Large-Scale Machine Learning over Graphs

Hanxiao Liu

CMU-LTI-18-013

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

Yiming Yang (Chair)	(Carnegie Mellon University)
Jaime G. Carbonell	(Carnegie Mellon University)
J. Zico Kolter	(Carnegie Mellon University)
Karen Simonyan	(DeepMind)

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies*

Keywords: Machine learning, optimization, semi-supervised learning, relational learning, graph-based learning, autoML, neural architecture search

For Yujie.

Abstract

Graphs are ubiquitous in a broad range of statistical modeling and machine learning applications. They are powerful mathematical concepts in representing not only structured data, but also structured computation procedures (e.g. neural network architectures). Despite their ubiquity, efficient learning over heterogeneous and/or complex graph structures remains a grand challenge, beyond much existing theory and algorithms. In this thesis, we address this challenge in several complementary aspects:

In part I we focus on learning across heterogeneous graphs. We propose a novel framework to fuse multiple heterogeneous graphs into a single homogeneous graph, on which the learning task can be formulated in a principled manner (Chapter 2). We then develop a method that imposes analogical structures among the heterogeneous nodes in the graphs for improved generalization (Chapter 3), which also theoretically unifies several representative models. In both cases, we develop scalable approximation algorithms to ensure linear scalability over the size of the input graphs.

In part II we focus on graph induction problems where a latent graph structure must be inferred directly from the data. We investigate the task in the graph spectral domain (of eigenvectors and eigenvalues), and propose an efficient non-parametric approach to recover the graph spectrum that best characterizes the underlying diffusion process over the data (Chapter 4).

Finally, in part III we focus on the optimization of neural network architectures, represented as acyclic graphs. We present a hierarchical representation scheme for neural network topologies, where smaller graph motifs are used as building blocks to form the larger ones (Chapter 5). We then relax the discrete architectures as continuous variables to enable efficient gradient-based optimization, leading to orders of magnitude speedup over several state-of-the-art non-differentiable techniques (Chapter 6). The automatically learned architectures achieve highly competitive performance for both image classification and language modeling, outperforming a large number of architectures manually designed by human experts.

Acknowledgments

First and foremost, I would like to thank my advisor Yiming Yang, who introduced me to the field of machine learning and granted me the greatest possible freedom to explore many interesting directions. This dissertation would not be possible without her insights and support. I also thank Jaime Carbonell, who has provided continuous help and guidance during my PhD just like my advisor. I am very grateful to the rest of the thesis committee, Zico Kolter and Karen Simonyan, for their generous assistance and valuable feedback on this work.

Thank you to all of my friends and collaborators at CMU: Siddharth Gopal, Konstantin Salomatin, Andrew Hsi, Guoqing Zheng, Zihang Dai, Ruochen Xu, Yuexin Wu, Wei-Cheng Chang, Guokun Lai, Jingzhou Liu, Bohan Li, Fatima Al-Raisi, Naoki Otani, Adams Wei Yu, Wanli Ma, Keerthiram Murugesan, Jay-Yoon Lee, Hector Liu, Zi Yang, Xuezhe Ma, Qizhe Xie, Zhilin Yang, Diyi Yang, Zichao Yang, Zhiting Hu, Han Zhao, Hao Zhang, Lu Jiang, Zhenzhong Lan, Jiachen Li, Bhuwan Dhingra, Bernie Huang, Agha Ali Raza, and many others not listed there. It has been such a pleasure to work and hangout with you. Life in Pittsburgh has been fun!

Besides, I would like to take the chance to thank Jian Zhang and Fang Wu, for their mentorship throughout my time at Citadel. Also many thanks to my collaborators at DeepMind, especially Karen Simonyan and Oriol Vinyals, for the fruitful discussions and their advice during and after my internship in London.

Last but not least, I would like to thank my wife Yujie, for her unwavering love, support and encouragement throughout this wonderful journey together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Overview

The thesis is organized into three parts. Part I focuses on scalable learning algorithms within or across heterogeneous graphs. Part II focuses on learning an unknown or partially specified graph from the data. Part III focuses on learning computation graphs, i.e., neural network architectures.

The motivations and contributions of the individual chapters are summarized as follows:

1.1.1 Learning over Heterogeneous Graphs

1.1.1.1 Relational Learning across Heterogeneous Graphs (Liu and Yang, 2015, 2016a)

Cross-graph relational learning refers to the task of predicting the strengths or labels of multi-relational tuples of heterogeneous object types, through the joint inference over multiple graphs which specify the internal connections among each type of the object. This is an open challenge in machine learning due to the combinatorially large number of all possible tuples to deal with and the usually extremely sparse labeled data.

In this chapter, we propose a convex formulation which enables transductive learning using both labeled and unlabeled tuples, and a scalable algorithm that enjoys a linear time complexity over the sizes of input graphs. Our method can be regarded as an extension of the classic spectral graph analysis developed for a single homogeneous graph to the case of multiple heterogeneous graphs. Experiments over citation networks and compound-protein interactions show that the proposed method can successfully scale to large cross-graph inference problems, outperforming other representative approaches significantly.

1.1.1.2 Analogical Inference for Knowledge Graphs (Liu et al., 2017c)

Large-scale relational embedding refers to the task of learning the latent representations for entities their relations in large knowledge graphs. An effective and scalable solution for this problem is crucial for the true success of knowledge-based inference in a broad range of applications.

In this chapter, we propose a novel framework for optimizing the latent representations with respect to the analogical properties of the embedded entities and relations. The proposed *ana-*

logical inference can be viewed as the generalization of the widely studied *analogical reasoning* in classic AI to the continuous vector space. By formulating the learning task in a differentiable fashion, our model enjoys both theoretical power and computational scalability, and significantly outperformed a large number of representative baselines on benchmark datasets. The model further offers a unification of several well-known methods in relational embedding, which can be proven to be special instantiations of our framework.

1.1.2 Learning with Graph Induction

1.1.2.1 Nonparametric Learning of Graph Diffusions (Liu and Yang, 2016b)

Graph-based semi-supervised learning (SSL) aims to leverage the intrinsic graph structure among both the labeled and unlabeled data when only limited supervision is available. A fundamental limitation of existing algorithms, however, is that the graph has to be fully specified based on some suboptimal metric independent of the data.

In this chapter, we propose a novel nonparametric graph-based SSL method which adapts the graph spectrum (that characterizes the label diffusion process) based on the underlying data manifold. Our formulation leads to a convex optimization problem that can be efficiently solved using the bundle method, and can be interpreted as to asymptotically minimize the generalization error bound of SSL with respect to the graph. Experiments over benchmark datasets in a variety of domains show advantageous performance of the proposed method over strong baselines.

1.1.3 Learning Neural Network Architectures

1.1.3.1 Architecture Search with Hierarchical Representations (Liu et al., 2017a)

In this chapter, we explore efficient neural architecture search methods and show that a simple yet powerful evolutionary algorithm can discover new architectures with excellent performance. Our approach combines a novel hierarchical genetic representation scheme that imitates the modularized design pattern commonly adopted by human experts, and an expressive search space that supports complex topologies. Our algorithm efficiently discovers architectures that outperform a large number of manually designed models for image classification, obtaining top-1 error of 3.6% on CIFAR-10 and 20.3% when transferred to ImageNet, which is competitive with the best existing neural architecture search approaches.

1.1.3.2 Differentiable Architecture Search (Liu et al., 2018)

In this chapter, we address the scalability challenge of architecture search by formulating the task in a differentiable manner. Unlike conventional approaches of applying evolution or reinforcement learning over a discrete and non-differentiable search space, our method is based on the continuous relaxation of the architecture representation, allowing efficient search of the architecture using gradient descent. Extensive experiments on CIFAR-10, ImageNet, Penn Treebank and WikiText-2 show that our algorithm excels in discovering high-performance convolutional architectures for image classification and recurrent architectures for language modeling, while being orders of magnitude faster than state-of-the-art non-differentiable techniques.

1.2 Thesis Publications

The cross-graph learning algorithms in Chapter 2 were published at ICML 2015 (Liu and Yang, 2015) (for two graphs) and ICML 2016 (Liu and Yang, 2016a) (for arbitrary number of graphs). The analogical inference algorithm, described in Chapter 3, was published at ICML 2017 (Liu et al., 2017c). The semi-supervised graph induction method in Chapter 4 was published at AISTATS 2016 (Liu and Yang, 2016b). The two architecture search algorithms, including evolutionary architecture search (Liu et al., 2017b) and differentiable architecture search (Liu et al., 2018), were published at ICLR 2018 and submitted to NIPS 2018, respectively.

The thesis also led to several other related works not listed as individual chapters, including a concept graph learning algorithm presented at JAIR 2016 (Liu et al., 2016a) and a graph-based multi-task learning algorithm published at NIPS 2016 (Liu et al., 2016b).

Part I

Learning over Heterogeneous Graphs

Chapter 2

Relational Learning across Heterogeneous Graphs

2.1 Background

Many important problems in multi-source relational learning could be cast as joint learning over multiple graphs about how heterogeneous types of objects interact with each other. In literature data analysis, for example, publication records provide rich information about how authors collaborate with each other in a co-authoring graph, how papers are linked in citation networks, how keywords are related via ontology, and so on. The challenging question is about how to combine such heterogeneous information in individual graphs for the labeling or scoring of the multi-relational associations in tuples like `(author,paper,keyword)`, given some observed instances of such tuples as the labeled training set. Automated labeling or scoring of unobserved tuples allows us to discover who have been active in the literature on what areas of research, and to predict who would become influential in which areas in the future. In protein data analysis, as another example, a graph of proteins with pairwise sequence similarities is often jointly studied with a graph of chemical compounds with their structural similarities for the discovery of interesting patterns in `(compound,protein)` pairs. We call the prediction problem in both examples *cross-graph learning of multi-relational associations*, or simply *cross-graph relational learning* (CGRL), where the multi-relational associations are defined by the tuples of heterogeneous types of objects, and each object type has its own graph with type-specific relational structure as a part of the provided data. The task is to predict the labels or the scores of unobserved multi-relational tuples, conditioned on a relatively small set of labeled instances.

CGRL is an open challenge in machine learning for several reasons. Firstly, the number of multi-relational tuples grows combinatorially in the numbers of individual graphs and the number of nodes in each graph. How to make cross-graph inference computationally tractable for large graphs is a tough challenge. Secondly, how to combine the internal structures or relations in individual graphs for joint inference in a principled manner is an open question. Thirdly, supervised information (labeled instances) is typically extremely sparse in CGRL due to the very large number of all possible combinations of heterogeneous objects in individual graphs. Consequently, the success of cross-graph learning crucially depends on effectively leveraging the massively available unlabeled tuples (and the latent relations among them) in addition to the labeled training data. In other words, how to make the learning transductive is crucial for the true success of CGRL. Research on transductive CGRL has been quite limited, to our knowledge.

Existing approaches in CGRL or CGRL-related areas can be outlined as those using tensors or graph-regularized tensors, and kernel machines that combine multiple kernels.

Tensor methods have been commonly used for combining multi-source evidence of the interactions among multiple types of objects (Nickel et al., 2011; Rendle et al., 2009; Kolda and Bader, 2009) as the combined evidence can be naturally represented as tuples. However, most of the tensor methods do not explicitly model the internal graph structure for each type of objects, although some of those methods implicitly leverage such information via graph-based regularization terms in their objective function that encourage similar objects within each graph to share similar latent factors (Narita et al., 2012; Cai et al., 2011). A major weakness in such tensor methods is the lack of convexity in their models, which leads to ill-posed optimization problems particularly in high-order scenarios. It has also been observed that tensor factorization models suffer from label-sparsity issue, which is typically severe in CGRL.

Kernel machines have been widely studied for supervised classifiers, where a kernel matrix corresponds to a similarity graph among a single type of objects. Multiple kernels can be combined, for example, by taking the tensor product of each individual kernel matrix, which results in a desired kernel matrix among cross-graph multi-relational tuples. The idea has been explored in relational learning combined with SVMs (Ben-Hur and Noble, 2005), perceptions (Basilico and Hofmann, 2004) or Gaussian process (Yu and Chu, 2008) for two types of objects and is generalizable to the multi-type scenario of CGRL. Although being generic, the complexity of such kernel-based methods grows exponentially in the number of individual kernels (graphs) and the size of each individual graph. As a result, kernel machines suffer from poor scalability in general. In addition, kernel machines are purely supervised (not for transductive learning), i.e., they cannot leverage the massive number of available non-observed tuples induced from individual graphs and the latent connections among them. Those limitations make existing kernel methods less powerful for solving the CGRL problem in large scale and under severely data-sparse conditions.

We propose a novel framework for CGRL which can be characterized as follows: (i) It uses graph products to map heterogeneous sources of information and the link structures in individual graphs onto a single *homogeneous* graph; (ii) It provides a convex formulation and approximation of the CGRL problem that ensure robust optimization and efficient computation; and (iii) It enables transductive learning in the form of label propagation over the induced homogeneous graph so that the massively available non-observed tuples and the latent connections among them can play an important role in effectively addressing the label-sparsity issue.

Our method in this chapter is most related to Liu and Yang (2015), where the authors formulated graph products for learning the edges of a bipartite graph. Our framework is fundamentally different in two aspects. First, our formulation and algorithms allow arbitrary number of individual graphs, while method in Liu and Yang (2015) is only applicable to two graphs. Secondly, the algorithms in Liu and Yang (2015) suffer from cubic complexity over the graphs sizes (quadratic by using a non-convex approximation), while our new algorithm enjoys both the convexity of the formulation and the low time complexity which is linear over the graph sizes.

Our method also shares the high-level goal with Statistical Relational Learning (SRL) (Getoor,

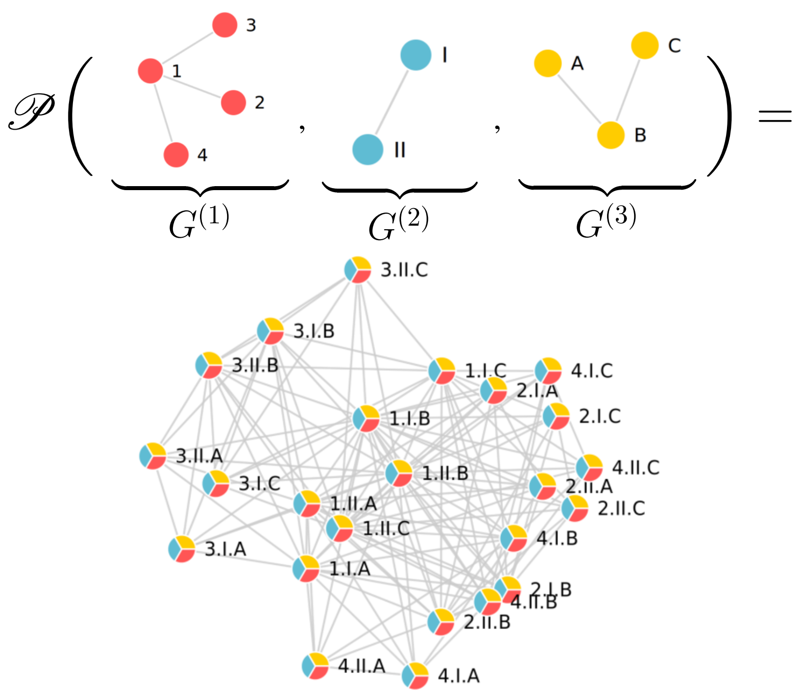

Figure 2.1: Product of three graphs $G^{(1)}$, $G^{(2)}$ and $G^{(3)}$. Each vertex in the resulting product graph $\mathscr{P}\big(G^{(1)}, G^{(2)}, G^{(3)}\big)$ corresponds to a multi-relation across the original graphs. For instance, vertex `3.II.B` in $\mathscr{P}$ corresponds to multi-relation `(3,II,B)` across $G^{(1)}$, $G^{(2)}$ and $G^{(3)}$.

2007) and Inductive Logic Programming (ILP) (Lavrac and Dzeroski, 1994) in terms of multirelational learning. However, both of our problem setting and formulation differ substantially from existing SRL/ILP approaches focusing on first-order logic and/or probabilistic reasoning over graphical models.

2.2 Spectral Graph Product

2.2.1 Notations

We are given J heterogeneous graphs where the j-th graph contains n_j vertices and is associated with an adjacency matrix $G^{(j)} \in \mathbb{R}^{n_j \times n_j}$. We use i_j to index the i_j-th vertex of graph j, and use a tuple $(i_1, \ldots, i_J)$ to index each multi-relation across the J graphs. The system predictions over all possible $\prod_{j=1}^{J} n_j$ multi-relations is summarized in an order-J tensor $f \in \mathbb{R}^{n_1 \times \cdots \times n_J}$, where $f_{i_1,i_2,\ldots,i_J}$ corresponds to the prediction about tuple $(i_1, \ldots, i_J)$.

Denote by $\otimes$ the Kronecker (Tensor) product. We use $\bigotimes_{j=1}^{J} x_j$ (or simply $\bigotimes_j x_j$) as the shorthand for $x_1 \otimes \cdots \otimes x_J$. Denote by $\times_j$ the j-mode product between tensors. We refer the readers to (Kolda and Bader, 2009) for a thorough introduction about tensor mode product.

2.2.2 Graph Product

In a nutshell, graph product (GP) [1] is a mapping from each cross-graph multi-relation to each vertex in a new graph $\mathscr{P}$, whose edges encode similarities among the multi-relations (illustrated in Fig. 2.1). A desirable property of GP is it provides a natural reduction from the original multi-relational learning problem over *heterogeneous* information sources (Task 2.2.1) to an equivalent graph-based learning problem over a *homogeneous* graph (Task 2.2.2).

Task 2.2.1. *Given J graphs $G^{(1)}, \ldots, G^{(J)}$ with a small set of labeled multi-relations $\mathcal{O} = \{(i_1, \ldots, i_J)\}$, predict labels of the unlabeled multi-relations.*

Task 2.2.2. *Given the product graph $\mathscr{P}\big(G^{(1)}, \ldots, G^{(J)}\big)$ with a small set of labeled vertices $\mathcal{O} = \{(i_1, \ldots, i_J)\}$, predict labels of its unlabeled vertices.*

2.2.3 Spectral Graph Product

We define a parametric family of GP operators named the spectral graph product (SGP), which is of particular interest as it subsumes the well-known Tensor GP and Cartesian GP (Table 2.1), is well behaved (Theorem 2.2.1) and allows efficient optimization routines (Section 2.3).

Let $\lambda^{(j)}_{i_j}$ and $v^{(j)}_{i_j}$ be the i_j-th eigenvalue and eigenvector for the graph j, respectively. We construct SGP by defining the eigensystem of its adjacency matrix based on the provided J heterogeneous eigensystems of $G^{(1)}, \ldots, G^{(J)}$.

Definition 2.2.1. *The SGP of $G^{(1)}, \ldots, G^{(J)}$ is a graph consisting of $\prod_j n_j$ vertices, with its adjacency matrix $\mathscr{P}_\kappa := \mathscr{P}_\kappa\left(G^{(1)}, \ldots, G^{(J)}\right)$ defined by the following eigensystem*

$$\Big\{\kappa\big(\lambda^{(1)}_{i_1}, \ldots, \lambda^{(J)}_{i_J}\big), \bigotimes_j v^{(j)}_{i_j}\Big\}_{i_1,\ldots,i_J} \tag{2.1}$$

where κ is a pre-specified nonnegative nondecreasing function over $\lambda^{(j)}_{i_j}, \forall j = 1, 2, \ldots, J$.

In other words, the $(i_1, \ldots, i_J)$-th eigenvalue of $\mathscr{P}_\kappa$ is defined by coupling the $\lambda^{(1)}_{i_1}, \ldots, \lambda^{(J)}_{i_J}$ with function κ, and the $(i_1, \ldots, i_J)$-th eigenvector of $\mathscr{P}_\kappa$ is defined by coupling $v^{(1)}_{i_1}, \ldots, v^{(J)}_{i_J}$ via tensor (outer) product.

Remark 2.2.1. *If each individual $\big\{v^{(j)}_{i_j}\big\}_{i_j=1}^{n_j}$ forms an orthogonal basis in $\mathbb{R}^{n_j}$, $\forall j \in 1, \ldots, J$, then $\big\{\bigotimes_j v^{(j)}_{i_j}\big\}_{i_1,\ldots,i_J}$ forms an orthogonal basis in $\mathbb{R}^{\prod_{j=1}^J n_j}$.*

In the following examples, we introduce two special kinds of SGPs, assuming $J = 2$ for brevity. Higher-order cases are later summarized in Table 2.1.

Example 2.2.1. *Tensor GP defines $\kappa(\lambda_{i_1}, \lambda_{i_2}) = \lambda_{i_1}\lambda_{i_2}$, and is equivalent to Kronecker product:*

$$\mathscr{P}_{\mathit{Tensor}}\big(G^{(1)}, G^{(2)}\big) = \sum_{i_1,i_2}(\lambda_{i_1}\lambda_{i_2})\big(v^{(1)}_{i_1} \otimes v^{(2)}_{i_2}\big)\big(v^{(1)}_{i_1} \otimes v^{(2)}_{i_2}\big)^\top \equiv G^{(1)} \otimes G^{(2)}$$

Example 2.2.2. *Cartesian GP defines $\kappa(\lambda_{i_1}, \lambda_{i_2}) = \lambda_{i_1} + \lambda_{i_2}$, which gives us the Kronecker sum:*

$$\mathscr{P}_{\mathit{Cartesian}}\big(G^{(1)}, G^{(2)}\big) = \sum_{i_1,i_2}(\lambda_{i_1} + \lambda_{i_2})\big(v^{(1)}_{i_1} \otimes v^{(2)}_{i_2}\big)\big(v^{(1)}_{i_1} \otimes v^{(2)}_{i_2}\big)^\top \equiv G^{(1)} \oplus G^{(2)}$$

SGP Type	$\kappa(\lambda^{(1)}_{i_1}, \cdots, \lambda^{(J)}_{i_J})$	$[\mathscr{P}_\kappa]_{(i_1,\cdots i_J),(i'_1,\cdots i'_J)}$
Tensor	$\prod_j \lambda^{(j)}_{i_j}$	$\prod_j G^{(j)}_{i_j,i'_j}$
Cartesian	$\sum_j \lambda^{(j)}_{i_j}$	$\sum_j G^{(j)}_{i_j,i'_j} \prod_{j' \neq j} \delta_{i_{j'}=i'_{j'}}$

Table 2.1: Tensor GP and Cartesian GP in higher-orders.

While Tensor GP and Cartesian GP provide mechanisms to associate multiple graphs in a multiplicative/additive manner, more complex cross-graph association patterns can be modeled by specifying κ. E.g., $\kappa(\lambda_{i_1}, \lambda_{i_2}, \lambda_{i_3}) = \lambda_{i_1}\lambda_{i_2} + \lambda_{i_2}\lambda_{i_3} + \lambda_{i_3}\lambda_{i_1}$ indicates pairwise associations are allowed among three graphs, but no triple-wise association is allowed as term $\lambda_{i_1}\lambda_{i_2}\lambda_{i_3}$ is not involved. Including higher order polynomials in κ amounts to incorporating higher-order associations among the graphs, which can be achieved by simply exponentiating κ.

Since what the product graph $\mathscr{P}$ offers is a similarity measure among multi-relations, shuffling the order of input graphs $G^{(1)}, \ldots, G^{(J)}$ should not affect $\mathscr{P}$'s topological structure. For SGP, this property is guaranteed by the following theorem:

Theorem 2.2.1. *SGP is commutative (up to graph isomorphism) when κ is commutative.*

We omit the proof. The theorem suggests that a SGP is well-behaved as long as its associated κ is commutative, which is true for both Tensor GP and Cartesian GP as both multiplication and addition operations are order-insensitive.

2.2.4 Optimization Objective

It is often more convenient to equivalently write tensor f as a multi-linear map. E.g., when $J = 2$, tensor (matrix) $f \in \mathbb{R}^{n_1 \times n_2}$ defines a bilinear map from $\mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ to $\mathbb{R}$ via $f(x_1, x_2) := x_1^\top f x_2$ and we have $f_{i_1,i_2} = f(e_{i_1}, e_{i_2})$. Such equivalence is analogous to high-order cases where f defines a multi-linear map from $\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_J}$ to $\mathbb{R}$.

To carry out transductive learning over $\mathscr{P}_\kappa$ (Task 2.2.2), we inject the structure of the product graph into f via a Gaussian random fields prior (Zhu et al., 2003). The negative log-likelihood of the prior $-\log p(f \mid \mathscr{P}_\kappa)$ is the same (up to constant) as the following squared semi-norm

$$\|f\|^2_{\mathscr{P}_\kappa} = vec(f)^\top \mathscr{P}_\kappa^{-1} vec(f) = \sum_{i_1, i_2, \ldots, i_J} \frac{f\left(v^{(1)}_{i_1}, \ldots, v^{(J)}_{i_J}\right)^2}{\kappa\left(\lambda^{(1)}_{i_1}, \ldots, \lambda^{(J)}_{i_J}\right)} \tag{2.2}$$

Our optimization objective is therefore defined as

$$\min_{f \in \mathbb{R}^{n_1 \times \cdots \times n_J}} \quad \ell_{\mathcal{O}}(f) + \frac{\gamma}{2}\|f\|^2_{\mathscr{P}_\kappa} \tag{2.3}$$

where $\ell_{\mathcal{O}}(\cdot)$ is a loss function to be defined later (Section 2.4), $\mathcal{O}$ is the set of training tuples, and γ is a tuning parameter controlling the strength of graph regularization.

[1] While traditional GP only applies to two graphs, we generalize it to the case of multiple graphs.

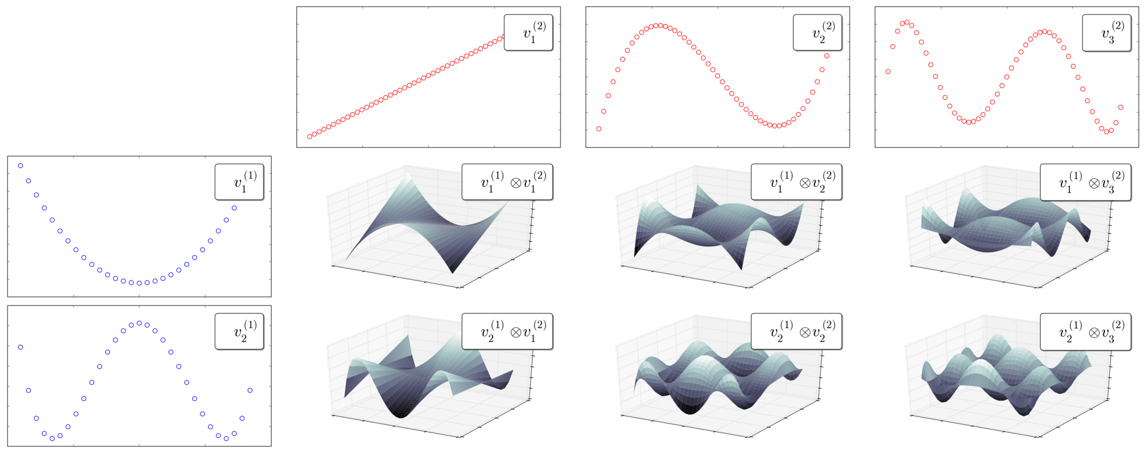

Figure 2.2: An illustration of the eigenvectors of $G^{(1)}$, $G^{(2)}$ and $\mathscr{P}\left(G^{(1)}, G^{(2)}\right)$. The leading nontrivial eigenvectors of $G^{(1)}$ and $G^{(2)}$ are denoted by blue and red curves, respectively. The induced leading nontrivial eigenvectors of $\mathscr{P}\left(G^{(1)}, G^{(2)}\right)$ are illustrated in 3D. If $G^{(1)}$ and $G^{(2)}$ are symmetrically normalized, their eigenvectors (corresponding to eigenvectors of the graph Laplacian) will be ordered by smoothness w.r.t. the graph structures. As a result, eigenvectors of $\mathscr{P}\left(G^{(1)}, G^{(2)}\right)$ will also be ordered by smoothness.

