsymbol{s}_d(\boldsymbol{x}))^2],$$

where $\boldsymbol{s}_m(\cdot)$ and $\boldsymbol{s}_d(\cdot)$ are the scores of the model and data distributions respectively. Intuitively, the SSM objective applies random projections to the scores in the Fisher divergence (Equation 2.29).

We observe, the random nature of the projections $\boldsymbol{v}$ to make learning inefficient. To address this, we study a method which finds projections along

Algorithm 4: Max-Sliced Score Matching

Input: Update frequency F of $\boldsymbol{V}$; Number of sampled eigenvectors M per training iteration; Data distribution $p_d(\boldsymbol{x})$; projection direction distribution $\hat{p}_{\boldsymbol{v}}(\boldsymbol{v})$; Model distribution $p_m(\boldsymbol{x};\boldsymbol{\theta})$.
step $\leftarrow 0$
repeat
  Sample $\boldsymbol{x} \sim p_d(\boldsymbol{x})$
  $\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \leftarrow \nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})$
  $\mathcal{L}_{\text{MSSM}}(\boldsymbol{x};\boldsymbol{\theta}) \leftarrow \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2$
  if *(step % F) ==0* **then**
    $\boldsymbol{V} \leftarrow \text{eig}(\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))$
  end
  for $i \leftarrow 1$ *to* M **do**
    $\mathcal{L}_{\text{MSSM}}(\boldsymbol{x};\boldsymbol{\theta}) \leftarrow \mathcal{L}_{\text{MSSM}}(\boldsymbol{x};\boldsymbol{\theta}) + \frac{\hat{p}_{\boldsymbol{v}}(\boldsymbol{v}_i)}{M} \boldsymbol{v}_i^T \nabla_{\boldsymbol{x}}(\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}_i)$
  end
  Update $\boldsymbol{\theta}$ via backpropagation
  step $\leftarrow$ step + 1
Return $\boldsymbol{\theta}$

directions that maximize the projected Fisher divergence via

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \quad \mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{V}^T(\boldsymbol{s}_d(\boldsymbol{x}) - \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))\|^2], \\ &\text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^{\boldsymbol{T}} = I, \end{aligned} \tag{5.1}$$

where $\boldsymbol{V} \in \mathbb{R}^{d\times d}$, denotes the matrix with its i-th column $\boldsymbol{v}_i \in \mathbb{R}^d$ being a projection direction. We use $\|\cdot\|$ to denote the 2-norm of a vector. The constraint $\boldsymbol{V}\boldsymbol{V}^{\boldsymbol{T}} = I$ is introduced to ensure that we make most use of the space by requiring orthogonality. Beneficially, the constraint ensures boundedness of the loss (vectors $\boldsymbol{v}_i$ cannot be scaled so that maximization reaches infinity). We refer to this objective as MSSM. In Figure 5.1, we illustrate the scores $\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})$, $\boldsymbol{s}_d(\boldsymbol{x})$ as well as the projection directions for a Gaussian mixture model. Since $\boldsymbol{s}_d(\boldsymbol{x})$ cannot be computed (the data distribution is unknown), we follow Hyvärinen *et al.* [83] and use integration-by-parts to obtain an objective without $\boldsymbol{s}_d(\boldsymbol{x})$ whose only difference from Equation 5.1

Algorithm 5: Lanczos Algorithm for Score Matching

Input: Number of iterations K. The score $s_m(\boldsymbol{x};\boldsymbol{\theta})$.
Initial $q_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
Choose $\boldsymbol{r} = \boldsymbol{q}_0$
$\beta_0 = \|\boldsymbol{q}_0\|$
while $k = 1, \cdots, K$ **do**
  $\boldsymbol{q}_k = \frac{\boldsymbol{r}}{\beta_{k-1}}$
  $\boldsymbol{r} = \nabla_{\boldsymbol{x}}(\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})^T \boldsymbol{q}_k)$
  $\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{q}_{k-1}\beta_{k-1}$
  $\alpha_k = \boldsymbol{q}_k^T \boldsymbol{r}$
  $\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{q}_k \alpha_k$
  $\beta_k = \|\boldsymbol{r}\|^2$
end
$\boldsymbol{T}_K = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \beta_2 & \alpha_3 & \\ & & \ddots & \beta_{K-1} \\ & & \beta_{K-1} & \alpha_K \end{bmatrix}$
$\boldsymbol{Q}_K = [\boldsymbol{q}_0, \cdots, \boldsymbol{q}_K]$
$\boldsymbol{S}_K = \mathrm{eig}(\boldsymbol{T}_K)$
Return $\boldsymbol{V}^* = \boldsymbol{Q}_K \boldsymbol{S}_K$

is a constant which is independent of $\boldsymbol{s}_m(\boldsymbol{x})$:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}}[\sum_{i=1}^{d} \boldsymbol{v}_i^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \boldsymbol{v}_i + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2], \\ \text{s.t.} \quad & \boldsymbol{V}\boldsymbol{V}^{\boldsymbol{T}} = I. \end{aligned} \tag{5.2}$$

We defer the proof of equivalence to Section C.2.

Consistency: In the following, we denote by $\hat{\boldsymbol{\theta}}_N$ the parameter resulting from optimizing the empirical mean of the objective in Equation 5.2 using a batch size N. Also, we call $\boldsymbol{\theta}^*$ the optimal parameter under the assumption of a well-specified model distribution $p_m(\boldsymbol{x};\boldsymbol{\theta})$, *i.e.*, $p_d(\boldsymbol{x}) = p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$ and $p_m(\boldsymbol{x};\boldsymbol{\theta}) \neq p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$ whenever $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$. Furthermore, suppose that the parameter space is compact, and that for all samples $\boldsymbol{x}$ and configurations $\boldsymbol{\theta}$, we have: $\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\| < \infty$, as well as $\|\nabla_{\boldsymbol{x}} \log \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|_\infty < \infty$. Then, we show that MSSM is consistent, *i.e.*, as the sample size N increases, the sampling distribution of the estimator $\hat{\boldsymbol{\theta}}_N$ becomes increasingly concentrated

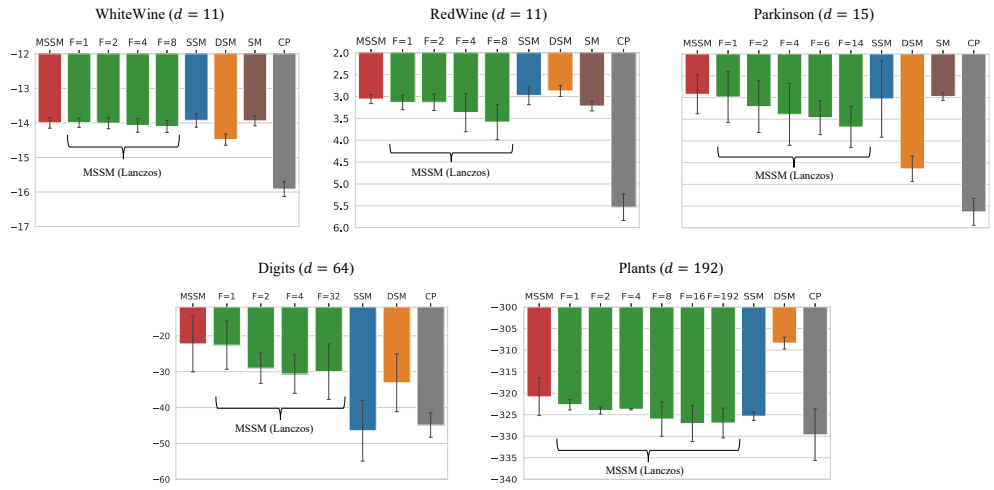


Figure 5.2: Log-likelihoods after training DKEF models on UCI datasets with different loss functions. Higher is better.

at the true parameter value $\boldsymbol{\theta}^*$. Formally,

$$\hat{\boldsymbol{\theta}}_N \xrightarrow{\mathrm{p}} \boldsymbol{\theta}^*, \quad N \to \infty.$$

The proof is detailed in Section C.1.1.

Asymptotic Normality: Suppose that the parameter space is compact, $\hat{\boldsymbol{\theta}}_N \xrightarrow{p} \boldsymbol{\theta}^*$, $\log p_m(\boldsymbol{x}; \boldsymbol{\theta})$ is twice continuously differentiable with respect to $\boldsymbol{\theta}$, $J(\boldsymbol{\theta}^*)$ is invertible at $\boldsymbol{\theta}^*$, and for all samples $\boldsymbol{x}$ and configurations $\boldsymbol{\theta}$, $\|\nabla^2_{\boldsymbol{\theta}} \|\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta})\|^2\|_\infty < \infty$, as well as $\|\nabla^2_{\boldsymbol{\theta}} [\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta})]_{ii}\|_\infty < \infty$, $\forall i \in \{1, \cdots, d\}$. Then, $\hat{\boldsymbol{\theta}}_N$ is asymptotically normally distributed around the true parameter $\boldsymbol{\theta}^*$ when the batch size N becomes very large, *i.e.*,

$$\sqrt{N}(\hat{\boldsymbol{\theta}}_N - \boldsymbol{\theta}^*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \quad N \to \infty.$$

Here, $\boldsymbol{\Sigma}$ is the covariance matrix. In Section C.1.2, we show that the asymptotic variance of our MSSM estimator is equal the score matching asymptotic variance and less than the sliced score matching one.

Furthermore, we argue that MSSM leads to a better maximum likelihood estimation than SSM. We show in Section C.3, that MSSM allows a better approximation of the partition function, as d vectors are needed for estimating the trace exactly, whereas 2^d vectors are needed for the sum.

Table 5.1: Log-likelihoods for NICE models trained on MNIST.

MLE	SSM		DSM		CP	Approx.	MSSM			
	-Plain-	-Staged-	$\sigma = 0.1$	$\sigma = 1.74$		BP	Exact	Laczos F=100	Laczos F=200	Laczos F=500
-791	-3355	-1102	-4363	-8082	-1517	-2288	-1044	-1067	-1085	-1235

Optimization for MSSM: In the following we first discuss how to address the program in Equation 5.2 before elaborating on efficiency.

Claim: The solution to the inner maximization problem in Equation 5.2 is the eigenvector basis of the Hessian $\mathbb{E}_{\boldsymbol{x}}[\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})]$.

Proof: The Lagrangian of the program in Equation 5.2 is: $\mathcal{L}(\boldsymbol{V}; \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}}[\sum_{i=1}^{d} \boldsymbol{v}_i^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta}) \boldsymbol{v}_i + \frac{1}{2} \|\boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})\|^2] - \sum_{i=1}^{d} \lambda_i (\|\boldsymbol{v}_i\|^2 - 1)$ with $\boldsymbol{v}_i^T \boldsymbol{v}_j = 0$, $(\forall i \neq j)$. For a given $\boldsymbol{v}_i$, $\frac{\partial \mathcal{L}(\boldsymbol{v}_i; \boldsymbol{\theta})}{\partial \boldsymbol{v}_i} = 0$ is equivalent to $\mathbb{E}_{\boldsymbol{x}}[\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})] \boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$. Hence, the direction $\boldsymbol{v}_i$ that solves the linear system corresponds to the eigenvector of the symmetric Hessian $\mathbb{E}_{\boldsymbol{x}}[\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})]$ with the eigenvalue λ_i. The condition $\boldsymbol{v}_i^T \boldsymbol{v}_j = 0$ is satisfied since the eigenvectors are orthogonal to each other. This concludes the proof. □

Unlike log-densities $\log p_d(\boldsymbol{x})$, which decrease as we move away from the data-manifold, the magnitude of the scores $\nabla_{\boldsymbol{x}} \log p_d(\boldsymbol{x})$ generally increases as we move away from the manifold. Hence, a good score estimation requires projection directions that encompass the entire space. The MSSM formulation uses a basis of d vectors to cover different directions in the d-dimensional space. As mentioned before, this is more efficient than sampling a large number of random directions to properly estimate the expectation in Equation 2.32.

However, computing the eigenvectors using traditional eigenvector decomposition is very expensive and requires explicit computation and storage of the Hessian $\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}; \boldsymbol{\theta})$, which defies the purpose of Score Matching. To address this concern we study the combination of three ideas: (1) use of the Lanczos algorithm [232] with K iterations to find the K most important directions $\boldsymbol{V} \in \mathbb{R}^{d \times K}$, while adapting the Lanczos algorithm to only operate on Hessian-vector products; (2) importance sampling following a distribution $\hat{p}_{\boldsymbol{v}}(\boldsymbol{v})$ to approximate the sum in Equation 5.2 which reduces the number of backpropagations; and (3) computing these vectors with a frequency F instead of at every iteration. Those ideas are summarized in Algorithm 4 and we discuss details next.

Table 5.2: Log-likelihoods on MNIST, estimated by AIS.

	VAE		WAE	
Latent Dim	8	32	8	32
ELBO	-96.87	-89.06	N/A	N/A
SSM	**-95.50**	-89.25	-98.24	-90.37
Stein	-96.71	-91.84	-99.05	-91.70
Spectral	-96.60	-94.67	-98.81	-92.55
MSSM (Exact)	-95.61	-89.26	**-97.62**	-89.53
MSSM (Laczos)	-95.75	**-88.99**	-98.25	**-89.46**

Lanczos Algorithm: The Lanczos algorithm [232] computes the spectrum of a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ by first reducing it to a tridiagonal form $\boldsymbol{T}_K \in \mathbb{R}^{K\times K}$ and then computing the spectrum $\boldsymbol{S}_K \in \mathbb{R}^{K\times K}$ of that matrix instead. The computation of the spectrum of a tridiagonal matrix is very efficient. The algorithm works in steps $k \in \{1, \dots, K\}$ and progressively builds an adapted orthonormal basis $\boldsymbol{Q}_k = [\boldsymbol{q}_0, \dots, \boldsymbol{q}_k] \in \mathbb{R}^{d\times k}$, that satisfies at each iteration $\boldsymbol{Q}_k^T \boldsymbol{A} \boldsymbol{Q}_k = \boldsymbol{T}_k$. Each of the k iterations of the algorithm requires a single Hessian-vector multiplication, that we compute implicitly in two steps: (1) First, we compute $\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})$; then (2) the derivative of the vector product $\nabla_{\boldsymbol{x}}(\boldsymbol{v}^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))$. Overall, for K steps, $K+1$ backpropagations are needed. The Lanczos algorithm is memory efficient as it only stores three vectors at every iteration k, *i.e.*, $\boldsymbol{q}_{k-1}$, $\boldsymbol{q}_k$ and $\boldsymbol{q}_{k+1}$. The eigenvectors $\boldsymbol{V}^*$ of the matrix $\boldsymbol{A}$ are then obtained via $\boldsymbol{V}^* = \boldsymbol{Q}_K \boldsymbol{S}_K \in \mathbb{R}^{d\times K}$, as illustrated in Algorithm 5. Note that we use $\boldsymbol{V}^*$ to denote the optimal directions obtained when using K steps of the Lanczos algorithm.

Sampling: After solving for the eigenvectors $\boldsymbol{V}^*$ of $\nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})$ obtained from Algorithm 5, we use importance sampling to avoid computing the sum over the projection directions in Equation 5.2, as this requires K backpropagations. Instead we address the following problem,

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{p_d(\boldsymbol{x})} \Big[\mathbb{E}_{\hat{p}_{\boldsymbol{v}}} \Big[\frac{\boldsymbol{v}^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \boldsymbol{v}}{\hat{p}_{\boldsymbol{v}}(\boldsymbol{v})} \Big] + \frac{1}{2} \| \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \|^2 \Big]. \tag{5.3}$$

We choose $\hat{p}_{\boldsymbol{v}}(\boldsymbol{v})$ to be the uniform distribution over the set of the computed eigenvectors $\boldsymbol{V}^*$. Note that the distribution $\hat{p}_{\boldsymbol{v}}$ differs from the distribution $p_{\boldsymbol{v}}$ over the space of all possible vectors $\boldsymbol{v}$ which is used for sliced score matching in Equation 2.32.

