AN END-TO-END FRAMEWORK FOR UNIFIED COMPRESSION AND LEARNING TO OPTIMIZE THE MINIMAX PROBLEM

A Dissertation

by

JIAYI SHEN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Zhangyang Wang
Committee Members,	Theodora Chaspari
	Shuiwang Ji
	Xiaoning Qian
Head of Department,	Scott Schaefer

December 2022

Major Subject: Computer Science

ABSTRACT

Deep neural networks (DNNs) are resource-intensive and call for efficient compression methods to reduce the resource cost. For a composite DNN with various modules, the optimal resource allocation among these modules remains a question. To address this issue, we propose a novel unified compression framework that compress the whole network in an end-to-end way, without any multi-stage heuristics nor expensive hyper-parameters tuning. We demonstrate the generality of this framework by showing its superior performance in compressing a recommendation system and a vision transformer. Furthermore, we discuss to what extent the optimizers learnt by learning to optimize(L2O) technique can be adapted to a special class of functions and outperform general-purpose optimizers for the minimax objective.

ACKNOWLEDGMENTS

This dissertation owes a lot of gratitude to many people who have helped me in my research and the completion of the dissertation.

First, I would like to say thank you to my advisor, Dr.Zhangyang (Atlas) Wang, for his support, advice and insights in this field that make this dissertation possible. His brilliant guidance helped me in all the time of research and writing of this dissertation.

Second, I would like to thank my dissertation committee members, Dr.Theodora Chaspari, Dr.Shuiwang Ji, and Dr.Xiaoning Qian, for their time, advice, and firm support.

Also, I would like to thank my lab mates at the VITA (Visual Informatics Group @ Texas A&M University) Lab in the Department of Computer Science and Engineering, and all the collaborators during my internship, for offering valuable advice on my work.

More importantly, I would like to express the sincerest gratitude to my parents and my boyfriend. Their love supports me spiritually throughout my Ph.D. program.

Last but not least, I would like to thank Texas A&M University and all the funding agencies for their all kinds of support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation (or) dissertation committee consisting of Dr.Zhangyang Wang [advisor], Dr.Theodora Chaspari and Dr.Shuiwang Ji of the Department of Computer Science and Engineering, and Dr.Xiaoning Qian of the Department of Electrical and Computer Engineering.

Part of the experimental results in Chapter 6 were provided by Shixing Yu and Tianlong Chen. Part of the experimental results in Chapter 8 were provided by Dr.Xiaohan Chen.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

The graduate study was supported by ARL A2I2 project.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION*

1.1 Unified Recommendation Systems Compression

As the core component of a recommendation system (RS), recommendation models (RM) based on ranking neural networks are widely adopted in general content recommendation and retrieval applications. In general, an effective recommendation model consists of two components: a feature embedding sub-model and a prediction sub-model, as illustrated in Figure 1.1. Usually, an RM adopts neural networks to serve two sub-models. Formally, we denote an RM as $f(\cdot;\mathcal{W})$, where $\mathcal{W}$ is the learnable parameters of f. For the inference, the model f takes the input feature data $\boldsymbol{x}$ to predict the confidence of the content, serving the recommendation applications. Specifically, we further define the embedding and prediction sub-models as $f_e(\cdot;\mathcal{W}_e)$ and $f_p(\cdot;\mathcal{W}_p)$ respectively, where $\mathcal{W}_e$ and $\mathcal{W}_p$ are their own learnable parameters and $\mathcal{W} = \{\mathcal{W}_e, \mathcal{W}_p\}$. The embedding feature, $\boldsymbol{v} \coloneqq f_e(\boldsymbol{x};\mathcal{W}_e)$, is the input of f_p with the input data $\boldsymbol{x}$. Hence, we can express the RM as $f(\cdot;\mathcal{W}) \triangleq f_p(f_e(\cdot;\mathcal{W}_e);\mathcal{W}_p)$. Given a ranking training loss $\ell(\cdot)$ (i.e., binary cross entropy (BCE) loss), the learning goal of the ranking model can be written as

$$\min_{\mathcal{W}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \ell(f(\boldsymbol{x};\mathcal{W}),\boldsymbol{y}),$$

where $\boldsymbol{y}$ is the ground-truth label, and $\mathcal{D}$ is the training dataset.