2.3 Approximation

The computational bottleneck for optimization (2.3) lies in evaluating $\|f\|^2_{\mathscr{P}_\kappa}$ and its first-order derivative, due to the extremely large size of $\mathscr{P}_\kappa$. In the following, we first identify the computation bottleneck of using the exact formulation, based on which we propose our convex approximation scheme that reduces the time complexity of evaluating the semi-norm $\|f\|^2_{\mathscr{P}_\kappa}$ from $O\left(\left(\sum_j n_j\right)\left(\prod_j n_j\right)\right)$ to $O\left(\prod_j d_j\right)$, where $d_j \ll n_j$ for $j = 1, \ldots, J$.

2.3.1 Complexity Analysis

The brute-force evaluation of $\|f\|^2_{\mathscr{P}_\kappa}$ according to (2.2) costs $O\left(\left(\prod_j n_j\right)^2\right)$, as one has to evaluate $O\left(\prod_j n_j\right)$ terms inside the summation where each term costs $O\left(\prod_j n_j\right)$. However, redundancies exist and the minimum complexity for the exact evaluation is given as follows

Proposition 2.3.1. *The exact evaluation of semi-norm* $\|f\|_{\mathscr{P}_\kappa}$ *takes* $O\left(\left(\sum_j n_j\right)\left(\prod_j n_j\right)\right)$ *flops.*

Proof. Notice that the collection of all numerators in (2.2), namely $\left[f\left(v^{(1)}_{i_1}, \ldots, v^{(J)}_{i_J}\right)\right]_{i_1,\cdots,i_J}$, is a tensor in $\mathbb{R}^{n_1\times\cdots\times n_J}$ that can be precomputed via

$$\left(\left(f \times_1 V^{(1)}\right) \times_2 V^{(2)}\right) \cdots \times_J V^{(J)} \tag{2.4}$$

where $\times_j$ stands for the j-mode product between a tensor in $\mathbb{R}^{n_1\times\cdots\times n_j \times\cdots\times n_J}$ and $V^{(j)} \in \mathbb{R}^{n_j\times n_j}$. The conclusion follows as the j-th mode product in (2.4) takes $O\left(n_j \prod_j n_j\right)$ flops, and one has to do this for each $j = 1, \ldots, J$. When $J = 2$, (2.4) reduces to the multiplication of three matrices ${V^{(1)}}^\top f V^{(2)}$ at the complexity of $O\left((n_1 + n_2)n_1 n_2\right)$. $\square$

2.3.2 Approximation via Tucker Decomposition

Equation (2.4) implies the key for complexity reduction is to reduce the cost of the j-mode multiplications $\cdot \times_j V^{(j)}$. Such multiplication costs $O\big(n_j \prod_j n_j\big)$ in general, but can be carried out more efficiently if f is structured.

Our solution is twofold: First, we include only the top-d_j eigenvectors in $V^{(j)}$ for each graph $G^{(i)}$, where $d_j \ll n_j$. Hence each $V^{(j)}$ becomes a thin matrix in $\mathbb{R}^{n_j \times d_j}$. Second, we restrict tensor f to be within the linear span of the top $\prod_{j=1}^{J} d_j$ eigenvectors of the product graph $\mathscr{P}_\kappa$

$$f = \sum_{k_1,\cdots,k_J=1}^{d_1,\cdots,d_J} \alpha_{k_1,\cdots,k_J} \bigotimes_j v_{k_j}^{(j)} \tag{2.5}$$

$$= \alpha \times_1 V^{(1)} \times_2 V^{(2)} \times_3 \cdots \times_J V^{(J)} \tag{2.6}$$

The combination coefficients $\alpha \in \mathbb{R}^{d_1 \times \cdots \times d_J}$ is known as the core tensor of Tucker decomposition. In the case where $J = 2$, the above is equivalent to saying $f \in \mathbb{R}^{n_1 \times n_2}$ is a low-rank matrix parametrized by $\alpha \in \mathbb{R}^{d_1 \times d_2}$ such that $f = \sum_{k_1,k_2} \alpha_{k_1,k_2} v_{k_1}^{(1)} v_{k_2}^{(2)\top} = V^{(1)} \alpha V^{(2)\top}$.

Combining (2.5) with the orthogonality property of eigenvectors leads to the fact that $f\big(v_{k_1}^{(1)}, \ldots, v_{k_J}^{(J)}\big) = \alpha_{k_1,\cdots,k_J}$. To see this for $J = 2$, notice $f\big(v_{k_1}^{(1)}, v_{k_2}^{(2)}\big) = v_{k_1}^{(1)\top} f v_{k_1}^{(2)} = v_{k_1}^{(1)\top} V^{(1)} \alpha V^{(2)\top} v_{k_1}^{(2)} = e_{k_1}^\top \alpha e_{k_2} = \alpha_{k_1,k_2}$. Therefore the semi-norm in (2.2) can be simplified as

$$\|f\|_{\mathscr{P}_\kappa}^2 = \|\alpha\|_{\mathscr{P}_\kappa}^2 = \sum_{k_1,\ldots,k_J=1}^{d_1,\cdots,d_J} \frac{\alpha_{k_1,\cdots,k_J}^2}{\kappa\big(\lambda_{k_1}^{(1)}, \ldots, \lambda_{k_J}^{(J)}\big)} \tag{2.7}$$

Comparing (2.7) with (2.2), the number of inside-summation terms is reduced from $O\big(\prod_j n_j\big)$ to $O\big(\prod_j d_j\big)$ where $d_j \ll n_j$. In addition, the cost for evaluating each term inside summation is reduced from $O\big(\prod_j n_j\big)$ to $O(1)$.

Denote by $V_{i_j}^{(j)} \in \mathbb{R}^{d_j}$ the i_j-th row of $V^{(j)}$, we obtain the following optimization by replacing f with α in (2.3)

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^{d_1 \times \cdots \times d_J}} \quad & \ell_{\mathcal{O}}(f) + \frac{\gamma}{2} \|\alpha\|_{\mathscr{P}_\kappa}^2 \\ \text{s.t.} \quad & f = \alpha \times_1 V^{(1)} \times_2 \cdots \times_J V^{(J)} \end{aligned} \tag{2.8}$$

Optimization above has intuitive interpretations. In principle, it is natural to emphasis bases in f that are "smooth" w.r.t. the manifold structure of $\mathscr{P}_\kappa$, and de-emphasis those that are "non-smooth" in order to obtain a parsimonious hypothesis with strong generalization ability. We claim this is exactly the role of regularizer (2.7). To see this, note any nonsmooth basis $\bigotimes_j v_{k_j}^{(j)}$ of $\mathscr{P}_\kappa$ is likely to be associated with small a eigenvalue $\kappa\big(\lambda_{k_1}^{(1)}, \ldots, \lambda_{k_J}^{(J)}\big)$ (illustrated in Fig. 2.2). The conclusion follows by noticing that $\alpha_{k_1,\ldots,k_J}$ is essentially the activation strength of $\bigotimes_j v_{k_j}^{(j)}$ in f (implied by (2.5)), and that (2.7) is going to give any $\alpha_{k_1,\ldots,k_J}$ associated with a small $\kappa\big(\lambda_{k_1}^{(1)}, \ldots, \lambda_{k_J}^{(J)}\big)$ a stronger penalty.

(2.8) is a convex optimization problem over α with any convex $\ell_{\mathcal{O}}(\cdot)$. Spectral approximation techniques for graph-based learning has been found successful in standard classification tasks

(Fergus et al., 2009), which are special cases under our framework when $J = 1$. We introduce this technique for multi-relational learning, which is particularly desirable as the complexity reduction will be much more significant for high-order cases ($J >= 2$).

While f in (2.5) is assumed to be in the Tucker form, other low-rank tensor representation schemes are potentially applicable. E.g., the Candecomp/Parafac (CP) form that further restricts α to be diagonal, which is more aggressive but substantially less expressive. The Tensor-Train decomposition (Oseledets, 2011) offers an alternative representation scheme in the middle of Tucker and CP, but the resulting optimization problem will suffer from non-convexity.

2.4 Optimization

We define $\ell_{\mathcal{O}}(f)$ to be the ranking ℓ_2-hinge loss

$$\ell_{\mathcal{O}}(f) = \frac{\sum_{\substack{(i_1,\dots,i_J) \in \mathcal{O} \\ (i'_1,\dots,i'_J) \in \bar{\mathcal{O}}}} \left(f_{i_1\dots i_J} - f_{i'_1\dots i'_J}\right)_+^2}{|\mathcal{O} \times \bar{\mathcal{O}}|} \tag{2.9}$$

where $(x)_+ = \max(0, 1-x)$, $\bar{\mathcal{O}}$ is the complement of $\mathcal{O}$ w.r.t. all possible multi-relations. Eq. (2.9) encourages the valid tuples in our training set $\mathcal{O}$ to be ranked higher than those corrupted ones in $\bar{\mathcal{O}}$, and is known to be a surrogate of AUC.

We use stochastic gradient descent for optimization as $|\mathcal{O}|$ is usually large. In each iteration, a random valid multirelation $(i_1, \dots, i_J)$ is uniformly drawn from $\mathcal{O}$, a random corrupted multirelation $(i'_1, \dots, i'_J)$ is uniformly drawn from $\bar{\mathcal{O}}$. Each noisy gradient is computed as

$$\nabla_\alpha = \frac{\partial \ell_{\mathcal{O}}}{\partial f}\left(\frac{\partial f_{i_1,\dots,i_J}}{\partial \alpha} - \frac{\partial f_{i'_1,\dots,i'_J}}{\partial \alpha}\right) + \gamma\alpha \oslash \kappa \tag{2.10}$$

where we abuse the notation by defining $\kappa \in \mathbb{R}^{d_1\times\cdots\times d_J}$, $\kappa_{k_1,\dots,k_J} := \kappa\big(\lambda_{k_1}^{(1)}, \dots, \lambda_{k_J}^{(J)}\big)$; $\oslash$ is the element-wise division between tensors. The gradient w.r.t. α in (2.10) is

$$\frac{\partial f_{i_1,\dots,i_J}}{\partial \alpha} = \frac{\partial\big(\alpha \times_1 V_{i_1}^{(1)} \times_2 \cdots \times_J V_{i_J}^{(J)}\big)}{\partial \alpha} \tag{2.11}$$

$$= \bigotimes_j V_{i_j}^{(j)} \quad \in \mathbb{R}^{d_1\times\dots d_J} \tag{2.12}$$

Each SGD iteration costs $O\big(\prod_j d_j\big)$ flops, which is independent from $n_1, n_2, \dots, n_J$. After obtaining the solution $\hat{\alpha}(\kappa)$ of optimization (2.8) for any given SGP $\mathscr{P}_\kappa$, our final predictions in $\hat{f}(\kappa)$ can be recovered via (2.5).

Following AdaGrad (Duchi et al., 2011), we allow adaptive step sizes for each element in α. That is, in the t-th iteration we use

$$\eta_{k_1,\dots,k_J}^{(t)} = \eta_0 \Big/ \Big[\sum_{\tau=1}^{t} \nabla_{\alpha_{k_1,\dots,k_J}}^{(\tau)}{}^2\Big]^{\frac{1}{2}} \tag{2.13}$$

as the step size for $\alpha_{k_1,\ldots,k_J}$, where $\left\{\nabla_{\alpha_{k_1,\ldots,k_J}}^{(\tau)}\right\}_{\tau=0}^{t}$ are the historical gradients associated with $\alpha_{k_1,\ldots,k_J}$ and $\eta_0 = 1$ is the initial step size. AdaGrad is particularly efficient with highly redundant gradients (Duchi et al., 2011), which is our case where the gradient is a regularized rank-2 tensor, according to (2.10) and (2.12).

In practice (especially for large J), the computation cost of tensor operations involving $\bigotimes_{j=1}^{J} V_{i_j}^{(j)} \in \mathbb{R}^{d_1,\ldots,d_J}$ is not ignorable even if $d_1, d_2, \ldots, d_J$ are small. Fortunately, such medium-sized tensor operations in our algorithm are highly parallelable over GPU.

2.5 Experiments and Results

2.5.1 Datasets

We evaluate our method on real-world data in two different domains: the Enzyme dataset (Yamanishi et al., 2008) for compound-protein interaction and the DBLP dataset of scientific publication records. Fig. 2.3 illustrates their heterogeneous objects and relational structures.

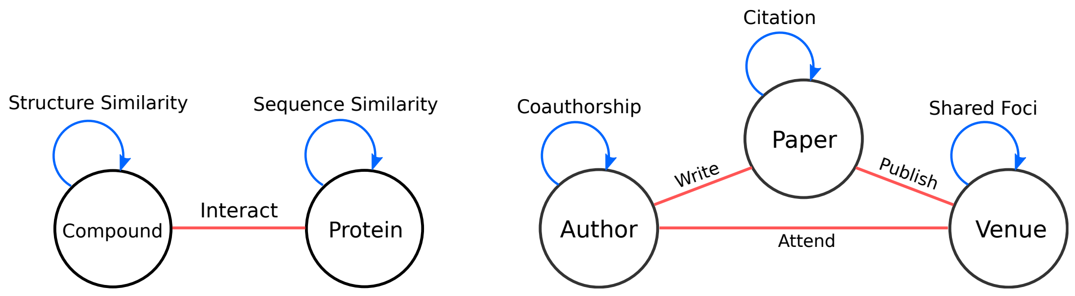

Figure 2.3: The heterogeneous types of objects (the circles) and the relational structures in Enzyme (left) and DBLP (right). The blue edges represent the within-graph relations and the red edges represent the cross-graph interactions. The corresponding tuples in Enzyme is in the form of `(Compound,Protein)`, and in DBLP is in the form of `(Author,Paper,Venue)`.

The Enzyme dataset has been used for modeling and predicting drug-target interactions, which contains a graph of 445 chemical compounds (drugs) and a graph of 664 proteins (targets). The prediction task is to label the unknown compound-protein interactions based on both the graph structures and a small set of 2,926 known interactions. The graph of compounds is constructed based on the SIMCOMP score (Hattori et al., 2003), and the graph of proteins is constructed based on the normalized SmithWaterman score (Smith and Waterman, 1981). While both graphs are provided in the dense form, we converted them into sparse kNN graphs where each vertex is connected with its top 1% neighbors.

As for the DBLP dataset, we use a subset of 34,340 DBLP publication records in the domain of Artificial Intelligence (Tang et al., 2008), from which 3 graphs are constructed as:

- For the author graph ($G^{(1)}$) we draw an edge between two authors if they have coauthored an overlapping set of papers, and remove the isolated authors using a DFS algorithm. We then obtain a symmetric kNN graph by connecting each author with her top 0.5% nearest neighbors using the count of co-authored papers as the proximity measure. The resulting graph has 5,517 vertices with 17 links per vertex on average.

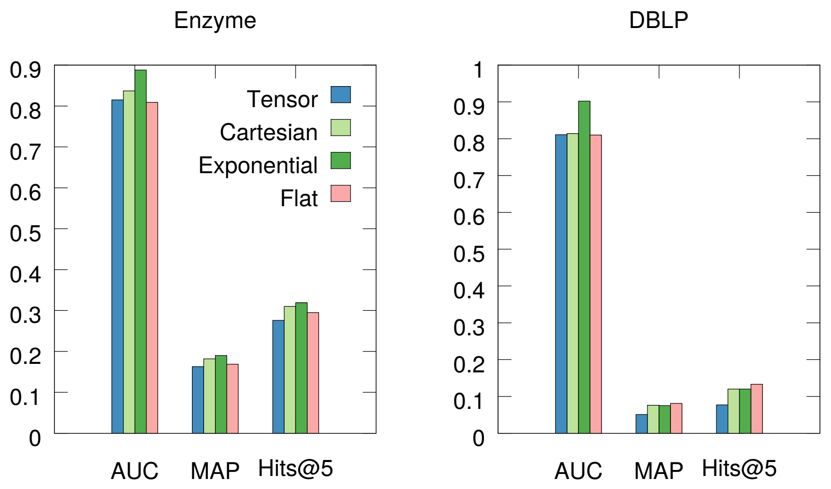

Figure 2.4: Performance of TOP with different SGPs.

Name	$\kappa(x,y)$ $(J=2)$	$\kappa(x,y,z)$ $(J=3)$
Tensor	xy	xyz
Cartesian	$x+y$	$x+y+z$
Exponential	e^{x+y}	$e^{xy+yz+xz}$
Flat	1	1

- For the paper graph ($G^{(2)}$) we connect two papers if both of them cite another paper, or are cited by another paper. Like $G^{(1)}$, we remove isolated papers using DFS and construct a symmetric 0.5%-NN graph. To measure the similarity of any given pair of papers, we represent each paper as a bag-of-citations and compute their cosine similarity. The resulted graph has 11,879 vertices and has an average degree of 50.
- For the venue graph ($G^{(3)}$) we connect two venues if they share similar research focus. The venue-venue similarity is measured by the total number of cross-citations in between, normalized by the size of the two venues involved. The symmetric venue graph has 22 vertices and an average degree of 7.

Tuples in the form of `(Author,Paper,Venue)` are extracted from the publication records, and there are 15,514 tuples (cross-graph interactions) after preprocessing.

2.5.2 Methods for Comparison

- Transductive Learning over Product Graph (**TOP**).
  The proposed method. We explore different choices of κ's for parametrizing the spectral graph product as in Table 2.5.2.

- Tensor Factorization (**TF**) and Graph-regularized TF (**GRTF**). In TF we factorize $f \in \mathbb{R}^{n_1\times\cdots\times n_J}$ as a set of dimensionality-reduced latent factors $C^{d_1,\times\cdots\times d_J}$, $U_1^{n_1\times d_1}, \ldots, U_J \in$

$\mathbb{R}^{n_J \times d_J}$. In GRTF, we further enhanced the traditional TF by adding graph regularizations to the objective function, which enforce the model to be aware of the context information in $G^{(j)}$'s (Narita et al., 2012; Cai et al., 2011);

- One-class Nearest Neighbor (**NN**). We score each tuple $(i_1, \dots, i_J)$ in the test set with $\hat{f}(i_1, \dots, i_J) = \max_{(i'_1,\dots,i'_J)\in\mathcal{O}} \prod_{j=1}^{J} G_{i_j i'_j}$. That is, we assume the tuple-tuple similarity can be factorized as the product of vertex-level similarities across different graphs. We experimented with several other similarity measures and empirically found the multiplicative similarity leads to the best overall performance. Note it does not rely on the presence of any negative examples.
- Ranking Support Vector Machines (Joachims, 2002) (**RSVM**). For the task of completing the missing paper in `(Author,?,Venue)`, we use a Learning-to-Rank strategy by treating `(Author,Venue)` as the query and `Paper` as the document to be retrieved. The query feature is constructed by concatenating the eigen-features of `Author` and `Venue`, where we define the eigen-feature of vertex i_j in graph j as $V_{i_j}^{(j)} \in \mathbb{R}^{d_j}$. The feature for each query-document pair is obtained by taking the tensor product of the query feature and document eigen-feature.
- Low-rank Tensor Kernel Machines (**LTKM**). While traditional tensor-based kernel construction methods for tuples suffer from poor scalability. We propose to speedup by replacing each individual kernel with its low-rank approximation before tensor product, leading to a low-rank kernel of tuples which allows more efficient optimization routines.

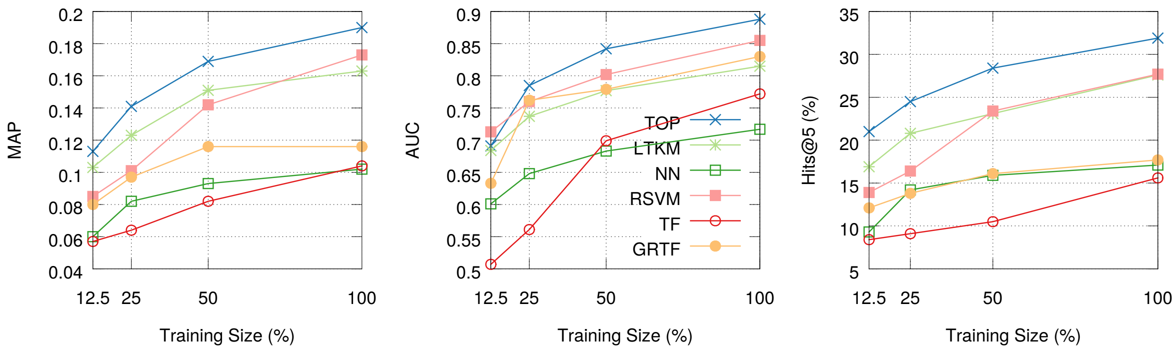

Figure 2.5: Test-set performance of different methods on Enzyme.

For fair comparison, TF, GRTF, RSVM and LTKM use exactly the same loss as that for TOP, i.e. e.q. (2.9). All algorithms are trained using a mini-batched stochastic gradient descent. We use the same eigensystems (eigenvectors and eigenvalues) of the $G^{(j)}$'s as the input for TOP, RSVM and LTKM. The number of top-eigenvalues/eigenvectors d_j for graph j is chosen such that $\lambda_1^{(j)}, \dots, \lambda_{d_j}^{(j)}$ approximately cover 80% of the total spectral energy of $G^{(j)}$. Under this criterion, we use $d_1 = 1,281$, $d_2 = 2,170$, $d_3 = 6$ for DBLP; $d_1 = 150$, $d_2 = 159$ for Enzyme.

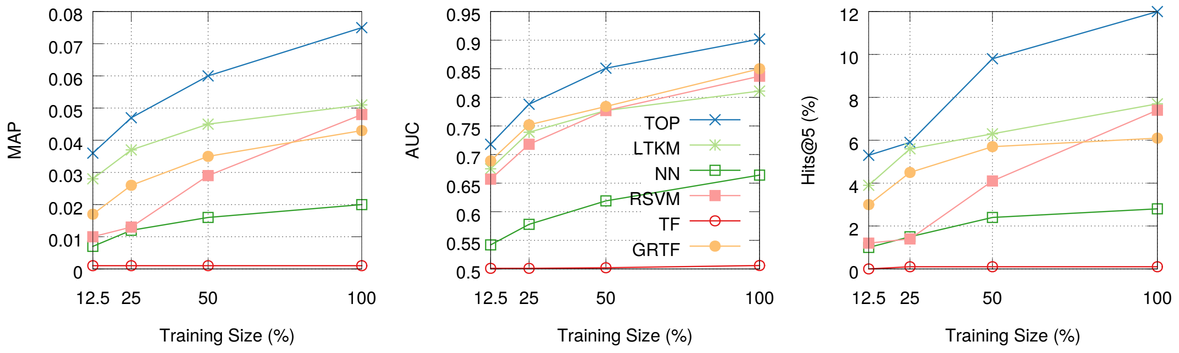

Figure 2.6: Test-set performance of different methods on DBLP.

2.5.3 Experiment Setups

For both datasets, we randomly sample one third of known interactions for training (denoted by $\mathcal{O}$), one third for validation and use the remaining ones for testing. Known interactions in the test set, denoted by $\mathcal{T}$, are treated as positive examples. All tuples not in $\mathcal{T}$, denoted by $\bar{\mathcal{T}}$, are treated as negative. Tuples present in $\mathcal{O}$ are removed from $\bar{\mathcal{T}}$ to avoid misleading results (Bordes et al., 2013).

We measure algorithm performance on Enzyme based on the quality of inferred target proteins given each compound, namely by the ability of completing `(Compound,?)`. For DBLP, the performance is measured by the quality of inferred papers given author and venue, namely by the ability of completing `(Author,?,Venue)`. We use Mean Average Prevision (MAP), Area Under the Curve (AUC) and Hits at Top 5 (Hits@5) as our evaluation metrics.

2.5.4 Results

Fig. 2.4 compares the results of TOP with various parameterizations of the spectral graph product (SGP). Among those, Exponential κ works better on average.

Figs. 2.5 and 2.6 show the main results, comparing TOP (with Exponential κ) with other representative baselines. Clearly, TOP outperforms all the other methods on both datasets in all the evaluation metrics of MAP [2], AUC and Hit@5.

Fig. 2.7 shows the performance curves of TOP on Enzyme over different model sizes (by varying the d_j's). With a relatively small model size compared with using the full spectrum, TOP's performance converges to the optimal point.

2.6 Summary

In this chapter, we presented a novel convex optimization framework for transductive CGRL and a scalable algorithmic solution with guaranteed global optimum and a time complexity that does not depend on the sizes of input graphs. Our experiments on multi-graph data sets provide strong

[2]MAP scores for random guessing are 0.014 on Enzyme and 0.00072 on DBLP, respectively.

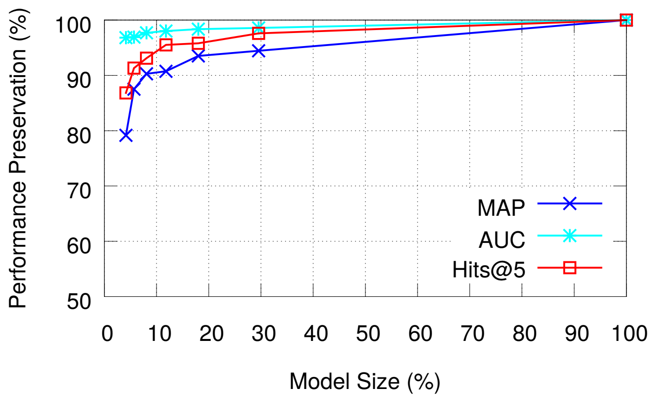

Figure 2.7: Performance of TOP v.s. model size on Enzyme.

evidence for the superior power of the proposed approach in modeling cross-graph inference and large-scale optimization.

Chapter 3

Analogical Inference for Knowledge Graphs

3.1 Background

Multi-relational embedding, or knowledge graph embedding, is the task of finding the latent representations of entities and relations for better inference over knowledge graphs. This problem has become increasingly important in recent machine learning due to the broad range of important applications of large-scale knowledge bases, such as Freebase (Bollacker et al., 2008), DBpedia (Auer et al., 2007) and Google's Knowledge Graph (Singhal, 2012), including question-answering (Ferrucci et al., 2010), information retrieval (Dalton et al., 2014) and natural language processing (Gabrilovich and Markovitch, 2009).

A knowledge base (KB) typically stores factual information as subject-relation-object triplets. The collection of such triplets forms a directed graph whose nodes are entities and whose edges are the relations among entities. Real-world knowledge graph is both extremely large and highly incomplete by nature (Min et al., 2013). How can we use the observed triplets in an incomplete graph to induce the unobserved triples in the graph presents a tough challenge for machine learning research.

Various statistical relational learning methods (Getoor, 2007; Nickel et al., 2015) have been proposed for this task, among which vector-space embedding models are most particular due to their advantageous performance and scalability (Bordes et al., 2013). The key idea in those approaches is to find dimensionality reduced representations for both the entities and the relations, and hence force the models to generalize during the course of compression. Representative models of this kind include tensor factorization (Singhal, 2012; Nickel et al., 2011), neural tensor networks (Socher et al., 2013; Chen et al., 2013), translation-based models (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015b), bilinear models and its variants (Yang et al., 2014; Trouillon et al., 2016), pathwise methods (Guu et al., 2015), embeddings based on holographic representations (Nickel et al., 2016), and product graphs that utilizes additional site information for the predictions of unseen edges in a semi-supervised manner (Liu and Yang, 2015, 2016a). Learning the embeddings of entities and relations can be viewed as a knowledge induction process, as those induced latent representations can be used to make inference about new triplets

$$\begin{tikzcd}
 & \mathit{nucleus} \arrow[rr, "\mathit{surrounded\_by}"] \arrow[drr, dashed, "\mathit{attract}"] & & \mathit{electrons} \arrow[d, "\mathit{made\_of}"] \\
\mathit{sun} \arrow[ur, "\mathit{scale\_down}"] \arrow[rr, "\mathit{surrounded\_by}"] \arrow[drr, "\mathit{attract}"'] & & \mathit{planets} \arrow[ur, "\mathit{scale\_down}"] \arrow[d, "\mathit{made\_of}"] & \mathit{charge} \\
 & & \mathit{mass} \arrow[ur, "\mathit{scale\_down}"] &
\end{tikzcd}$$

Figure 3.1: Commutative diagram for the analogy between the Solar System (blue) and the Rutherford-Bohr Model (red) (atom system). By viewing the atom system as a "miniature" of the solar system (via the $scale\_down$ relation), one is able to complete missing facts (triplets) about the latter by mirroring the facts about the former. The analogy is built upon three basic analogical structures (parallelograms): "sun is to $planets$ as $nucleus$ is to $electrons$", "sun is to $mass$ as $nucleus$ is to $charge$" and "$planets$ are to $mass$ as $eletrons$ are to $charge$".

that have not been seen before.