Overall the method requires $\mathcal{O}(M + \frac{K}{F})$ backpropagations. Here, M is the number of sampled vectors to approximate the expectation over $\boldsymbol{v}$ in

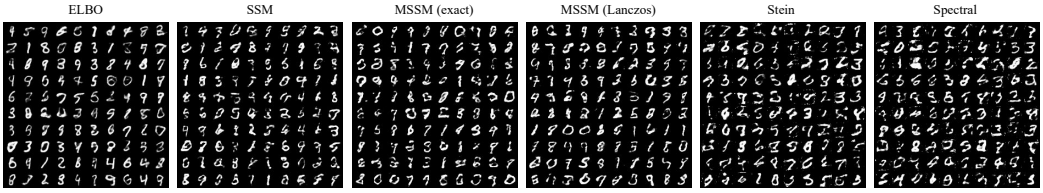


Figure 5.3: VAE generated samples on MNIST for a latent dimension $\boldsymbol{z} = 32$.

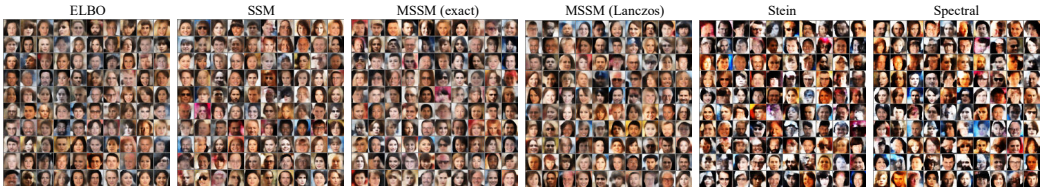


Figure 5.4: VAE generated samples on CelebA for a latent dimension $\boldsymbol{z} = 32$.

Equation 5.3. In practice, we chose $M = 1$.

5.2 Experiments

We study the MSSM approach on estimating (1) densities (Section 5.2.1) and (2) scores for modeling implicit distributions (Section 5.2.2). Following Song *et al.* [88], we set the number of vectors M to be 1 in Equation 5.3.

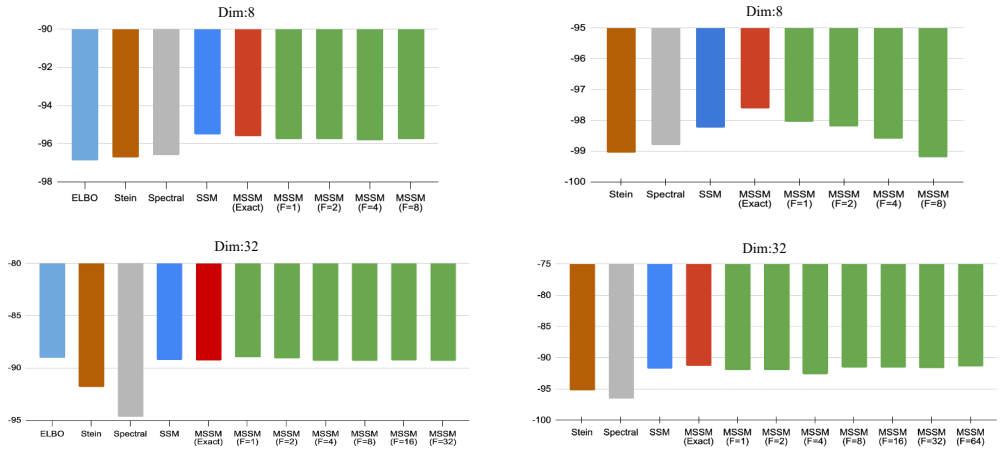
5.2.1 Density Estimation

We consider two density estimation experiments: (1) estimating the parameters of an energy-based model, *i.e.*, "deep kernel exponential families" (DKEF) and (2) learning a deepflow model.

Baselines: We use the following baselines: (1) Score matching (SM) [83], (2) Sliced score matching with reduced variance (SSM) and a multivariate Rademacher distribution p_v [88], (3) Denoising score matching (DSM) [86], (4) Approximate backpropagation (approx. BP) [84] and (5) Curvature propagation (CP) [85]. We report results of MSSM with exact eigenvector computation and with an approximate one using the Lanczos algorithm.

Metric: As evaluation metric we use maximum likelihood which directly relates to the score matching loss, as explained in Section C.3.

Deep Kernel Exponential Families (DKEF): DKEF parameterizes the unnormalized log-likelihood via $\log p_f(\boldsymbol{x}) = \sum_{l=1}^{L} \alpha_l k(\boldsymbol{x}, \boldsymbol{z}_l) + \log q_0(\boldsymbol{x})$, where



(a) Log-likelihoods of VAE on MNIST. (b) Log-likelihoods of WAE on MNIST.

Figure 5.5: Log-likelihoods of VAE and WAE with implicit distributions trained on MNIST dataset.

$q_0(\boldsymbol{x})$ is a fixed function, α_l are weights, and $k(\boldsymbol{x}, \boldsymbol{z}_l)$ is a mixture of R Gaussian kernels defined on the feature space $\boldsymbol{\phi}_r$ of a deepnet and evaluated at L different inducing points $\boldsymbol{z}_l$:

$$k(\boldsymbol{x}, \boldsymbol{z}_l) = \sum_{r=1}^{R} \rho_r \exp\left(\frac{-1}{2\sigma_r^2}\|\boldsymbol{\phi}_r(\boldsymbol{x}) - \boldsymbol{\phi}_r(\boldsymbol{z}_l)\|^2\right). \tag{5.4}$$

The training parameters include the inducing points $\boldsymbol{z}_l$, the deep net $\boldsymbol{\phi}_r$, length scales σ_r and the non-negative mixture coefficients ρ_r. A closed-form expression for the weights α_l can be derived as a function of the other model parameters (see Section C.4 for training details). We follow the same setup as [88]. All models are trained with 15 different random seeds and training is stopped when the validation loss does not improve for 200 steps.

The obtained log-likelihoods on RedWine ($d = 11$), WhiteWine ($d = 11$), Parkinsons ($d = 15$), Digits ($d = 64$) and Plants ($d = 192$) datasets [233] are presented in Figure 5.2. Note that the likelihoods are estimated using AIS [234], using a zero mean Gaussian proposal distribution $\mathcal{N}(0, 2I)$ and 1,000,000 samples. We observe that MSSM performance is comparable to SSM for datasets with small dimensionality, *i.e.*, RedWine, WhiteWine and Parkinsons. MSSM however scales better to the high-dimensional datasets, *i.e.*, Digits and Plants. Exact MSSM is consistently better than the Lanczos version. This is expected as we limit the number of iterations of the Lanczos algorithm to K, which may lead to smaller eigenvalues not converging. As

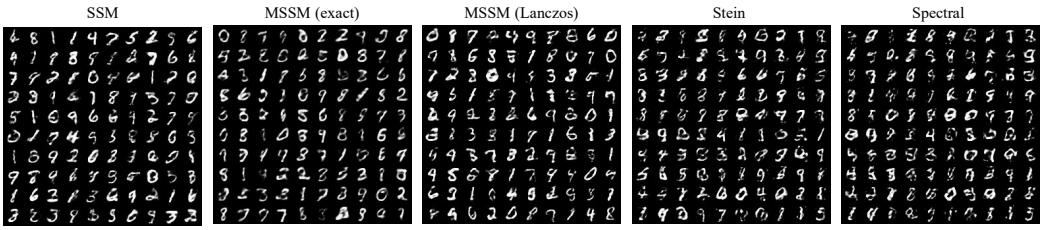


Figure 5.6: WAE generated samples on MNIST for a latent dimension $\boldsymbol{z} = 32$.

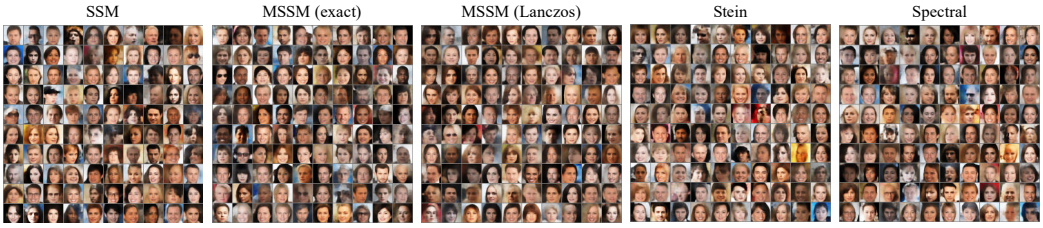


Figure 5.7: WAE generated samples on CelebA for a latent dimension $\boldsymbol{z} = 32$.

we increase the frequency F of the eigenvector computation, MSSM (Lanczos) performance decreases. This is expected as well, since the computed eigenvectors are no longer accurate and SSM ends-up sampling more diverse directions in the space. Our MSSM models, both exact and Lanczos-based with small update frequencies F, have lower variance than SSM for RedWine, WhiteWine, Parkinsons and Digits datasets. DSM performance strongly depends on the choice of the noise rate σ, which is hard to tune. Similarly to work by Song *et al.* [88], we run a grid search over a range of values. CP performance is constantly worse as it injects noise to each node in the computation graph. This is not efficient for training large deepnet models. Computing the exact score matching loss becomes already prohibitively slow for the digits and plants datasets. We omit the results for approx. BP, as the log-likelihoods were smaller than -10^6 for all datasets.

Deep Flow Models: Since Deep flow models enable access to a tractable maximum likelihood, we train a NICE [229] model to better understand the relationship between MSSM, SSM and MLE. For this we consider MNIST data ($d = 728$) generation. Results are reported in Table 5.1. Since score matching is a weak form of MLE, as explained in Section C.3, we obtain a better maximum likelihood when applying score matching loss on the intermediate layers of the flow model (SSM-staged). When trained with the SSM loss applied only to the last layer, the log-likelihood fluctuates and the best achieved performance does not correspond to the one at the end of training.

Table 5.3: FID scores of different methods versus number of training iterations on the CelebA dataset.

		10k	40k	70k	100k
VAE	ELBO	**96.20**	73.70	69.42	66.32
	SSM	108.52	70.28	66.52	62.50
	Stein	126.60	118.87	120.51	126.76
	Spectral	131.90	125.04	128.36	133.93
	MSSM (exact)	100.09	69.17	64.37	**60.68**
	MSSM (Lanczos, F=1)	100.26	**64.14**	65.45	60.78
	MSSM (Lanczos, F=2)	98.38	65.34	64.91	62.46
	MSSM (Lanczos, F=8)	101.15	65.34	65.78	63.76
	MSSM (Lanczos, F=16)	100.65	64.71	**62.68**	63.33
	MSSM (Lanczos, F=32)	107.24	66.30	65.31	65.91
WAE	SSM	84.11	61.09	56.23	54.33
	Stein	82.93	63.46	58.53	57.61
	Spectral	82.30	62.47	58.03	55.96
	MSSM (exact)	87.12	**60.93**	**56.17**	**53.98**
	MSSM (Lanczos, F=1)	84.08	71.68	57.29	54.03
	MSSM (Lanczos, F=2)	82.33	71.00	60.66	54.23
	MSSM (Lanczos, F=8)	82.92	73.37	59.48	54.98
	MSSM (Lanczos, F=16)	**82.21**	73.48	59.62	58.49
	MSSM (Lanczos, F=32)	83.75	70.54	59.37	59.27

Indeed, Song *et al.* [88] report results from the best checkpoint. To further improve upon SSM-staged, we replace the SSM loss with MSSM at the output layer. We observe that the log-likelihood training curve has a monotonically increasing behavior. We improve upon SSM-staged at the end of the training when using MSSM (exact) or MSSM with eigenvectors computed every 100 or 200 iterations using the Lanczos algorithm.

5.2.2 Score Estimation

We consider replacing simple explicit encoder distributions in Variational Auto-Encoders (VAE) and Wasserstein Auto-Encoders (WAE), with more expressive implicit ones that we train with an additional score matching loss. As baselines, we consider *Stein* [91] and *Spectral* [93] score estimation techniques.

VAE with Implicit Encoders: A classical VAE is trained by minimizing the ELBO loss:

$$\min_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{p_d(\boldsymbol{x})} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z}) + \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})]. \tag{5.5}$$

Here, $p(\boldsymbol{z})$ is the prior distribution, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is the encoder distribution and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is the decoder one. In order to compute a closed form of the loss, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is usually chosen to be a simple explicit distribution (*e.g.*, a Gaus-

sian). This obviously limits the expressivity of the model. Extending ELBO with a score matching loss permits to model arbitrarily complex distributions $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ implicitly. This better captures the multi-modality and intricacies of common dataset. The latter is achieved by replacing the entropy term $-\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ with the term $-\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{z}} \log q_{\boldsymbol{\phi}(\boldsymbol{z}|\boldsymbol{x})}(\boldsymbol{z}|\boldsymbol{x})^T \boldsymbol{z}]$. It can be easily checked that both terms have the same derivative with respect to $\boldsymbol{\phi}$, *i.e.*,

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\Big[\nabla_{\boldsymbol{z}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})^T \nabla_{\boldsymbol{\phi}} \boldsymbol{z}\Big].$$

The gradient $\nabla_{\boldsymbol{z}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is approximated with a deepnet $\boldsymbol{s}_{\psi}(\boldsymbol{z}|\boldsymbol{x})$ using a score matching loss, circumventing the computation of $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. The objective from Equation 5.5 becomes:

$$\begin{aligned}
&\min_{\boldsymbol{\phi},\boldsymbol{\theta},\boldsymbol{\psi}} \max_{\boldsymbol{V}} \mathbb{E}_{p_d(\boldsymbol{x})}\, \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z}) + \\
&\boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})^T \boldsymbol{z} + \sum_{i=1}^{d} \boldsymbol{v}_i^T \nabla_{\boldsymbol{z}} \boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}) \boldsymbol{v}_i + \frac{1}{2} \|\boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})\|^2], \\
&\text{s.t. } \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}.
\end{aligned}$$

We report negative log-likelihoods on MNIST, as estimated by AIS [234] with 1000 intermediate distributions, in Table 5.2 for latent dimensions of size 8 and 32. We present samples in Figure 5.3. In Figure 5.5 (a), we present results for different Lanczos update frequencies. We observe MSSM to perform comparably to SSM. Both MSSM and SSM outperform the ELBO, which emphasizes the expressive power of an implicit encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$.

In Table 5.3, we evaluate sample quality on CelebA with FID scores [235] computed after 10k, 40k, 70k and 100k iterations respectively. We observe that MSSM and its variants all converge faster, having low FID scores after 10k iterations already. In contrast, SSM does not perform as well early during training. Three MSSM variants outperform the ELBO loss. Samples are presented in Figure 5.4.

WAE with Implicit Encoders: While VAE computes a KL-divergence between the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ of each sample and the prior distribution $p(\boldsymbol{z})$, WAE compares the distribution of the entire encoder space $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ to the prior $p(\boldsymbol{z})$. This alleviates the issue of erroneous reconstruction due to overlapping latent codes $\boldsymbol{z}$ originating from different samples $\boldsymbol{x}$. It also

reduces the prior-hole problem. Concretely, WAE minimizes the objective:

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}_{p_d(\boldsymbol{x})} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[c(\boldsymbol{x}, p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) - \lambda \log p(\boldsymbol{z}) - \lambda q_{\boldsymbol{\phi}}(\boldsymbol{z})],$$

composed of the reconstruction loss $c(\boldsymbol{x}, p_\theta(\boldsymbol{x}|\boldsymbol{z}))$ and the KL-divergence between the prior $p(\boldsymbol{z})$ and the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. We model $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ implicitly. Following the VAE case we approximate $\nabla_{\boldsymbol{z}} q_{\boldsymbol{\phi}}(\boldsymbol{z})$ using a deepnet $\boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z})$, resulting in the objective

$$\begin{aligned}\min_{\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{\psi}} \max_{\boldsymbol{V}} \mathbb{E}_{p_d(\boldsymbol{x})} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z})}[c(\boldsymbol{x}, p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) - \lambda \log p(\boldsymbol{z}) - \\ \lambda \boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z})\boldsymbol{z} + \boldsymbol{v}^T \nabla_{\boldsymbol{z}} \boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z})\boldsymbol{v} + \frac{1}{2}\|\boldsymbol{s}_{\boldsymbol{\psi}}(\boldsymbol{z})\|^2], \\ \text{s.t. } \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}.\end{aligned}$$

On MNIST, MSSM achieves the best log-likelihoods for both latent dimensions across all baselines (Table 5.2, Figure 5.5 (b), as well as the best FID scores for CelebA data (Table 5.3). We visualize generated samples obtained from training on MNIST data in Figure 5.6. Samples after training on CelebA are given in Figure 5.7.