Nowadays, extremely large-scale data have been poured into the recommendation system to predict user behavior in many applications. In the online inference procedure, the heaviest computation component is the layer-wise product between the hidden output vectors and the model

* Part of this chapter is reprinted with permission from "Learning a minimax optimizer: A pilot study " [2] by Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., Wang, Z. (2020, September) in International Conference on Learning Representations, Copyright 2020 held by the authors. Part of this chapter is reprinted with permission from "UMEC: Unified model and embedding compression for efficient recommendation systems" [5] by Shen, J., Wang, H., Gui, S., Tan, J., Wang, Z., Liu, J. (2020, September). in International Conference on Learning Representations, Copyright 2020 held by the authors. Part of this chapter is reprinted with permission from "Unified visual transformer compression" [3] by Yu, S., Chen, T., Shen, J., Yuan, H., Tan, J., Yang, S., ... Wang, Z. (2022). in International Conference on Learning Representations, Copyright 2022 held by the authors.

parameters $\mathcal{W}$, for a neural network-based RM. A slimmed neural network structure would save a great amount of power consumption during the inference. Hence, the main idea of an RM compression is to slim the structural complexity of $f(\mathcal{W})$ and reduce the dimension of hidden output vectors.

To obtain an efficient ranking model (for example, MLP based) for an RS, one may apply existing model compression methods to MLPs directly. For example, [6] removes the entire filters in the network together with their connecting feature maps in terms of magnitudes, which can also be applied to MLP structures to remove a specific neuron as well as its connections. [7] approximates the importance of a neuron (filter) to the final loss by using the first and second-order Taylor expansions, which can also be applied to pruning the neurons without hassle. There are also some compression methods focusing on dimension reduction of embedding feature vectors. [8] proposes a mixed dimension embedding scheme by designing non-uniform embedding dimensions scaling with popularity of features. [9] uses Neuron Input Search (NIS) to learn the embedding dimensions for the sparse categorical features.

However, the performance of these compression methods highly depends on the hyper-parameter tuning and the background knowledge of the specific recommendation model. For example, an embedding dimension compression method may require the user to search for the best dimension with multiple training and search rounds.

We would like to solve the RM compression problem with a unified resource-constrained optimization framework. It only relies on the resource consumption of the RM $f(\mathcal{W})$ to compress both the MLP and the embedding dimensions, without any multi-stage heuristics nor expensive hyper-parameters tuning. Our novel unified model and embedding compression (**UMEC**) framework directly satisfies both the requirement of resource consumption of the ranking neural network model and the prediction accuracy goal of the ranking model, with end-to-end gradient-based training. We reformulate the optimization of training loss associated with hard constraints into a minimax optimization problem solved by the alternating direction method of multipliers (ADMM) method [10].