Despite the substantial efforts and great successes so far in the research on multi-relational embedding, one important aspect is missing, i.e., to study the solutions of the problem from the *analogical inference* point of view, by which we mean to rigorously define the desirable analogical properties for multi-relational embedding of entities and relations, and to provide algorithmic solution for optimizing the embeddings w.r.t. the analogical properties. We argue that analogical inference is particularly desirable for knowledge base completion, since for instance if system A (a subset of entities and relations) is analogous to system B (another subset of entities and relations), then the unobserved triples in B could be inferred by mirroring their counterparts in A. Figure 3.1 uses a toy example to illustrate the intuition, where system A corresponds to the solar system with three concepts (entities), and system B corresponds the atom system with another three concepts. An analogy exists between the two systems because B is a "miniature" of A. As a result, knowing how the entities are related to each other in system A allows us to make inference about how the entities are related to each other in system B by analogy.

Although *analogical reasoning* was an active research topic in classic AI (artificial intelligence), early computational models mainly focused on non-differentiable rule-based reasoning (Gentner, 1983; Falkenhainer et al., 1989; Turney, 2008), which can hardly scale to very large KBs such as Freebase or Google's Knowledge Graph. How to leverage the intuition of analogical reasoning via statistical inference for automated embedding of very large knowledge graphs has not been studied so far, to our knowledge.

It is worth mentioning that analogical structures have been observed in the output of several word/entity embedding models (Mikolov et al., 2013; Pennington et al., 2014). However, those observations stopped there as merely empirical observations. Can we mathematically formulate the desirable analogical structures and leverage them in our objective functions to improve multi-relational embedding? In this case, can we develop new algorithms for tractable inference for the embedding of very large knowledge graphs? These questions present a fundamental challenge which has not been addressed by existing work, and answering these questions are the main contributions we aim in this section. We name this open challenge as the *analogical inference* problem, for the distinction from rule-based *analogical reasoning* in classic AI.

Our specific novel contributions are the following:

1. A new framework that, for the first time, explicitly models analogical structures in multi-relational embedding, and state-of-the-art performance on benchmark datasets;
2. The algorithmic solution for conducting analogical inference in a differentiable manner, whose implementation is as scalable as the fastest known relational embedding algorithms;
3. The theoretical insights on how our framework provides a unified view of several representative methods as its special (and restricted) cases, and why the generalization of such cases lead to the advantageous performance of our method as empirically observed.

3.2 Analogical Inference

Analogical reasoning is known to play a central role in human induction about knowledge (Gentner, 1983; Minsky, 1988; Holyoak et al., 1996; Hofstadter, 2001). Here we provide a mathematical formulation of the analogical structures of interest in multi-relational embedding in a latent semantic space, to support algorithmic inference about the embeddings of entities and relations in a knowledge graph.

3.2.1 Notations

Let $\mathcal{E}$ and $\mathcal{R}$ be the space of all entities and their relations. A knowledge base $\mathcal{K}$ is a collection of triplets $(s, r, o) \in \mathcal{K}$ where $s \in \mathcal{E}, o \in \mathcal{E}, r \in \mathcal{R}$ stand for the subject, object and their relation, respectively. Denote by $v \in \mathbb{R}^{|\mathcal{E}| \times m}$ a look-up table where $v_e \in \mathbb{R}^m$ is the vector embedding for entity e, and denote by tensor $W \in \mathbb{R}^{|\mathcal{R}| \times m \times m}$ another look-up table where $W_r \in \mathbb{R}^{m \times m}$ is the matrix embedding for relation r. Both v and W are to be learned from $\mathcal{K}$.

3.2.2 Relations as Linear Maps

We formulate each relation r as a linear map that, for any given $(s, r, o) \in \mathcal{K}$, transforms the subject s from its original position in the vector space to somewhere near the object o. In other words, we expect the latent representations for any valid (s, r, o) to satisfy

$$v_s^\top W_r \approx v_o^\top \tag{3.1}$$

The degree of satisfaction in the approximated form of (3.1) can be quantified using the inner product of $v_s^\top W_r$ and v_o. That is, we define a bilinear score function as:

$$\phi(s, r, o) = \langle v_s^\top W_r, v_o \rangle = v_s^\top W_r v_o \tag{3.2}$$

Our goal is to learn v and W such that $\phi(s, r, o)$ gives high scores to valid triples, and low scores to the invalid ones. In contrast to some previous models (Bordes et al., 2013) where relations are modeled as additive translating operators, namely $v_s + w_r \approx v_o$, the multiplicative formulation in (3.1) offers a natural analogy to the first-order logic where each relation is treated as a predicate operator over input arguments (subject and object in our case). Clearly, the linear transformation defined by a matrix is a richer operator than the additive transformation defined by a vector. Multiplicative models are also found to substantially outperform additive models empirically (Nickel et al., 2011; Yang et al., 2014).

3.2.3 Normal Transformations

Instead using arbitrary matrices to implement linear maps, a particular family of matrices has been studied for "well-behaved" linear maps. This family is named as the *normal matrices*.

Definition 3.2.1 (Normal Matrix)**.** *A real matrix A is normal if and only if $A^\top A = AA^\top$.*

Normal matrices have nice theoretical properties which are often desirable form relational modeling, e.g., they are unitarily diagonalizable and hence can be conveniently analyzed by the spectral theorem (Dunford et al., 1971). Representative members of the normal family include:

- Symmetric Matrices for which $W_r W_r^\top = W_r^\top W_r = W_r^2$. These includes all diagonal matrices and positive semi-definite matrices, and the symmetry implies $\phi(s, r, o) = \phi(o, r, s)$. They are suitable for modeling symmetric relations such as $is\_identical$.
- Skew-/Anti-symmetric Matrices for which $W_r W_r^\top = W_r^\top W_r = -W_r^2$, which implies $\phi(s, r, o) = -\phi(o, r, s)$. These matrices are suitable for modeling asymmetric relations such as is_parent_of.
- Rotation Matrices for which $W_r W_r^\top = W_r^\top W_r = I_m$, which suggests that the relation r is invertible as W_r^{-1} always exists. Rotation matrices are suitable for modeling 1-to-1 relationships (bijections).
- Circulant Matrices (Gray et al., 2006), which have been implicitly used in recent work on holographic representations (Nickel et al., 2016). These matrices are usually related to the learning of latent representations in the Fourier domain.

In the remaining parts, we denote all the real normal matrices in $\mathbb{R}^{m \times m}$ as $\mathcal{N}_m(\mathbb{R})$.

3.2.4 Analogical Structures

Consider the famous example in the word embedding literature (Mikolov et al., 2013; Pennington et al., 2014), for the following entities and relations among them:

"man is to $king$ as $woman$ is to $queen$"

In an abstract notion we denote the entities by a (as man) , b (as $king$), c (as $woman$) and d (as $queen$), and the relations by r (as $crown$) and r' (as $male \mapsto female$), respectively. These give us the subject-relation-object triplets as follows:

$$a \xrightarrow{r} b, \quad c \xrightarrow{r} d, \quad a \xrightarrow{r'} c, \quad b \xrightarrow{r'} d \tag{3.3}$$

For multi-relational embeddings, r and r' are members of $\mathcal{R}$ and are modeled as linear maps.

The relational maps in (3.3) can be visualized using a commutative diagram (Adámek et al., 2004; Brown and Porter, 2006) from the Category Theory, as shown in Figure 3.2, where each node denotes an entity and each edge denotes a linear map that transforms one entity to the other. We also refer to such a diagram as a "parallelogram" to highlight its particular *algebraic structure*[1].

$$\begin{tikzcd}
\mathrm{a} \arrow[rr, "r"] \arrow[dr, "r'"] & & \mathrm{b} \arrow[dr, "r'"] & \\
& \mathrm{c} \arrow[rr, "r"] & & \mathrm{d}
\end{tikzcd}$$

Figure 3.2: Parallelogram diagram for the analogy of "a is to b as c is to d", where each edge denotes a linear map.

The parallelogram in Figure 3.2 represents a very basic analogical structure which could be informative for the inference about unknown facts (triplets). To get a sense about why analogies would help in the inference about unobserved facts, we notice that for entities a, b, c, d which form an analogical structure in our example, the parallelogram structure is fully determined by symmetry. This means that if we know $a \xrightarrow{r} b$ and $a \xrightarrow{r'} c$, then we can induce the remaining triplets of $c \xrightarrow{r} d$ and $b \xrightarrow{r'} d$. In other words, understanding the relation between man and $king$ helps us to fill up the unknown relation between $woman$ and $queen$.

Analogical structures are not limited to parallelograms, of course, though parallelograms often serve as the building blocks for more complex analogical structures. As an example, in Figure 3.1 of §3.1 we show a compound analogical structure in the form of a triangular prism, for mirroring the correspondent entities/relations between the atom system and the solar system. Formally define the desirable analogical structures in a computationally tractable objective for optimization is the key for solving our problem, which we will introduce next.

3.2.5 Commutative Constraints

Although it is tempting to explore all potentially interesting parallelograms in the modeling of analogical structure, it is computationally intractable to examine the entire powerset of entities as the candidate space of analogical structures. A more reasonable strategy is to identify some desirable properties of the analogical structures we want to model, and use those properties as constraints for reducing the candidate space.

An desirable property of the linear maps we want is that all the directed paths with the same starting node and end node form the *compositional equivalence*. Denoting by "$\circ$" the composition operator between two relations, the parallelogram in Figure 3.2 contains two equivalent compositions as:

$$r \circ r' = r' \circ r \tag{3.4}$$

which means that a is connected to d via either path. We call this the *commutativity* property of the linear maps, which is a necessary condition for forming commutative parallelograms and

[1]Notice that this is different from parallelograms in the geometric sense because each edge here is a linear map instead of the difference between two nodes in the vector space.

therefore the corresponding analogical structures. Yet another example is given by Figure 3.1, where sun can traverse to $charge$ along multiple alternative paths of length three, implying the commutativity of relations $surrounded\_by$, $made\_of$, $scale\_down$.

The composition of two relations (linear maps) is naturally implemented via matrix multiplication (Yang et al., 2014; Guu et al., 2015), hence equation (3.4) indicates

$$W_{r \circ r'} = W_r W_{r'} = W_{r'} W_r \tag{3.5}$$

One may further require the commutative constraint (3.5) to be satisfied for any pair of relations in $\mathcal{R}$ because they may be simultaneously present in the same commutative parallelogram for certain subsets of entities. In this case, we say the relations in $\mathcal{R}$ form a commuting family.

It is worth mentioning that $\mathcal{N}_m(\mathbb{R})$ is not closed under matrix multiplication. As the result, the composition rule in eq. (3.5) may not always yield a legal new relation—$W_{r \circ r'}$ may no longer be normal. However, any commuting family in $\mathcal{N}_m(\mathbb{R})$ is indeed closed under multiplication. This explains the necessity of having a commuting family of relations from an alternative perspective.

3.2.6 Optimization

The generic goal for multi-relational embedding is to find entity and relation representations such that positive triples labeled as $y = +1$ receive higher score than the negative triples labeled as $y = -1$. This can be formulated as

$$\min_{v,W} \; \mathbb{E}_{s,r,o,y \sim \mathcal{D}} \; \ell \left(\phi_{v,W}(s,r,o), y\right) \tag{3.6}$$

where $\phi_{v,W}(s,r,o) = v_s^\top W_r v_o$ is our score function based on the embeddings, ℓ is our loss function, and $\mathcal{D}$ is the data distribution constructed based on the training set $\mathcal{K}$.

To impose analogical structures among the representations, we in addition require the linear maps associated with relations to form a commuting family of normal matrices.

This gives us the objective function for ANALOGY:

$$\min_{v,W} \; \mathbb{E}_{s,r,o,y \sim \mathcal{D}} \; \ell \left(\phi_{v,W}(s,r,o), y\right) \tag{3.7}$$

$$\text{s.t.} \;\; W_r W_r^\top = W_r^\top W_r \quad \forall r \in \mathcal{R} \tag{3.8}$$

$$W_r W_{r'} = W_{r'} W_r \quad \forall r, r' \in \mathcal{R} \tag{3.9}$$

where constraints (3.8) and (3.9) are corresponding to the normality and commutativity requirements, respectively. Such a constrained optimization may appear to be computationally expensive at the first glance. In §3.3, however, we will recast it as a simple lightweight problem for which each SGD update can be carried out efficiently in $O(m)$ time.

3.3 An Efficient Algorithm

The constrained optimization (3.7) is computationally challenging due to the large number of model parameters in tensor W, the matrix normality constraints, and the quadratic number of pairwise commutative constraints in (3.9).

Interestingly, by exploiting the special properties of commuting normal matrices, we will show in Corollary 3.3.2.1 that ANALOGY can be alternatively solved via an another formulation of substantially lower complexity. Our findings are based on the following lemma and theorem:

Lemma 3.3.1. *(Wilkinson and Wilkinson, 1965) For any real normal matrix A, there exists a real orthogonal matrix Q and a block-diagonal matrix B such that $A = QBQ^\top$, where each diagonal block of B is either*

1. *A real valued scalar.*
2. *A 2-dimensional real matrix in the form of $\begin{bmatrix} x & -y \\ y & x \end{bmatrix}$, where both x, y are real scalars.*

The lemma suggests any real normal matrix can be block-diagonalized into an almost-diagonal canonical form. In the following, we use $\mathcal{B}_m^n$ to denote all $m \times m$ almost-diagonal matrices with $n < m$ real scalars on the diagonal.

Theorem 3.3.2. *(Adapted from (Grone et al., 1987)) If a set of real normal matrices $A_1, A_2, \ldots$ form a commuting family, namely*

$$A_i A_j = A_j A_i \; \forall i, j \tag{3.10}$$

then they can be block-diagonalized by the same real orthogonal basis Q.

The theorem implies that the set of dense relational matrices $\{W_r\}_{r\in\mathcal{R}}$, if mutually commutative, can always be *simultaneously block-diagonalized* into another set of sparse almost-diagonal matrices $\{B_r\}_{r\in\mathcal{R}}$.

Corollary 3.3.2.1. *For any given solution (v^*, W^*) of optimization (3.7), there always exists an alternative set of embeddings (u^*, B^*) such that $\phi_{v^*,W^*}(s,r,o) \equiv \phi_{u^*,B^*}(s,r,o)$, $\forall(s,r,o)$, and (u^*, B^*) is given by the solution of:*

$$\min_{u,B} \; \mathbb{E}_{s,r,o,y\sim\mathcal{D}} \; \ell\left(\phi_{u,B}(s,r,o), y\right) \tag{3.11}$$

$$B_r \in \mathcal{B}_m^n \quad \forall r \in \mathcal{R} \tag{3.12}$$

The corollary offers a equivalent but highly efficient formulation for ANALOGY.

proof sketch. With commutative constraints, there must exist some orthogonal matrix Q such that $W_r = QB_rQ^\top$, $B_r \in \mathcal{B}_m^n$, $\forall r \in \mathcal{R}$. We can plug-in these expressions into optimization (3.7) and let $u = vQ$, obtaining

$$\phi_{v,W}(s,r,o) = v_s^\top W_r v_o \tag{3.13}$$

$$= v_s^\top Q B_r Q^\top v_o \tag{3.14}$$

$$= u_s^\top B_r u_o = \phi_{u,B}(s,r,o) \tag{3.15}$$

In addition, it is not hard to verify that constraints (3.8) and (3.9) are automatically satisfied by exploiting the facts that Q is orthogonal and $\mathcal{B}_m^n$ is a commutative normal family. $\square$

Constraints (3.12) in the alternative optimization problem can be handled by simply binding together the coefficients within each of those 2×2 blocks in B_r. Note each B_r consists of only m free parameters, allowing efficient evaluation of the gradient w.r.t. any given triple in $O(m)$.

3.4 A Unified View of Representative Methods

Here we provide a unified view of several embedding models (Yang et al., 2014; Trouillon et al., 2016; Nickel et al., 2016), by showing that they are restricted versions under our framework, hence are implicitly imposing analogical properties. This explains their strong empirical performance as compared to other baselines (§3.5).

3.4.1 DistMult

DistMult (Yang et al., 2014) embeds both entities and relations as vectors, and defines the score function as

$$\phi(s,r,o) = \langle v_s, v_r, v_o \rangle \tag{3.16}$$

$$\text{where } \; v_s, v_r, v_o \in \mathbb{R}^m, \forall s, r, o \tag{3.17}$$

where $\langle \cdot, \cdot, \cdot \rangle$ denotes the generalized inner product.

Proposition 3.4.1. *DistMult embeddings can be fully recovered by ANALOGY embeddings when* $n = m$.

Proof. This is trivial to verify as the score function (3.17) can be rewritten as $\phi(s,r,o) = v_s^\top B_r v_o$ where B_r is a diagonal matrix given by $B_r = \operatorname{diag}(v_r)$. □

Entity analogies are encouraged in DistMult as the diagonal matrices $\operatorname{diag}(v_r)$'s are both normal and mutually commutative. However, DistMult is restricted to model symmetric relations only, since $\phi(s,r,o) \equiv \phi(o,r,s)$.

3.4.2 Complex Embeddings (ComplEx)

ComplEx (Trouillon et al., 2016) extends the embeddings to the complex domain $\mathbb{C}$, where

$$\phi(s,r,o) = \Re\left(\langle v_s, v_r, \overline{v_o} \rangle\right) \tag{3.18}$$

$$\text{where } \; v_s, v_r, v_o \in \mathbb{C}^m, \forall s, r, o \tag{3.19}$$

where $\overline{x}$ denotes the complex conjugate of x.

Proposition 3.4.2. *ComplEx embeddings of embedding size* m *can be fully recovered by ANALOGY embeddings of embedding size* $2m$ *when* $n = 0$.

Proof. Let $\Re(x)$ and $\Im(x)$ be the real and imaginary parts of any complex vector x. We recast ϕ in (3.18) as

$$\begin{aligned} \phi(r,s,o) = &+ \left\langle \Re(v_r), \Re(v_s), \Re(v_o) \right\rangle && (3.20)\\ &+ \left\langle \Re(v_r), \Im(v_s), \Im(v_o) \right\rangle && (3.21)\\ &+ \left\langle \Im(v_r), \Re(v_s), \Im(v_o) \right\rangle && (3.22)\\ &- \left\langle \Im(v_r), \Im(v_s), \Re(v_o) \right\rangle = {v'_s}^\top B_r v'_o && (3.23) \end{aligned}$$

The last equality is obtained via a change of variables: For any complex entity embedding $v \in \mathbb{C}^m$, we define a new real embedding $v' \in \mathbb{R}^{2m}$ such that

$$\begin{cases} (v')_{2k} &= \Re(v)_k \\ (v')_{2k-1} &= \Im(v)_k \end{cases} \quad \forall k = 1, 2, \dots m \tag{3.24}$$

The corresponding B_r is a block-diagonal matrix in $\mathcal{B}^0_{2m}$ with its k-th block given by

$$\begin{bmatrix} \Re(v_r)_k & -\Im(v_r)_k \\ \Im(v_r)_k & \Re(v_r)_k \end{bmatrix} \tag{3.25}$$

□

3.4.3 Holographic Embeddings (HolE)

HolE (Nickel et al., 2016) defines the score function as

$$\phi(s, r, o) = \langle v_r, v_s * v_o \rangle \tag{3.26}$$

$$\text{where } \; v_s, v_r, v_o \in \mathbb{R}^m, \forall s, r, o \tag{3.27}$$

where the association of s and o is implemented via circular correlation denoted by $*$. This formulation is motivated by the holographic reduced representation (Plate, 2003).

To relate HolE with ANALOGY, we rewrite (3.27) in a bilinear form with a circulant matrix $C(v_r)$ in the middle

$$\phi(r, s, o) = v_s^\top C(v_r) v_o \tag{3.28}$$

where entries of a circulant matrix are defined as

$$C(x) = \begin{bmatrix} x_1 & x_m & \cdots & x_3 & x_2 \\ x_2 & x_1 & x_m & & x_3 \\ \vdots & x_2 & x_1 & \ddots & \vdots \\ x_{m-1} & & \ddots & \ddots & x_m \\ x_m & x_{m-1} & \cdots & x_2 & x_1 \end{bmatrix} \tag{3.29}$$

It is not hard to verify that circulant matrices are normal and commute (Gray et al., 2006), hence entity analogies are encouraged in HolE, for which optimization (3.7) reduces to an unconstrained problem as equalities (3.8) and (3.9) are automatically satisfied when all W_r's are circulant.

We further reveal the equivalence between HolE and ComplEx with minor relaxation:

Proposition 3.4.3. *HolE embeddings can be obtained via the following score function*

$$\phi(s, r, o) = \Re\left(\langle v_s, v_r, \overline{v_o} \rangle\right) \tag{3.30}$$

$$\textit{where } \; v_s, v_r, v_o \in \mathfrak{F}(\mathbb{R}^m), \forall s, r, o \tag{3.31}$$

where $\mathfrak{F}(\mathbb{R}^m)$ denotes the image of $\mathbb{R}^m$ in $\mathbb{C}^m$ through the Discrete Fourier Transform (DFT). In particular, the above reduces to ComplEx by relaxing $\mathfrak{F}(\mathbb{R}^m)$ to $\mathbb{C}^m$.

Proof. Let $\mathfrak{F}$ be the DFT operator defined by $\mathfrak{F}(x) = Fx$ where $F \in \mathbb{C}^{m\times m}$ is called the Fourier basis of DFT. A well-known property for circulant matrices is that any $C(x)$ can always be diagonalized by F, and its eigenvalues are given by Fx (Gray et al., 2006).

Hence the score function can be further recast as

$$\begin{aligned}
\phi(r,s,o) &= v_s^\top F^{-1}\operatorname{diag}(Fv_r)Fv_o && (3.32)\\
&= \frac{1}{m}\overline{(Fv_s)}^\top \operatorname{diag}(Fv_r)(Fv_o) && (3.33)\\
&= \frac{1}{m}\langle \overline{\mathfrak{F}(v_s)}, \mathfrak{F}(v_r), \mathfrak{F}(v_o)\rangle && (3.34)\\
&= \Re\left[\frac{1}{m}\langle \overline{\mathfrak{F}(v_s)}, \mathfrak{F}(v_r), \mathfrak{F}(v_o)\rangle\right] && (3.35)
\end{aligned}$$

Let $v_s' = \overline{\mathfrak{F}(v_s)}$, $v_o' = \overline{\mathfrak{F}(v_o)}$ and $v_r' = \frac{1}{m}\mathfrak{F}(v_r)$, we obtain exactly the same score function as used in ComplEx

$$\phi(s,r,o) = \Re\left(\langle v_s', v_r', \overline{v_o'}\rangle\right) \tag{3.36}$$

(3.36) is equivalent to (3.18) apart from an additional constraint that v_s', v_r', v_o' are the image of $\mathbb{R}$ in the Fourier domain. □

3.5 Experiments and Results

3.5.1 Datasets

We evaluate ANALOGY and the baselines over two benchmark datasets for multi-relational embedding released by previous work (Bordes et al., 2013), namely a subset of Freebase (FB15K) for generic facts and WordNet (WN18) for lexical relationships between words.

The dataset statistics are summarized in Table 3.1.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	#train	#valid	#test
FB15K	14,951	1,345	483,142	50,000	59,071
WN18	40,943	18	141,442	5,000	5,000

Table 3.1: Dataset statistics for FB15K and WN18.

3.5.2 Methods for Comparison

We compare the performance of ANALOGY against a variety types of multi-relational embedding models developed in recent years. Those models can be categorized as:

- Translation-based models where relations are modeled as translation operators in the embedding space, including TransE (Bordes et al., 2013) and its variants TransH (Wang et al., 2014), TransR (Lin et al., 2015b), TransD (Ji et al., 2015), STransE (Nguyen et al., 2016) and RTransE (Garcia-Duran et al., 2015).

- Multi-relational latent factor models including LFM (Jenatton et al., 2012) and RESCAL (Nickel et al., 2011) based collective matrix factorization.
- Models involving neural network components such as neural tensor networks (Socher et al., 2013) and PTransE-RNN (Lin et al., 2015b), where RNN stands for recurrent neural networks.
- Pathwise models including three different variants of PTransE (Lin et al., 2015a) which extend TransE by explicitly taking into account indirect connections (relational paths) between entities.
- Models subsumed under our proposed framework (§3.4), including DistMult (Yang et al., 2014) based simple multiplicative interactions, ComplEx (Trouillon et al., 2016) using complex coefficients and HolE (Nickel et al., 2016) based on holographic representations. Those models are implicitly leveraging analogical structures per our previous analysis.
- Models enhanced by external information. We use Node+LinkFeat (NLF) (Toutanova and Chen, 2015) as a representative example, which leverages textual mentions derived from the ClueWeb corpus.

3.5.3 Evaluation Metrics

Following the literature of multi-relational embedding, we use the conventional metrics of Hits@k and Mean Reciprocal Rank (MRR) which evaluate each system-produced ranked list for each test instance and average the scores over all ranked lists for the entire test set of instances.

The two metrics would be flawed for the *negative instances* created in the test phase as a ranked list may contain some positive instances in the training and validation sets (Bordes et al., 2013). A recommended remedy, which we followed, is to remove all training- and validation-set triples from all ranked lists during testing. We use "filt." and "raw" to indicate the evaluation metrics with or without filtering, respectively.

In the first set of our experiments, we used on Hits@k with k=10, which has been reported for most methods in the literature. We also provide additional results of ANALOGY and a subset of representative baseline methods using MRR, Hits@1 and Hits@3, to enable the comparison with the methods whose published results are in those metrics.

3.5.4 Implementation Details

Loss Function: We use the logistic loss for ANALOGY throughout all experiments, namely $\ell(\phi(s,r,o),y) = -\log\sigma(y\phi(s,r,o))$, where σ is the sigmoid activation function. We empirically found this simple loss function to perform reasonably well as compared to more sophisticated ranking loss functions.

Asynchronous AdaGrad: Our C++ implementation[2] runs over a CPU, as ANALOGY only requires lightweight linear algebra routines. We use asynchronous stochastic gradient descent (SGD) for optimization, where the gradients w.r.t. different mini-batches are simultaneously evaluated in multiple threads, and the gradient updates for the shared model parameters are carried

[2]Code available at https://github.com/quark0/ANALOGY.

out without synchronization. While being efficient, asynchronous SGD causes little performance drop when parameters associated with different mini-batches are mutually disjoint with a high probability (Recht et al., 2011). The learning rate is adapted based on historical gradients as in AdaGrad (Duchi et al., 2011).

Creation of Negative Samples: Since only valid triples (positive instances) are explicitly given in the training set, invalid triples (negative instances) need to be artificially created. Specifically, for every positive example (s, r, o), we generate three negative instances (s', r, o), (s, r', o), (s, r, o') by corrupting s, r, o with random entities/relations $s' \in \mathcal{E}$, $r' \in \mathcal{R}$, $o' \in \mathcal{E}$. The union of all positive and negative instances defines our data distribution $\mathcal{D}$ for SGD updates.