5.3 Conclusion

We propose max-sliced score matching (MSSM) which improves upon sliced score matching (SSM) by finding the most informative projection directions for minimizing the difference between the data and model scores. We show that these directions correspond to the eigenvectors of the Hessian of the log-likelihood, which can be computed efficiently using the Lanczos algorithm, importance sampling to approximate the sum over the eigenvectors as well as a schedule for updating the eigenvectors at specific frequencies. We show improved performance on learning unnormalized statistical models and estimating scores for implicit distributions in VAEs and WAEs.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Main Message

In the preceding chapters, we proposed different models for scaling inference (Chapter 3, Chapter 4) to higher-order potentials and learning (Chapter 5) to high-dimensional spaces, in structured models. In Chapter 3, we extended Gaussian conditional random fields, traditionally uni-modal and only limited to modeling pairwise interactions, to model multi-modal distributions, capture higher-order potentials and incorporate external constraints at run-time, while keeping the inference exact. We demonstrated compelling results on the task of diverse image colorization. In Chapter 4, we showed that we can learn heuristics for solving inference in higher-order conditional random fields for the task of semantic segmentation, using reinforcement learning, while scaling linearly with potential orders and number of variables. Finally, in Chapter 5, we proposed a new loss, max-sliced score matching (MSSM) for learning scalable structured models. Unlike sliced score matching (SSM) that projects the data and model scores into random directions and computes their average difference along those directions, MSSM only considers the most informative directions. We proved that MSSM is consistent, asymptotically normal and has lower variance than SSM. We showed empirically improved performance on estimation of densities and scores for implicit distributions in Variational and Wasserstein auto-encoders. In the following, we discuss a few future directions to improve the proposed models.

6.2 Future Work

Numerous extensions for future work, certainly not limited to the directions summarized subsequently, are possible.

Applications. We showed compelling results of the proposed Gaussian Conditional Random Field and the RL-based inference engine for the tasks of diverse image colorization and semantic segmentation, respectively. We, however, think that those models could be useful for many more challenges where higher-order potentials encoding prior knowledge, would lead to better capturing of the structure of the input/output spaces. For instance, image captioning [236], holistic scene understanding [237], stereo estimation [46] and 3D reconstruction [238]. Generalizing our models to these applications would require further exploration into learning the right embeddings for the entities (*e.g.*, words, pixels) over which the structure is defined, such that these embeddings can capture features useful for minimizing the energy function of interest.

Spacio-temporal VAE based G-CRFs. While in Chapter 3, we only showed results on learning diverse image colorization, it would be interesting to extend the model to capture correlations in the temporal dimension, *i.e.*, modeling spatial distance between pixels in different frames. To achieve a good performance at this scale, it might be useful to include spacial distances between the pixels in different frames. This would lead to placing more confidence on the short-range connections than on the long-range ones. This could be achieved by weighting the terms in the A matrix differently, or learning approximate spatio-temporal distance between two pixels using feature embeddings as in [239].

Better rewards and learning algorithms for RL-based minimization of the energy function. In Chapter 4, we studied two schemes for generating rewards to minimize the energy function. We showed that the first scheme where rewards are the difference between the current energy value and the previous one did not lead to good performance. The second scheme, where rewards are discretized to $+1$ in case the selected label leads to the lowest energy compared to all other possible labels and -1 otherwise, led to much better segmentations. However, this scheme does not necessarily lead to rewards that reflect the ground-truth labels. Also, assigning a reward for the entire segmentation mask to fix this issue, for instance in the context of policy gradients, is not efficient due to the large state space that scales linearly with the number of nodes and labels. An interesting future direction

to explore is using imitation learning. In particular, it might be possible to learn better rewards with inverse-reinforcement learning. Specifically, the solution of the energy minimization obtained by an exact solver on small instances of the problem can be used to learn reward functions, which then would allow learning a good policy.

More efficient RL-based inference engine. The proposed RL-base inference engine in Chapter 4, labels the superpixels in the image sequentially. This procedure could be accelerated by labeling groups of superpixels at once (*e.g.*, superpixels with similar embeddings within the same bounding box in semantic segmentation), similarly to move-making algorithms. Hierarchical reinforcement learning methods could be applicable. Besides, this would make the training of the inference engine faster, which would enable using it as a subroutine inside of a learning loop, hence enabling end-to-end learning of the models parameters. Currently, the energy is calibrated on a held-out set of the training data, which is sub-optimal.

A score matching loss with better maximum-likelihood approximation. While we were able to improve upon slices score matching by finding more informative projection directions in Chapter 5, we did to beat the maximum likelihood results, as demonstrated by the flow model experiments. This is due to score matching being a "weaker" form of maximum likelihood, since it corresponds to the second-order Taylor approximation of the contrastive divergence loss with 1-step Langevin Monte-Carlo as a sampler, as explained in Section C.3. Empirically, we noticed that the variance of the score matching estimator is due to the absence of contrastive term which ensures that the scores are correct outside of the manifold, *i.e.*, larger in magnitude and the Hessian being positive definite. Intuitively, score matching variants enforce the concavity of the log-likelihood at the data samples. When clusters are close, in the presence of outliers, we observed that this does not lead to learning the right curvature of the log-likelihood outside of the manifold, which results in a lot variance across different runs of the same experiment. Extending score matching with a contrastive loss, *e.g.*, an adversarial loss, might help better learning of scores outside of the manifold, and hence making learning the ones on the manifold easier.

APPENDIX A

GAUSSIAN CONDITIONAL RANDOM FIELDS BASED VARIATIONAL AUTO-ENCODERS

We use a variational auto-encoder model enriched with a mixture density network and a Gaussian conditional random field to generate diverse and globally consistent colorizations while enabling controllability through sparse user edits. In this appendix, we present additional results on the LFW, LSUN-Church, ILSVRC-2015 and ImageNet datasets (Section A.1). We also explore endowing VAE and BEGAN [240] with a structured output-space distribution through the G-CRF formulation, for the task of image generation (Section A.2).

A.1 Additional Results

Figure A.1: Qualitative comparison of our results with the baselines on LFW.

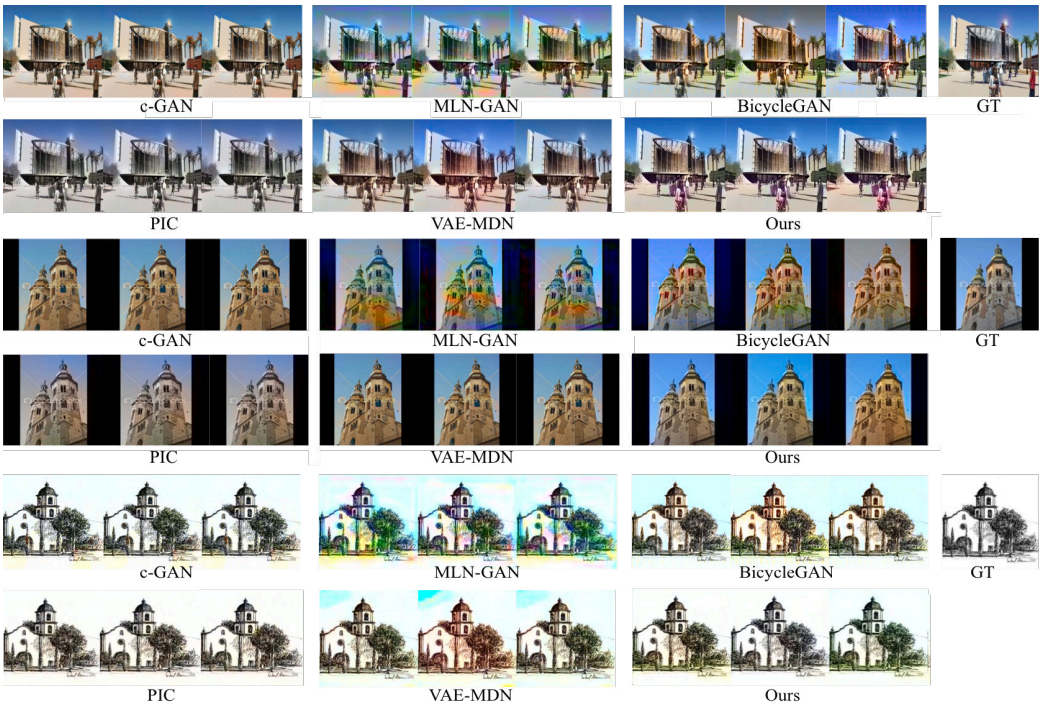


Figure A.2: Qualitative comparison of our results with the baselines on LSUN.

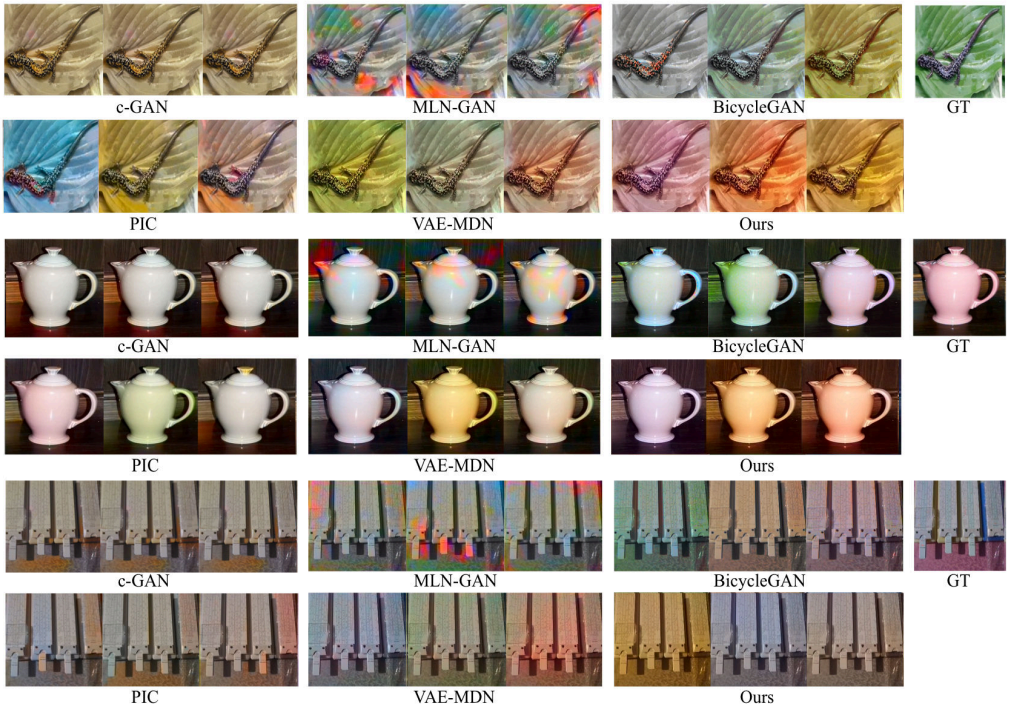


Figure A.3: Qualitative comparison of our results with the baselines on ILSVRC-2015.

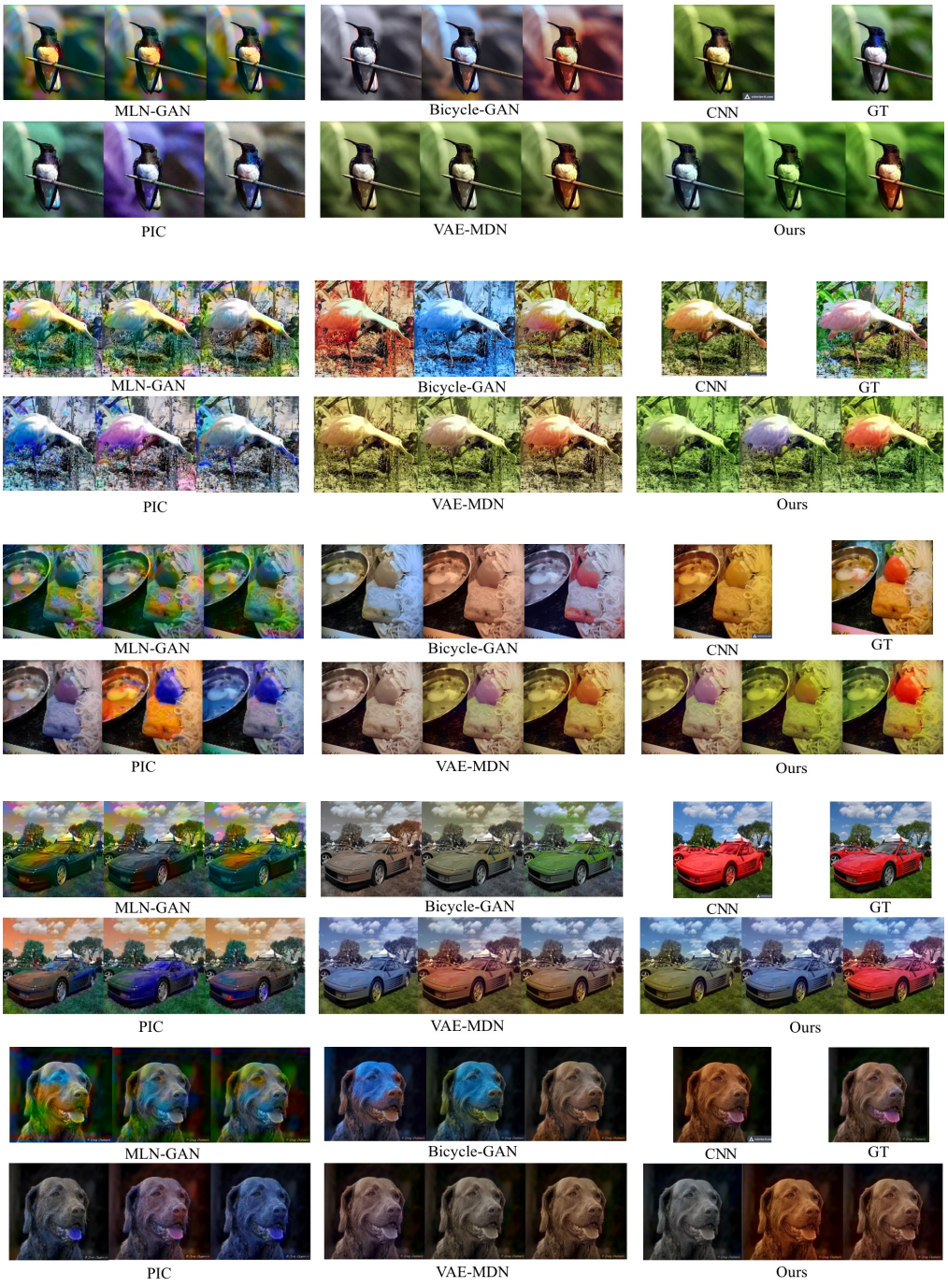


Figure A.4: Qualitative comparison of our results with the baselines on ImageNet.

Figure A.5: Qualitative comparison of our results with the baselines on the Shoes dataset [241].

A.2 G-CRF for Structured Generative Models

Beyond colorization, we explore the effect of endowing two different generative models, namely variational auto-encoders and boundary equilibrium generative adversarial network (BEGAN) [240] with a structured output space through our G-CRF formulation. We show the results in Figure A.6 and Figure A.7 using the Toronto Face Dataset (TFD) [242]. Quantitative results are reported in Table A.1 using (1) KL divergence between the distributions of generated and real data, (2) sharpness by gradient magnitude, (3) by edge width and (4) by variance of the Laplacian. The results are normalized with respect to real data measurements.