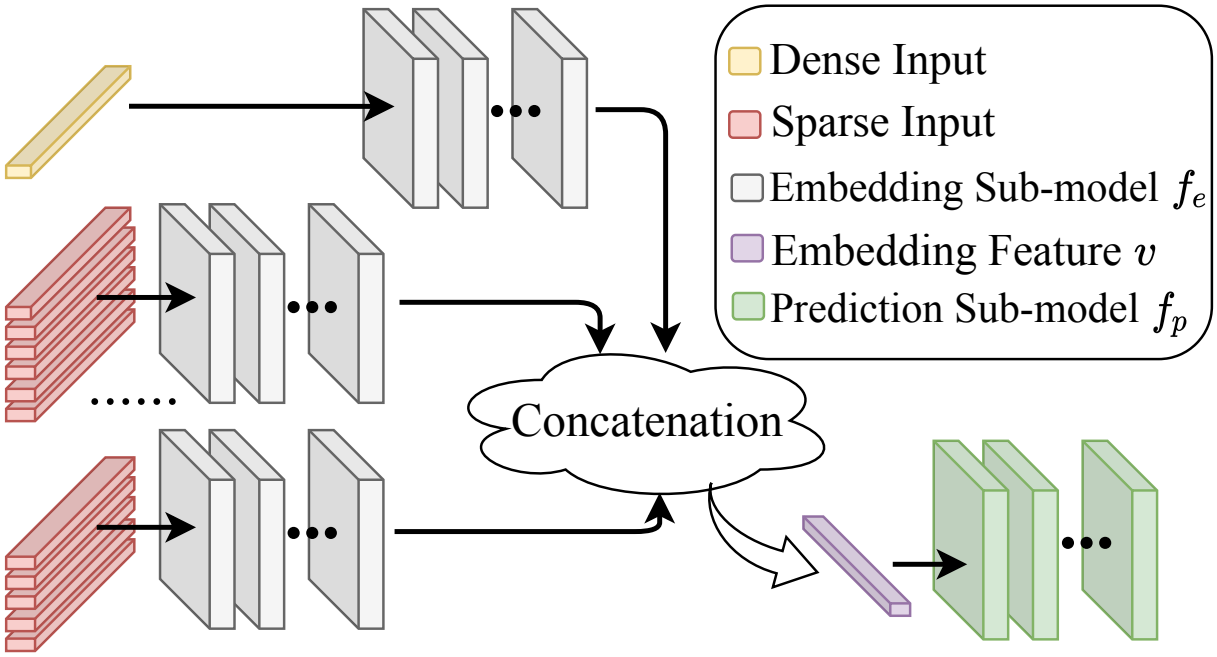


Figure 1.1: An example of the neural network-based recommendation system: the deep learning recommendation model (DLRM), proposed by [1]. Reprinted from [2].

1.2 Unified Visual Transformer Compression

Convolution neural networks (CNNs) [11–13] have been the *de facto* architecture choice for computer vision tasks in the past decade. Their training and inference cost significant and ever-increasing computational resources. Recently, drawn by the scaling success of attention-based models [14] in natural language processing (NLP) such as BERT [15], various works seek to leverage the Transformer architecture to computer vision [16–18]. The Vision Transformer (ViT) architecture [19], and its variants, have been demonstrated achieves comparable or superior results on a series of image understanding tasks compared to the state of the art CNNs, especially when pretrained on datasets with sufficient model capacity [20].

Despite the emerging power of ViTs, such architecture is shown to be even more resource-intensive than CNNs, making its deployment impractical under resource-limited scenarios. That is due to the absence of customized image operators such as convolution, the stack of self-attention modules that suffer from quadratic complexity with regard to the input size, among other factors.

Owing to the substantial architecture differences between CNNs and ViTs, although there is a large wealth of successful CNN compression techniques [6, 21–23], it is not immediately clear whether they are the same effective for ViTs. One further open question is how to best integrate their powers for ViT compression, as one often needs to jointly exploit multiple compression means for CNNs [24–26].

On the other hand, the NLP literature has widely explored the compression of BERT [27], ranging from unstructured pruning [28, 29], attention head pruning [30] and encoder unit pruning [31]; to knowledge distillation [32], layer factorization [33], quantization [34, 35] and dynamic width/depth inference [36]. Lately, earlier works on compressing ViTs have also drawn ideas from those similar aspects: examples include weight/attention pruning [37–39], input feature (token) selection [39, 40], and knowledge distillation [41, 42]. Yet up to our best knowledge, there has been no systematic study that strives to either compare or compose (even naively cascade) multiple individual compression techniques for ViTs – not to mention any joint optimization like [24–26] did for CNNs. We conjecture that may potentially limit the performance gain of ViT compression.

1.3 Learning A Minimax Optimizer

Many popular applications can be formulated into solving continuous minimax optimization, such as generative adversarial networks (GAN) [43], distributionally robust learning [44], domain adaptation [45], distributed computing [46, 47], privacy protection [48, 49], among many more. In the above proposed unified compression problem, the ultimate objective to be optimized is also a minimax problem, that can be elaborated as follows: we consider a cost function $f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ and the min-max game $\min_x \max_y f(x, y)$. We aim to find the *saddle point* (x^*, y^*) of f:

$$f(x^*, y) \leq f(x^*, y^*) \leq f(x, y^*), \quad \forall (x, y) \in \mathcal{X} \times \mathcal{Y}, \tag{1.1}$$

where $\mathcal{X} \subset \mathbb{R}^m$ and $\mathcal{Y} \subset \mathbb{R}^n$. If $\mathcal{X} = \mathbb{R}^m$ and $\mathcal{Y} = \mathbb{R}^n$, (x^*, y^*) is called a *global* saddle point; if $\mathcal{X} \times \mathcal{Y}$ is a neighborhood near (x^*, y^*), (x^*, y^*) is a *local* saddle point.

The main challenge to solve problem (1.1) is the unstable dynamics of iterative algorithms.