Model Selection: We conducted a grid search to find the hyperparameters of ANALOGY which maximize the filtered MRR on the validation set, by enumerating all combinations of the embedding size $m \in \{100, 150, 200\}$, ℓ_2 weight decay factor $\lambda \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ of model coefficients v and W, and the ratio of negative over positive samples $\alpha \in \{3, 6\}$. The resulting hyperparameters for the WN18 dataset are $m = 200, \lambda = 10^{-2}, \alpha = 3$, and those for the FB15K dataset are $m = 200, \lambda = 10^{-3}, \alpha = 6$. The number of scalars (1×1 blocks) on the diagonal of B_r is a hyperparameter manually set to be $\frac{m}{2}$. We set the initial learning rate to be 0.1 for both datasets and adjust it using AdaGrad during optimization. All models are trained for 500 epochs.

3.5.5 Results

Table 3.2 compares the Hits@10 score of ANALOGY with that of 23 competing methods using the published scores for these methods in the literature on the WN18 and FB15K datasets. For the methods not having both scores, the missing slots are indicated by "–". The best score on each dataset is marked in the bold face; if the differences among the top second or third scores are not statistically significant from the top one, then these scores are also bold faced. We used one-sample proportion test (Yang and Liu, 1999) at the 5% p-value level for testing the statistical significances[3].

Table 3.3 compares the methods (including ours) whose results in additional metrics are available. The usage of the bold faces is the same as those in Table 3.2.

In both tables, ANALOGY performs either the best or the 2nd best which is in the equivalent class with the best score in each case according statistical significance test. Specifically, on the harder FB15K dataset in Table 3.2, which has a very large number of relations, our model outperforms all baseline methods. These results provide a good evidence for the effective modeling of analogical structures in our approach. We are pleased to see in Table 3.3 that ANALOGY outperforms DistMult, ComplEx and HolE in all the metrics, as the latter three can be viewed as more constrained versions of our method (as discussed in (§3.4)). Furthermore, our assertion on HolE for being a special case of ComplEx (§3.4) is justified in the same table by the fact that the performance of HolE is dominated by ComplEx.

In Figure 3.3 we show the empirical scalability of ANALOGY, which not only completes one epoch in a few seconds on both datasets, but also scales linearly in the size of the embedding problem. As compared to single-threaded AdaGrad, our asynchronous AdaGrad over 16 CPU

[3]Note proportion tests only apply to performance scores as proportions, including Hits@k, but are not applicable to non-proportional scores such as MRR. Hence we only conducted the proportion tests on the Hits@k scores.

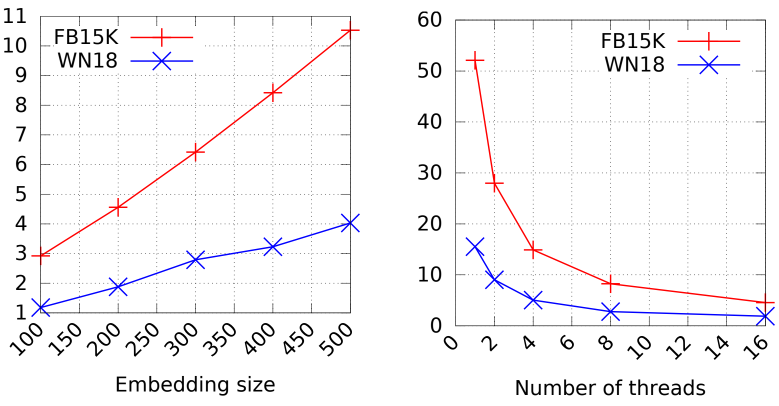

Figure 3.3: CPU run time per epoch (secs) of ANALOGY. The left figure shows the run time over increasing embedding sizes with 16 CPU threads; The right figure shows the run time over increasing number of CPU threads with embedding size 200.

threads offers 11.4x and 8.3x speedup on FB15K and WN18, respectively, on a single commercial desktop.

3.6 Summary

We presented a novel framework for explicitly modeling analogical structures in multi-relational embedding, along with a differentiable objective function and a linear-time inference algorithm for large-scale embedding of knowledge graphs. The proposed approach obtains the state-of-the-art results on two popular benchmark datasets, outperforming a large number of strong baselines in most cases. Although we only focused on the multi-relational inference for knowledge-base embedding, we believe that analogical structures exist in many other machine learning problems beyond the scope of this section. We hope this work shed light on a broad range of important problems where scalable inference for analogical analysis would make an impact, such as machine translation and image captioning (both require modeling cross-domain analogies).

Table 3.2: Hits@10 (filt.) of all models on WN18 and FB15K categories into three groups: (i) 19 baselines without modeling analogies; (ii) 3 baselines and our proposed ANALOGY which implicitly or explicitly enforce analogical properties over the induced embeddings (see §3.4); (iii) One baseline relying on large external data resources in addition to the provided training set.

Models	WN18	FB15K
Unstructured (Bordes et al., 2013)	38.2	6.3
RESCAL (Nickel et al., 2011)	52.8	44.1
NTN (Socher et al., 2013)	66.1	41.4
SME (Bordes et al., 2012)	74.1	41.3
SE (Bordes et al., 2011)	80.5	39.8
LFM (Jenatton et al., 2012)	81.6	33.1
TransH (Wang et al., 2014)	86.7	64.4
TransE (Bordes et al., 2013)	89.2	47.1
TransR (Lin et al., 2015b)	92.0	68.7
TKRL (Xie et al., 2016a)	–	73.4
RTransE (Garcia-Duran et al., 2015)	–	76.2
TransD (Ji et al., 2015)	92.2	77.3
CTransR (Lin et al., 2015b)	92.3	70.2
KG2E (He et al., 2015)	93.2	74.0
STransE (Nguyen et al., 2016)	93.4	79.7
DistMult (Yang et al., 2014)	93.6	82.4
TransSparse (Ji et al., 2016)	93.9	78.3
PTransE-MUL (Lin et al., 2015a)	–	77.7
PTransE-RNN (Lin et al., 2015a)	–	82.2
PTransE-ADD (Lin et al., 2015a)	–	84.6
NLF (with external corpus) (Toutanova and Chen, 2015)	*94.3*	*87.0*
ComplEx (Trouillon et al., 2016)	**94.7**	84.0
HolE (Nickel et al., 2016)	**94.9**	73.9
Our ANALOGY	**94.7**	**85.4**

Table 3.3: MRR and Hits@$\{1,3\}$ of a subset of representative models on WN18 and FB15K. The performance scores of TransE and REACAL are cf. the results published in (Trouillon et al., 2016) and (Nickel et al., 2016), respectively.

	WN18				FB15			
Models	MRR (filt.)	MRR (raw)	Hits@1 (filt.)	Hits@3 (filt.)	MRR (filt.)	MRR (raw)	Hits@1 (filt.)	Hits@3 (filt.)
RESCAL (Nickel et al., 2011)	89.0	60.3	84.2	90.4	35.4	18.9	23.5	40.9
TransE (Bordes et al., 2013)	45.4	33.5	8.9	82.3	38.0	22.1	23.1	47.2
DistMult (Yang et al., 2014)	82.2	53.2	72.8	91.4	65.4	24.2	54.6	73.3
HolE (Nickel et al., 2016)	93.8	61.6	**93.0**	**94.5**	52.4	23.2	40.2	61.3
ComplEx (Trouillon et al., 2016)	94.1	58.7	**93.6**	**94.5**	69.2	24.2	59.9	75.9
Our ANALOGY	**94.2**	**65.7**	**93.9**	**94.4**	**72.5**	**25.3**	**64.6**	**78.5**

Part II

Learning with Graph Induction

Chapter 4

Nonparametric Learning of Graph Diffusions

4.1 Background

Graph representation of data is ubiquitous in machine learning. In many scenarios, we are given a partially labeled graph with only a small number of labeled vertices, and the task is to predict the missing labels of the large number of unlabeled vertices.

With limited supervision, it is often crucial to leverage the intrinsic manifold structure of both the labeled and unlabeled vertices during the training phase. A variety of graph-based semi-supervised learning (SSL) algorithms have been proposed under this motivation, including label propagation (Zhu and Ghahramani, 2002), Gaussian random fields (Zhu et al., 2003) and Laplacian Support Vector Machines (Melacci and Belkin, 2011). Many of those approaches rely on the assumption that strongly connected vertices are likely to share the same labels, and fall under the manifold regularization framework (Belkin et al., 2006) where the graph Laplacian (Chung, 1997) plays a key role.

Given a graph, the graph Laplacian characterizes how the label of each vertex diffuses (propagates) from itself to its *direct* neighbors. While the graph Laplacian in its original form may not be sufficiently expressive for modeling complex graph transduction patterns, it has been shown that a rich family of important graph transduction patterns under various assumptions, including multi-step random walk, heat diffusion (Kondor and Lafferty, 2002) and von-Neumann diffusion (Ito et al., 2005), can be incorporated into SSL by transforming the spectrum[1] of the graph Laplacian with nonnegative nondecreasing functions (Smola and Kondor, 2003; Zhu et al., 2004; Johnson and Zhang, 2008). The collection of those functions are referred to as the *Spectral Transformation* (ST) family.

Despite of the expressiveness of the ST family, how to find the optimal ST for any problem in hand is an open challenge. While manual specification (Smola and Kondor, 2003; Johnson and Zhang, 2008) is clearly suboptimal, various approaches have been proposed to automatically find the optimal ST. Among the existing works, parametric approaches assume the optimal ST belongs some pre-specified function family (e.g. the polynomial or exponential), and then find

[1]In this chapter, we refer to the spectrum of a matrix as the multiset of its eigenvalues.

the function hyperparameter via grid search or curve-fitting (Kunegis and Lommatzsch, 2009). However, the fundamental question about how to choose the function family is left unanswered, and it is not clear whether commonly used parametric function families are rich enough to subsume the true optimal ST. On the other hand, a more flexible nonparametric framework based on kernel-target alignment has been studied in (Zhu et al., 2004), where the optimization of ST is efficiently solved via quadratically constrained quadratic programming (QCQP). However, the target matrix itself may be unreliable as it is constructed based a very small number of observed labels, and it is not conclusive whether a better alignment score always leads to a better prediction performance.

Note all the above approaches adopt *two-step* procedures, where the optimal ST is empirically estimated in some preprocessing step before SSL is carried out (with the ST obtained in the previous step). We argue that the separation of ST optimization from SSL may result in suboptimal performance, as combining the two steps together will allow the learned ST to better adapt to the problem structure.

This section addresses the aforementioned challenge by proposing a principled optimization framework which *simultaneously* conducts SSL and finds the optimal ST for the graph Laplacian used in SSL. Starting with the natural formulation of the joint optimization, we show how it can be reformulated as an equivalent *convex* optimization problem via Lagrangian duality, and then derive an efficient algorithm using the bundle method. We refer to our new approach as *Adaptive Spectral Transform* (AST), meaning that the ST is automatically adapted to the problem in hand and its target domain.

Besides improved performance over benchmark datasets across various domains, insights are provided regarding the advantageous performance of AST by revisiting an existing theorem on SSL from a new angle. Specifically, we show that AST actually aims to asymptotically minimize the generalization error bound of SSL.

4.2 Nonparametric Adaptation of Graph Spectrum

4.2.1 Manifold Regularization

Given a graph G of m vertices, where each vertex denotes an instance and each edge encodes the affinity between a pair of instances. Suppose only a very small set $\mathcal{T}$ of l vertices has been labeled where $l \ll m$, our task is to predict the missing labels of the remaining $m - l$ vertices based on both the l labeled vertices and the intrinsic manifold structure of G.

Denote by y_i the true label and by $\mathrm{f}_i \in \mathbb{R}$ the system-estimated score for vertex i, resp. In order to leverage the labels, we hope f_i and y_i to be as close as possible for all $i \in \mathcal{T}$. Meanwhile, to leverage the large amount of unlabeled vertices, we want the scores for all (both labeled and unlabeled) vertices to be smooth w.r.t. the graph structure of G. The two desired properties entail the following optimization problem:

$$\min_{\mathrm{f} \in \mathbb{R}^m} \quad \frac{1}{l} \sum_{i \in \mathcal{T}} \ell(\mathrm{f}_i, y_i) + \gamma \mathrm{f}^\top \mathcal{L} \mathrm{f} \tag{4.1}$$

where the first term is the empirical loss of the system-predicted scores $\mathrm{f} \in \mathbb{R}^m$, $\mathcal{L}$ in the second term is the normalized graph Laplacian matrix associated with G characterizing G's manifold structure. Specifically, denote by A the adjacency matrix of G, by D a diagonal matrix of degrees with $d_{ii} = \sum_j a_{ij}$ and by $L = D - A$ the graph Laplacian. The normalized graph Laplacian is defined as $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$, with its eigensystem denoted by $\{(\lambda_i, \phi_i)\}_{i=1}^m$. For convenience, we assume the eigenvalues of $\mathcal{L}$ are in the increasing order: $\lambda_1 \leq \lambda_2 \ldots \leq \lambda_m$. It is well known that the smallest eigenvalue λ_1 is always zero, and that ϕ_i's with small indices tend to be "smoother" over the data manifold than those with large indices (Chung, 1997).

In (4.1), the label information is encoded in the empirical loss $\ell(\mathrm{f}_i, y_i)$. E.g., one could specify $\ell(\mathrm{f}_i, y_i)$ to be $(\mathrm{f}_i - y_i)^2$. The manifold assumption is encoded in the second term (a.k.a. the manifold regularizer) involving the graph Laplacian, satisfying

$$\mathrm{f}^\top \mathcal{L} \mathrm{f} = \frac{1}{2} \sum_{i \sim j} a_{ij} \left(\frac{\mathrm{f}_i}{\sqrt{d_{ii}}} - \frac{\mathrm{f}_j}{\sqrt{d_{jj}}} \right)^2 \tag{4.2}$$

$$= \sum_{i=1}^m \lambda_i \langle \phi_i, \mathrm{f} \rangle^2 \tag{4.3}$$

Eq. (4.2) suggests that the regularizer essentially encourages scores f_i, f_j (normalized by the squared root of degrees) to be close when vertices i, j are strongly connected in G, namely when a_{ij} is large. An alternative perspective, as implied by (4.3), is to think of the regularizer as penalizing the projection of f onto different bases (the $\phi_i's$) with different weights (the $\lambda_i's$), where the smooth components in f are going to receive lighter penalty than the nonsmooth ones.

4.2.2 Spectral Transform

Although the graph Laplacian gives a nice characterization about how vertices in G influence their direct neighbors, it is not sufficiently expressive for modeling complex label propagation patterns, such as multi-step influence from a given vertex to its indirect neighbors and the decay of such influence. As a simple remedy to incorporate a richer family of label propagation patterns over the manifold, various methods have been proposed based on transforming the spectrum of $\mathcal{L}$ using some nonnegative nondecreasing function, known as the spectral transformation (Smola and Kondor, 2003; Zhu et al., 2004; Johnson and Zhang, 2008).

As an example, by taking the exponential of the Laplacian spectrum, one gets $\sum_{i=1}^m e^{\beta\lambda_i} \phi_i \phi_i^\top = e^{\beta\mathcal{L}}$ where β is a nonnegative scalar. The transformed Laplacian has a neat physical interpretation in terms of heat diffusion process, and is closely related to infinite random walk with decay over the manifold (Kondor and Lafferty, 2002). From (4.3)'s perspective, the replacement of λ_i with e^{λ_i} can be viewed as a way to exaggerate the difference in weighing the bases. That is, the nonsmooth components in f are going to receive a larger relative penalty during the optimization after the exponential transformation.

Formally, we define the *Spectral Transformation* (ST) over $\mathcal{L}$ as $\sigma(\mathcal{L}) := \sum_{i=1}^m \sigma(\lambda_i) \phi_i \phi_i^\top$, where $\sigma : \mathbb{R}_+ \mapsto \mathbb{R}_+$ is a nondecreasing function which transforms each Laplacian eigenvalue to a nonnegative scalar. Besides the aforementioned diffusion kernel where $\sigma(x) = e^{\beta x}$, other commonly used STs include $\sigma(x) = x + \beta$ (Gaussian field), $\sigma(x) = \frac{1}{(\alpha - x)^\beta}$ (multi-step random walk), $\sigma(x) = \left[\cos\left(\frac{\pi}{4}x\right)\right]^{-1}$ (inverse cosine) (Smola and Kondor, 2003; Kunegis and Lommatzsch, 2009), etc.

The ST-enhanced SSL is formulated as

$$\min_{\mathrm{f}\in\mathbb{R}^m} \quad \frac{1}{l}\sum_{i\in\mathcal{T}} \ell(\mathrm{f}_i, y_i) + \gamma \mathrm{f}^\top \sigma(\mathcal{L})\mathrm{f} \tag{4.4}$$

4.2.3 Adapting the Spectral Transform

The nature of SSL described in (4.4) crucially depends our choice of ST. It is a common practice to manually specify σ (Smola and Kondor, 2003; Johnson and Zhang, 2008) or to learn the hyperparameter of σ within a pre-specified function family (Shawe-Taylor and Kandola, 2002; Kunegis and Lommatzsch, 2009). Both methods are suboptimal when the true σ^* lies in a broader function space.

In this section, we focus on automatically learning σ^* from data with no prior assumption on its function form. In terms of SSL, we argue it suffices to learn $\{\sigma^*(\lambda_i)\}_{i=1}^m$ instead of the analytical expression of σ^*, as the objective in (4.4) is uniquely determined by these m transformed eigenvalues. Therefore, in the following we switch our focus from the task of making σ adaptive to the equivalent task of making each $\sigma(\lambda_i)$ adaptive.

Define $\theta \in \mathbb{R}^m$ where $\theta_i := \sigma(\lambda_i)^{-1}$. We are going to focus on learning θ as notation-wise it is more convenient to work with the reciprocals. After substituting the ST σ with θ in (4.4), the optimization becomes

$$\min_{\mathrm{f}\in\mathbb{R}^m} \quad \underbrace{\frac{1}{l}\sum_{i\in\mathcal{T}} \ell(\mathrm{f}_i, y_i) + \gamma \sum_{i=1}^{m} \theta_i^{-1} \left\langle \phi_i, \mathrm{f}\right\rangle^2}_{\mathcal{C}(\mathrm{f};\theta)} \tag{4.5}$$

When $\theta_i = 0$, we define $\theta_i^{-1} := 0$ as its pseudo-inverse. For brevity, in the following we assume all the θ_i's are strictly positive. The singular case where some θ_i's are exactly zero will be studied specifically in Section 4.3.4.

To determine θ for (4.5), Zhu et al. (Zhu et al., 2004) proposed a two-step procedure based on empirical kernel-target alignment. In the first step, an empirical estimation about θ is obtained by maximizing the alignment score between the kernel matrix implied by θ, i.e. $\sum_i \theta_i \phi_i \phi_i^\top$, and a target kernel matrix induced from a small amount of observed labels. In the second step, the estimated $\hat{\theta}$ is plugged-into the SSL objective (4.5) for learning f.

Different from existing (manual/parametric/two-step) approaches, we argue that it is beneficial to put the task of finding the optimal θ^* and the task of SSL into a unified optimization framework, as the two procedures can mutually reinforce each other, thus making θ^* more adapted to the problem structure.

It may appear straightforward to approach the aforementioned goal by minimizing (4.5) w.r.t. f and w.r.t. θ in an alternating manner. Unfortunately, the resulting optimization is non-convex, and a meaningless solution can be obtained by simply setting all the θ_i^{-1}'s to zero.

Instead, we propose to achieve this goal by solving the following optimization problem (AST)

$$\min_{\theta\in\Theta} \left(\min_{\mathrm{f}\in\mathbb{R}^m} \quad \mathcal{C}(\mathrm{f}, \theta) \right) + \tau \|\theta\|_1 \tag{4.6}$$

where $\underset{\sim}{C}(\mathrm{f};\theta)$ is the SSL objective defined in (4.5), τ is a positive scalar-valued tuning parameter, and Θ denotes the set of all possible reciprocals of the transformed Laplacian spectrum

$$\begin{aligned}\Theta &= \left\{\theta : \theta_i = \sigma(\lambda_i)^{-1}, \forall i = 1, 2 \ldots m,\ \sigma \text{ is a valid ST}\right\} \\ &= \{\theta : \theta_1 \geq \theta_2, \ldots \geq \theta_m \geq 0\}\end{aligned} \tag{4.7}$$

The second equality is derived based on the facts that (i) the λ_i's are in the increasing order (ii) σ can be *any* nonnegative nondecreasing function.

The intuition behind optimization (4.6) is that we want the optimal θ^* (and the associated optimal ST) to simultaneously satisfy the following criteria:

(a) It should tend to minimize the SSL objective (4.5). As in multiple kernel learning (Lanckriet et al., 2004; Bach et al., 2004), this is arguably the most natural and effective way to make θ^* adaptive to the problem structure.

(b) It should have a moderate ℓ_1-norm. Namely the "transformed" data manifold should have a moderate total effective resistance (Boyd, 2006). As we will see later, this additional requirement is crucial as it precludes degenerate solutions. It also makes our bundle method more efficient by sparsifying θ (Section 4.3).

4.3 Optimization

Let us present our optimization strategies for solving (4.6), starting with the following theorem

Theorem 4.3.1 (Convexity of AST)**.**
(4.6) *is a convex optimization problem over* θ*.*

After presenting the proof for Theorem 4.3.1 (Section 4.3.1), we propose our method to compute the gradient for (4.6)'s structured objective function in Section 4.3.2, and offer a bundle method for efficient optimization in Section 4.3.3. We will study the singular case where some θ_i's are allowed to be exactly zero in Section 4.3.4, which can be particularly useful in large-scale scenarios. The SSL subroutine for AST is discussed in Section 4.3.5.

4.3.1 Convexity

We proof Theorem 4.3.1 by first reformulating optimization (4.6)'s objective function into an equivalent minimax-type function via Lagrangian duality, and then showing the convexity of the equivalent optimization problem.

The Lagrangian dual for $\underset{\sim}{C}(\mathrm{f};\theta)$ is

$$\underbrace{-\omega(-u) - \frac{1}{4\gamma}\sum_{i=1}^{m} \theta_i \left\langle \phi_i, u \right\rangle^2}_{\bar{C}(u;\theta)} \tag{4.8}$$

where $\omega(\cdot)$ is the conjugate function for $\sum_{i\in\mathcal{T}} \ell(\mathrm{f}_i, y_i)$. It is not hard to verify that the Slater's condition holds for optimization (4.5), i.e. $\min_{\mathrm{f}\in\mathbb{R}^m} \underset{\sim}{C}(\mathrm{f};\theta)$, thus strong duality ensures that

$$\min_{\mathrm{f}\in\mathbb{R}^m} \underset{\sim}{C}(\mathrm{f};\theta) = \max_{u\in\mathbb{R}^m} \bar{C}(u;\theta) \tag{4.9}$$

and optimization (4.6) for AST can be recast as

$$\min_{\theta\in\Theta} \underbrace{\left(\max_{u\in\mathbb{R}^m} \;\bar{C}(u;\theta)\right)}_{g(\theta)} + \tau\|\theta\|_1 \tag{4.10}$$

We claim the resulting equivalent problem (4.10) is convex over θ. To see this, notice that $\bar{C}(u;\theta)$ defined in (4.8) is an affine over θ for each given u, and recall that the pointwise maximum of any set of convex functions (affines) is still convex, the first structured term $g(\theta)$ in optimization (4.10), i.e. $\max_{u\in\mathbb{R}^m} \bar{C}(u;\theta)$, is hence convex over θ. The conclusion follows by further noticing the second term $\|\theta\|_1$ in (4.10) is also a convex function, and that Θ in (4.7) is a convex domain.

4.3.2 Deriving the Structured Gradient

In this section, we discuss our method to compute the gradient of $g(\theta) := \max_u \bar{C}(u;\theta)$ in (4.10), denoted by $\nabla_\theta g(\theta)$, as a prerequisite for subsequent optimization algorithms. We rely on Danskin's Theorem (Danskin, 1966) as $g(\theta)$ is the maximum of infinite number of functions:

Theorem 4.3.2 (Danskin's Theorem)**.** *If function $g(\theta)$ is in the form of $g(\theta) := \max_{u\in\mathcal{U}} \bar{C}(u;\theta)$ where $\mathcal{U}$ is a compact space and $\bar{C}(\cdot;\theta)$ is a differentiable function with $\bar{C}(u;\theta)$ and $\nabla\bar{C}(u;\theta)$ depending continuously on u and θ, then the subgradient of $g(\theta)$, i.e. $\partial_\theta g(\theta)$, is given by $\partial_\theta\bar{C}(\hat{u};\theta)$ where $\hat{u} \in \operatorname{argmax}_{u\in\mathcal{U}} \bar{C}(u;\theta)$.*

For our case $\mathcal{U} := \mathbb{R}^m$ and the subgradient $\partial_\theta g(\theta)$ can be substituted with gradient $\nabla_\theta g(\theta)$ as the function of interest is differentiable. Recall that we have assumed all θ_i's to be positive, $\bar{C}(u;\theta)$ is strictly convex over u and therefore $\hat{u} := \operatorname{argmax}_u \bar{C}(u;\theta)$ is always unique.

Suppose $\hat{u}$ is given, following Theorem 4.3.2 we have

$$\nabla_\theta g(\theta) = \nabla_\theta \bar{C}(\hat{u};\theta) = -\frac{\left(\langle\phi_1,\hat{u}\rangle^2, \ldots, \langle\phi_m,\hat{u}\rangle^2\right)^\top}{4\gamma} \tag{4.11}$$

To compute the R.H.S. of (4.11), we have to get $\hat{u}$ in advance via solving $\max_u \bar{C}(u;\theta)$. In case the conjugate function involved in $\bar{C}(u;\theta)$ is hard to work with, it is more convenient to first obtain the primal solution $\hat{\mathrm{f}}$ by solving the corresponding primal problem $\min_{\mathrm{f}} \underline{C}(\mathrm{f};\theta)$ described in (4.5), and then recover the dual solution $\hat{u}$ from $\hat{\mathrm{f}}$ via the K.K.T. condition.

According to the stationarity condition, $\hat{u}$ and $\hat{\mathrm{f}}$ must satisfy

$$\hat{u} = 2\gamma\left(\sum_{i=1}^m \theta_i^{-1}\phi_i\phi_i^\top\right)\hat{\mathrm{f}} \tag{4.12}$$

This suggests an alternative to (4.11), i.e. to compute the gradient of $g(\theta)$ directly based on the primal variable via

$$\nabla_\theta g(\theta) = -\gamma\left(\frac{\langle\phi_1,\hat{\mathrm{f}}\rangle^2}{\theta_1^2}, \ldots, \frac{\langle\phi_m,\hat{\mathrm{f}}\rangle^2}{\theta_m^2}\right)^\top \tag{4.13}$$

where $\hat{\mathrm{f}} := \operatorname{argmin}_{\mathrm{f}} \underline{C}(\mathrm{f},\theta)$ is obtained by applying any SSL algorithm[2] to (4.5).

[2]Many off-the-shelf SSL solvers can be easily modified for solving the primal problem (4.5).

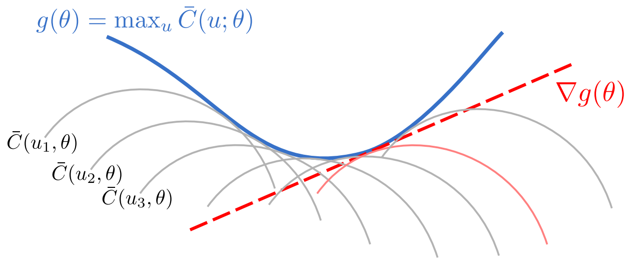

Figure 4.1: A visual interpretation of Danskin's theorem. Computing the derivative of $\nabla g(\theta)$ is equivalent to solving for $\hat{u}$ and computing the derivative of $\bar{C}(\hat{u};\theta)$.