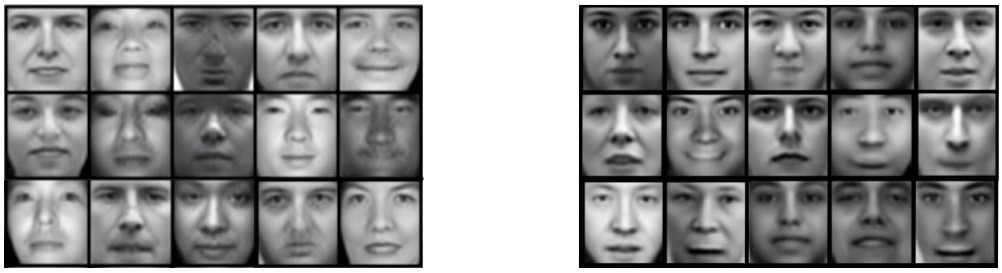
For the variational auto-encoder model, The G-CRF is added on top of the decoder. Additionally, the reconstruction loss is augmented with the feature loss from [243]. We compare our results with the ones obtained from a classical VAE and a VAE trained with the feature loss without the G-CRF layer. Figure A.6 shows that our model results in sharper, higher-quality and more diverse faces.

For the BEGAN model, we add our G-CRF layer on top of the discriminator. Hence, the model implicitly penalizes generated samples which have different statistics than the real samples, at the output layer level. In Figure A.7, we compare our results with the classical BEGAN for the hyper-parameter γ set to 0.5 after approx. 120,000 iterations. We observe our model

to generate diverse and better quality samples.

(a) (b) (c)

Figure A.6: Randomly generated samples from (a) a classical VAE [184], (b) a deep feature consistent VAE [243] and (c) our structured output space feature consistent VAE, trained on TFD.



(a) (b)

Figure A.7: Randomly generated samples from (a) a classical BEGAN [240] and (b) our BEGAN with a structured output space discriminator trained on TFD.

Table A.1: Results of the CRF extension for generative models.

	KLD	*Gradient*	*Edge width*	*Laplacian*
Classical VAE	0.33	66.5%	52.65%	16.3%
Feature Consistent VAE	0.16	91.5%	72.9%	30.17%
S-VAE (Ours)	**0.13**	**97**%.	**95.5**%	**92.**%
BEGAN	0.09	98.54 %	99.4 %	96.37%
S-BEGAN (Ours)	**0.07**	**99.1**%	**100.**%	**98.76**%

APPENDIX B

INFERENCE IN GENERAL GRAPHICAL MODELS USING REINFORCEMENT LEARNING

In Chapter 4, we show that we can learn program heuristics, *i.e.*, policies, for solving inference in higher-order CRFs for the task of semantic segmentation, using reinforcement learning with dense rewards. Our method solves inference tasks efficiently without imposing any constraints on the form of the potentials. In Figure B.1 and Figure B.2, we present further segmentation results of our model on examples from the Pascal VOC and the MOTS datasets respectively. The pairwise potential, together with the superpixel segmentation helped reduce inconsistencies in the unaries obtained from PSPNet/TrackR-CNN across all the examples. HOP1 resulted in better learning the boundaries of the objects. The energy with the HOP2 potential provides the best results across all energies as it helped better segment overlapping objects. We include additional results comparing PSPNet/TrackR-CNN, DQN and MCTS outputs for the energy function with unary, pairwise, HOP1 and HOP2 potentials in Figure B.6 and Figure B.5. The policies trained with DQN/MCTS improve over the PSPNet/TrackR-CNN results across almost all our experiments. Additional failure cases are presented in Figure B.3 and Figure B.4. Figure B.7 presents results further visualizations of the learned policies.

Figure B.1: Output of our method for different potentials on MOTS.

Figure B.2: Output of our method for different potentials on Pascal VOC.

Figure B.3: Additional failure cases on MOTS.

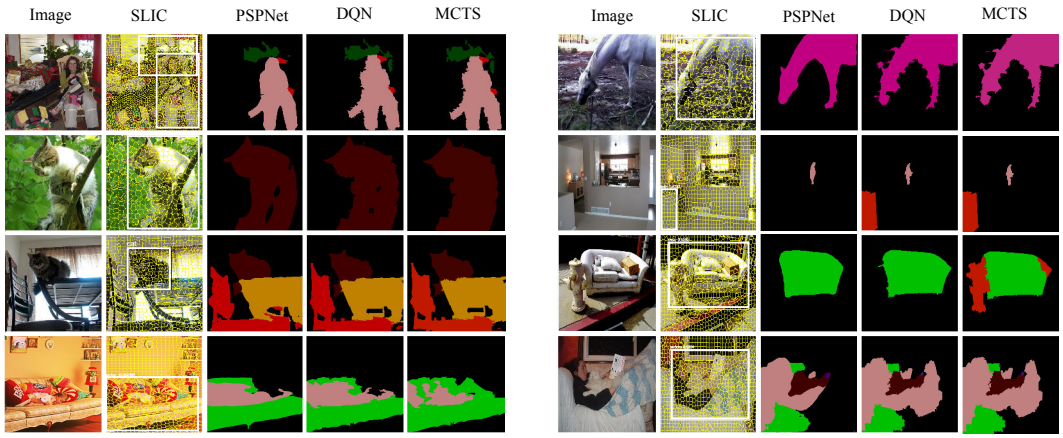


Figure B.4: Additional failure cases on Pascal VOC.

Figure B.5: Additional success cases on MOTS.

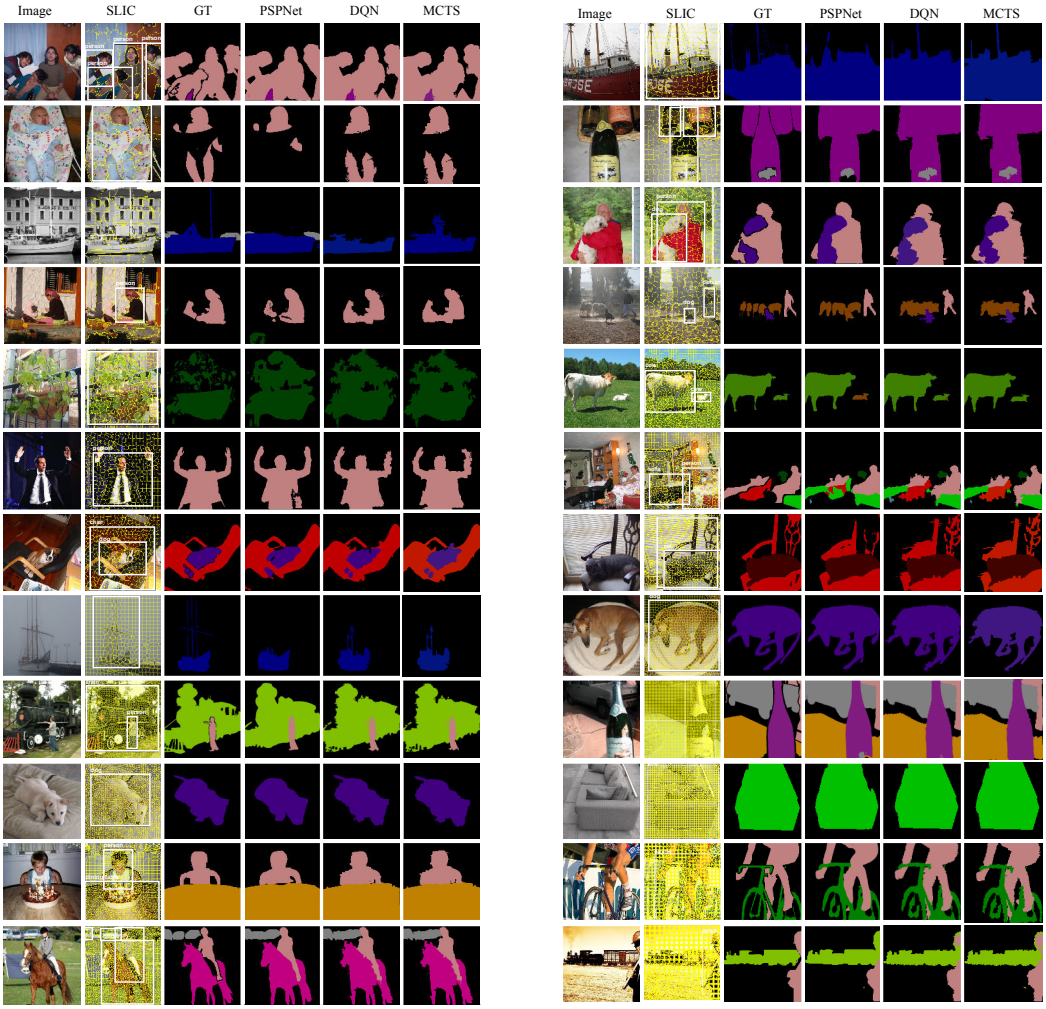


Figure B.6: Additional success cases on Pascal VOC.

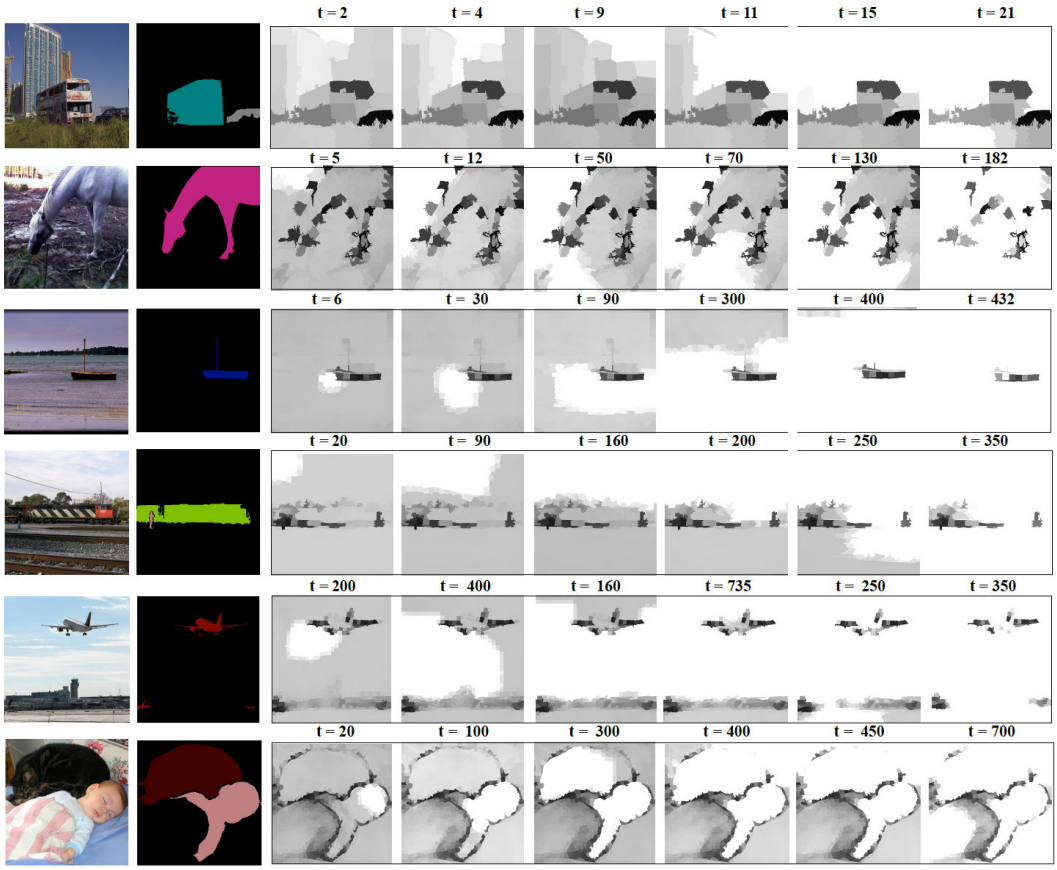


Figure B.7: Visualization of our learned policy.

APPENDIX C

MAX-SLICED SCORE MATCHING FOR LEARNING STRUCTURED MODELS AT SCALE

Sliced score matching (SSM) enabled scaling score matching (SM) to more complex problems and high-dimensional datasets. The main idea: instead of directly matching the high-dimensional scores, we match their projections along random directions, *i.e.*,

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{v}\sim p_{\boldsymbol{v}}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[(\boldsymbol{v}^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{v}^T \boldsymbol{s}_d(\boldsymbol{x}))^2]. \tag{C.1}$$

Here, $p_d(\boldsymbol{x})$ and $p_m(\boldsymbol{x};\boldsymbol{\theta})$ denote the data and the model distributions respectively. Moreover, $\boldsymbol{s}_d(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p_d(\boldsymbol{x})$ and $\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\boldsymbol{\theta})$ are the corresponding scores. Currently, high variance is the major drawback for the SSM loss. In order to address this, we propose Max-Sliced Score Matching (MSSM), a variant of SSM that chooses the most informative projection directions $\boldsymbol{V} \in \mathbb{R}^{d\times d}$ instead of averaging over random ones. We start our derivation by applying the max-slice to the Fisher divergence, *i.e.*,

$$\begin{aligned} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{V}^T(\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{s}_d(\boldsymbol{x}))\|^2], \\ \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^{\boldsymbol{T}} = I. \end{aligned} \tag{C.2}$$

Since the score of the data distribution $\boldsymbol{s}_d(\boldsymbol{x})$ is unknown, our MSSM optimizes an equivalent loss that does not depend on that data score:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \quad & \mathbb{E}_{\boldsymbol{x}\sim p_d}[\sum_{i=1}^{d} \boldsymbol{v}_i^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \boldsymbol{v}_i + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2], \\ \text{s.t.} \quad & \boldsymbol{V}\boldsymbol{V}^{\boldsymbol{T}} = I, \end{aligned} \tag{C.3}$$

where $\boldsymbol{v}_i$ is the i-th column vector of matrix $\boldsymbol{V}$. Note, Equation C.2 and Equation C.3 are the same as Equation 5.1 and Equation 5.2 in the main text.

In the following, we denote by $\mathcal{L}(\boldsymbol{\theta})$ the MSSM loss from Equation C.3, *i.e.*,

$$\mathcal{L}(\boldsymbol{\theta}) = \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{V}^T(\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{s}_d(\boldsymbol{x}))\|^2], \tag{C.4}$$

$J(\boldsymbol{\theta})$ the MSSM loss from Equation C.2, *i.e.*,

$$J(\boldsymbol{\theta}) = \max_{\boldsymbol{V}} \quad \mathbb{E}_{\boldsymbol{x}\sim p_d}[\sum_{i=1}^{d} \boldsymbol{v}_i^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}_i + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2], \tag{C.5}$$

and by $J_N(\boldsymbol{\theta})$ the loss estimator resulting from empirically evaluating $J(\boldsymbol{\theta})$ on a batch size N, *i.e.*,

$$J_N(\boldsymbol{\theta}) = \max_{\boldsymbol{V}} \quad \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{d} \boldsymbol{v}_i^T \nabla_{\boldsymbol{x}^{(j)}} \boldsymbol{s}_m(\boldsymbol{x}^{(j)};\boldsymbol{\theta})\boldsymbol{v}_i + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x}^{(j)};\boldsymbol{\theta})\|^2. \tag{C.6}$$

Here, $\boldsymbol{v}_i$ is the i-th column vector of $\boldsymbol{V}$. We also assume that $p_m(\boldsymbol{x};\boldsymbol{\theta})$ is well specified, and that $\boldsymbol{\theta}^*$ is the parameter of the data distribution, *i.e.*, $p_d(\boldsymbol{x}) = p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$. We denote by $\boldsymbol{\theta}$, $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}_N$ the estimates resulting from minimizing $\mathcal{L}(\boldsymbol{\theta})$, $J(\boldsymbol{\theta})$ and $J_N(\boldsymbol{\theta})$ under the constraint $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}$, respectively, *i.e.*,

$$\boldsymbol{\theta} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \quad \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}, \tag{C.7}$$

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \quad \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}, \tag{C.8}$$

$$\hat{\boldsymbol{\theta}}_N = \operatorname*{argmin}_{\boldsymbol{\theta}} J_N(\boldsymbol{\theta}) \quad \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}. \tag{C.9}$$

We also define $f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)$ as follows:

$$f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*) = \sum_{i=1}^{d} \boldsymbol{v}_i^{*T} \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}_i^* + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}^*)\|^2.$$

Here, $\boldsymbol{V}^*$ is the optimal basis obtained from solving the inner optimization program given in Equation C.2. Hence, $J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_d}[f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)]$ and $J_N(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}[f(\boldsymbol{\theta};\boldsymbol{x}^{(i)},\boldsymbol{V}^*)]$.