Simplest algorithms such as gradient descent ascent (GDA) can cycle around the saddle point or even diverge [50–52]. Plenty of works have been developed recently to address this issue [53–58]. However, the convergence is still sensitive to the parameters in these algorithms. Even if the cost function is only changed by scaling, those parameters have to be re-tuned to ensure convergence.

A recent trend of *learning to optimize* (**L2O**) parameterizes training algorithms to be learnable from data, such that the meta-learned optimizers can be adapted to a special class of functions and outperform general-purpose optimizers. That is particularly meaningful, when one has to solve a large number of yet similar optimization problems repeatedly and quickly. Specifically, for existing L2O methods that operate in the space of continuous optimization, almost all of them solve some minimization problem [4, 59, 60], leveraging an LSTM or a reinforcement learner to model their optimizer. Different from classic optimization results that often provide worst-case convergence, most L2O methods have little or no convergence guarantees, especially on problem or data instances distinct from what is seen in training, leaving their *generalizability* in practice questionable [61]. Motivated by L2O's success in learning efficient minimization solvers from data, we seek to answer: *whether we could accomplish strong minimax L2O solvers* as well; and if yes, *how generalizable they could be?*

As it might look straightforward at first glance, such extension is **highly nontrivial** due to facing several unique challenges. Firstly, while continuous minimization has a magnitude of mature and empirically stable solvers, for general minimax optimization, even state-of-the-art analytical algorithms can exhibit instability or even divergence. To the best of our knowledge, most state-of-the-art convergence analysis of minimax optimization is built on the convex-concave assumption [57, 58, 62], and some recent works relax the assumption to nonconvex-concave [52, 63]. Convergence for general minimax problems is still open. That makes a prominent concern on whether a stable minimax L2O is feasible. Secondly, given the two groups of min and max variables simultaneously, it is unclear to what extent their optimization strategies can be modeled and interact within one unified framework – a new question that would never be met in minimization. Thirdly, the noisy and sometimes cyclic dynamics of minimax optimization will provide noisier

guidance (e.g., *reward*) to L2O; not to say that, it is not immediately clear how to define the reward: for minimization, the reward is typically defined as the negative cumulative objective values along the history [60]. However, for minimax optimization the objective cannot simply be decreased or increased monotonically.

1.4 Dissertation Contributions

1.4.1 Unified Recommendation Systems Compression

To summarize our contributions, we present the merits of **UMEC** as the following:

- To the best of our knowledge, **UMEC** is the first unified optimization framework for the recommendation system scenario. Unlike those existing works that treat the selection of input feature and compression of the model as two individual problems, **UMEC** jointly learns both together via unifying both original prediction learning goal and the model compression related hard constraints.

- We reformulate the joint input feature selection and model compression task as a constrained optimization problem. We convert resource constraints and L_0 sparsity constraints into soft optimization energy terms and solve the whole optimization using ADMM methods.

- Extensive experiments performed over large-scale public benchmarks show that our method largely outperforms previous state-of-the-art input feature selection methods and model compression methods, endorsing the benefits of the proposed end-to-end optimization.

1.4.2 Unified Visual Transformer Compression

We establish the first *all-in-one compression framework* that organically integrate (structured) pruning. Rather than ad-hoc composition, we for the first time propose a *unified vision transformer compression* (**UVC**) framework, which seamlessly integrates the three effective compression and jointly optimizes towards the task utility goal under the budget constraints. UVC is mathematically formulated as a constrained optimization problem and solved using the primal-dual algorithm from end to end. Our main contributions are outlined as follows:

- We present UVC that unleashes the potential of ViT compression, by jointly leveraging multiple ViT compression means for the first time. UVC only requires to specify a global resource budget, and can automatically optimize the composition of different techniques.

- We formulate and solve UVC as a unified constrained optimization problem. It simultaneously learns model weights, layer-wise pruning ratios/masks, under an overall budget constraint.

- Extensive experiments are conducted with a DeiT backbone, outperforming or being competitive with past methods, which consistently verify the effectiveness of our proposal across different architectures. In detail, UVC on DeiT-Tiny yields around 50% FLOPs reduction, with little performance degradation (only 0.3%/0.9% loss compared to the baseline).