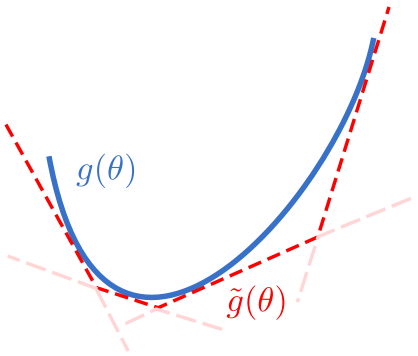

Figure 4.2: We maintain a piecewise lowerbound $\tilde{g}(\theta)$, which keeps being refined during optimization to better approximate $g(\theta)$.

4.3.3 Bundle Method

After obtaining $\nabla_\theta g(\theta)$ according to section 4.3.2, it is straightforward to minimize the AST objective in (4.10): $g(\theta) + \gamma\|\theta\|_1$ via the subgradient method or proximal gradient method. However, both algorithms have slow convergence rate, and it can be tricky to choose a suitable step size to ensure efficient convergence.

We propose to use the bundle method for (4.10) (equivalently, (4.6)), which has been found particularly efficient in solving problems involving structured loss functions (Kloft et al., 2009; Sra et al., 2012). Our method is a variant of bundle method for regularized risk minimization (BMRM) (Teo et al., 2010), and subsumes the semi-infinite linear programming (SILP) for large-scale multiple kernel learning (Sonnenburg et al., 2006).

The key idea is to replace the "tough" part in (4.10), i.e. $g(\theta)$, with an "easy" piecewise linear function $\tilde{g}(\theta)$ that lowerbounds the original $g(\theta)$, as shown in Figure 4.2. After the replacement, optimization (4.10) becomes

$$\min_{\theta\in\Theta} \quad \tilde{g}(\theta) + \tau\|\theta\|_1 \tag{4.14}$$

We then alternate between solving the surrogate problem (4.14) and refining the lowerbound $\tilde{g}(\theta)$ until convergence. Note (4.14) is a Linear Programming (LP), as its objective function is piecewise linear and its feasible set Θ defined in (4.7) is a polyhedron.

To obtain a piecewise lowerbound $\tilde{g}(\theta)$ for $g(\theta)$, recall any convex function can be lowerbounded by its tangents. Hence it suffices to let $\tilde{g}(\theta)$ be the supremum of a set of tangents associated with historical iterations. Specifically, we define $\tilde{g}(\theta)$ at the t-th iteration as

$$\tilde{g}^{(t)}(\theta) := \max_{0\leq i\leq t-1} \quad g(\theta^{(i)}) + \left\langle \nabla g\left(\theta^{(i)}\right), \theta - \theta^{(i)} \right\rangle \tag{4.15}$$

where superscript "$^{(i)}$" indexes the quantity associated with the i-th iteration. It is not hard to verify that $g^{(t)}(\theta) \leq g(\theta)$ always holds, and that $g^{(t)}(\theta)$ tends to better approximate $g(\theta)$ as t increases.

Details of the bundle method for AST is presented in Algorithm 1.

Algorithm 1: Bundle Method for AST

```
Input
  $\epsilon$   desired convergence accuracy
  $\mathcal{L}$   normalized graph Laplacian of $G$
  $\ell$   loss function based on available labels
  $\gamma$   tuning parameter for manifold regularization in (4.5)
  $\tau$   tuning parameter for the $\ell_1$-norm in (4.6)
Output
  f   system-inferred vertex labels
  $\theta$   system-inferred reciprocals of
        the transformed Laplacian eigenvalues
Initialization
$t \leftarrow 0$;
/*take pseudo-inverse when necessary*/;
$\{\lambda_i, \phi_i\}_{i=1}^m \leftarrow \mathrm{eig}(\mathcal{L})$, $\{\theta_i^{(0)} \leftarrow \lambda_i^{-1}\}_{i=1}^m$;
do
    /*solve (4.5) via standard SSL*/;
    $\mathrm{f}^{(t)} \leftarrow \mathrm{argmin}_{\mathrm{f}\in\mathbb{R}^m}\ \underset{\sim}{C}(\mathrm{f};\theta^{(t)})$;
    $g(\theta^{(t)}) \leftarrow \underset{\sim}{C}(\mathrm{f}^{(t)};\theta^{(t)})$;
    /*according to (4.13)*/;
    $\nabla g(\theta^{(t)}) \leftarrow -\gamma\left(\frac{\langle\phi_1,\mathrm{f}^{(t)}\rangle^2}{\theta_1^{(t)2}},\dots,\frac{\langle\phi_m,\mathrm{f}^{(t)}\rangle^2}{\theta_m^{(t)2}}\right)^\top$;
    $t \leftarrow t+1$;
    /*update the piecewise-linear lowerbound*/;
    $\tilde{g}^{(t)}(\theta) \leftarrow \max_{0\le i\le t-1}\ g(\theta^{(i)}) + \left\langle \nabla g(\theta^{(i)}), \theta-\theta^{(i)}\right\rangle$;
    /*solve the linear programming*/;
    $\theta^{(t)} \leftarrow \mathrm{argmin}_{\theta\in\{\theta|\theta_1\ge\theta_2\ge\dots\ge\theta_m\ge 0\}}\ \tilde{g}^{(t)}(\theta)+\tau\|\theta\|_1$;
while $g(\theta^{(t-1)}) + \|\theta^{(t-1)}\|_1 - \tilde{g}^{(t)}(\theta^{(t)}) - \|\theta^{(t)}\|_1 > \epsilon$;
/*terminate when the piecewise-linear lowerbound is sufficiently close to the original
function*/;
```

4.3.4 Exploiting Singularity

Now let us focus on the singular cases where some θ_i's (and their pseudo-inverse θ_i^{-1}'s) are exactly zero. This may happen in two scenarios:

(a) During the bundle method, some θ_i's are shrunk to zero after solving the LP (4.14) due to the presence of the ℓ_1-regularization over θ.

(b) Small-valued θ_i's associated with those nonsmooth ϕ_i's are truncated to be zero for the sake of scalability. This strategy will substantially reduce the parameter size of SSL, and has been successfully applied to large-scale problems (Fergus et al., 2009).

In the following, we will assume $\theta_i > 0$ for $1 \leq i \leq k$ and $\theta_i = 0$ for $k < i \leq m$, where $k \ll m$. To handle the singular case, we modify $\underset{\sim}{C}(\mathrm{f}; \theta)$ in (4.5) as

$$\frac{1}{l}\sum_{i\in\mathcal{T}} \ell(\mathrm{f}_i, y_i) + \gamma \sum_{1\leq i\leq k} \theta_i^{-1} \langle \phi_i, \mathrm{f}\rangle^2 + \sum_{k<i\leq m} \mathbf{1}_{\{\langle\phi_i,\mathrm{f}\rangle=0\}} \tag{4.16}$$

where $\mathbf{1}_{\{\cdot\}}$ equals zero if the inside-bracket condition is satisfied and equals $+\infty$ otherwise. The third term in (4.16) is crucial in that otherwise the projection of f onto ϕ_i for any $k < i \leq m$ will be left unregularized and the resulting model can easily over-fit.

The solution f^* for minimizing (4.16) must lie in the span of $\{\phi_i\}_{i=1}^k$ as otherwise the indicator function will go to infinity. Let $\mathrm{f} := \sum_{1\leq j\leq k} \alpha_j\phi_j$. (4.16) can be reduced to consist of only k ($k \ll m$) parameters

$$\frac{1}{l}\sum_{i\in\mathcal{T}} \ell\Bigg(e_i^\top \sum_{1\leq j\leq k} \alpha_j\phi_j, y_i\Bigg) + \gamma \sum_{1\leq i\leq k} \theta_i^{-1}\alpha_i^2 \tag{4.17}$$

where e_i stands for the i-th unit vector in $\mathbb{R}^m$.

Applying similar analysis[3] in the previous subsections to the modified $\underset{\sim}{C}(\mathrm{f}; \theta)$ in (4.16), for singular cases the gradient of $g(\theta)$ during bundle method is given by

$$\nabla_\theta g(\theta) = -\gamma\left(\frac{\langle\phi_1, \hat{\mathrm{f}}\rangle^2}{\theta_1^2}, \dots \frac{\langle\phi_k, \hat{\mathrm{f}}\rangle^2}{\theta_k^2}, 0, \dots 0\right)^\top \tag{4.18}$$

$$\equiv -\gamma\left(\frac{\hat{\alpha_1}^2}{\theta_1^2}, \dots \frac{\hat{\alpha_k}^2}{\theta_k^2}, 0, \dots 0\right)^\top \tag{4.19}$$

where $\hat{\mathrm{f}}$ and $\hat{\alpha}$ are solutions for minimizing (4.16) and minimizing (4.17), respectively. Eq. (4.19) holds because $\langle\phi_i, \hat{\mathrm{f}}\rangle = \sum_{1\leq j\leq k} \hat{\alpha}_j\langle\phi_i, \phi_j\rangle = \hat{\alpha}_i$.

To carry out bundle method for the singular case, we need to compute $\nabla_\theta g(\theta)$ via (4.19), which requires $\hat{\alpha}$ as the solution of minimizing (4.17). Compared to solving optimization (4.5) w.r.t. $\mathrm{f} \in \mathbb{R}^m$ for the non-singular case, minimizing (4.17) w.r.t. $\alpha \in \mathbb{R}^k$ can be performed much more efficiently due to the substantially reduced parameter size (recall $k \ll m$). In fact, once the top-k eigenvalues/eigenvectors $\{\lambda_j, \phi_j\}_{j=1}^k$ of $\mathcal{L}$ is obtained, the time/space complexity for both the LP subroutine and the SSL subroutine (4.17) in AST will become independent from m, which is desirable for large problems.

4.3.5 Inner Optimization

Both the original and the singular AST involve solving a standard SSL problem as their intermediate subroutines, i.e. minimizing (4.5) w.r.t. f or minimizing (4.17) w.r.t. α. Here we use the later to demonstrate how existing off-the-self machine learning toolkits can be conveniently leveraged for this purpose.

We specify $\ell(\cdot, \cdot)$ as the squared hinge loss. Besides large-margin property, its smoothness often leads to efficient optimization (Chang et al., 2008). In this case, minimizing (4.17) can be

[3]The analysis follows Sections 4.3.1, 4.3.3 and 4.3.2. We omit the details due to the space limit.

formulated as

$$\min_{\alpha\in\mathbb{R}^k} \frac{1}{l}\sum_{i\in\mathcal{T}} \max\left(1 - y_i e_i^\top \Phi\alpha, 0\right)^2 + \gamma\alpha^\top \mathrm{diag}(\theta_1^{-1}, \theta_2^{-1}, \dots \theta_k^{-1})\alpha \tag{4.20}$$

where $\Phi = [\phi_1, \phi_2, \dots \phi_k] \in \mathbb{R}^{m\times k}$. By defining $C := (\gamma l)^{-1}$ and

$$w_j := \alpha_j\sqrt{\frac{2}{\theta_j}} \quad 1 \le j \le k \tag{4.21}$$

$$x_i := \mathrm{diag}\left(\sqrt{\frac{\theta_1}{2}}, \sqrt{\frac{\theta_2}{2}} \dots \sqrt{\frac{\theta_k}{2}}\right)\Phi^\top e_i \quad \forall i \in \mathcal{T} \tag{4.22}$$

optimization (4.20) can be recast as

$$\min_{w\in\mathbb{R}^k} C\sum_{i\in\mathcal{T}} \max\left(1 - y_i\langle x_i, w\rangle, 0\right)^2 + \frac{1}{2}\|w\|_2^2 \tag{4.23}$$

Note that (4.23) is the standard formulation of L2-SVM and can be efficiently solved via existing solvers such as LIBLINEAR (Fan et al., 2008). After obtaining the solution $\hat{w}$ for (4.23), the solution $\hat{\alpha}$ for (4.20) can be easily recovered by rescaling $\hat{w}$, and then be plugged-into (4.18) to compute $\nabla_\theta g(\theta)$ required by the bundle method.

4.4 Generalization Bound

In this section we provide theoretical intuitions to justify the proposed method. We are going to show that AST can be interpreted as an automatic procedure to asymptotically minimize the SSL generalization error bound w.r.t. different STs.

Our analysis is based an existing theorem on the relationship between the generalization performance of SSL and any given (fixed) graph-Laplacian spectrum (Johnson and Zhang, 2008). While proving the theorem is not the contribution of this section, our method provides a new angle to utilize the theorem. To the best of our knowledge, none of the previous work, including (Johnson and Zhang, 2008), have formulated or provided any algorithmic solution to *automatically* determine the optimal spectrum among all candidate spectrums in this manner (i.e. formulating and solving optimization (4.6)).

Theorem 4.4.1 (Adapted from (Johnson and Zhang, 2008))**.** *Suppose indices of the labeled vertices in $\mathcal{T}$ are sampled from $\{1, 2, \dots, m\}$ uniformly at random. Let $\hat{\mathrm{f}}(\mathcal{T})$ be the system-predicted scores in $\mathbb{R}^m$ obtained via solving optimization (4.4) for any given $\mathcal{T}$, and let ℓ be a convex loss function such that $|\nabla\ell| \le b$. We have*

$$\frac{1}{m-l}\mathbb{E}_{\mathcal{T}}\sum_{i\notin\mathcal{T}} \ell\left(\hat{\mathrm{f}}_i(\mathcal{T}), y_i\right) \le \left(\min_{\mathrm{f}\in\mathbb{R}^m} \frac{1}{m}\sum_{i=1}^{m} \ell\left(\mathrm{f}_i, y_i\right) + \gamma\mathrm{f}^\top\sigma(\mathcal{L})\mathrm{f}\right) + \frac{b^2\mathrm{tr}\left(\sigma(\mathcal{L})^{-1}\right)}{2\gamma l m} \tag{4.24}$$

The L.H.S. of (4.24) stands for the empirical risk of SSL for any given ST σ.

To see the connections between AST and Theorem 4.4.1, let $\tau = \frac{b^2}{2\gamma l m}$ and recall that $\sigma(\mathcal{L}) = \sum_{i=1}^{m} \sigma(\lambda_i)\phi_i\phi_i^\top = \sum_{i=1}^{m} \theta_i^{-1}\phi_i\phi_i^\top$, we rewrite the R.H.S. of (4.24) as

$$\left(\min_{\mathrm{f}\in\mathbb{R}^m} \frac{1}{m}\sum_{i=1}^{m} \ell\left(\mathrm{f}_i, y_i\right) + \gamma \sum_{i=1}^{m} \theta_i^{-1} \left\langle \phi_i, \mathrm{f}\right\rangle^2\right) + \tau\|\theta\|_1 \tag{4.25}$$

By comparing the AST objective function in (4.6) with (4.25), we see that AST is essentially trying to minimize a surrogate of (4.25) where the true loss $\frac{1}{m}\sum_{i=1}^{m} \ell\left(\mathrm{f}_i, y_i\right)$ based on all the m vertex labels is substituted by the empirical loss $\frac{1}{l}\sum_{i\in\mathcal{T}} \ell\left(\mathrm{f}_i, y_i\right)$ based on l partially observed vertex labels. The two loss functions are asymptotically equivalent as $l \to m$. This substitution is necessary since in practice it is impossible for us to access all of the m vertex labels during the training phase.

Notice there is an additional isotonic constraint $\theta_1 \geq \theta_2 \dots \theta_m \geq 0$ for AST when minimizing the generalization error bound (4.25) w.r.t. θ, indicating AST always favours the smooth components over the non-smooth ones in the final prediction $\hat{\mathrm{f}}$.

4.5 Experiments and Results

4.5.1 Methods for Comparison

We compare the performance of the following methods in our experiments:

(a) **SSL** is the standard SSL in (4.1) with squared hinge loss. This amounts to taking the ST in (4.4) to be the identity function $\sigma(x) = x$.

(b) **Diffusion** is the ST-enhanced SSL described in (4.4), where σ is parametrized as $\sigma(x) = e^{\beta x}$ a.k.a. the heat diffusion kernel. Prior to SSL, β is empirically estimated by maximizing the kernel alignment score (Shawe-Taylor and Kandola, 2002) via grid search over $[10^{-4}, 10^4]$.

(c) **GRF** is another ST-enhanced SSL algorithm with $\sigma(x) = x+\beta$, a.k.a. the kernel of Gaussian random field. As in Diffusion, β is empirically estimated before SSL via kernel alignment over $[10^{-5}, 10^3]$.

(d) **NKTA** is nonparametric kernel-target alignment (Zhu et al., 2004), a two-step procedure for ST-enhanced SSL. Prior to SSL, we find σ that maximizes the kernel alignment score without assuming its parametric form. Then, we solve (4.4) with the empirically estimated ST. We follow the formulation of (Zhu et al., 2004) and solve the QCQP subroutine using SeDuMi[4].

(e) **AST** is our proposed method of Adaptive Spectral Transform. Different from the aforementioned two-step kernel alignment approaches, the optimal ST is obtained *along with* SSL by solving the convex optimization problem (4.6) via bundle method.

[4]`http://sedumi.ie.lehigh.edu/downloads`

4.5.2 Experimental Setup

We compare AST against the baselines on benchmark datasets in three different domains:

1. **20NewsGroup** for document classification. We use the PC-vs-Mac subset consisting of 1,993 documents with binary labels. Following (Zhu et al., 2004), a symmetrized unweighted 10-nearest neighbor (10NN) graph is constructed based on the cosine similarity between documents.
2. **Isolet** for spoken letter recognition consisting of 7,797 instances from 26 classes [5]. We construct a 10NN graph using the Euclidean distance between the audio features.
3. **MNIST** for pattern recognition of the handwritten digits. We use the full training set consisting of 60,000 images from 10 classes (digits 0-9). A 10NN graph is constructed based on the Euclidean distance among the images.

For all datasets, parameter γ for manifold regularization is fixed to be 10^{-3} for all methods as we find the results are not sensitive to the choice of γ. Instead of tuning the hyperparameter τ for our method AST, we simply fix it to be 10^{-2} across all experiments. For all datasets, only the top-50 Laplacian eigenvectors are used for SSL. For AST we use the singular version as described in Section 4.3.4 with $k = 50$.

Given a dataset of m data points, we randomly sample l labeled vertices and predict the remaining unlabeled $m - l$ vertices with methods for comparison. The training size l gradually increases from 2^4 to 2^7, and the experiment is repeated for 30 times for each given training size. The mean and standard variance of the prediction accuracy are reported.

4.5.3 Results

Results are presented in Figure 4.3. For all aforementioned baselines, the prediction accuracy improves and the variance tends to decrease as we gradually enlarge the training size.

First, it is evident that all ST-enhanced methods outperform the traditional SSL on average, which justifies the effectiveness of allowing richer graph transduction patterns over the data manifold.

Secondly, among two-step methods based on empirical kernel-target alignment, it is evident that the nonparametric method NKTA outperforms the two parametric methods Diffusion and GRF, which justifies our previous argument that pre-specifying ST to be within some common function family is too restrictive to accurately capture the "true" graph transduction pattern.

Finally, between nonparametric methods, we observe that the performance of AST dominates NKTA over all datasets. This confirms our intuition that ST-finding and SSL are able to mutually reinforce each other during the joint optimization. The advantageous empirical performance of AST also justifies our previous theoretical analysis in Section 4.4.

We also notice AST yields much more stable performance than NKTA. We conjecture that NKTA might be subject to noise as it is trying to fit the target kernel matrix—a quantity induced from only a very limited amount of labels. On the other hand, AST is designed to be adaptive to the problem structure—an arguably more robust reference.

[5]All the algorithms for comparison can be trivially extended to the multi-class case by decomposing the original problem into multiple binary SSL tasks.

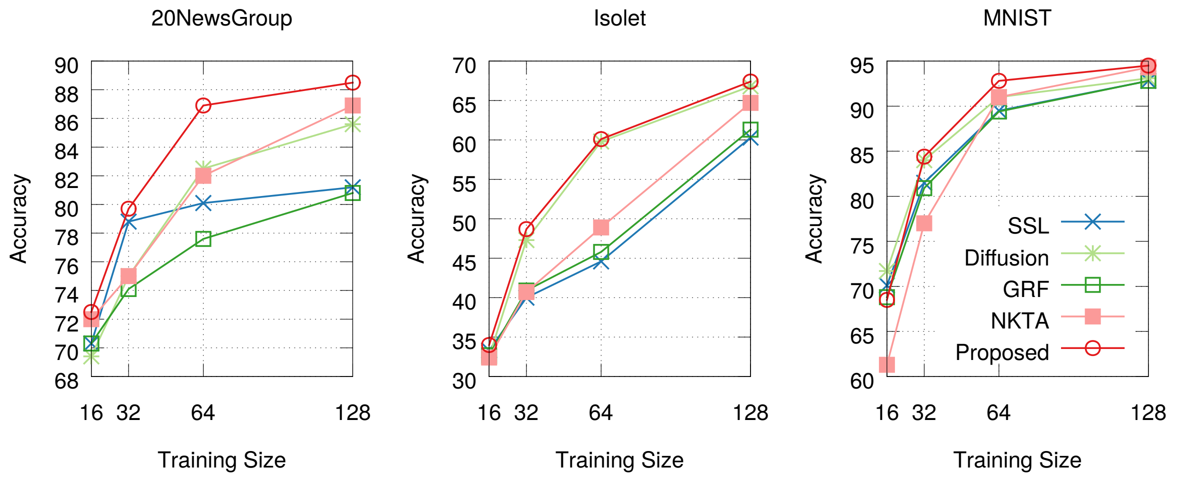

Figure 4.3: Classification accuracy on 20NewsGroup, Isolet and MNIST

We plotted out the STs produced by different baseline methods over MNIST when $l = 128$ in Figure 4.4. Each sub-figure contains 30 curves in total corresponding to the 30 different runs. From the figure we see that while the STs produced by Diffusion and GRF are restricted to specific parametric forms, STs produced by NKTA and AST are more flexible. Figure 4.4 also shows that STs produced by AST tend to be have lower variance than those produced by NKTA, which justifies our previous stability claim about AST.

An empirical comparison of the speed of all the baseline methods is presented in Table 4.1.

Table 4.1: Speed comparison of different methods on MNIST when $l = 128$ given the top-50 eigenvalues/eigenvectors. We use convergence tolerance $\epsilon = 10^{-3}$ for AST.

Method	SSL	Diffusion	GRF	NKTA	AST
Time (secs)	0.148	0.564	0.738	24.152	2.556

4.6 Summary

In this chapter, we proposed a new nonparametric framework for carrying out SSL and finding the Laplacian spectrum of the data manifold simultaneously. Different from existing two-step approaches based on manual specification or kernel-target alignment, our approach unifies both tasks into a joint optimization problem and is naturally adaptive to the problem structure. Our formulation enjoys convexity and can be efficiently solved using the bundle method. Theoretical insights are provided to show that the proposed algorithm attempts to asymptotically minimize the SSL generalization error bound w.r.t. the Laplacian spectrum. The merits of our framework are verified by its advantageous empirical performance over strong baselines.

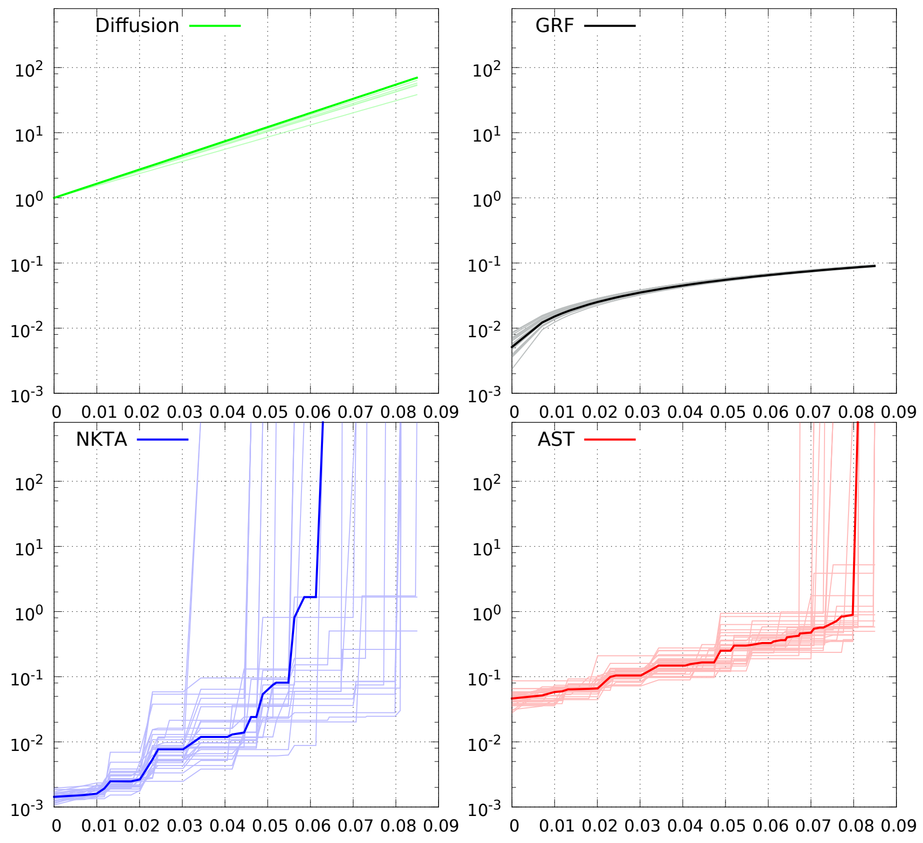

Figure 4.4: STs produced by all methods on the MNIST dataset (each sub-figure contains the results of 30 different runs), where the x-axis and y-axis (log-scale) correspond to the original spectrum λ_i's and the transformed spectrum $\sigma(\lambda_i)$'s, resp.

Part III

Learning Neural Network Architectures

Chapter 5

Architecture Search with Hierarchical Representations

5.1 Background

Discovering high-performance neural network architectures required years of extensive research by human experts through trial and error. As far as the image classification task is concerned, state-of-the-art convolutional neural networks are going beyond deep, chain-structured layout (Simonyan and Zisserman, 2014; He et al., 2016a) towards increasingly more complex, graph-structured topologies (Szegedy et al., 2015, 2016, 2017; Larsson et al., 2016; Xie et al., 2016b; Huang et al., 2016). The combinatorial explosion in the design space makes handcrafted architectures not only expensive to obtain, but also likely to be suboptimal in performance.

Recently, there has been a surge of interest in using algorithms to automate the manual process of architecture design. Their goal can be described as finding the optimal architecture in a given search space such that the validation accuracy is maximized on the given task. Representative architecture search algorithms can be categorized as random with weights prediction (Brock et al., 2017), Monte Carlo Tree Search (Negrinho and Gordon, 2017), evolution (Stanley and Miikkulainen, 2002; Xie and Yuille, 2017; Miikkulainen et al., 2017; Real et al., 2017), and reinforcement learning (Baker et al., 2016; Zoph and Le, 2016; Zoph et al., 2017; Zhong et al., 2017), among which reinforcement learning approaches have demonstrated the strongest empirical performance so far.

Architecture search can be computationally very intensive as each evaluation typically requires training a neural network. Therefore, it is common to restrict the search space to reduce complexity and increase efficiency of architecture search. Various constraints that have been used include: growing a convolutional "backbone" with skip connections (Real et al., 2017), a linear sequence of filter banks (Brock et al., 2017), or a directed graph where every node has exactly two predecessors (Zoph et al., 2017). In this work we constrain the search space by imposing a hierarchical network structure, while allowing flexible network topologies (directed acyclic graphs) at each level of the hierarchy. Starting from a small set of primitives such as convolutional and pooling operations at the bottom level of the hierarchy, higher-level computation graphs, or motifs, are formed by using lower-level motifs as their building blocks. The motifs at the top of the hierarchy are stacked multiple times to form the final neural network. This approach enables search algorithms to implement powerful hierarchical modules where any change in the motifs is propagated across the whole network immediately. This is analogous to the modularized design patterns used in many hand-crafted architectures, e.g. VGGNet (Simonyan and Zisserman, 2014), ResNet (He et al., 2016a), and Inception (Szegedy et al., 2016) are all comprised of building blocks. In our case, a hierarchical architecture is discovered through evolutionary or random search.