In Section C.1, we prove consistency and asymptotic-normality of the MSSM objective. In Section C.2, we derive an equivalent version of the MSSM objective that does not depend on the unknown data score $\boldsymbol{s}_d(\boldsymbol{x})$. We derive the relationship between MSSM loss and MLE loss in Section C.3. Further training details are presented in Section C.4.

C.1 Proof: Consistency and Asymptotic Normality

In the following, we show consistency and asymptotic normality of MSSM. This is important to assess the convergence and asymptotic variance of the estimated parameters around the true parameters.

C.1.1 Consistency

We will show that $\hat{\boldsymbol{\theta}}_N$ is consistent, *i.e.*, as the sample size N increases, the sampling distribution of the estimator $\hat{\boldsymbol{\theta}}_N$ becomes increasingly concentrated at the true parameter value $\boldsymbol{\theta}^*$. Formally,

$$\hat{\boldsymbol{\theta}}_N \xrightarrow{\text{p}} \boldsymbol{\theta}^*, \quad N \to \infty.$$

Here, "$\xrightarrow{\text{p}}$" denotes convergence in probability, *i.e.*, $\lim_{n\to\infty} P(|\hat{\boldsymbol{\theta}}_N - \boldsymbol{\theta}^*| > \boldsymbol{\epsilon}) = 0$, $\forall \boldsymbol{\epsilon} > \mathbf{0}$. The proof follows two steps:

1. First, we prove that $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$, in **Lemma 1** and **Theorem 1**.
2. Second, in **Theorem 2**, we prove that $\hat{\boldsymbol{\theta}}_N \xrightarrow{\text{p}} \hat{\boldsymbol{\theta}}$ when $N \to \infty$. For this, we show that under the assumptions: (a) the parameter space $\boldsymbol{\Theta}$ is compact, (b) $\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\| < \infty$ and (c) $\|\nabla_{\boldsymbol{x}} \log \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|_\infty < \infty$, we have $|f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)| < \infty$. As a result, the conditions in Lemma 2.4 [244] are satisfied. So, $\sup_{\boldsymbol{\theta}} |J_N(\boldsymbol{\theta}) - J(\boldsymbol{\theta})| \xrightarrow{\text{p}} 0$. Thus, from Theorem 2.1 [244], we conclude $\hat{\boldsymbol{\theta}}_N \xrightarrow{\text{p}} \hat{\boldsymbol{\theta}}$.

Lemma 1 *Assume that $p_m(\boldsymbol{x};\boldsymbol{\theta})$ is well specified, i.e., $p_d(\boldsymbol{x}) = p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$, and $p_m(\boldsymbol{x};\boldsymbol{\theta}) \neq p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$ whenever $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$. In the following, we show that, when Equation C.2 converges to 0, $\boldsymbol{\theta}$ converges to $\boldsymbol{\theta}^*$, i.e.,*

$$\max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{V}^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{V}^T \boldsymbol{s}_d(\boldsymbol{x})\|^2] = 0 \iff \boldsymbol{\theta} = \boldsymbol{\theta}^*.$$

Proof: The proof is based on the following sequence of equivalences:

$$
\begin{aligned}
&\max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{V}^T\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{V}^T\boldsymbol{s}_d(\boldsymbol{x})\|^2] = 0, \quad \text{s.t. } \boldsymbol{V}\boldsymbol{V}^T = I \\
\Leftrightarrow &\begin{cases} \max_{\boldsymbol{V}} (\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{s}_d(\boldsymbol{x}))^T \boldsymbol{V}\boldsymbol{V}^T (\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{s}_d(\boldsymbol{x})) = 0, \forall \boldsymbol{x} \sim p_d(\boldsymbol{x}), \\ \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = I \end{cases} \\
\overset{(i)}{\Leftrightarrow} &\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{s}_d(\boldsymbol{x})\|^2 = 0, \quad \forall \boldsymbol{x} \sim p_d(\boldsymbol{x}) \\
\Leftrightarrow &\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{s}_d(\boldsymbol{x}), \quad \forall \boldsymbol{x} \sim p_d(\boldsymbol{x}) \\
\Leftrightarrow &\log p_m(\boldsymbol{x};\boldsymbol{\theta}) = \log p_d(\boldsymbol{x}) + \text{Const.}, \quad \forall \boldsymbol{x} \sim p_d(\boldsymbol{x}) \\
\overset{(ii)}{\Leftrightarrow} &p_d(\boldsymbol{x}) = p_m(\boldsymbol{x};\boldsymbol{\theta}), \quad \forall \boldsymbol{x} \sim p_d(\boldsymbol{x})
\end{aligned}
$$

Note, (i) holds because $\boldsymbol{V}\boldsymbol{V}^T = I$. Further, (ii) holds as $p_m(\boldsymbol{x},\boldsymbol{\theta})$ and $p_d(\boldsymbol{x})$ are normalized probability density functions. Finally, $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ is a simple proof by contradiction. Suppose $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$ is true, then it implies $p_d(\boldsymbol{x}) \neq p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$ according to our assumption. This is not true, so $\boldsymbol{\theta} = \boldsymbol{\theta}^*$.

Theorem 1 *Assume the results from Lemma 1. Given the equivalence Equation C.2 $\iff$ Equation C.3 (Section C.2), the equality $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$ holds.*

Proof: In Lemma 1, we proved that $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ holds. Besides, in Section C.2, we show that the programs in Equation C.2 and Equation C.3 are equivalent. This results in the equivalence $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$. As a result, $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$.

Theorem 2 *Assume, (1) the parameter space $\boldsymbol{\Theta}$ is compact, (2) $f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)$ is continuous, (3) $\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\| < \infty$ and (4) $\|\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|_\infty < \infty$. Then, $\hat{\boldsymbol{\theta}}_N \xrightarrow{p} \hat{\boldsymbol{\theta}}$.*

Proof: We start by showing that $|f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)| < \infty$. Given that $\|\boldsymbol{v}_j\|^2 = 1$ $\forall j \in \{1,\cdots,K\}$, then $|\boldsymbol{v}_j^{(k)}\boldsymbol{v}_j^{(l)}| < 1, \forall k,l \in \{1,\cdots,d\}$. Since $\|\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\| < \infty$, we conclude $|\boldsymbol{v}_j^{*T}\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}_j^*| < \infty$, $\forall j$. Besides, since $\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 < \infty$, we conclude that $|f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)| < \infty$. Hence, all the assumptions for Lemma 2.4 [244] are satisfied. So, $\sup_{\boldsymbol{\theta}} |J_N(\boldsymbol{\theta}) - J(\boldsymbol{\theta})| \xrightarrow{\text{p}} 0$. Thus, from Theorem 2.1 [244], we deduce $\hat{\boldsymbol{\theta}}_N \xrightarrow{\text{p}} \hat{\boldsymbol{\theta}}$.

C.1.2 Asymptotic Normality

We start by introducing some notation. We denote by $\boldsymbol{H}_{ij}$ a matrix that depends on $p_m(\boldsymbol{x};\boldsymbol{\theta})$ as follows:

$$\boldsymbol{H}_{ij} = \mathbb{E}_{p_d}[(\nabla_{\boldsymbol{\theta}}[\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_{ii} + \frac{1}{2}\nabla_{\boldsymbol{\theta}}([\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_i)^2) \\ (\nabla_{\boldsymbol{\theta}}[\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_{jj} + \frac{1}{2}\nabla_{\boldsymbol{\theta}}([\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_j)^2)^T],$$

where, $[\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})]_{ii}$ is the ii-th entry of the matrix $\nabla^2_{\boldsymbol{x}} p_m(\boldsymbol{x};\boldsymbol{\theta})$ and $[\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})]_i$ is the i-th entry of the vector $\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})$. Also, we denote $\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}$ and $J(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}$ as $\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)$, and $J(\boldsymbol{\theta}^*)$, respectively.

In the following, we show that $\hat{\boldsymbol{\theta}}_N$ is asymptotically normally distributed around the true parameter $\boldsymbol{\theta}^*$ when the batch size N becomes very large, *i.e.*,

$$\sqrt{N}(\hat{\boldsymbol{\theta}}_N - \boldsymbol{\theta}^*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}).$$

Here "$\xrightarrow{d}$" denotes a convergence in distribution and $\boldsymbol{\Sigma}$ is the covariance matrix. Specifically, we prove the theorem:

Theorem 3 *If (1) $\boldsymbol{\Theta}$ is compact, (2) $\hat{\boldsymbol{\theta}}_N \xrightarrow{p} \boldsymbol{\theta}^*$, (3) $\log p_m(\boldsymbol{x};\boldsymbol{\theta})$ is twice continuously differentiable with respect to $\boldsymbol{\theta}$, (4) $J(\boldsymbol{\theta}^*)$ is invertible at $\boldsymbol{\theta}^*$, (5) $\|\nabla^2_{\boldsymbol{\theta}}\|\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})\|^2\|_\infty < \infty$, and (6) $\|\nabla^2_{\boldsymbol{\theta}}[\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x};\boldsymbol{\theta})]_{ii}\|_\infty < \infty$, $\forall i \in \{1,\cdots,d\}$, then,*

$$\sqrt{N}(\hat{\boldsymbol{\theta}}_N - \boldsymbol{\theta}^*) \xrightarrow{d} \mathcal{N}\Big(\mathbf{0}, [\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*)]^{-1}\Big(\sum_{i,j=1}^{d} \boldsymbol{H}_{ij}\Big)[\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*)]^{-1}\Big).$$

The proof is based on Theorem 3.1 [244]. To apply it, we first prove the following:

1. $\sqrt{N}\nabla_{\boldsymbol{\theta}} J_N(\boldsymbol{\theta}^*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \sum_{i,j=1}^{d} \boldsymbol{H}_{ij})$, in **Lemma 2**.
2. $\sup_{\boldsymbol{\theta}} |\nabla^2_{\boldsymbol{\theta}} J_N(\boldsymbol{x};\boldsymbol{\theta}) - \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{x};\boldsymbol{\theta})| \xrightarrow{p} \mathbf{0}$ in **Lemma 3**. Note that the absolute value is applied element wise to all the elements of the matrix.

Lemma 2 *If $\log p_m(\boldsymbol{x};\boldsymbol{\theta})$ is twice continuously differentiable with respect to $\boldsymbol{\theta}$, then, $\sqrt{N}\nabla_{\boldsymbol{\theta}} J_N(\boldsymbol{\theta}^*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \sum_{i,j=1}^{d} \boldsymbol{H}_{ij})$.*

Proof: The central limit theorem gives,

$$\sqrt{N}\sum_{i=1}^{d}\nabla_{\boldsymbol{\theta}}f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)\xrightarrow{d}\mathcal{N}(\mathbf{0},\mathrm{Var}[\nabla_{\boldsymbol{\theta}}f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)]).$$

Note that $\mathbb{E}[\nabla_{\boldsymbol{\theta}}f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V})]=0$ as $\boldsymbol{\theta}^*$ is the optimal parameter for $f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)$. We compute $\mathrm{Var}[\nabla_{\boldsymbol{\theta}}f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)]$ as follows:

$$\begin{aligned}
&\mathbb{E}_{p_d}[\nabla_{\boldsymbol{\theta}}f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)\nabla_{\boldsymbol{\theta}}f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)^T]\\
&=\mathbb{E}_{p_d}[(\sum_{i=1}^{d}\nabla_{\boldsymbol{\theta}}\boldsymbol{v}_i^T\nabla_{\boldsymbol{x}}^2\boldsymbol{p}_m(\boldsymbol{x};\boldsymbol{\theta}^*)\boldsymbol{v}_i+\frac{1}{2}\nabla_{\boldsymbol{\theta}}\|\nabla_{\boldsymbol{x}}\boldsymbol{p}_m(\boldsymbol{x};\boldsymbol{\theta}^*)\|^2)(\sum_{j=1}^{d}\nabla_{\boldsymbol{\theta}}\boldsymbol{v}_j^T\nabla_{\boldsymbol{x}}^2\boldsymbol{p}_m(\boldsymbol{x};\boldsymbol{\theta}^*)\boldsymbol{v}_j+\\
&\frac{1}{2}\nabla_{\boldsymbol{\theta}}\|\nabla_{\boldsymbol{x}}\boldsymbol{p}_m(\boldsymbol{x};\boldsymbol{\theta}^*)\|^2)^T]\\
&=\mathbb{E}_{p_d}[(\sum_{i=1}^{d}\nabla_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{x}}^2\log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_i+\frac{1}{2}\nabla_{\boldsymbol{\theta}}([\nabla_{\boldsymbol{x}}\log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_i)^2)(\sum_{j=1}^{d}\frac{1}{2}\nabla_{\boldsymbol{\theta}}([\nabla_{\boldsymbol{x}}\log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_j)^2\\
&+\nabla_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{x}}^2\log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)]_j)^T]\\
&=\sum_{i,j}^{d}H_{ij}
\end{aligned}$$

Hence, $\sqrt{N}\nabla_{\boldsymbol{\theta}}J_N(\boldsymbol{\theta}^*)\xrightarrow{d}\mathcal{N}(\mathbf{0},\sum_{i,j=1}^{d}\boldsymbol{H}_{ij})$.□

Lemma 3 *Assuming $\boldsymbol{\Theta}$ is compact and $p_m(\boldsymbol{x};\boldsymbol{\theta})$ being twice continuously differentiable with respect to $\boldsymbol{\theta}$, then*

$$\sup_{\boldsymbol{\theta}}|\nabla_{\boldsymbol{\theta}}^2J_N(\boldsymbol{\theta}^*)-\nabla_{\boldsymbol{\theta}}^2J(\boldsymbol{\theta}^*)|\xrightarrow{p}\mathbf{0}.$$

Proof: We write down the second order derivative of $f(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{V}^*)$ as:

$$\nabla_{\boldsymbol{\theta}}^2f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)=\sum_{i=1}^{d}\nabla_{\boldsymbol{\theta}}^2[\boldsymbol{v}_i^{*T}\nabla_{\boldsymbol{x}}^2\log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)\boldsymbol{v}_i^*]+\frac{1}{2}\nabla_{\boldsymbol{\theta}}^2\|\nabla_{\boldsymbol{x}}\log p_m(\boldsymbol{x};\boldsymbol{\theta}^*)\|^2$$

By assumption, we have (1) $\|\nabla_{\boldsymbol{\theta}}^2[\nabla_{\boldsymbol{x}}^2\log p_m(\boldsymbol{x};\boldsymbol{\theta})]_{ii}\|_\infty<\infty$, as well as (2) $\|\nabla_{\boldsymbol{\theta}}^2\|\nabla_{\boldsymbol{x}}\log p_m(\boldsymbol{x};\boldsymbol{\theta})\|^2\|_\infty<\infty$. Hence, $\|\nabla_{\boldsymbol{\theta}}^2f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)\|<\infty$. The assumptions for Lemma 2.4 [244] are satisfied. So,

$$\sup_{\boldsymbol{\theta}}|\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{\theta}}^2f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)-\mathbb{E}_{\boldsymbol{x}\sim p_d}[\nabla_{\boldsymbol{\theta}}^2f(\boldsymbol{\theta}^*;\boldsymbol{x},\boldsymbol{V}^*)]|\xrightarrow{p}\mathbf{0},$$

which is equivalent to:

$$\sup_{\boldsymbol{\theta}} |\nabla^2_{\boldsymbol{\theta}} J_N(\boldsymbol{\theta}^*) - \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*)| \xrightarrow{p} \mathbf{0}.\square$$

Combining **Theorem 3** assumptions with **Lemma 2** and **Lemma 3** results, satisfies all the assumptions of Theorem 3.1 [244]. Hence,

$$\sqrt{N}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*) \to \mathcal{N}\Big(\mathbf{0}, (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1} \left(\sum_{i,j=1}^{d} \boldsymbol{H}_{ij}\right) (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1}\Big).\square \quad \text{(C.10)}$$

Remark: As a result, the asymptotic variance of MSSM is the same as SM's asymptotic variance and smaller than the one for SSM. Proofs are presented in Appendix B.4 by Song *et al.* [88]:

- **SSM** with projection directions following a normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$:

$$\sqrt{N}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*) \to \mathcal{N}\Big(\mathbf{0}, (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1} \left(\sum_{i,j=1}^{d} H_{ij} + \frac{2}{M}\left(\sum_{i=1}^{d} H_{ii} + \sum_{\substack{i,j=1 \\ i\neq j}}^{d} W_{ij}\right)\right) (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1}\Big).$$

Here,

$$W_{ij} = \mathbb{E}_{p_d}[(\nabla_{\boldsymbol{\theta}}[\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta}^*)]_{ij} + \frac{1}{2}\nabla_{\boldsymbol{\theta}}([\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta}^*)]_i [\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta}^*)]_j))$$
$$(\nabla_{\boldsymbol{\theta}}[\nabla^2_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta}^*)]_{ij} + \frac{1}{2}\nabla_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta}^*)]_i [\nabla_{\boldsymbol{x}} \log p_m(\boldsymbol{x}; \boldsymbol{\theta}^*)]_j)^T].$$

- **SSM** with projection directions following a Rademacher distribution:

$$\sqrt{N}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*) \to \mathcal{N}\Big(\mathbf{0}, (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1} \left(\sum_{i,j=1}^{d} H_{ij} + \frac{2}{M} \sum_{\substack{i,j=1 \\ i\neq j}}^{d} W_{ij}\right) (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1}\Big).$$

- **SM** (Equation 2.33):

$$\sqrt{N}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*) \to \mathcal{N}\Big(\mathbf{0}, (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1} \left(\sum_{i,j=1}^{d} H_{ij}\right) (\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^*))^{-1}\Big).$$

C.2 Proof: Equation C.2 $\Leftrightarrow$ Equation C.3

We assume that the model's and data score functions, *i.e.*, $\boldsymbol{s}_m(\boldsymbol{x}, \boldsymbol{\theta})$ and $\boldsymbol{s}_d(\boldsymbol{x})$ respectively, are differentiable and satisfy $\mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{s}_d(\boldsymbol{x})\|^2] < \infty$ and $\mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2] < \infty$. Besides, we assume that the boundary condition holds, *i.e.*, $\lim_{x\to+\infty} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})p_d(\boldsymbol{x}) = 0$, $\forall\boldsymbol{\theta}$. In the following, we prove that the MSSM objective

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \quad \mathbb{E}_{\boldsymbol{x}\sim p_d}[\|\boldsymbol{V}^T(\boldsymbol{s}_d(\boldsymbol{x}) - \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))\|^2], \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^{\boldsymbol{T}} = I, \tag{C.11}$$

is equivalent to:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \quad & \mathbb{E}_{\boldsymbol{x}\sim p_d}[\boldsymbol{V}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{V} + \frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2], \\ \text{s.t.} \quad & \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}. \end{aligned}$$

Proof: We show how to derive a loss that doesn't depend on the unknown distribution $p_d(\boldsymbol{x})$ using: (1) the constraint $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}$ and (2) Integration by parts (cf. Lemma 4 by Hyvärinen *et al.* [83]):

$$\begin{cases} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\frac{1}{2}\|\boldsymbol{V}^T\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{w}) - \boldsymbol{V}^T\boldsymbol{s}_d(\boldsymbol{x})\|^2], \\ \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I} \end{cases}$$

$$\overset{(i)}{\Leftrightarrow} \begin{cases} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \frac{1}{2}\|\boldsymbol{s}_d(\boldsymbol{x})\|^2 - (\boldsymbol{V}^T\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))^T(\boldsymbol{V}^T\boldsymbol{s}_d(\boldsymbol{x}))], \\ \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I} \end{cases}$$

$$\overset{(ii)}{\Leftrightarrow} \begin{cases} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \frac{1}{2}\|\boldsymbol{s}_d(\boldsymbol{x})\|^2 + \sum_{j=1}^{d} \boldsymbol{v}_j^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}_j], \\ \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I} \end{cases}$$

$$\overset{(iii)}{\Leftrightarrow} \begin{cases} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \sum_{j=1}^{d} \boldsymbol{v}_j^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}_j], \\ \text{s.t.} \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}. \end{cases}$$

Note, (i) is due to the fact that $\boldsymbol{V}\boldsymbol{V}^T = I$. Further, (iii) is due to the fact that $\boldsymbol{s}_d(\boldsymbol{x})$ does not depend on either $\boldsymbol{\theta}$ or $\boldsymbol{V}$. Moreover, (ii) is due to the

integration by parts trick, as we show in the following:

$$
\begin{aligned}
&\max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\sum_{j=1}^{d}(\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^T \boldsymbol{s}_d(\boldsymbol{x}))] \\
\iff &\max_{\boldsymbol{V}} \sum_{j=1}^{d} \int_{\boldsymbol{x}} p_d(\boldsymbol{x})(\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^T \boldsymbol{s}_d(\boldsymbol{x}))\boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} \sum_{j=1}^{d} \int_{\boldsymbol{x}} p_d(\boldsymbol{x})(\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^T \nabla_{\boldsymbol{x}} \log p_d(\boldsymbol{x}))\boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} \sum_{j=1}^{d} \int_{\boldsymbol{x}} (\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^T \nabla_{\boldsymbol{x}} p_d(\boldsymbol{x}))\boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} \sum_{j=1}^{d} \int_{\boldsymbol{x}} (\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^T \nabla_{\boldsymbol{x}} p_d(\boldsymbol{x}))\boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} \sum_{j=1}^{d}\sum_{i=1}^{d} \int_{\boldsymbol{x}} (\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^{(i)} \frac{\partial p_d(\boldsymbol{x})}{\partial \boldsymbol{x}_i})\boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} \sum_{j=1}^{d}\sum_{i=1}^{d} \int_{\boldsymbol{x}} (\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^{(i)} \frac{\partial p_d(\boldsymbol{x})}{\partial \boldsymbol{x}_i})\boldsymbol{dx} \\
\overset{(iv)}{\iff} &\max_{\boldsymbol{V}} \sum_{j=1}^{d}\sum_{i=1}^{d} \Big[(\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^{(i)} p_d(\boldsymbol{x}))\Big]_{-\infty}^{+\infty} - \int_{\boldsymbol{x}} \boldsymbol{v}_j^{(i)} p_d(\boldsymbol{x}) \boldsymbol{v}_j^T \frac{\partial \boldsymbol{s}_m(\boldsymbol{x})}{\partial \boldsymbol{x}_i}\boldsymbol{dx} \\
\overset{(v)}{\iff} &\max_{\boldsymbol{V}} \sum_{j=1}^{d}\sum_{i=1}^{d} - \int_{\boldsymbol{x}} \boldsymbol{v}_j^{(i)} p_d(\boldsymbol{x}) \boldsymbol{v}_j^T \frac{\partial \boldsymbol{s}_m(\boldsymbol{x})}{\partial \boldsymbol{x}_i}\boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} - \sum_{j=1}^{d} \int_{\boldsymbol{x}} p_d(\boldsymbol{x}) \boldsymbol{v}_j^T \nabla_{\boldsymbol{x}} s_m(\boldsymbol{x}) \boldsymbol{v}_j \boldsymbol{dx} \\
\iff &\max_{\boldsymbol{V}} - \sum_{j=1}^{d} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\boldsymbol{v}_j^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x}) \boldsymbol{v}_j]
\end{aligned}
$$

Note, (iv) is due to partial integration. Further, (v) is the boundary condition, *i.e.*, $p_d(\boldsymbol{x})$ vanishes at infinity. Hence,

$$\max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[\sum_{j=1}^{d}(\boldsymbol{v}_j^T \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))(\boldsymbol{v}_j^T \boldsymbol{s}_d(\boldsymbol{x}))] \Leftrightarrow \max_{\boldsymbol{V}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[- \sum_{j=1}^{d} \boldsymbol{v}_j^T \nabla_{\boldsymbol{x}} \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) \boldsymbol{v}_j].$$

C.3 Proof: MSSM Relationship to MLE

As explained by Hyvarinen *et al.* [245], the objective: $\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}}[\frac{1}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \operatorname{tr}(\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))$, can be derived using the 2^{nd}-order Taylor approximation of the contrastive divergence loss:

$$L_{\text{CD}}(\boldsymbol{x};\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[\log p_m(\boldsymbol{x};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{x'}|\boldsymbol{x}}[\log p_m(\boldsymbol{x'};\boldsymbol{\theta})]],$$

with a 1-step Langevin Monte-Carlo as the sampler from the model distribution $p_m(\boldsymbol{x};\boldsymbol{\theta})$. The Langevin Monte-Carlo sampler generates samples via:

$$\boldsymbol{x'} = \boldsymbol{x} + \frac{\mu}{2}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) + \sqrt{\mu}\boldsymbol{\sigma}, \quad \boldsymbol{\sigma} \sim \mathcal{N}(0, I).$$

The Taylor expansion gives:

$$\begin{aligned}
\log p_m(\boldsymbol{x'};\boldsymbol{\theta}) &= \log p_m(\boldsymbol{x};\boldsymbol{\theta}) + \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})^T(\boldsymbol{x}-\boldsymbol{x'}) + (\boldsymbol{x'}-\boldsymbol{x})^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})(\boldsymbol{x'}-\boldsymbol{x}) \\
&= \log p_m(\boldsymbol{x};\boldsymbol{\theta}) + \boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})^T\left(\frac{\mu}{2}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) + \sqrt{\mu}\boldsymbol{\sigma}\right) + \\
&\quad \left(\frac{\mu}{2}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) + \sqrt{\mu}\boldsymbol{\sigma}\right)^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\left(\frac{\mu}{2}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}) + \sqrt{\mu}\boldsymbol{\sigma}\right).
\end{aligned}$$

Leveraging the fact that $\mathbb{E}[\boldsymbol{\sigma}] = 0$ and $\mathbb{E}[\boldsymbol{\sigma}\boldsymbol{\sigma}^T] = I$, we obtain:

$$\mathbb{E}_{\boldsymbol{x'}|\boldsymbol{x}}[p_m(\boldsymbol{x'};\boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[\log p_m(\boldsymbol{x};\boldsymbol{\theta}) + \frac{\mu}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \mu\operatorname{tr}(\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})) + \frac{\mu^2}{4}\boldsymbol{s}_m^T(\boldsymbol{x};\boldsymbol{\theta})\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})].$$

Hence, Equation C.3 becomes: $L_{\text{CD}}(\boldsymbol{x};\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[\frac{\mu}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \mu\ \operatorname{tr}(\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})) + \frac{\mu^2}{4}\boldsymbol{s}_m^T(\boldsymbol{x};\boldsymbol{\theta})\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})]$, resulting in,

$$L_{\text{CD}}(\boldsymbol{x};\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[\frac{\mu}{2}\|\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \mu\ \operatorname{tr}(\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})) + \mathcal{O}(\mu^2)].$$

SSM approximates the trace operator via the Hutchinson's trick, *i.e.*, the trace operator $\operatorname{tr}(\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta}))$ is replaced with $\mathbb{E}_{p_{\boldsymbol{v}}}[\boldsymbol{v}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}]$, such that $\mathbb{E}_{p_{\boldsymbol{v}}}[\boldsymbol{v}\boldsymbol{v}^T] = \boldsymbol{I}$. In case the projection directions follow a Rademacher distribution, *i.e.*, $\boldsymbol{v} \in \{-1, 1\}^d$, 2^d vectors are needed to exactly estimate the trace operator. However for MSSM, the equality $\operatorname{tr}(\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})) = \sum_{i=1}^{d}\boldsymbol{v}^T\nabla_{\boldsymbol{x}}\boldsymbol{s}_m(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{v}$ holds when only considering d vectors.

C.4 Training Details

We closely follow the training setup of Song *et al.* [88] for the different applications. In the following, we derive a closed form expression for the weight parameters $\boldsymbol{\alpha} = \{\alpha_l\}_{l=1}^L$ of the Kernel Exponential Families (DKEF) model, in case of the MSSM loss. DKEF parameterizes the unnormalized log-likelihood via $\log p_f(\boldsymbol{x}) = \log q_0(\boldsymbol{x}) + \sum_{l=1}^L \alpha_l k(\boldsymbol{x}, \boldsymbol{z}_l)$, where $q_0(\boldsymbol{x})$ is a fixed function, and $k(\boldsymbol{x}, \boldsymbol{z}_l)$ is a mixture of R Gaussian kernels defined on the feature space $\boldsymbol{\phi}_r$ of a deep net and evaluated at L different inducing points $\boldsymbol{z}_l$:

$$k(\boldsymbol{x}, \boldsymbol{z}_l) = \sum_{r=1}^{R} \rho_r \exp\left(\frac{-1}{2\sigma_r^2}\|\boldsymbol{\phi}_r(\boldsymbol{x}) - \boldsymbol{\phi}_r(\boldsymbol{z}_l)\|^2\right).$$

The kernel parameters $\{\rho_r\}_{r=1}^R$,$\{\sigma_r\}_{r=1}^R$, $\{\boldsymbol{\phi}_r\}_{r=1}^R$ and the inducing points $\{\boldsymbol{z}_l\}_{l=1}^L$ are learned via gradient descent. We prove that for fixed ρ_r, σ_r, $\boldsymbol{\phi}_r$, $\boldsymbol{z}_l$ and $\boldsymbol{V}^*$, finding $\boldsymbol{\alpha}$ that minimizes the program,

$$\min_{\boldsymbol{\alpha}} \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d} \boldsymbol{v}_j^{*T} \nabla^2_{\boldsymbol{x}^{(i)}} \log p_f(\boldsymbol{x}^{(i)}) \boldsymbol{v}_j^*}_{J_1} + \underbrace{\frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}\|\nabla_{\boldsymbol{x}} \log p_f(\boldsymbol{x}^{(i)})\|^2}_{J_2} + \frac{1}{2}\lambda_{\boldsymbol{\alpha}}\|\boldsymbol{\alpha}\|^2,$$

is equivalent to solving a linear system:

$$(\boldsymbol{G} + \lambda_{\boldsymbol{\alpha}} \boldsymbol{I})\boldsymbol{\alpha} = -\boldsymbol{b} - \boldsymbol{c},$$

with $\lambda_{\boldsymbol{\alpha}}$ being a regularization parameters, and $\boldsymbol{G} \in \mathbb{R}^{L\times L}$ and $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^L$ are defined as follows:

$$\begin{aligned}
G_{l,l'} &= \frac{1}{N}\sum_{i=1}^{N} (\nabla_{\boldsymbol{x}^{(i)}} k(\boldsymbol{x}^{(i)}, \boldsymbol{z}_l))^T (\nabla_{\boldsymbol{x}^{(i)}} k(\boldsymbol{x}^{(i)}, \boldsymbol{z}_{l'})), \quad \forall l, l' \in [1, L] \\
b_l &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d} \boldsymbol{v}_j^{*T} \nabla^2_{\boldsymbol{x}^{(i)}} k(\boldsymbol{x}^{(i)}, \boldsymbol{z}_l) \boldsymbol{v}_j^*, \quad \forall l \in [1, L] \\
c_l &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d} \nabla_{\boldsymbol{x}^{(i)}} q_0(\boldsymbol{x}^{(i)})^T k(\boldsymbol{x}^{(i)}, \boldsymbol{z}), \quad \forall l \in [1, L].
\end{aligned}$$

Note, N is the batch size and d is the number of eigenvectors.