The evolution of neural architectures was studied as a sub-task of *neuroevolution* (Holland, 1975; Miller et al., 1989; Yao, 1999; Stanley and Miikkulainen, 2002; Floreano et al., 2008), where the topology of a neural network is simultaneously evolved along with its weights and hyperparameters. The benefits of indirect encoding schemes, such as multi-scale representations, have historically been discussed in Gruau et al. (1994); Kitano (1990); Stanley (2007); Stanley et al. (2009). Despite these pioneer studies, evolutionary or random architecture search has not been investigated at larger scale on image classification benchmarks until recently (Real et al., 2017; Miikkulainen et al., 2017; Xie and Yuille, 2017; Brock et al., 2017; Negrinho and Gordon, 2017). Our work shows that the power of simple search methods can be substantially enhanced using well-designed search spaces.

Our experimental setup resembles Zoph et al. (2017), where an architecture found using reinforcement learning obtained the state-of-the-art performance on ImageNet. Our work reveals that random or evolutionary methods, which so far have been seen as less efficient, can scale and achieve competitive performance on this task if combined with a powerful architecture representation, whilst utilizing significantly less computational resources.

To summarize, our main contributions are:

1. We introduce hierarchical representations for describing neural network architectures.
2. We show that competitive architectures for image classification can be obtained even with simplistic random search, which demonstrates the importance of search space construction.
3. We present a scalable variant of evolutionary search which further improves the results and achieves the best published results[1] among evolutionary architecture search techniques.

5.2 Architecture Representations

We first describe flat representations of neural architectures (Sect. 5.2.1), where each architecture is represented as a single directed acyclic graph of primitive operations. Then we move on to hierarchical representations (Sect. 5.2.2) where smaller graph motifs are used as building blocks to form larger motifs. Primitive operations are discussed in Sect. 5.2.3.

5.2.1 Flat Architecture Representation

We consider a family of neural network architectures represented by a single-source, single-sink computation graph that transforms the input at the source to the output at the sink. Each node of the graph corresponds to a feature map, and each directed edge is associated with some primitive

[1]See Real et al. (2018) for a more recent study of evolutionary methods for architecture search.

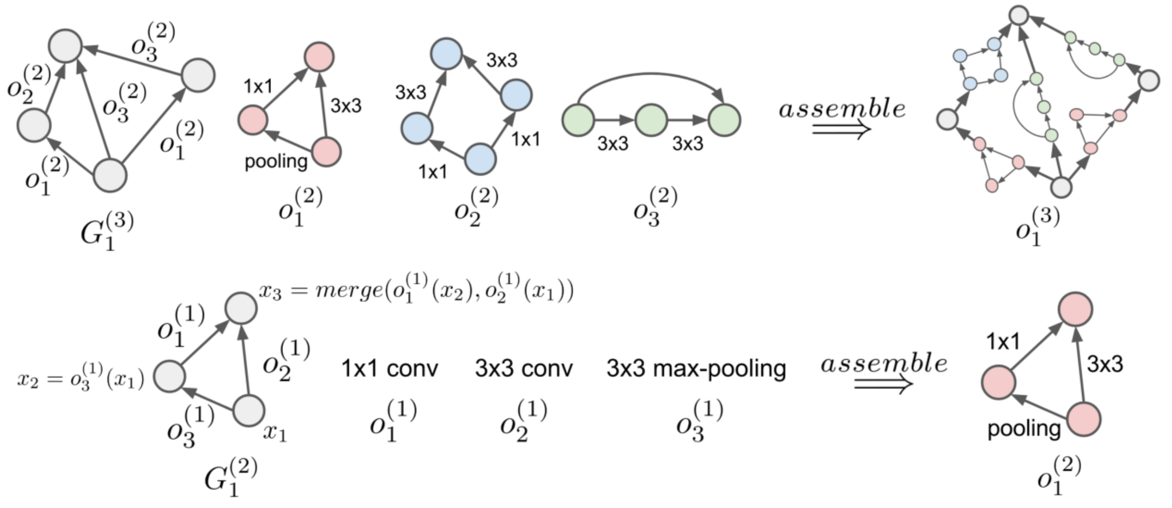

Figure 5.1: An example of a three-level hierarchical architecture representation. The bottom row shows how level-1 primitive operations $o_1^{(1)}, o_2^{(1)}, o_3^{(1)}$ are assembled into a level-2 motif $o_1^{(2)}$. The top row shows how level-2 motifs $o_1^{(2)}, o_2^{(2)}, o_3^{(2)}$ are then assembled into a level-3 motif $o_1^{(3)}$.

operation (e.g. convolution, pooling, etc.) that transforms the feature map in the input node and passes it to the output node.

Formally, an architecture is defined by the representation $(G, \boldsymbol{o})$, consisting of two ingredients:

1. A set of available operations $\boldsymbol{o} = \{o_1, o_2, \dots\}$.
2. An adjacency matrix G specifying the neural network graph of operations, where $G_{ij} = k$ means that the k-th operation o_k is to be placed between nodes i and j.

The architecture is obtained by assembling operations $\boldsymbol{o}$ according to the adjacency matrix G:

$$arch = assemble(G, \boldsymbol{o}) \tag{5.1}$$

in a way that the resulting neural network sequentially computes the feature map x_i of each node i from the feature maps x_j of its predecessor nodes j following the topological ordering:

$$x_i = merge\left[\{o_{G_{ij}}(x_j)\}_{j<i}\right], \quad i = 2, \dots, |G| \tag{5.2}$$

Here, $|G|$ is the number of nodes in a graph, and $merge$ is an operation combining multiple feature maps into one, which in our experiments was implemented as depthwise concatenation. An alternative option of element-wise addition is less flexible as it requires the incoming feature maps to contain the same number of channels, and is strictly subsumed by concatenation if the resulting x_i is immediately followed by a 1×1 convolution.

5.2.2 Hierarchical Architecture Representation

The key idea of the hierarchical architecture representation is to have several motifs at different levels of hierarchy, where lower-level motifs are used as building blocks (operations) during the construction of higher-level motifs.

Consider a hierarchy of L levels where the ℓ-th level contains M_ℓ motifs. The highest-level $\ell = L$ contains only a single motif corresponding to the full architecture, and the lowest level $\ell = 1$ is the set of primitive operations. We recursively define $o_m^{(\ell)}$, the m-th motif in level ℓ, as the composition of lower-level motifs $\boldsymbol{o}^{(\ell-1)} = \left\{o_1^{(\ell-1)}, o_2^{(\ell-1)}, ..., o_{M_{(\ell-1)}}^{(\ell-1)}\right\}$ according to its network structure $G_m^{(\ell)}$:

$$o_m^{(\ell)} = assemble\left(G_m^{(\ell)}, \boldsymbol{o}^{(\ell-1)}\right), \quad \forall \ell = 2, \ldots, L \tag{5.3}$$

A hierarchical architecture representation is therefore defined by $\left(\left\{\{G_m^{(\ell)}\}_{m=1}^{M_\ell}\right\}_{\ell=2}^{L}, \boldsymbol{o}^{(1)}\right)$, as it is determined by network structures of motifs at all levels and the set of bottom-level primitives. The assembly process is illustrated in Fig. 5.1.

5.2.3 Primitive Operations

We consider the following six primitives at the bottom level of the hierarchy ($\ell = 1, M_\ell = 6$):

- 1×1 convolution of C channels
- 3×3 depthwise convolution
- 3×3 separable convolution of C channels
- 3×3 max-pooling
- 3×3 average-pooling
- identity

If applicable, all primitives are of stride one and the convolved feature maps are padded to preserve their spatial resolution. All convolutional operations are followed by batch normalization and ReLU activation (Ioffe and Szegedy, 2015); their number of channels is fixed to a constant C. We note that convolutions with larger receptive fields and more channels can be expressed as motifs of such primitives. Indeed, large receptive fields can be obtained by stacking 3×3 convolutions in a chain structure (Simonyan and Zisserman, 2014), and wider convolutions with more channels can be obtained by merging the outputs of multiple convolutions through depthwise concatenation.

We also introduce a special $none$ op, which indicates that there is no edge between nodes i and j. It is added to the pool of operations at each level.

5.3 Evolutionary Architecture Search

Evolutionary search over neural network architectures can be performed by treating the representations of Sect. 5.2 as genotypes. We first introduce an action space for mutating hierarchical genotypes (Sect. 5.3.1), as well as a diversification-based scheme to obtain the initial population (Sect. 5.3.2). We then describe tournament selection and random search in Sect. 5.3.3, and our distributed implementation in Sect. 5.3.4.

5.3.1 Mutation

A single mutation of a hierarchical genotype consists of the following sequence of actions:

1. Sample a target non-primitive level $\ell \geq 2$.
2. Sample a target motif m in the target level.
3. Sample a random successor node i in the target motif.
4. Sample a random predecessor node j in the target motif.
5. Replace the current operation $o_k^{(\ell-1)}$ between j and i with a randomly sampled operation $o_{k'}^{(\ell-1)}$.

In the case of flat genotypes which consist of two levels (one of which is the fixed level of primitives), the first step is omitted and ℓ is set to 2. The mutation can be summarized as:

$$[G_m^{(\ell)}]_{ij} = k' \tag{5.4}$$

where ℓ, m, i, j, k' are randomly sampled from uniform distributions over their respective domains. Notably, the above mutation process is powerful enough to perform various modifications on the target motif, such as:

1. **Add a new edge**: if $o_k^{(\ell-1)} = none$ and $o_{k'}^{(\ell-1)} \neq none$.
2. **Alter an existing edge**: if $o_k^{(\ell-1)} \neq none$ and $o_{k'}^{(\ell-1)} \neq none$ and $o_{k'}^{(\ell-1)} \neq o_k^{(\ell-1)}$.
3. **Remove an existing edge**: if $o_k^{(\ell-1)} \neq none$ and if $o_{k'}^{(\ell-1)} = none$.

5.3.2 Initialization

To initialize the population of genotypes, we use the following strategy:

1. Create a "trivial" genotype where each motif is set to a chain of identity mappings.
2. Diversify the genotype by applying a large number (e.g. 1000) of random mutations.

In contrast to several previous works where genotypes are initialized by trivial networks (Stanley and Miikkulainen, 2002; Real et al., 2017), the above diversification-based scheme not only offers a good initial coverage of the search space with non-trivial architectures, but also helps to avoid an additional bias introduced by handcrafted initialization routines. In fact, this strategy ensures initial architectures are reasonably well-performing even without any search, as suggested by our random sample results in Table 5.1.

5.3.3 Search Algorithms

Our evolutionary search algorithm is based on tournament selection (Goldberg and Deb, 1991). Starting from an initial population of random genotypes, tournament selection provides a mechanism to pick promising genotypes from the population, and to place its mutated offspring back into the population. By repeating this process, the quality of the population keeps being refined over time. We always train a model from scratch for a fixed number of iterations, and we refer to the training and evaluation of a single model as an evolution step. The genotype with the highest

Algorithm 2: AsyncEvo Asynchronous Evolution (Controller)

```
Input: Data queue $\mathcal{Q}$ containing initial genotypes; Memory table $\mathcal{M}$ recording evaluated
       genotypes and their fitness.
while True do
    if hasIdleWorker() then
        $genotype \leftarrow$ asyncTournamentSelect($\mathcal{M}$)
        $genotype' \leftarrow$ Mutate($genotype$)
        $\mathcal{Q} \leftarrow \mathcal{Q} \cup genotype'$
```

Algorithm 3: AsyncEvo Asynchronous Evolution (Worker)

```
Input: Training set $\mathcal{T}$, validation set $\mathcal{V}$; Shared memory table $\mathcal{M}$ and data queue $\mathcal{Q}$.
while True do
    if $|\mathcal{Q}| > 0$ then
        $genotype \leftarrow \mathcal{Q}$.pop()
        $arch \leftarrow$ Assemble($genotype$)
        $model \leftarrow$ Train($arch, \mathcal{T}$)
        $fitness \leftarrow$ Evaluate($model, \mathcal{V}$)
        $\mathcal{M} \leftarrow \mathcal{M} \cup (genotype, fitness)$
```

fitness (validation accuracy) among the entire population is selected as the final output after a fixed amount of time.

A tournament is formed by a random set of genotypes sampled from the current effective population, among which the individual with the highest fitness value wins the tournament. The selection pressure is controlled by the tournament size, which is set to 5% of the population size in our case. We do not remove any genotypes from the population, allowing it to grow with time, maintaining architecture diversity. Our evolution algorithm is similar to the binary tournament selection used in a recent large-scale evolutionary method (Real et al., 2017).

We also investigated random search, a simpler strategy which has not been sufficiently explored in the literature, as an alternative to evolution. In this case, a population of genotypes is generated randomly, the fitness is computed for each genotype in the same way as done in evolution, and the genotype with the highest fitness is selected as the final output. The main advantage of this method is that it can be run in parallel over the entire population, substantially reducing the search time.

5.3.4 Implementation

Our distributed implementation is asynchronous, consisting of a single controller responsible for performing evolution over the genotypes, and a set of workers responsible for their evaluation. Both parties have access to a shared tabular memory $\mathcal{M}$ recording the population of genotypes and their fitness, as well as a data queue $\mathcal{Q}$ containing the genotypes with unknown fitness which should be evaluated.

Specifically, the controller will perform tournament selection of a genotype from $\mathcal{M}$ whenever a worker becomes available, followed by the mutation of the selected genotype and its insertion into $\mathcal{Q}$ for fitness evaluation (Algorithm 2). A worker will pick up an unevaluated genotype from $\mathcal{Q}$ whenever there is one available, assemble it into an architecture, carry out training and validation, and then record the validation accuracy (fitness) in $\mathcal{M}$ (Algorithm 3). Architectures are trained from scratch for a fixed number of steps with random weight initialization. We do not rely on weight inheritance as in (Real et al., 2017), though incorporating it into our system is possible. Note that during architecture evolution no synchronization is required, and all workers are fully occupied.

5.4 Experiments and Results

5.4.1 Experimental Setup

In our experiments, we use the proposed search framework to learn the architecture of a convolutional cell, rather than the entire model. The reason is that we would like to be able to quickly compute the fitness of the candidate architecture and then transfer it to a larger model, which is achieved by using less cells for fitness computation and more cells for full model evaluation. A similar approach has recently been used in (Zoph et al., 2017; Zhong et al., 2017).

Architecture search is carried out entirely on the CIFAR-10 training set, which we split into two sub-sets of 40K training and 10K validation images. Candidate models are trained on the training subset, and evaluated on the validation subset to obtain the fitness. Once the search process is over, the selected cell is plugged into a large model which is trained on the combination of training and validation sub-sets, and the accuracy is reported on the CIFAR-10 test set. We note that the test set is never used for model selection, and it is only used for final model evaluation. We also evaluate the cells, learned on CIFAR-10, in a large-scale setting on the ImageNet challenge dataset (Sect. 5.4.3).

For CIFAR-10 experiments we use a model which consists of 3×3 convolution with c_0 channels, followed by 3 groups of learned convolutional cells, each group containing N cells. After each cell (with c input channels) we insert 3×3 separable convolution which has stride 2 and $2c$ channels if it is the last cell of the group, and stride 1 and c channels otherwise. The purpose of these convolutions is to control the number of channels as well as reduce the spatial resolution. The last cell is followed by global average pooling and a linear softmax layer.

For fitness computation we use a smaller model with $c_0 = 16$ and $N = 1$, shown in Fig. 5.2 (top-left). It is trained using SGD with 0.9 momentum for 5000 steps, starting with the learning rate 0.1, which is reduced by 10x after 4000 and 4500 steps. The batch size is 256, and the weight decay value is $3 \cdot 10^{-4}$. We employ standard training data augmentation where a 24×24 crop is randomly sampled from a 32×32 image, followed by random horizontal flipping. The evaluation is performed on the full size 32×32 image.

A note on variance. We found that the variance due to optimization was non-negligible, and we believe that reporting it is important for performing a fair comparison and assessing model capabilities. When training CIFAR models, we have observed standard deviation of up to 0.2% using the exact same setup. The solution we adopted was to compute the fitness as the average

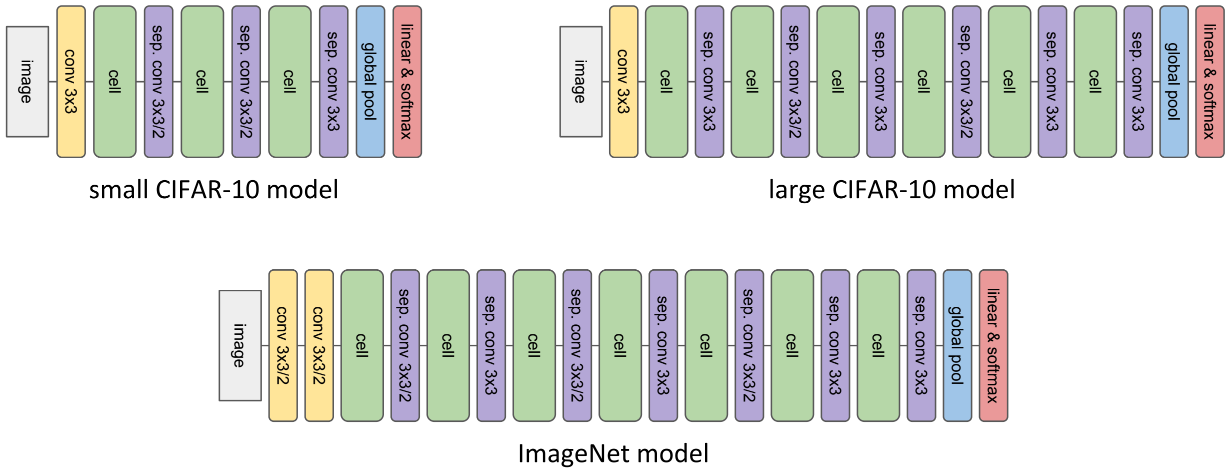

Figure 5.2: Image classification models constructed using the cells optimized with architecture search. *Top-left:* small model used during architecture search on CIFAR-10. *Top-right:* large CIFAR-10 model used for learned cell evaluation. *Bottom:* ImageNet model used for learned cell evaluation.

accuracy over 4 training-evaluation runs.

For the evaluation of the learned cell architecture on CIFAR-10, we use a larger model with $c_0 = 64$ and $N = 2$, shown in Fig. 5.2 (top-right). The larger model is trained for 80K steps, starting with a learning rate 0.1, which is reduced by 10x after 40K, 60K, and 70K steps. The rest of the training settings are the same as used for fitness computation. We report mean and standard deviation computed over 5 training-evaluation runs.

For the evaluation on the ILSVRC ImageNet challenge dataset (Russakovsky et al., 2015), we use an architecture similar to the one used for CIFAR, with the following changes. An input 299×299 image is passed through two convolutional layers with 32 and 64 channels and stride 2 each. It is followed by 4 groups of convolutional cells where the first group contains a single cell (and has $c_0 = 64$ input channels), and the remaining three groups have $N = 2$ cells each (Fig. 5.2, bottom). We use SGD with momentum which is run for 200K steps, starting with a learning rate of 0.1, which is reduced by 10x after 100K, 150K, and 175K steps. The batch size is 1024, and weight decay is 10^{-4}. We did not use auxiliary losses, weight averaging, label smoothing or path dropout empirically found effective in (Zoph et al., 2017). The training augmentation is the same as in (Szegedy et al., 2016), and consists in random crops, horizontal flips and brightness and contrast changes. We report the single-crop top-1 and top-5 error on the ILSVRC validation set.

5.4.2 Architecture Search on CIFAR-10

We run the evolution on flat and hierarchical genotypes for 7000 steps using 200 GPU workers. The initial size of the randomly initialized population is 200, which later grows as a result of tournament selection and mutation (Sect. 5.3). For the hierarchical representation, we use three levels ($L = 3$), with $M_1 = 6, M_2 = 6, M_3 = 1$. Each of the level-2 motifs is a graph with $|G^{(2)}| = 4$ nodes, and the level-3 motif is a graph with $|G^{(3)}| = 5$ nodes. Each level-2 motif

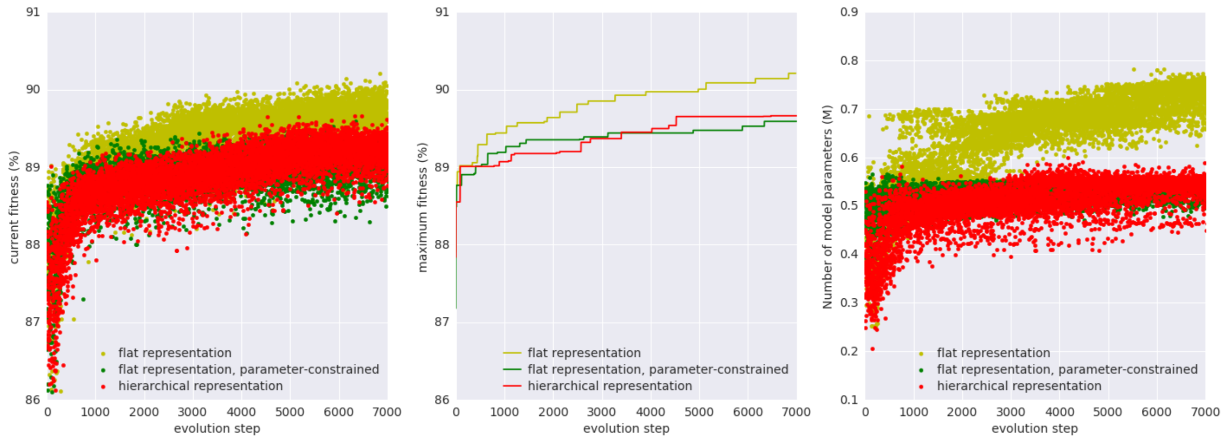

Figure 5.3: Fitness and number of parameters vs evolution step for flat and hierarchical representations. *Left:* fitness of a genotype generated at each evolution step. *Middle:* maximum fitness across all genotypes generated before each evolution step. *Right:* number of parameters in the small CIFAR-10 model constructed using the genotype generated at each evolution step.

is followed by a 1×1 convolution with the same number of channels as on the motif input to reduce the number of parameters. For the flat representation, we used a graph with 11 nodes to achieve a comparable number of edges.

The evolution process is visualized in Fig. 5.3. The left plot shows the fitness of the genotype generated at each step of evolution: the fitness grows fast initially, and plateaus over time. The middle plot shows the best fitness observed by each evolution step. Since the first 200 steps correspond to a random initialization and mutation starts after that, the best architecture found at step 200 corresponds to the output of random search over 200 architectures.

Fig. 5.3 (right) shows the number of parameters in the small network (used for fitness computation), constructed using the genotype produced at each step. Notably, flat genotypes achieve higher fitness, but at the cost of larger parameter count. We thus also consider a parameter-constrained variant of the flat genotype, where only the genotypes with the number of parameters under a fixed threshold are permitted; the threshold is chosen so that the flat genotype has a similar number of parameters to the hierarchical one. In this setting hierarchical and flat genotypes achieve similar fitness.

To demonstrate that improvement in fitness of the hierarchical architecture is correlated with the improvement in the accuracy of the corresponding large model trained till convergence, we plot the relative accuracy improvements in Fig. 5.4.

As far as the search cost is concerned, it takes 1 hour to compute the fitness of one architecture on a single P100 GPU (which involves 4 rounds of training and evaluation). Using 200 GPUs, it thus takes 1 hour to perform random search over 200 architectures and 1.5 days to do the evolutionary search with 7000 steps. This is significantly faster than 11 days using 250 GPUs reported by Real et al. (2017) and 4 days using 450 GPUs reported by Zoph et al. (2017).

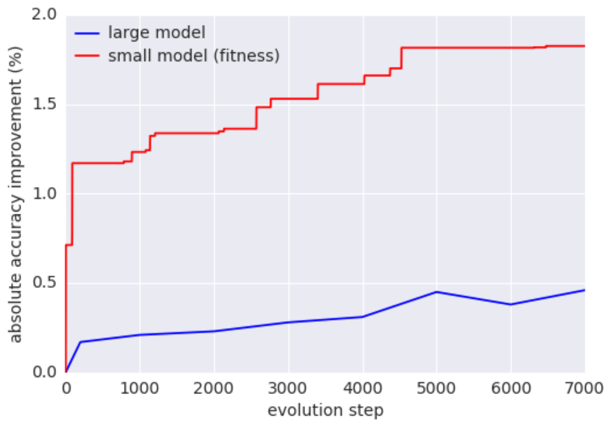

Figure 5.4: Accuracy improvement over the course of evolution, measured with respect to the first random genotype. The small model is the model used for fitness computation during evolution (its absolute fitness value is shown with the red curve in Fig. 5.3 (middle)). The large model is the model where the evolved cell architecture is deployed for training and evaluation.

Search Method	**CIFAR-10 error (%)**	**ImageNet Top-1/Top-5 error (%)**
Flat repr-n, random architecture	4.56 ± 0.11	$21.4/5.8$
Flat repr-n, random search (200 samples)	4.02 ± 0.11	$20.8/5.7$
Flat repr-n, evolution (7000 samples)	3.92 ± 0.06	$20.6/5.6$
Flat repr-n, parameter-constrained, evolution (7000 samples)	4.17 ± 0.08	$21.2/5.8$
Hier. repr-n, random architecture	4.21 ± 0.11	$21.5/5.8$
Hier. repr-n, random search (200 samples)	4.04 ± 0.2	$20.4/5.3$
Hier. repr-n, random search (7000 samples)	3.91 ± 0.15	$21.0/5.5$
Hier. repr-n, evolution (7000 samples)	$\mathbf{3.75 \pm 0.12}$	$\mathbf{20.3/5.2}$

Table 5.1: Classification results on the CIFAR-10 test set and ILSVRC validation set obtained using the architectures found using various representations and search methods.

5.4.3 Architecture Evaluation on CIFAR-10 and ImageNet

We now turn to the evaluation of architectures found using random and evolutionary search on CIFAR-10 and ImageNet. The results are presented in Table 5.1.

First, we note that randomly sampled architectures already perform surprisingly well, which we attribute to the representation power of our architecture spaces. Second, random search over 200 architectures achieves very competitive results on both CIFAR-10 and ImageNet, which is remarkable considering it took 1 hour to carry out. This demonstrates that well-constructed architecture representations, coupled with diversified sampling and simple search form a simple but strong baseline for architecture search. Our best results are achieved using evolution over hierarchical representations: $3.75\% \pm 0.12\%$ classification error on the CIFAR-10 test set (using $c_0 = 64$ channels), which is further improved to $3.63\% \pm 0.10\%$ with more channels ($c_0 = 128$). On the ImageNet validation set, we achieve 20.3% top-1 classification error and 5.2% top-5 error. We put these results in the context of the state of the art in Tables 5.2 and 5.3. We achieve the best published results on CIFAR-10 using evolutionary architecture search, and also demonstrate competitive performance compared to the best published methods on both CIFAR-10 and ImageNet. Our ImageNet model has 64M parameters, which is comparable to Inception-ResNet-v2 (55.8M) but larger than NASNet-A (22.6M).