Proof:
In the following, we express the quadratic (J_1) and squared norm (J_2) terms of the loss in Equation C.4 as functions of $\boldsymbol{\alpha}$, $\boldsymbol{G}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. Starting with J_1:

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d}\boldsymbol{v}_j^{*T}\nabla_{\boldsymbol{x}}^2\log p_f(\boldsymbol{x})\boldsymbol{v}_j^* &= \frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{L}\sum_{j=1}^{d}\alpha_l\boldsymbol{v}_j^{*T}\nabla_{\boldsymbol{x}^{(i)}}k(\boldsymbol{x}^{(i)},\boldsymbol{z}_l)\boldsymbol{v}_j^* + \text{Const}\\ &= \sum_{l=1}^{L}\alpha_l b_l + \text{Const}\\ &= \boldsymbol{\alpha}^T\boldsymbol{b} + \text{Const.}\end{aligned}$$

Next, we express J_2 as a function of $\boldsymbol{\alpha}$, $\boldsymbol{G}$ and $\boldsymbol{c}$:

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|\nabla_{\boldsymbol{x}}\log p_f(\boldsymbol{x})\|^2\\ &=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|\sum_{l=1}^{L}\alpha_l\nabla_{\boldsymbol{x}^{(i)}}k(\boldsymbol{x}^{(i)},\boldsymbol{z}_l)+\log q_0(\boldsymbol{x}^{(i)})\|^2\\ &=\frac{1}{2N}\sum_{i=1}^{N}\Big(\sum_{l,l'}^{L}\alpha_l\alpha_{l'}\nabla_{\boldsymbol{x}^{(i)}}k(\boldsymbol{x}^{(i)},\boldsymbol{z}_l)^T\nabla_{\boldsymbol{x}^{(i)}}k(\boldsymbol{x}^{(i)},\boldsymbol{z}_{l'})+2\sum_{l=1}^{L}\alpha_l\nabla_{\boldsymbol{x}^{(i)}}k(\boldsymbol{x}^{(i)},\boldsymbol{z}_l)^T\\ &\nabla_{\boldsymbol{x}^{(i)}}q_0(\boldsymbol{x}^{(i)})\Big)+\text{Const}\\ &=\frac{1}{2}\sum_{l,l'}^{L}\alpha_l\alpha_{l'}G_{l,l'}+\sum_{l=1}^{L}\alpha_l c_l+\text{Const}\\ &=\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{G}\boldsymbol{\alpha}+\boldsymbol{\alpha}^T\boldsymbol{c}+\text{Const.}\end{aligned}$$

Hence, the optimization problem in Equation C.4 becomes:

$$\begin{aligned}\boldsymbol{\alpha}&=\underset{\boldsymbol{\alpha}}{\text{argmin}}\,\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d}[\boldsymbol{v}_j^{*T}\nabla_{\boldsymbol{x}^{(i)}}^2\log p_f(\boldsymbol{x}^{(i)})\boldsymbol{v}_j^*+\frac{1}{2}\|\nabla_{\boldsymbol{x}}\log p_f(\boldsymbol{x}^{(i)})\|^2]+\frac{1}{2}\lambda_{\boldsymbol{\alpha}}\|\boldsymbol{\alpha}\|^2\\ &=\underset{\boldsymbol{\alpha}}{\text{argmin}}\,\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{G}\boldsymbol{\alpha}+\boldsymbol{\alpha}^T\boldsymbol{b}+\boldsymbol{\alpha}^T\boldsymbol{c}+\frac{1}{2}\lambda_{\boldsymbol{\alpha}}\|\boldsymbol{\alpha}\|^2\\ &=-(\boldsymbol{G}+\lambda_{\boldsymbol{\alpha}}\boldsymbol{I})^{-1}(\boldsymbol{b}+\boldsymbol{c}).\end{aligned}$$

REFERENCES

[1] S. Messaoud, D. Forsyth, and A. Schwing, "Structural consistency and controllability for diverse colorization," in *Procs. ECCV*, 2018.

[2] A. Senior, J. Jumper, D. Hassabis, and P. Kohli, "Alphafold: Using artificial intelligence for scientific discovery." [Online]. Available: https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery

[3] J. Huang, Y. Xie, F. Yu, Q. Ke, M. Abadi, E. Gillum, and Z. M. Mao, "Socialwatch: Detection of online service abuse via large-scale social graphs," in *Procs. AsiaCCS*, 2013.

[4] R. Zafarani, M. A. Abbasi, and H. Liu, *Social Media Mining: An Introduction*. Cambridge University Press, 2014.

[5] M. V. Devarakonda, S. Messaoud, and C.-H. Tsou, "Medical record problem list generation," U.S. Patent WO/2018/220550, Dec. 2018.

[6] C.-H. Tsou, S. Messaoud, J. J. Liang, and M. V. Devarakonda, "Automatic problem identification by disease category using deep learning," IBM Research, Tech. Rep., 2016.

[7] S. Messaoud, Y. Varatharajah, M. A. Younkin, X. Wang, A. Athreya, R. Iyer, and N. Ertekin-Taner, "Unsupervised analysis of transcriptomic data for demystifying the cause of Alzheimer's disease," *Alzheimer's & Dementia*, 2017.

[8] S. Messaoud and T. Ogasawara, "Accelerating genomic data parsing on field programmable gate arrays," US Patent App. 16/668,166, Feb. 2020.

[9] S. Messaoud, "Accelerating SAM to BAM parsing on FPGA," IBM Research Tokyo, Tech. Rep., 2015.

[10] M. J. Wainwright and M. I. Jordan, *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[11] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[12] R. Bellman, "Dynamic programming," *Science*, 1966.

[13] J. S. Yedidia, "Message-passing algorithms for inference and optimization," *J. Stat. Phys.*, 2011.

[14] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *PAMI*, 2004.

[15] D. Schlesinger and B. Flach, "Transforming an arbitrary minsum problem into a binary one," TU Dresden, Tech. Rep., 2006.

[16] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *PAMI*, 2001.

[17] C. Rother, V. Kolmogorov, and A. Blake, "GrabCut: interactive foreground extraction using iterated graph cuts," *TOG*, 2004.

[18] Y. Li, J. Sun, and H.-Y. Shum, "Video object cut and paste," in *Procs. SIGGRAPH*, 2005.

[19] R. Gupta, A. A. Diwan, and S. Sarawagi, "Efficient inference with cardinality-based clique potentials," in *Procs. ICML*, 2007.

[20] D. Tarlow, I. Givoni, and R. Zemel, "Hop-map: Efficient message passing with high order potentials," in *Procs. ICAIS*, 2010.

[21] P. Kohli, P. H. Torr et al., "Robust higher-order potentials for enforcing label consistency," *IJCV*, 2009.

[22] D. Tarlow, G. Elidan, D. Koller, and J. C. Duchi, "Using combinatorial optimization within max-product belief propagation," in *Procs. NeurIPS*, 2007.

[23] F. Amat, F. Moussavi, L. R. Comolli, G. Elidan, K. H. Downing, and M. Horowitz, "Markov random field based automatic image alignment for electron tomography," *JSB*, 2008.

[24] A. Lodi and G. Zarpellon, "On learning and branching: A survey," *TOP*, 2017.

[25] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to run heuristics in tree search," in *Procs. IJCAI*, 2017.

[26] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Procs. AAAI*, 2016.

[27] M. Shlezinger, "Syntactic analysis of two-dimensional visual signals in the presence of noise," *Cybernetics*, 1976.

[28] A. M. Koster, S. P. Van Hoesel, and A. W. Kolen, "The partial constraint satisfaction problem: Facets and lifting theorems," *Oper. Res. Lett.*, 1998.

[29] C. Chekuri, S. Khanna, J. Naor, and L. Zosin, "A linear programming formulation and approximation algorithms for the metric labeling problem," *SIAM*, 2004.

[30] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *PAMI*, 2006.

[31] A. Globerson and T. S. Jaakkola, "Approximate inference using planar graph decomposition," in *Procs. NeurIPS*, 2007.

[32] T. Meltzer, A. Globerson, and Y. Weiss, "Convergent message passing algorithms: A unifying view," *UAI*, 2009.

[33] D. Sontag and T. Jaakkola, "Tree block coordinate descent for map in graphical models," in *Procs. AISTATS*, 2009.

[34] T. Hazan, J. Peng, and A. Shashua, "Tightening fractional covering upper bounds on the partition function for high-order region graphs," in *UAI*, 2012.

[35] D. Tarlow, D. Batra, P. Kohli, and V. Kolmogorov, "Dynamic tree block coordinate ascent," in *Procs. ICML*, 2011.

[36] J. K. Johnson, "Convex relaxation methods for graphical models: Lagrangian and maximum entropy approaches," Ph.D. dissertation, 2008.

[37] V. Jojic, S. Gould, D. Koller et al., "Accelerated dual decomposition for map inference," in *Procs. ICML*, 2010.

[38] P. Ravikumar, A. Agarwal, and M. J. Wainwright, "Message-passing for graph-structured linear programs: Proximal methods and rounding schemes," *JMLR*, 2010.

[39] T. Werner, "Revisiting the linear programming relaxation approach to gibbs energy minimization and weighted constraint satisfaction," *PAMI*, 2009.

[40] A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing, "An augmented Lagrangian approach to constrained MAP inference," in *Procs. ICML*, 2011.

[41] O. Meshi and A. Globerson, "An alternating direction method for dual map lp relaxation," in *Procs. ECML*, 2011.

[42] J. H. Kappes, B. Savchynskyy, and C. Schnörr, "A bundle approach to efficient map-inference by Lagrangian relaxation," in *Procs. CVPR*, 2012.

[43] N. Komodakis, N. Paragios, and G. Tziritas, "MRF energy minimization and beyond via dual decomposition," *PAMI*, 2010.

[44] V. Srikumar, G. Kundu, and D. Roth, "On amortizing inference cost for structured prediction," in *Procs. EMNLP*, 2012.

[45] K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume, and J. Langford, "Learning to search better than your teacher," in *Procs. ICML*, 2015.

[46] K. Yamaguchi, T. Hazan, D. McAllester, and R. Urtasun, "Continuous Markov random fields for robust stereo estimation," in *ECCV*, 2012.

[47] L. Sigal, M. Isard, B. H. Sigelman, and M. J. Black, "Attractive people: Assembling loose-limbed models using non-parametric belief propagation," in *Procs. NeurIPS*, 2004.

[48] E. B. Sudderth, M. I. Mandel, W. T. Freeman, and A. S. Willsky, "Visual hand tracking using nonparametric belief propagation," in *Procs. CVPR Workshop*, 2004.

[49] N. Friedman, M. Goldszmidt et al., "Discretizing continuous attributes while learning Bayesian networks," 1996.

[50] A. Mabrouk, C. Gonzales, K. Jabet-Chevalier, and E. Chojnaki, "Multivariate cluster-based discretization for Bayesian network structure learning," in *Procs. SUM*, 2015.

[51] S. Monti and G. F. Cooper, "A multivariate discretization method for learning Bayesian networks from mixed data," *UAI*, 2013.

[52] D. Margaritis and S. Thrun, "A Bayesian multiresolution independence test for continuous variables," *UAI*, 2013.

[53] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory and Applications*, 2005.

[54] K. Ristovski, V. Radosavljevic, S. Vucetic, and Z. Obradovic, "Continuous conditional random fields for efficient regression in large fully connected graphs," in *Procs. AAAI*, 2013.

[55] R. Vemulapalli, O. Tuzel, M.-Y. Liu, and R. Chellapa, "Gaussian conditional random field network for semantic segmentation," in *Procs. CVPR*, 2016.

[56] S. Chandra and I. Kokkinos, "Fast, exact and multi-scale inference for semantic image segmentation with deep Gaussian CRFs," in *Procs. ECCV*, 2016.

[57] J. Jancsary, S. Nowozin, T. Sharp, and C. Rother, "Regression tree fields: An efficient, non-parametric approach to image labeling problems," in *Procs. CVPR*, 2012.

[58] M. F. Tappen, C. Liu, E. H. Adelson, and W. T. Freeman, "Learning Gaussian conditional random fields for low-level vision," in *Procs. CVPR*, 2007.

[59] R. Vemulapalli, O. Tuzel, and M.-Y. Liu, "Deep Gaussian conditional random field network: A model-based deep network for discriminative denoising," in *Procs. CVPR*, 2016.

[60] A. Deshpande, J. Rock, and D. Forsyth, "Learning large-scale automatic image colorization," in *Procs. ICCV*, 2015.

[61] M. Salzmann, "Continuous inference in graphical models with polynomial energies," in *Procs. CVPR*, 2013.

[62] S. Wang, A. Schwing, and R. Urtasun, "Efficient inference of continuous markov random fields with polynomial potentials," in *Procs. NeurIPS*, 2014.

[63] A. Ihler and D. McAllester, "Particle belief propagation," in *Procs. AISTATS*, 2009.

[64] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Elsevier, 2014.

[65] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation and facial appearance estimation," in *Procs. CVPR*, 2002.

[66] J. Pacheco and E. Sudderth, "Proteins, particles, and pseudo-max-marginals: a submodular approach," in *Procs. ICML*, 2015.

[67] D. Wang, Z. Zeng, and Q. Liu, "Stein variational message passing for continuous graphical models," in *Procs. ICML*, 2018.

[68] V. Lempitsky, C. Rother, S. Roth, and A. Blake, "Fusion moves for Markov random field optimization," *PAMI*, 2009.

[69] K. G. Samuel and M. F. Tappen, "Learning optimized MAP estimates in continuously-valued MRF models," in *Procs. CVPR*, 2009.

[70] B. Taskar, C. Guestrin, and D. Koller, "Max-margin Markov networks," in *Procs. NeurIPS*, 2004.

[71] D. Munoz, J. A. Bagnell, N. Vandapel, and M. Hebert, "Contextual classification with functional max-margin Markov networks," in *Procs. CVPR*, 2009.

[72] M. Szummer, P. Kohli, and D. Hoiem, "Learning CRFs using graph cuts," in *Procs. ECCV*, 2008.

[73] S. Gould et al., "Max-margin learning for lower linear envelope potentials in binary Markov random fields." in *Procs. ICML*, 2011.

[74] J. Besag, "Statistical analysis of non-lattice data," *J. Royal Stat. Soc.*, 1975.

[75] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *NC*, 2002.

[76] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Procs. ICML*, 2009.

[77] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *Procs. NeurIPS*, 2019.

[78] D. Gamerman and H. F. Lopes, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. CRC Press, 2006.

[79] E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu, "Learning non-convergent non-persistent short-run MCMC toward energy-based model," in *Procs. NeurIPS*, 2019.

[80] B. Dai, Z. Liu, H. Dai, N. He, A. Gretton, L. Song, and D. Schuurmans, "Exponential family estimation via adversarial dynamics embedding," in *Procs. NeurIPS*, 2019.

[81] J. Lawson, G. Tucker, B. Dai, and R. Ranganath, "Energy-inspired models: Learning with sampler-induced distributions," in *Procs. NeurIPS*, 2019.

[82] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Procs. ICAIS*, 2010.

[83] A. Hyvärinen, "Estimation of non-normalized statistical models by score matching," *JMLR*, 2005.

[84] D. P. Kingma and Y. L. Cun, "Regularized estimation of image statistics by score matching," in *Procs. NeurIPS*, 2010.

[85] J. Martens, I. Sutskever, and K. Swersky, "Estimating the Hessian by back-propagating curvature," in *Procs. ICML*, 2012.

[86] P. Vincent, "A connection between score matching and denoising autoencoders," *NC*, 2011.

[87] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *Procs. NeurIPS*, 2019.

[88] Y. Song, S. Garg, J. Shi, and S. Ermon, "Sliced score matching: A scalable approach to density and score estimation," *AUAI*, 2019.

[89] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines," *Commun. Stat. Simul. Comput.*, 1989.

[90] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, and R. Zemel, "Cutting out the middle-man: Training and evaluating energy-based models without sampling," *arXiv preprint arXiv:2002.05616*, 2020.

[91] Y. Li and R. E. Turner, "Gradient estimators for implicit models," in *Procs. ICLR*, 2018.

[92] Q. Liu, J. Lee, and M. Jordan, "A kernelized stein discrepancy for goodness-of-fit tests," in *Procs. ICML*, 2016.

[93] J. Shi, S. Sun, and J. Zhu, "A spectral approach to gradient estimation for implicit distributions," in *Procs. ICML*, 2018.

[94] J. M. Alvarez, Y. LeCun, T. Gevers, and A. M. Lopez, "Semantic road segmentation via multi-scale ensembles of learned features," in *Procs. ECCV*, 2012.

[95] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," in *Procs. ICLR*, 2015.

[96] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *PAMI*, 2012.

[97] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from RGB-D images for object detection and segmentation," in *Procs. ECCV*, 2014.