Model	**Error (%)**
ResNet-1001 + pre-activation (He et al., 2016b)	4.62
Wide ResNet-40-10 + dropout (Zagoruyko and Komodakis, 2016)	3.8
DenseNet (k=24) (Huang et al., 2016)	3.74
DenseNet-BC (k=40) (Huang et al., 2016)	3.46
MetaQNN (Baker et al., 2016)	6.92
NAS v3 (Zoph and Le, 2016)	3.65
Block-QNN-A (Zhong et al., 2017)	3.60
NASNet-A (Zoph et al., 2017)	3.41
Evolving DNN (Miikkulainen et al., 2017)	7.3
Genetic CNN (Xie and Yuille, 2017)	7.10
Large-scale Evolution (Real et al., 2017)	5.4
SMASH (Brock et al., 2017)	4.03
Evolutionary search, hier. repr., $c_0 = 64$	3.75 ± 0.12
Evolutionary search, hier. repr., $c_0 = 128$	3.63 ± 0.10

Table 5.2: Classification error on the CIFAR-10 test set obtained using state-of-the-art models as well as the best-performing architecture found using the proposed architecture search framework. Existing models are grouped as (from top to bottom): handcrafted architectures, architectures found using reinforcement learning, and architectures found using random or evolutionary search.

Model	**Top-1 error (%)**	**Top-5 error (%)**
Inception-v3 (Szegedy et al., 2016)	21.2	5.6
Xception (Chollet, 2016)	21.0	5.5
Inception-ResNet-v2 (Szegedy et al., 2017)	19.9	4.9
NASNet-A (Zoph et al., 2017)	19.2	4.7
Evolutionary search, hier. repr., $c_0 = 64$	20.3	5.2

Table 5.3: Classification error on the ImageNet validation set obtained using state-of-the-art models as well as the best-performing architecture found using our framework.

5.5 Summary

In this chapter, we presented an efficient evolutionary method that identifies high-performing neural architectures based on a novel hierarchical representation scheme, where smaller operations are used as the building blocks to form the larger ones. Notably, we show that strong results can be obtained even using simplistic search algorithms, such as evolution or random search, when coupled with a well-designed architecture representation. Our best architecture yields the state-of-the-art result on CIFAR-10 among evolutionary methods and successfully scales to ImageNet with highly competitive performance. In the future, we are interested in combining hierarchical representations with search algorithms beyond evolution and random search, such as simulated annealing and gradient-based optimization (Chapter 6).

Chapter 6

Differentiable Architecture Search

6.1 Background

Discovering state-of-the-art neural network architectures requires substantial effort of human experts. Recently, there has been a growing interest in developing algorithmic solutions to automate the manual process of architecture design. The automatically searched architectures have achieved highly competitive performance in tasks such as image classification (Zoph and Le, 2016; Zoph et al., 2017; Liu et al., 2017b,a; Real et al., 2018) and object detection (Zoph et al., 2017).

The best existing architecture search algorithms are computationally demanding despite their remarkable performance. For example, obtaining a state-of-the-art architecture for CIFAR-10 and ImageNet required 1800 GPU days of reinforcement learning (RL) (Zoph et al., 2017) or 3150 GPU days of evolution (Real et al., 2018). Several approaches for speeding up have been proposed, such as imposing a particular structure of the search space (Liu et al., 2017b,a), weights or performance prediction for each individual architecture (Brock et al., 2017; Baker et al., 2018) and weight sharing across multiple architectures (Pham et al., 2018b; Cai et al., 2018), but the fundamental challenge of scalability remains. An inherent cause of inefficiency for the dominant approaches, e.g. based on RL, evolution, MCTS (Negrinho and Gordon, 2017), SMBO (Liu et al., 2017a) or Bayesian optimization (Kandasamy et al., 2018), is the fact that architecture search is treated as a black-box optimization problem over a discrete domain, which leads to a large number of architecture evaluations required.

In this work, we approach the problem from a different angle, and propose a method for efficient architecture search called DARTS (Differentiable ARchiTecture Search). Instead of searching over a discrete set of candidate architectures, we relax the search space to be continuous, so that the architecture can be optimized with respect to its validation set performance by gradient descent. The data efficiency of gradient-based optimization, as opposed to inefficient black-box search, allows DARTS to achieve competitive performance with the state of the art using orders of magnitude less computation resources. It also outperforms another recent efficient architecture search method, ENAS (Pham et al., 2018b). Notably, DARTS is simpler than many existing approaches as it does not involve any controllers (Zoph and Le, 2016; Baker et al., 2016; Zoph et al., 2017; Pham et al., 2018b), hypernetworks (Brock et al., 2017) or performance predictors (Liu et al., 2017a), yet it is generic enough to search for both convolutional and recurrent architectures.

The idea of searching architectures within a continuous domain is not new (Saxena and Verbeek, 2016; Ahmed and Torresani, 2017; Shin et al., 2018), but there are several major distinctions. While prior works seek to fine-tune a specific aspect of an architecture, such as filter shapes or branching patterns in a convolutional network, DARTS is able to discover high-performance architectures with complex graph topologies within a rich search space. Moreover, DARTS is not restricted to any specific architecture family, and is able to discover both convolutional and recurrent networks.

In our experiments (Sect. 6.3) we show that DARTS is able to design a convolutional cell that achieves $2.83 \pm 0.06\%$ test error on CIFAR-10 for image classification, which is competitive with the state-of-the-art result by regularized evolution (Real et al., 2018) obtained using three orders of magnitude more computation resources. The same convolutional cell also achieves 26.9% top-1 error when transferred to ImageNet (mobile setting), which is comparable to the best RL method (Zoph et al., 2017). On the language modeling task, DARTS discovers a recurrent cell that achieves 56.1 perplexity on Penn Treebank (PTB) in a single GPU day, outperforming both extensively tuned LSTM (Melis et al., 2017) and all the existing automatically searched cells based on NAS (Zoph and Le, 2016) and ENAS (Pham et al., 2018b).

Our contributions can be summarized as follows:

- We introduce a novel algorithm for differentiable network architecture search that is applicable to both convolutional and recurrent architectures.
- Through extensive experiments on image classification and language modeling tasks we show that gradient-based architecture search achieves highly competitive results on CIFAR-10 and outperforms the state of the art on PTB. This is a very interesting result, considering that so far the best architecture search methods used non-differentiable search techniques, e.g. based on RL (Zoph et al., 2017) or evolution (Real et al., 2018; Liu et al., 2017b).
- We achieve remarkable architecture search efficiency (with 4 GPUs: 2.83% error on CIFAR-10 in 1 day; 56.1 perplexity on PTB in 6 hours) which we attribute to the use of gradient-based optimization as opposed to non-differentiable search techniques.
- We show that the architectures learned by DARTS on CIFAR-10 and PTB are transferable to ImageNet and WikiText-2, respectively.

The implementation of DARTS is available at https://github.com/quark0/darts

6.2 Differentiable Architecture Search

We describe our search space in general form in Sect. 6.2.1, where the computation procedure for an architecture (or a cell in it) is represented as a directed acyclic graph. We then introduce a simple continuous relaxation scheme for our search space which leads to a differentiable learning objective for the joint optimization of the architecture and its weights (Sect. 6.2.2). Finally, we propose an approximation technique to make the algorithm computationally feasible and efficient (Sect. 6.2.3).

6.2.1 Search Space

Following Zoph et al. (2017); Real et al. (2018); Liu et al. (2017a,b), we search for a computation cell as the building block of the final architecture. The learned cell could either be stacked to form a convolutional network or recursively connected to form a recurrent network.

A cell is a directed acyclic graph consisting of an ordered sequence of N nodes. Each node $x^{(i)}$ is a latent representation (e.g. a feature map in convolutional networks) and each directed edge (i, j) is associated with some operation $o^{(i,j)}$ that transforms $x^{(i)}$. We assume the cell to have two input nodes and a single output node. For convolutional cells, the input nodes are defined as the cell outputs in the previous two layers (Zoph et al., 2017). For recurrent cells, these are defined as the input at the current step and the state carried from the previous step. The output of the cell is obtained by applying a reduction operation (e.g. concatenation) to all the intermediate nodes.

Each intermediate node is computed based on all of its predecessors:

$$x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)}) \tag{6.1}$$

A special *zero* operation is also included to indicate a lack of connection between two nodes. The task of learning the cell therefore reduces to learning the operations on its edges.

6.2.2 Continuous Relaxation and Optimization

Let $\mathcal{O}$ be a set of candidate operations (e.g., convolution, max pooling, *zero*) where each operation represents some function $o(\cdot)$ to be applied to $x^{(i)}$. To make the search space continuous, we relax the categorical choice of a particular operation as a softmax over all possible operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{6.2}$$

where the operation mixing weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. The task of architecture search then reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$, as illustrated in Fig. 6.1. At the end of search, a discrete architecture is obtained by replacing each mixed operation $\bar{o}^{(i,j)}$ with the most likely operation, i.e., $o^{(i,j)} = \text{argmax}_{o\in\mathcal{O}}\ \alpha_o^{(i,j)}$. In the following, we refer to α as the (encoding of the) architecture.

After relaxation, our goal is to jointly learn the architecture α and the weights w within all the mixed operations (e.g. weights of the convolution filters). Analogous to architecture search using RL (Zoph and Le, 2016; Zoph et al., 2017; Pham et al., 2018b) or evolution (Liu et al., 2017b; Real et al., 2018) where the validation set performance is treated as the reward or fitness, DARTS aims to optimize the validation loss, but using gradient descent.

Denote by $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ the training and the validation loss, respectively. Both losses are determined not only by the architecture α, but also the weights w in the network. The goal for architecture search is to find α^* that minimizes the validation loss $\mathcal{L}_{val}(w^*, \alpha^*)$, where the weights w^* associated with the architecture are obtained by minimizing the training loss $w^* = \text{argmin}_w\ \mathcal{L}_{train}(w, \alpha^*)$.

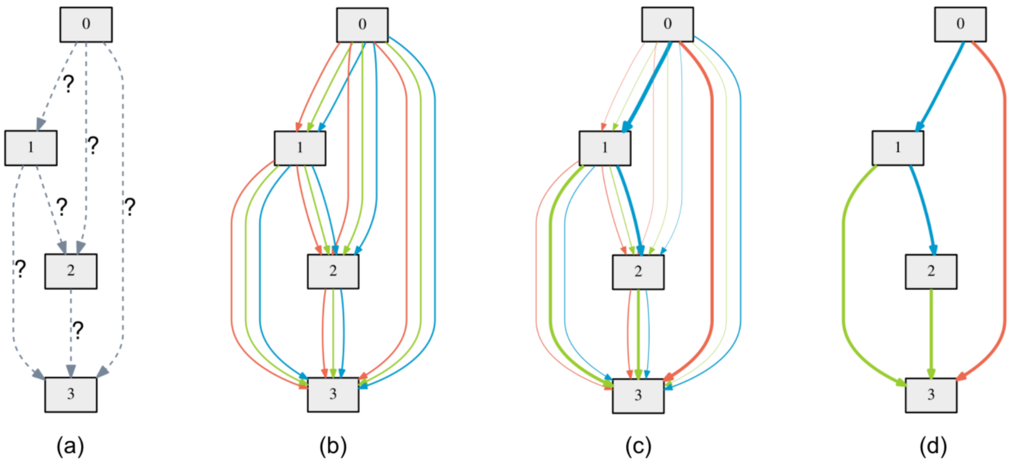

Figure 6.1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Algorithm 4: DARTS – Differentiable Architecture Search

```
Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i,j)$
while not converged do
    1. Update weights $w$ by descending $\nabla_w \mathcal{L}_{train}(w,\alpha)$
    2. Update architecture $\alpha$ by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w,\alpha),\alpha)$
Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o\in\mathcal{O}}\ \alpha_o^{(i,j)}$ for each edge $(i,j)$
```

This implies a bilevel optimization problem (Anandalingam and Friesz, 1992; Colson et al., 2007) with α as the upper-level variable and w as the lower-level variable:

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha),\alpha) \tag{6.3}$$

$$\text{s.t.} \quad w^*(\alpha) = \mathrm{argmin}_w \ \mathcal{L}_{train}(w,\alpha) \tag{6.4}$$

The nested formulation also arises in gradient-based hyperparameter optimization (Maclaurin et al., 2015; Pedregosa, 2016; Franceschi et al., 2018), which is related in a sense that the architecture α could be viewed as a special type of hyperparameter, although its dimension is substantially higher than scalar-valued hyperparameters such as the learning rate, and it is harder to optimize.

6.2.3 Approximation

Solving the bilevel optimization exactly is prohibitive, as it would require recomputing $w^*(\alpha)$ by solving the inner problem (6.4) whenever there is any change in α. We thus propose an approximate iterative optimization procedure where w and α are optimized by alternating between gradient descent steps in the weight and architecture spaces respectively (Alg. 4). At step k, given the current architecture α_{k-1}, we obtain w_k by moving w_{k-1} in the direction of minimising the training loss $\mathcal{L}_{train}(w_{k-1}, \alpha_{k-1})$. Then, keeping the weights w_k fixed, we update the architecture so as to minimize the the validation loss *after a single step of gradient descent w.r.t. the weights*:

$$\mathcal{L}_{val}(w_k - \xi \nabla_w \mathcal{L}_{train}(w_k, \alpha_{k-1}), \alpha_{k-1}) \tag{6.5}$$

where ξ is the learning rate for this virtual gradient step. The motivation behind (6.5) is that we would like to find an architecture which has a low validation loss when its weights are optimized by (a single step of) gradient descent, where the one-step unrolled weights serve as the surrogate for $w^*(\alpha)$. Related approaches have been used in meta-learning for model transfer (Finn et al., 2017) and gradient-based hyperparameter tuning (Luketina et al., 2016). Notably, the dynamics of our iterative algorithm define a Stackelberg game (Von Stackelberg, 1934) between α's optimizer (leader) and w's optimizer (follower), which typically requires the leader to anticipate the follower's next-step move in order to achieve an equilibrium. While we are not currently aware of the convergence guarantees for our optimization algorithm, in practice it is able to converge with a suitable choice of ξ[1]. We also note that when momentum is enabled for weight optimisation, the one-step forward learning objective (6.5) is modified accordingly and all of our analysis still applies.

The architecture gradient is given by differentiating (6.5) w.r.t. α (we omit the step index k for brevity):

$$\nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \tag{6.6}$$

where $w' = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$ denotes the weights for a one-step forward model. The gradient (6.6) contains a matrix-vector product in its second term, which is expensive to compute. Fortunately, the complexity can be substantially reduced using the finite difference approximation. Let ϵ be a small scalar [2] and $w^{\pm} = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. Then:

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon} \tag{6.7}$$

Evaluating the finite difference requires only two forward passes for the weights and two backward passes for α, and the complexity is reduced from $O(|\alpha||w|)$ to $O(|\alpha| + |w|)$.

First-order Approximation: When $\xi = 0$, the second-order derivative in (6.6) will then disappear. In this case, the architecture gradient is given by $\nabla_\alpha \mathcal{L}_{val}(w, \alpha)$, corresponding to the simple heuristic of optimizing the validation loss by assuming α and w are independent of each other. This leads to some speed-up but empirically worse performance, according to our experimental results in Table 6.1 and Table 6.2. In the following, we refer to the case of $\xi = 0$ as the first-order approximation, and refer to the gradient formulation with $\xi > 0$ as the second-order approximation.

6.2.4 Deriving Discrete Architectures

After obtaining the continuous architecture encoding α, the discrete architecture is derived by

[1]A simple working strategy is to set ξ equal to the learning rate for w's optimizer.

[2]We found $\epsilon = 0.01/\|\nabla_{w'} \mathcal{L}_{val}(w', \alpha)\|_2$ to be sufficiently accurate in all of our experiments.

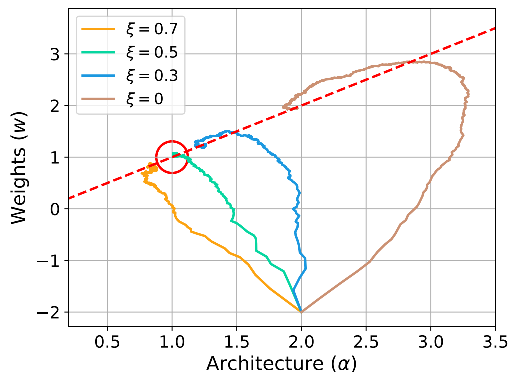

Figure 6.2: Learning dynamics of our iterative algorithm when $\mathcal{L}_{val}(w, \alpha) = \alpha w - 2\alpha + 1$ and $\mathcal{L}_{train}(w, \alpha) = w^2 - 2\alpha w + \alpha^2$, starting from $(\alpha^{(0)}, w^{(0)}) = (2, -2)$. The analytical solution for the corresponding bilevel optimization problem is $(\alpha^*, w^*) = (1, 1)$, which is highlighted in the red circle. The dashed red line indicates the feasible set where constraint (6.4) is satisfied exactly (namely, weights in w are optimal for the given architecture α). The example shows that a suitable choice of ξ helps to converge to a better local optimum.

1. Retaining k strongest predecessors for each intermediate node, where the strength of an edge is defined as $\max_{o \in \mathcal{O}, o \neq zero} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})}$. To make our derived architecture comparable with those in the existing works, we use $k = 2$ for convolutional cells (Zoph et al., 2017; Real et al., 2018) and $k = 1$ for recurrent cells (Pham et al., 2018b).
2. Replacing every mixed operation as the most likely operation by taking the argmax.

6.3 Experiments and Results

Our experiments on CIFAR-10 and PTB consist of two stages, architecture search (Sect. 6.3.1) and architecture evaluation (Sect. 6.3.2). In the first stage, we search for the cell architectures using DARTS, and determine the best cells based on their validation performance. In the second stage, we use these cells to construct larger architectures, which we train from scratch and report their performance on the test set. Finally, we investigate the transferability of the best cells learned on CIFAR-10 and PTB by evaluating them on ImageNet and WikiText-2 (WT2) respectively (Sect. 6.3.4).

6.3.1 Architecture Search

6.3.1.1 Searching for Convolutional Cells on CIFAR-10

We include the following operations in $\mathcal{O}$: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and $zero$. All operations are of stride one (if applicable) and the convolved feature maps are padded to preserve their spatial resolution. We use the ReLU-Conv-BN order for convolutional operations, and each separable convolution is always applied twice (Zoph et al., 2017; Real et al., 2018; Liu et al., 2017a).

Our convolutional cell consists of $N = 7$ nodes, among which the output node is defined as the depthwise concatenation of all the intermediate nodes (input nodes excluded). The rest of the setup follows Zoph et al. (2017); Liu et al. (2017a); Real et al. (2018), where a network is then formed by stacking multiple cells together. The first and second nodes of cell k are set equal to the outputs of cell $k-2$ and cell $k-1$, respectively, and 1×1 convolutions are inserted as necessary. Cells located at the $1/3$ and $2/3$ of the total depth of the network are reduction cells, in which all the operations adjacent to the input nodes are of stride two. The architecture encoding therefore is $(\alpha_{normal}, \alpha_{reduce})$, where α_{normal} is shared by all the normal cells and α_{reduce} is shared by all the reduction cells.

Since the architecture will be varying throughout the search process, we always use batch-specific statistics for batch normalization rather than the global moving average. Learnable affine parameters in all batch normalizations are disabled during the search process to avoid rescaling the outputs of the candidate operations.

To carry out architecture search, we hold out half of the CIFAR-10 training data as the validation set. A small network consisting of 8 cells is trained using DARTS for 50 epochs, with batch size 64 (for both the training and validation sets) and the initial number of channels 16. The numbers were chosen to ensure the network can fit into a single GPU. We use momentum SGD to optimize the weights w, with initial learning rate $\eta_w = 0.025$ (annealed down to zero following a cosine schedule), momentum 0.9, and weight decay 3×10^{-4}. We use Adam as the optimizer for the architecture variables (the α's in both the normal and reduction cells), with initial learning rate $\eta_\alpha = 3 \times 10^{-4}$, momentum $\beta = (0.5, 0.999)$ and weight decay 10^{-3}. The search takes one day on a single GPU[3].

6.3.1.2 Searching for Recurrent Cells on Penn Treebank

Our set of available operations includes the special *zero* operation, as well as linear transformations followed by $\tanh$, relu, $\mathrm{sigmoid}$, and identity mapping, respectively. The choice of these candidate operations follows Zoph and Le (2016); Pham et al. (2018b).

Our recurrent cell consists of $N = 12$ nodes. The very first intermediate node is obtained by linearly transforming the two input nodes, adding up the results and then passing through a $\tanh$ activation function, as done in the ENAS cell (Pham et al., 2018b). The rest of the cell is learned. Other settings are similar to ENAS, where each operation is enhanced with a highway bypass (Zilly et al., 2016) and the cell output is defined as the average of all the intermediate nodes. As in ENAS, we enable batch normalization in each node to prevent gradient explosion during architecture search, and disable it during architecture evaluation. Learnable affine parameters in batch normalization are disabled, as we did for convolutional cells. Our recurrent network consists of only a single cell. Namely, we do not assume any repetitive patterns within the architecture by vertically stacking the cells.

For architecture search, both the embedding and the hidden sizes are 300. The linear transformation parameters across all candidate operations on the same edge are shared (their shapes are all 300 $\times$ 300). This allows us to fit the continuous architecture within a single GPU. The network is then trained for 50 epochs using SGD without momentum, with learning rate $\eta_w = 20$, batch size 256, BPTT length 35, and weight decay 5×10^{-7}. We apply variational dropout (Gal and Ghahramani, 2016) of 0.2 to word embeddings, 0.75 to the cell input, and 0.25 to all the hidden nodes. A dropout of 0.75 is also applied to the output layer. Other training settings are

[3]All of our experiments were performed using NVIDIA GTX 1080Ti GPUs.

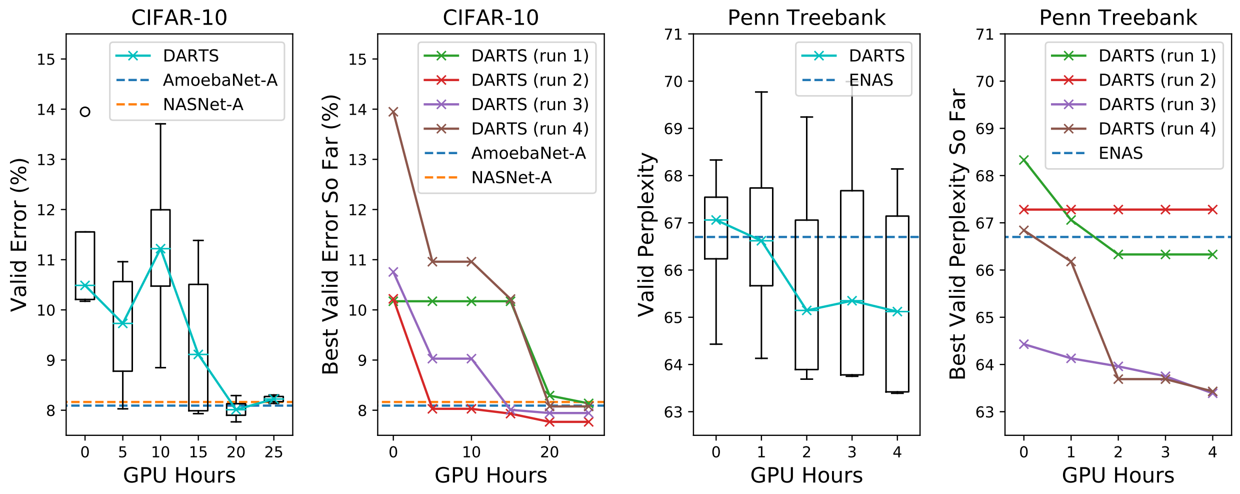

Figure 6.3: Search progress of DARTS for convolutional cells on CIFAR-10 and recurrent cells on PTB. We keep track of the most recent architectures over time. Each architecture snapshot is re-trained from scratch using the training set (for 100 epochs on CIFAR-10 and for 300 epochs on PTB) and then evaluated on the validation set. For each task, we repeat the experiments for 4 times with different random seeds, and report the median and the best (per run) validation performance of the architectures over time. As references, we also report the results (with the same evaluation setup and comparable number of parameters) of the best existing cells discovered using RL or evolution, including NASNet-A (Zoph et al., 2017) (1800 GPU days), AmoebaNet-A (3150 GPU days) (Real et al., 2018) and ENAS (0.5 GPU day) (Pham et al., 2018b).

identical to those in Merity et al. (2017); Yang et al. (2017). Similarly to the convolutional architectures, we use Adam for the optimization of α, with initial learning rate $\eta_\alpha = 3 \times 10^{-3}$, momentum $\beta = (0.9, 0.999)$ and weight decay 10^{-3}. The search takes 6 hours on a single GPU.

6.3.2 Architecture Evaluation

To select the architecture for evaluation, we run DARTS four times with different random seeds and pick the best cell based on the validation performance. This is particularly important for recurrent cells, as the optimization outcomes can be initialization-sensitive (Fig. 6.3).

To evaluate the selected architecture, we randomly initialize its weights (weights learned during the search process are discarded), train it from scratch, and report its performance on the test set. We note the test set is never used for architecture search or architecture selection.

6.3.2.1 CIFAR-10

A large network of 20 cells is trained for 600 epochs with batch size 96. Other hyperparameters remain the same as the ones used for architecture search. Following existing works (Pham et al., 2018b; Zoph et al., 2017; Liu et al., 2017a; Real et al., 2018), additional enhancements include cutout (DeVries and Taylor, 2017), path dropout of probability 0.3 and auxiliary towers with weight 0.4. The training takes 1.5 days on a single GPU with our implementation in PyTorch

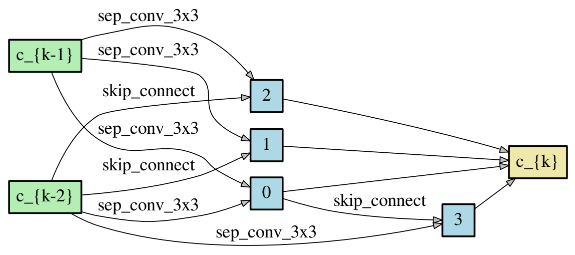

Figure 6.4: Normal cell learned on CIFAR-10.

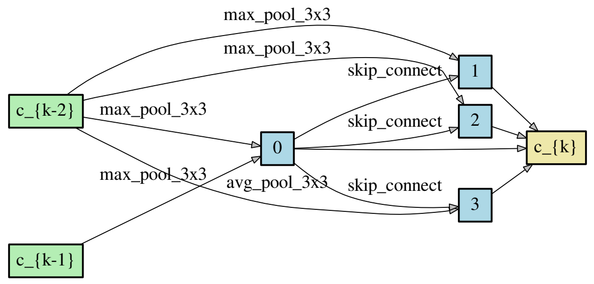

Figure 6.5: Reduction cell learned on CIFAR-10.

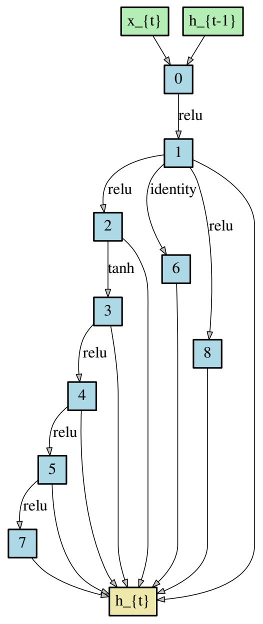

Figure 6.6: Recurrent cell learned on PTB.

(Paszke et al., 2017). Since the CIFAR results are subject to high variance even with exactly the same setup (Liu et al., 2017b), we report the mean and standard deviation of 4 independent runs for our full model.

To avoid any discrepancy between different implementations and/or training settings (e.g. the batch sizes), we incorporated the NASNet-A cell (Zoph et al., 2017) and the AmoebaNet-A cell (Real et al., 2018) into our training framework and reported their results under the same settings as our cells.

6.3.2.2 Penn Treebank

A single-layer recurrent network with the discovered cell is trained for 1600 epochs with batch size 64 using averaged SGD (Polyak and Juditsky, 1992) (ASGD), with learning rate $\eta_w = 20$ and weight decay 8×10^{-7}. To speedup, we start with SGD and trigger ASGD using the same protocol as in Yang et al. (2017); Merity et al. (2017). Both the embedding and the hidden sizes are set to 850 to ensure our model size is comparable with other baselines. Other hyperparameters, including dropouts, remain the same as those for architecture search. For fair comparison, we do not finetune our model at the end of the optimization, nor do we use any additional enhancements such as dynamic evaluation (Krause et al., 2017) or continuous cache (Grave et al., 2016). The training takes 1.5 days on a 1080Ti GPU with our PyTorch implementation. To account for implementation discrepancies, we also incorporated the ENAS cell (Pham et al., 2018b) into our codebase and trained their network under the same setup as our discovered cells.

Table 6.1: Comparison with state-of-the-art image classifiers on CIFAR-10. Results marked with † were obtained by training the corresponding architectures using our setup.

Architecture	**Test Error (%)**	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	manual
NASNet-A + cutout (Zoph et al., 2017)	2.65	3.3	1800	RL
NASNet-A + cutout (Zoph et al., 2017)†	2.83	3.1	1800	RL
AmoebaNet-A + cutout (Real et al., 2018)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	evolution
Hierarchical evolution (Liu et al., 2017b)	3.75 ± 0.12	15.7	300	evolution
PNAS (Liu et al., 2017a)	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	RL
Random search baseline‡ + cutout	3.41 ± 0.08	2.9	4	random
DARTS (first order) + cutout	2.95 ± 0.07	3.0	1.5	gradient-based
DARTS (second order) + cutout	2.83 ± 0.06	3.4	4	gradient-based

‡ Best architecture among 24 samples according to the validation error after 100 training epochs.

6.3.3 Results Analysis

The CIFAR-10 results for convolutional architectures are presented in Table 6.1. Notably, DARTS achieved comparable results with the state of the art (Zoph et al., 2017; Real et al., 2018) while using three orders of magnitude less computation resources (i.e. 1.5 or 4 GPU days vs 1800 GPU days for NASNet and 3150 GPU days for AmoebaNet). Moreover, with slightly longer search time, DARTS outperformed ENAS (Pham et al., 2018b) by discovering cells with comparable error rates but less parameters. The longer search time is due to the fact that we have repeated the search process for four times for cell selection. This practice is less important for convolutional cells however, because the performance of discovered architectures does not strongly depend on initialization (Fig. 6.3).

Table 6.2 presents the results for recurrent architectures on PTB, where a cell discovered by DARTS achieved the test perplexity of 56.1. This is competitive with the state-of-the-art model enhanced by a mixture of softmaxes (Yang et al., 2017), and better than all the rest of the existing architectures that are either manually or automatically discovered. To the best of our knowledge, this is the first time an automatically searched architecture outperforms the extensively tuned LSTM (Melis et al., 2017), demonstrating the importance of architecture search in addition to hyperparameter search. In terms of efficiency, the overall cost (4 runs in total) is within 1 GPU day, which is comparable to ENAS and significantly faster than NAS (Zoph and Le, 2016).

It is interesting to note that random search is competitive for both convolutional and recurrent models, which reflects the importance of the search space design. Nevertheless, with comparable or less search cost, DARTS is able to significantly improve upon random search in both cases (2.83 ± 0.06 vs 3.41 ± 0.08 on CIFAR-10; 56.6 vs 59.4 on PTB).

To understand the necessity of bilevel optimization, we also investigated joint optimization of

Table 6.2: Comparison with state-of-the-art language models on Penn Treebank. Results marked with † were obtained by training the corresponding architectures using our setup.

Architecture	**Perplexity**		**Params**	**Search Cost**	**Search**
	valid	test	**(M)**	**(GPU days)**	**Method**
Variational RHN (Zilly et al., 2016)	67.9	65.4	23	–	manual
LSTM (Merity et al., 2017)	60.7	58.8	24	–	manual
LSTM + skip connections (Melis et al., 2017)	60.9	58.3	24	–	manual
LSTM + 5 softmax experts (Yang et al., 2017)	–	57.4	–	–	manual
LSTM + 15 softmax experts (Yang et al., 2017)	58.1	56.0	22	–	manual
NAS (Zoph and Le, 2016)	–	64.0	25	1e4 CPU days	RL
ENAS (Pham et al., 2018b)*	68.3	63.1	24	0.5	RL
ENAS (Pham et al., 2018b)†	60.8	58.6	24	0.5	RL
Random search baseline‡	61.8	59.4	23	2	random
DARTS (first order)	60.2	57.6	23	0.5	gradient-based
DARTS (second order)	58.8	56.6	23	1	gradient-based
DARTS (second order) + 1e3 more training epochs	58.3	56.1	23	1	gradient-based

* The results were obtained using the code (Pham et al., 2018a) publicly released by the authors.

‡ Best architecture among 8 samples according to the validation perplexity after 300 training epochs.

α and w over the union of the training and validation set, namely by solving $\min_{\alpha,w} \mathcal{L}_{train+val}(w, \alpha)$ using coordinate descent. The best cell among 4 runs yielded $4.16 \pm 0.16\%$ error rate on CIFAR-10 test set with 3.1M parameters, which is worse than the cell obtained using random search. We hypothesize that this simplistic optimization heuristic would cause α to overfit the training data, leading to poor generalization to the validation and test sets.

6.3.4 Transferability of Learned Architectures

6.3.4.1 ImageNet

We consider the *mobile* setting where the input size is 224×224 and the number of multiply-add operations in the model is restricted to be less than 600M. A network of 14 cells is trained for 250 epochs with batch size 128, weight decay 3×10^{-5} and initial SGD learning rate 0.1 (decayed by a factor of 0.97 after each epoch). Other hyperparameters follow Zoph et al. (2017); Real et al. (2018); Liu et al. (2017a)[4]. The training takes 12 days on a single GPU.

Table 6.3 shows that the cell learned on CIFAR-10 is indeed transferable to ImageNet. It is worth noticing that DARTS achieves competitive performance with the state-of-the-art RL method (Zoph et al., 2017) while using three orders of magnitude less computation resources.

6.3.4.2 WikiText-2

We use embedding and hidden sizes 700, weight decay 5×10^{-7}, and hidden-node variational dropout 0.15. Other hyperparameters remain the same as those for PTB.

[4]We did not conduct extensive hyperparameter tuning due to limited computation resources.

Table 6.3: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	**Test Error (%)** top-1	top-5	**Params (M)**	**$+\times$ (M)**	**Search Cost (GPU days)**	**Search Method**
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet $2\times$ (v1) (Zhang et al., 2017)	29.1	10.2	$\sim$5	524	–	manual
ShuffleNet $2\times$ (v2) (Zhang et al., 2017)	26.3	–	$\sim$5	524	–	manual
NASNet-A (Zoph et al., 2017)	26.0	8.4	5.3	564	1800	RL
NASNet-B (Zoph et al., 2017)	27.2	8.7	5.3	488	1800	RL
NASNet-C (Zoph et al., 2017)	27.5	9.0	4.9	558	1800	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2017a)	25.8	8.1	5.1	588	$\sim$225	SMBO
DARTS (searched on CIFAR-10)	26.9	9.0	4.9	595	4	gradient-based

Table 6.4: Comparison with state-of-the-art language models on WT2. Results marked with † were obtained by training the corresponding architectures using our setup.

Architecture	**Perplexity** valid	test	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
LSTM + augmented loss (Inan et al., 2017)	91.5	87.0	28	–	manual
LSTM + continuous cache pointer (Grave et al., 2016)	–	68.9	–	–	manual
LSTM (Merity et al., 2017)	69.1	66.0	33	–	manual
LSTM + skip connections (Melis et al., 2017)	69.1	65.9	24	–	manual
LSTM + 15 softmax experts (Yang et al., 2017)	66.0	63.3	33	–	manual
ENAS (Pham et al., 2018b)† (searched on PTB)	72.4	70.4	33	0.5	RL
DARTS (searched on PTB)	69.5	66.9	33	1	gradient-based

Table 6.4 shows that the cell identified by DARTS transfers better than ENAS on WT2, though the overall results are less strong than those presented in Table 6.2 for PTB. The weaker transferability between PTB and WT2 (as compared to that between CIFAR-10 and ImageNet) could be explained by the relatively small size of the source dataset (PTB) for architecture search. The issue of transferability could potentially be circumvented by directly optimizing the architecture on the task of interest.

6.4 Summary

In this chapter, we presented the first differentiable architecture search algorithm for both convolutional and recurrent networks. By searching in a continuous space, our method is able to match or outperform the state-of-the-art non-differentiable architecture search methods on image classification and language modeling tasks with remarkable efficiency improvement by several orders of magnitude. Note the differentiable search technique is orthogonal to the hierarchical representation that we developed in Chapter 5. For example, one could assign a continuous architecture encoding α to each motif in the hierarchy, hence the hierarchical architecture can be optimized using gradient descent instead of evolution for improved efficiency.

Chapter 7

Conclusion and Future Work

We have investigated several complementary directions related to learning over graphs, including principled methods for incorporating the structural information over heterogeneous graphs (part I), efficient induction of data-dependent graphs (Part II) and scalable optimization of computation graphs (Part III). While the advancement in each of the above direction is of great significance for a variety of applications, we are unable to cover all the important topics related to graphs. Some of them will be left as the future work.

- *Graphical Models*: In the aforementioned chapters, links in the graphs either refer to similarities (Chapter 2, 4), predicates (Chapter 3) or learnable functions (Chapter 5, 6). However, links may also carry the meaning of statistical dependencies or causalities in the field of probabilistic graphical models (Wainwright et al., 2008; Koller and Friedman, 2009). We note there has been an intensive literature on the scalable inference over the graph of random variables, beyond the scope of this thesis.
- *Graph Networks*: There has been a surge of recent interest in developing deep learning architectures with graph-structured/relational inductive biases. Many of these works, such as graph convolution networks in various forms (Kipf and Welling, 2016; Veličković et al., 2017; Gilmer et al., 2017; Lai et al., 2017), can be viewed as nonlinear extensions of the classic label propagation algorithm (Zhu et al., 2003) or manifold regularization (Belkin et al., 2006), and can be potentially combined with the techniques developed in Chapter 2, 3 and 4. A comprehensive review of this topic can be found at Battaglia et al. (2018).
- *Graph Generation*: While we have mostly focused on discriminative learning, there has been a growing body of research on the generative modeling over graphs. Related works include earliest schemes based on random graph models (Erdos and Rényi, 1960; Barabási and Albert, 1999), as well as more recent efforts using deep generative models (Wang et al., 2017; Johnson; Li et al., 2018). Incorporating generative models (e.g. as prior) into discriminative learning should help further improve the robustness and generalization.
- *Meta-Learning*: Most graph-based algorithms are restricted to in-sample nodes provided at training. Out-of-sample nodes or sub-graphs are traditionally handled using the Nyström method (Bengio et al., 2004; Fowlkes et al., 2004), which usually relies on strong assumptions. Meanwhile, the recent development of meta-learning has enabled deep neural networks to better generalize to unseen tasks by the explicitly training over the distribu-

tion of tasks (Koch et al., 2015; Vinyals et al., 2016; Andrychowicz et al., 2016; Santoro et al., 2016; Finn et al., 2017). Similar ideas of this kind could potentially help address the out-of-sample issue for graph-based approaches.

- *Improved Architecture Search Methods*: Although the traditional paradigm of architecture design has been factorized as search space design plus search algorithm design (Chapter 5, 6), both still require nontrivial amount of prior knowledge to obtain state-of-the-art results (Zoph et al., 2017; Liu et al., 2017b; Real et al., 2018; Pham et al., 2018b; Liu et al., 2018). There are several interesting directions to explore:
    - *Large Search Spaces*: It would be interesting to investigate how far we can achieve in a substantially enriched search space. This could be achieved in at least three ways: (1) One may learn the connections among the cells instead of manually stacking them according to a chain-structured topology, namely by removing prior assumptions on the top of the architecture hierarchy; (2) One may further decompose our current handcrafted convolutional primitives as batchnorms, activation functions and basic (linear) convolution operations. This means less prior assumptions on the bottom of the hierarchy. (3) Besides addition and concatenation, one may also consider multiplicative interactions among the operations (Hu et al., 2017).
    - *Improved Optimization Algorithms*: The performance of DARTS may suffer form approximation discrepancies due to single-step unrolling during the optimization, and the argmax in the last step. It would be interesting to implement multi-step unrolling (assuming sufficient memory), which can be formulated as back-propagation over time wrt the architecture encoding. It would also be very interesting to anneal the temperature of the softmaxes, allowing the continuous architecture to exactly reduce to a discrete architecture at the end of search. The memory relatively high consumption of DARTS (due to the fully connected graph) could be alleviated by doing sampling according to α in the forward/backward passes, combined with Gumbel-Softmax (Jang et al., 2016; Maddison et al., 2016) or the REINFORCE trick.

Bibliography

Jiří Adámek, Horst Herrlich, and George E Strecker. Abstract and concrete categories. the joy of cats. 2004. 3.2.4

Karim Ahmed and Lorenzo Torresani. Connectivity learning in multi-branch networks. *arXiv preprint arXiv:1709.09582*, 2017. 6.1

G Anandalingam and TL Friesz. Hierarchical optimization: An introduction. *Annals of Operations Research*, 34(1):1–11, 1992. 6.2.2

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016. 7

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007. 3.1

Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004. a

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 5.1, 5.4.3, 6.1

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *ICLR Workshop*, 2018. 6.1

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999. 7

Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first international conference on Machine learning*, page 9. ACM, 2004. 2.1

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 7

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, 7:2399–2434, 2006. 4.1, 7

Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl 1):i38–i46, 2005. 2.1

Yoshua Bengio, Jean-françcois Paiement, Pascal Vincent, Olivier Delalleau, Nicolas L Roux, and Marie Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *Advances in neural information processing systems*, pages 177–184, 2004. 7

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008. 3.1

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Conference on artificial intelligence*, number EPFL-CONF-192344, 2011. 3.2

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. Joint learning of words and meaning representations for open-text semantic parsing. In *AISTATS*, volume 22, pages 127–135, 2012. 3.2

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, pages 2787–2795, 2013. 2.5.3, 3.1, 3.2.2, 3.5.1, 3.5.2, 3.5.3, 3.2, 3.3

Stephen Boyd. Convex optimization of graph laplacian eigenvalues. In *Proceedings of the International Congress of Mathematicians*, volume 3, pages 1311–1319, 2006. b

Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Smash: One-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 5.1, 5.4.3, 6.1

Ronald Brown and Tim Porter. Category theory: an abstract setting for analogy and comparison. In *What is category theory*, volume 3, pages 257–274, 2006. 3.2.4

Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. Graph regularized nonnegative matrix factorization for data representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8):1548–1560, 2011. 2.1, 2.5.2

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. *AAAI*, 2018. 6.1

Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *The Journal of Machine Learning Research*, 9:1369–1398, 2008. 4.3.5

Danqi Chen, Richard Socher, Christopher D Manning, and Andrew Y Ng. Learning new facts from knowledge bases with neural tensor networks and semantic word vectors. *arXiv preprint arXiv:1301.3618*, 2013. 3.1

François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016. 5.4.3

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997. 4.1, 4.2.1

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007. 6.2.2

Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 365–374. ACM, 2014. 3.1

John M Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966. 4.3.2

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 6.3.2.1

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011. 2.4, 2.4, 3.5.4

Nelson Dunford, Jacob T Schwartz, William G Bade, and Robert G Bartle. *Linear operators*. Wiley-interscience New York, 1971. 3.2.3

Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960. 7

Brian Falkenhainer, Kenneth D Forbus, and Dedre Gentner. The structure-mapping engine: Algorithm and examples. *Artificial intelligence*, 41(1):1–63, 1989. 3.1

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008. 4.3.5

Rob Fergus, Yair Weiss, and Antonio Torralba. Semi-supervised learning in gigantic image collections. In *Advances in neural information processing systems*, pages 522–530, 2009. 2.3.2, b

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010. 3.1

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017. 6.2.3, 7

Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008. 5.1

Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystrom method. *IEEE transactions on pattern analysis and machine intelligence*, 26(2): 214–225, 2004. 7

Luca Franceschi, Paolo Frasconi, Saverio Salzo, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910*, 2018. 6.2.2

Evgeniy Gabrilovich and Shaul Markovitch. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34:443–498, 2009. 3.1

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016. 6.3.1.2

Alberto Garcia-Duran, Antoine Bordes, and Nicolas Usunier. *Composing relationships with translations*. PhD thesis, CNRS, Heudiasyc, 2015. 3.5.2, 3.2

Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7 (2):155–170, 1983. 3.1, 3.2

Lise Getoor. *Introduction to statistical relational learning*. MIT press, 2007. 2.1, 3.1

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017. 7

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991. 5.3.3

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016. 6.3.2.2, 6.4

Robert M Gray et al. Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239, 2006. 3.2.3, 3.4.3, 3.4.3

Robert Grone, Charles R Johnson, Eduardo M Sa, and Henry Wolkowicz. Normal matrices. *Linear Algebra and its Applications*, 87:213–225, 1987. 3.3.2

Frdric Gruau, L'universite Claude Bernard lyon I, Of A Diplome De Doctorat, M. Jacques Demongeot, Examinators M. Michel Cosnard, M. Jacques Mazoyer, M. Pierre Peretto, and M. Darell Whitley. Neural network synthesis using cellular encoding and the genetic algorithm., 1994. 5.1

Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094*, 2015. 3.1, 3.2.5

Masahiro Hattori, Yasushi Okuno, Susumu Goto, and Minoru Kanehisa. Heuristics for chemical compound matching. *Genome Informatics*, 14:144–153, 2003. 2.5.1

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. 5.1

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016b. 5.4.3

Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 623–632. ACM, 2015. 3.2

Douglas R Hofstadter. Analogy as the core of cognition. *The analogical mind: Perspectives from cognitive science*, pages 499–538, 2001. 3.2

John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1975. 5.1

Keith J Holyoak, Keith James Holyoak, and Paul Thagard. *Mental leaps: Analogy in creative thought*. MIT press, 1996. 3.2

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 6.3

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017. 7

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 5.1, 5.4.3

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017. 6.1

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *International Conference on Learning Representations*, 2017. 6.4

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 5.2.3

Takahiko Ito, Masashi Shimbo, Taku Kudo, and Yuji Matsumoto. Application of kernels to link analysis. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 586–592. ACM, 2005. 4.1

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 7

Rodolphe Jenatton, Nicolas L Roux, Antoine Bordes, and Guillaume R Obozinski. A latent factor model for highly multi-relational data. In *Advances in Neural Information Processing Systems*, pages 3167–3175, 2012. 3.5.2, 3.2

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL (1)*, pages 687–696, 2015. 3.5.2, 3.2

Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 985–991, 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11982`. 3.2

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002. 2.5.2

Daniel D. Johnson. Learning graphical state transitions. 7

Rie Johnson and Tong Zhang. Graph-based semi-supervised learning and spectral kernel design.

Information Theory, IEEE Transactions on, 54(1):275–288, 2008. 4.1, 4.2.2, 4.2.3, 4.4, 4.4.1

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018. 6.1

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 7

Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476, 1990. 5.1

Marius Kloft, Ulf Brefeld, Pavel Laskov, Klaus-Robert Müller, Alexander Zien, and Sören Sonnenburg. Efficient and accurate lp-norm multiple kernel learning. In *Advances in neural information processing systems*, pages 997–1005, 2009. 4.3.3

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015. 7

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51 (3):455–500, 2009. 2.1, 2.2.1

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 7

Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, volume 2, pages 315–322, 2002. 4.1, 4.2.2

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. *arXiv preprint arXiv:1709.07432*, 2017. 6.3.2.2

Jérôme Kunegis and Andreas Lommatzsch. Learning spectral graph transformations for link prediction. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 561–568. ACM, 2009. 4.1, 4.2.2, 4.2.3

Guokun Lai, Hanxiao Liu, and Yiming Yang. Learning graph convolution filters from data manifold. *arXiv preprint arXiv:1710.11577*, 2017. 7

Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004. a

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016. 5.1

Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *WLP*, pages 146–160. Springer, 1994. 2.1

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018. 7

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379*, 2015a. 3.5.2, 3.2

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187, 2015b. 3.1, 3.5.2, 3.2

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017a. 1.1.3.1, 6.1, 6.2.1, 6.3.1.1, 6.3.2.1, 6.1, 6.3.4.1, 6.3

Hanxiao Liu and Yiming Yang. Bipartite edge prediction via transductive learning over product graphs. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1880–1888, 2015. 1.1.1.1, 1.2, 2.1, 3.1

Hanxiao Liu and Yiming Yang. Cross-graph learning of multi-relational associations. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2235–2243, 2016a. 1.1.1.1, 1.2, 3.1

Hanxiao Liu and Yiming Yang. Semi-supervised learning with adaptive spectral transform. In *Artificial Intelligence and Statistics*, pages 902–910, 2016b. 1.1.2.1, 1.2

Hanxiao Liu, Wanli Ma, Yiming Yang, and Jaime Carbonell. Learning concept graphs from online educational data. *Journal of Artificial Intelligence Research*, 55:1059–1090, 2016a. 1.2

Hanxiao Liu, Keerthiram Murugesan, Jaime Carbonell, and Yiming Yang. Adaptive smoothed online multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4296–4304, 2016b. 1.2

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *International Conference on Learning Representations*, 2017b. 1.2, 6.1, 6.2.1, 6.2.2, 6.3.2.1, 6.1, 7

Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2168–2178. PMLR, 2017c. 1.1.1.2, 1.2

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1.1.3.2, 1.2, 7

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*, pages 2952–2960, 2016. 6.2.3

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015. 6.2.2

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 7

Stefano Melacci and Mikhail Belkin. Laplacian support vector machines trained in the primal. *The Journal of Machine Learning Research*, 12:1149–1184, 2011. 4.1

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017. 6.1, 6.3.3, 6.2, 6.4

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017. 6.3.1.2, 6.3.2.2, 6.2, 6.4

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017. 5.1, 5.4.3

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 3.1, 3.2.4

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989. 5.1

Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, pages 777–782, 2013. 3.1

Marvin Minsky. *Society of mind*. Simon and Schuster, 1988. 3.2

Atsuhiro Narita, Kohei Hayashi, Ryota Tomioka, and Hisashi Kashima. Tensor factorization using auxiliary information. *Data Mining and Knowledge Discovery*, 25(2):298–324, 2012. 2.1, 2.5.2

Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017. 5.1, 6.1

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. Stranse: a novel embedding model of entities and relationships in knowledge bases. *arXiv preprint arXiv:1606.08140*, 2016. 3.5.2, 3.2

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 809–816, 2011. 2.1, 3.1, 3.2.2, 3.5.2, 3.2, 3.3

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *arXiv preprint arXiv:1503.00759*, 2015. 3.1

Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1955–1961, 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12484`. (document), 3.1, 3.2.3, 3.4, 3.4.3, 3.5.2, 3.2, 3.3

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317, 2011. 2.3.2

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 6.3.2.1

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*, pages 737–746, 2016. 6.2.2

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014. 3.1, 3.2.4

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Authors' implementation of "Efficient Neural Architecture Search via Parameter Sharing". `https://github.com/melodyguan/enas/tree/2734eb2657847f090e1bc5c51c2b9cbf0be51887`, 2018a. Accessed: 2018-04-05. 6.2

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018b. (document), 6.1, 6.2.2, 1, 6.3.1.2, 6.3, 6.3.2.1, 6.3.2.2, 6.1, 6.3.3, 6.2, 6.4, 7

Tony A Plate. Holographic reduced representation: Distributed representation for cognitive structures. 2003. 3.4.3

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. 6.3.2.2

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017. 5.1, 5.3.2, 5.3.3, 5.3.4, 5.4.2, 5.4.3

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. (document), 1, 6.1, 6.2.1, 6.2.2, 1, 6.3.1.1, 6.3, 6.3.2.1, 6.1, 6.3.3, 6.3.4.1, 6.3, 7

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011. 3.5.4

Steffen Rendle, Leandro Balby Marinho, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, 2009. 2.1

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 5.4.1

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016. 7

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pages 4053–4061, 2016. 6.1

N Shawe-Taylor and A Kandola. On kernel target alignment. *Advances in neural information processing systems*, 14:367, 2002. 4.2.3, b

Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. In *Workshop at International Conference on Learning Representations*, 2018. 6.1

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5.1, 5.2.3

Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 2012. 3.1, 3.1

Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981. 2.5.1

Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003. 4.1, 4.2.2, 4.2.3

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013. 3.1, 3.5.2, 3.2

Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006. 4.3.3

Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012. 4.3.3

Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007. 5.1

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. 5.1, 5.3.2

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009. 5.1

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 5.1, 6.3

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016. 5.1, 5.4.1, 5.4.3

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017. 5.1, 5.4.3

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM, 2008. 2.5.1

Choon Hui Teo, SVN Vishwanthan, Alex J Smola, and Quoc V Le. Bundle methods for regularized risk minimization. *The Journal of Machine Learning Research*, 11:311–365, 2010. 4.3.3

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, 2015. 3.5.2, 3.2

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard.

Complex embeddings for simple link prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2071–2080, 2016. URL http://jmlr.org/proceedings/papers/v48/trouillon16.html. (document), 3.1, 3.4, 3.4.2, 3.5.2, 3.2, 3.3

Peter D Turney. The latent relation mapping engine: Algorithm and experiments. *Journal of Artificial Intelligence Research*, 33:615–655, 2008. 3.1

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 7

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016. 7

Heinrich Von Stackelberg. *Marktform und gleichgewicht*. J. springer, 1934. 6.2.3

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008. 7

Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267*, 2017. 7

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119. Citeseer, 2014. 3.1, 3.5.2, 3.2

James Hardy Wilkinson and James Hardy Wilkinson. *The algebraic eigenvalue problem*, volume 87. Clarendon Press Oxford, 1965. 3.3.1

Lingxi Xie and Alan Yuille. Genetic cnn. *arXiv preprint arXiv:1703.01513*, 2017. 5.1, 5.4.3

Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Representation learning of knowledge graphs with hierarchical types. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2965–2971, 2016a. 3.2

Saining Xie, Ross Girshick, Piotr Dollr, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016b. 5.1

Yoshihiro Yamanishi, Michihiro Araki, Alex Gutteridge, Wataru Honda, and Minoru Kanehisa. Prediction of drug–target interaction networks from the integration of chemical and genomic spaces. *Bioinformatics*, 24(13):i232–i240, 2008. 2.5.1

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2014. URL http://arxiv.org/abs/1412.6575. 3.1, 3.2.2, 3.2.5, 3.4, 3.4.1, 3.5.2, 3.2, 3.3

Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49. ACM, 1999. 3.5.5

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: a high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017. 6.3.1.2, 6.3.2.2, 6.3.3, 6.2, 6.4

Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999. 5.1

Kai Yu and Wei Chu. Gaussian process models for link analysis and transfer learning. In *Advances in Neural Information Processing Systems*, pages 1657–1664, 2008. 2.1

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 5.4.3

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017. 6.3

Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552*, 2017. 5.1, 5.4.1, 5.4.3

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Citeseer, 2002. 4.1

Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, volume 3, pages 912–919, 2003. 2.2.4, 4.1, 7

Xiaojin Zhu, Jaz Kandola, Zoubin Ghahramani, and John D Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In *Advances in neural information processing systems*, pages 1641–1648, 2004. 4.1, 4.2.2, 4.2.3, d, 1

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016. 6.3.1.2, 6.2

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 5.1, 5.4.3, 6.1, 6.2.2, 6.3.1.2, 6.3.3, 6.2

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017. (document), 5.1, 5.4.1, 5.4.1, 5.4.2, 5.4.3, 5.4.3, 6.1, 6.2.1, 6.2.2, 1, 6.3.1.1, 6.3, 6.3.2.1, 6.1, 6.3.3, 6.3.4.1, 6.3, 7