[98] Y. Ganin and V. Lempitsky, "Neural network nearest neighbor fields for image transforms," in *Procs. ACCV*, 2014.

[99] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Procs. CVPR*, 2015.

[100] P. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *Procs. ICML*, 2014.

[101] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation," in *Procs. ECCV*, 2014.

[102] M. Mostajabi, P. Yadollahpour, and G. Shakhnarovich, "Feedforward semantic segmentation with zoom-out features," in *Procs. CVPR*, 2015.

[103] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Procs. NeurIPS*, 2014.

[104] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep structured output learning for unconstrained text recognition," in *Procs. ICLR*, 2015.

[105] S. Chandra, N. Usunier, and I. Kokkinos, "Dense and low-rank Gaussian CRFs using deep embeddings," in *Procs. ICCV*, 2017.

[106] D. Belanger and A. McCallum, "Structured prediction energy networks," in *Procs. ICML*, 2016.

[107] C. Graber, O. Meshi, and A. Schwing, "Deep structured prediction with nonlinear output transformations," in *Procs. NeurIPSs*, 2018.

[108] C. Graber and A. Schwing, "Graph structured prediction energy networks," in *Procs. NeurIPS*, 2019.

[109] M. Gygli, M. Norouzi, and A. Angelova, "Deep value networks learn to evaluate and iteratively refine structured outputs," in *Procs. ICML*, 2017.

[110] L.-C. Chen, A. Schwing, A. Yuille, and R. Urtasun, "Learning deep structured models," in *Procs. ICML*, 2015.

[111] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr, "Conditional random fields as recurrent neural networks," in *Procs. ICCV*, 2015.

[112] Z. Liu, X. Li, P. Luo, C.-C. Loy, and X. Tang, "Semantic image segmentation via deep parsing network," in *Procs. ICCV*, 2015.

[113] L. Tu and K. Gimpel, "Learning approximate inference networks for structured prediction," in *Procs. ICLR*, 2018.

[114] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Procs. ICASSP*, 2013.

[115] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Procs. ICML*, 2006.

[116] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Procs. ICASSP*, 2011.

[117] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *JMLR*, 2003.

[118] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Procs. NeurIPS*, 2014.

[119] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Procs. ICLR*, 2015.

[120] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *NC*, 1997.

[121] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Procs. EMNLP*, 2014.

[122] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *TNNLS*, 2016.

[123] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Procs. ICML*, 2015.

[124] W. Zhang and T. G. Dietterich, "Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling," *JAIR*, 2000.

[125] J. Boyan and A. W. Moore, "Learning evaluation functions to improve optimization by local search," *JMLR*, 2000.

[126] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Procs. NeurIPS*, 2015.

[127] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," in *https://arxiv.org/abs/1611.09940*, 2016.

[128] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-Prop: Sample-efficient policy gradient with an off-policy critic," in *Procs. ICLR*, 2017.

[129] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Procs. NeurIPS*, 2017.

[130] A. Laterre, Y. Fu, M. K. Jabri, A.-S. Cohen, D. Kas, K.Hajjar, T. S. Dahl, A. Kerkeni, and K. Beguir, "Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization," in *Procs. NeurIPS Workshop on Deep RL*, 2018.

[131] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky, "Deep reinforcement learning for dialogue generation," in *Procs. EMNLP*, 2016.

[132] J. Williams, K. Asadi, and G. Zweig, "Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning," in *Procs. ACL*, 2017.

[133] D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T.-Y. Liu, and W.-Y. Ma, "Dual learning for machine translation," in *Procs. NeurIPS*, 2016.

[134] R. Bunel, M. Hausknecht, J. Devlin, R. Singh, and P. Kohli, "Leveraging grammar and reinforcement learning for neural program synthesis," in *Procs. ICLR*, 2018.

[135] C. Liang, M. Norouzi, J. Berant, Q. Le, and N. Lao, "Memory augmented policy optimization for program synthesis with generalization," in *Procs. NeurIPS*, 2017.

[136] T. Pierrot, G. Ligner, S. E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. Freitas, "Learning compositional neural programs with recursive tree search and planning," in *Procs. NeurIPS*, 2019.

[137] C. Liang, J. Berant, Q. Le, K. Forbus, and N. Lao, "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision," in *Procs. ACL*, 2016.

[138] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Procs. ICLR*, 2017.

[139] A. Sharaf and H. D. III, "Structured prediction via learning to search under bandit feedback," in *Procs. ACL Workshop on Structured Prediction for NLP*, 2017.

[140] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "Sequence level training with recurrent neural networks," in *Procs. ICLR*, 2016.

[141] M. Norouzi, S. Bengio, N. Jaitly, M. Schuster, Y. Wu, D. Schuurmans et al., "Reward augmented maximum likelihood for neural structured prediction," in *Procs. NeurIPS*, 2016.

[142] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio, "An actor-critic algorithm for sequence prediction," in *Procs. ICLR*, 2017.

[143] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," in *Procs. ICLR*, 2018.

[144] S. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *Procs. CVPR*, 2017.

[145] W. Xiong, T. Hoang, and W. Y. Wang, "Deeppath: A reinforcement learning method for knowledge graph reasoning," in *Procs. EMNLP*, 2017.

[146] R. Nogueira and K. Cho, "Task-oriented query reformulation with reinforcement learning," in *Procs. EMNLP*, 2017.

[147] C. Buck, J. Bulian, M. Ciaramita, W. Gajewski, A. Gesmundo, N. Houlsby, and W. Wang, "Ask the right questions: Active question reformulation with reinforcement learning," in *Procs. ICLR*, 2018.

[148] K. Narasimhan, A. Yala, and R. Barzilay, "Improving information extraction by acquiring external evidence with reinforcement learning," in *Procs. EMNLP*, 2016.

[149] P. Qin, W. Xu, and W. Y. Wang, "Robust distant supervision relation extraction via deep reinforcement learning," in *Procs. ACL*, 2018.

[150] M. G. Lagoudakis and M. L. Littman, "Learning to select branching rules in the DPLL procedure for satisfiability," in *Procs. ENDM*, 2001.

[151] H. Samulowitz and R. Memisevic, "Learning to solve QBF," in *Procs. AAAI*, 2007.

[152] H. He, H. Daume, and J. M. Eisner, "Learning to search in branch and bound algorithms," in *Procs. NeurIPS*, 2014.

[153] E. B. Khalil, P. L. Bodic, L. Song, G. L. Nemhauser, and B. N. Dilkina, "Learning to branch in mixed integer programming," in *Procs. AAAI*, 2016.

[154] K. Li and J. Malik, "Learning to optimize," in *Procs. ICLR*, 2017.

[155] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. D. Freitas, "Learning to learn by gradient descent by gradient descent," in *Procs. NeurIPS*, 2016.

[156] E. Learned-Miller, G. B. Huang, A. Roy Chowdhury, H. Li, and G. Hua, "Labeled faces in the wild: A survey," in *Advances in Face Detection and Facial Image Analysis*, 2016.

[157] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *CoRR*, 2015.

[158] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., "Imagenet large scale visual recognition challenge," *IJCV*, 2015.

[159] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Procs. CVPR*, 2017.

[160] Y. Cao, Z. Zhou, W. Zhang, and Y. Yu, "Unsupervised diverse colorization via generative adversarial networks," in *Procs. ECML PKDD*, 2017.

[161] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman, "Toward multimodal image-to-image translation," in *NeurIPS*, 2017.

[162] A. Royer, A. Kolesnikov, and C. H. Lampert, "Probabilistic image colorization," in *Procs. BMVC*, 2017.

[163] A. Deshpande, J. Lu, M.-C. Yeh, and D. Forsyth, "Learning diverse image colorization," in *Procs. CVPR*, 2017.

[164] T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring color to greyscale images," *Procs. SIGGRAPH*, 2002.

[165] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," in *Procs. SIGGRAPH*, 2004.

[166] A. Y. S. Chia, S. Zhuo, R. K. Gupta, Y. W. Tai, S. Y. Cho, P. Tan, and S. Lin, "Semantic colorization with internet images," *TOG*, 2011.

[167] R. K. Gupta, A. Y.-S. Chia, D. Rajan, E. S. Ng, and H. Zhiyong, "Image colorization using similar images," *ACM Multimedia*, 2012.

[168] D. Cohen-Or and D. Lischinski, "Colorization by example," *EGSR*, 2005.

[169] Y. Morimoto, Y. Taguchi, and T. Naemura, "Automatic colorization of grayscale images using multiple images on the web," in *Procs. SIGGRAPH*, 2009.

[170] G. Charpiat, M. Hofmann, and B. Schölkopf, "Automatic image colorization via multimodal predictions," in *Procs. ECCV*, 2008.

[171] Z. Cheng, Q. Yang, and B. Sheng, "Deep colorization," in *Procs. ICCV*, 2015.

[172] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification," in *Procs. SIGGRAPH*, 2016.

[173] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," in *Procs. ECCV*, 2016.

[174] D. Varga and T. Szirányi, "Twin deep convolutional neural network for example-based image colorization," *CAIP*, 2017.

[175] R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu, and A. A. Efros, "Real-time user-guided image colorization with learned deep priors," *Procs. SIGGRAPH*, 2017.

[176] D. Varga and T. Szirányi, "Twin deep convolutional neural network for example-based image colorization," *ICCAIP*, 2017.

[177] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves et al., "Conditional image generation with PixelCNN decoders," in *Procs. NeurIPS*, 2016.

[178] S. Guadarrama, R. Dahl, D. Bieber, M. Norouzi, J. Shlens, and K. Murphy, "Pixcolor: Pixel recursive colorization," in *Procs. BMVC*, 2017.

[179] C. M. Bishop, "Mixture density networks," Tech. Rep., 1994.

[180] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays, "Transient attributes for high-level understanding and editing of outdoor scenes," in *Procs. SIGGRAPH*, 2014.

[181] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Procs. CVPR*, 2016.

[182] A. Yu and K. Grauman, "Fine-grained visual comparisons with local learning," in *Procs. CVPR*, 2014.

[183] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," in *Procs. ECCV*, 2016.

[184] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Procs. ICLR*, 2014.

[185] K. Sohn, X. Yan, and H. Lee, "Learning structured output representation using deep conditional generative models," in *Proc. NIPS*, 2015.

[186] L. Wang, A. G. Schwing, and S. Lazebnik, "Diverse and accurate image description using a variational auto-encoder with an additive Gaussian encoding space," in *Procs. NeurIPS*, 2017.

[187] U. Jain*, Z. Zhang*, and A. G. Schwing, "Creativity: Generating diverse questions using variational autoencoders," in *Procs. CVPR*, 2017, * equal contribution.

[188] G. Huang, M. Mattar, H. Lee, and E. G. Learned-Miller, "Learning to align from scratch," in *Procs. NeurIPS*, 2012.

[189] Y. Endo, S. Iizuka, Y. Kanamori, and J. Mitani, "Deepprop: Extracting deep features from a single image for edit propagation," *Eurographics*, 2016.

[190] J. T. Barron and B. Poole, "The fast bilateral solver," in *Procs. ECCV*, 2016.

[191] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *TIP*, 2004.

[192] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "MOTS: Multi-object tracking and segmentation," in *Procs. CVPR*, 2019.

[193] M. Everingham, L. van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," *IJCV*, 2010.

[194] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Procs. CVPR*, 2017.

[195] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Procs. CVPR*, 2016.

[196] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *Procs. ICML*, 2016.

[197] S. Konishi and A. L. Yuille, "Statistical cues for domain specific image segmentation with performance analysis," in *Procs. CVPR*, 2000.

[198] X. He, R. S. Zemel, and M. A. Carreira-Perpiñán, "Multiscale conditional random fields for image labeling," in *Procs. CVPR*, 2004.

[199] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Procs. ICML*, 2001.

[200] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton University Press, 1962.

[201] A. Goldberg and R. Tarjan, "A new approach to the maximum flow problem," *JACM*, 1988.

[202] D. Greig, B. Porteous, and A. Seheult, "Exact maximum a posteriori estimation for binary images," *J. of the Royal Statistical Society*, 1989.

[203] Y. Boykov, O. Veksler, and R. Zabih, "Markov random fields with efficient approximations," in *Procs. CVPR*, 1998.

[204] Y. Boykov and M. P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images," in *Procs. ICCV*, 2001.

[205] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," in *Procs. EMMCVPR*, 2001.

[206] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *PAMI*, 2001.

[207] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph-cuts?" *PAMI*, 2004.

[208] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "TextonBoost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation," in *Procs. ECCV*, 2006.

[209] X. Li, Z. Liu, P. Luo, C. Change, and X. Tang, "Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade," in *Procs. CVPR*, 2017.

[210] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Procs. ICCV*, 2015.

[211] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Procs. MICCAI*, 2015.

[212] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Procs. ICLR*, 2016.

[213] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[214] L. C. Chen, A. G. Schwing, A. Yuille, and R. Urtasun, "Learning deep structured models," in *Procs. ICML*, 2015, * equal contribution.

[215] A. G. Schwing and R. Urtasun, "Fully Connected Deep Structured Networks," in *https://arxiv.org/abs/1503.02351*, 2015.

[216] A. Guisti, D. Ciresan, J. Masci, L. Gambardella, and J. Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," in *Procs. ICIP*, 2013.

[217] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," in *Procs. ICLR*, 2014.

[218] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Procs. CVPR*, 2015.

[219] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr, "Conditional random fields as recurrent neural networks," in *Procs. ICCV*, 2015.

[220] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv:1712.01815*, 2017.

[221] A. Arnab, S. Jayasumana, S. Zheng, and P. Torr, "Higher-order conditional random fields in deep neural networks," in *Procs. ECCV*, 2016.

[222] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *Procs. ICCV*, 2011.

[223] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *PAMI*, 2012.

[224] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Procs. AAAI*, 1982.

[225] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Tree-reweighted belief propagation algorithms and approximate ML estimation by pseudo-moment matching," in *Procs. AISTATS*, 2003.

[226] B. Andres, J. H. Kappes, T. Beier, U. Köthe, and F. A. Hamprecht, "The lazy flipper: Efficient depth-limited exhaustive search in discrete graphical models," in *Procs. ECCV*, 2012.

[227] A. Fix, A. Gruber, E. Boros, and R. Zabih, "A graph cut algorithm for higher-order Markov random fields," in *Procs. ICCV*, 2011.

[228] L. Wenliang, D. Sutherland, H. Strathmann, and A. Gretton, "Learning deep kernels for exponential family densities," in *Procs. ICML*, 2019.

[229] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[230] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," in *Procs. ICLR*, 2018.

[231] Y. Du and I. Mordatch, "Implicit generation and generalization in energy-based models," *arXiv preprint arXiv:1903.08689*, 2019.

[232] C. Lanczos, *An Iteration Method For The Solution Of The Eigenvalue Problem of Linear Differential and Integral Operators*, 1950.

[233] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[234] R. M. Neal, "Annealed importance sampling," *Statistics and Computing*, 2001.

[235] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash-equilibrium," *arXiv preprint arXiv:1706.08500*, 2017.

[236] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *ACM Computing Surveys*, 2019.

[237] D. Lin, S. Fidler, and R. Urtasun, "Holistic scene understanding for 3d object detection with RGB-D cameras," in *ICCV*, 2013.

[238] M. Salzmann and R. Urtasun, "Beyond feature points: Structured prediction for monocular non-rigid 3d reconstruction," in *ECCV*, 2012.

[239] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps," *TPAMI*, 2012.

[240] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.

[241] A. Yu and K. Grauman, "Fine-grained visual comparisons with local learning," in *Procs. CVPR*, 2014.

[242] J. M. Susskind, A. K. Anderson, and G. E. Hinton, "The Toronto face database. department of computer science, University of Toronto, Toronto, ON," Tech. Rep., 2010.

[243] X. Hou, L. Shen, K. Sun, and G. Qiu, "Deep feature consistent variational autoencoder," in *Procs. WACV*, 2017.

[244] W. K. Newey and D. McFadden, "Large sample estimation and hypothesis testing," *Handbook of Econometrics*, 1994.

[245] A. Hyvarinen, "Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables," *IEEE Trans. Neural Netw. Learn. Syst*, 2007.