1.4.3 Learning A Minimax Optimizer

We conduct a pilot study into minimax L2O. We start by establishing the first dedicated minimax L2O framework, called *Twin-L2O*. It is composed of two LSTMs sharing one objective-based reward, separately responsible for updating min and max variables. By ablations of the design options, we find this decoupled design facilitate meta-learning most, particularly when the min and max updates are highly non-symmetric. We demonstrate the superior convergence of Twin-L2O on several testbed problems, compared against a number of analytical solvers.

On top of that, we further investigate how to enhance the generalizability of the learned minimax solver[1], and discuss two complementary alternatives with experimental validations. The first alternative is an empirical toolkit that is applicable for general minimax L2O. We introduce *curriculum learning* to training L2O models for the first time, by recognizing that not all problem instances are the same difficult to learn to solve. After plugging in that idea, we show that Twin-L2O can be trained to stably solve a magnitude more problem instances (in terms of parameter varying range). The second alternative explores a theoretical mechanism called *safeguarding*, particularly for the important *special case* of convex-concave problems. When solving a testing instance, safeguarding identifies when an L2O failure would occur and provides an analytical fall-back option [64]. That guarantees convergence for convex-concave problems and, in practice, converges faster even when the problem parameters are drawn from a different distribution from training.

[1]We differentiate the usages of two terms: **parameters** and **variables**, throughout the dissertation. For example, $\min_u \max_v \; ax^2 - by^2$, we call a, b parameters and x, y variables. For simplicity, this paper only discusses the L2O generalizability when the testing instances' parameter distribution differs from the training.

2. RELATED WORK*

2.1 Model Compression

Model compression aims to reduce the complexity of Deep Neural Network (DNN) models to achieve inference efficiency and resource saving. Generally, model compression techniques can be categorized into pruning [7, 65, 66], quantization [67–69] and distillation [70–72]. [65] proposes a magnitude-based and element-wise pruning technique, which does not guarantee the reduction of computation efficiency, energy and memory costs. Structured pruning can solve this issue by removing whole certain types of structures from the network, such as filter pruning and channel pruning methods [6,7,22]. Recent works [73–79] reveal that with appropriate compression techniques, the identified sparse subnetworks are capable of training in isolation to match the dense model performance. In the meanwhile, researchers also develop great interest in investigating hardware-aware compression, which utilizes practical resource requirements(e.g., energy, latency, Flops) to guide the compression [80–82].

2.2 Recommendation Models

With the recent development of deep learning techniques, plenty of works have proposed learning-based recommendation systems to grapple with personalized recommendation tasks. [83] applies recurrent neural networks (RNN) on long session data to model the whole session and achieve better accuracy than the traditional matrix factorization approaches. [84] proposes a high-level recommendation algorithm for YouTube videos by specifying a candidate generation model followed by a separate ranking model, which demonstrates a significant performance boost thanks to neural networks. [85] proposes a recommendation algorithm based on the graph neural network

* Part of this chapter is reprinted with permission from "Learning a minimax optimizer: A pilot study " [2] by Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., Wang, Z. (2020, September) in International Conference on Learning Representations, Copyright 2020 held by the authors. Part of this chapter is reprinted with permission from "UMEC: Unified model and embedding compression for efficient recommendation systems" [5] by Shen, J., Wang, H., Gui, S., Tan, J., Wang, Z., Liu, J. (2020, September). in International Conference on Learning Representations, Copyright 2020 held by the authors. Part of this chapter is reprinted with permission from "Unified visual transformer compression" [3] by Yu, S., Chen, T., Shen, J., Yuan, H., Tan, J., Yang, S., ... Wang, Z. (2022). in International Conference on Learning Representations, Copyright 2022 held by the authors.

to better capture the complex transitions among items, achieving a more accurate item embedding. Their method outperforms traditional sequential approaches. [1] exploits categorical features and designs a parallelism scheme for the embedding tables to alleviate the limited memory problem. [86] proposes a multi-attention based model to mitigate the low efficiency problem in the group recommendation scenarios.

2.3 Input Feature Compression For Recommendation Models

The numerous input features for practical recommendation scenarios necessitate the selection of useful ones to save memory resources and facilitate computational efficiency during inference. In factorization machines(FM)-based and context-aware recommendation systems, for example, [87] conducts input feature selection in an automatic manner by feature ranking and feature sub-sampling, while managing to improve the prediction quality. [8] designs a mixed dimension embedding layers scheme and adjusts the dimension of a particular embedding feature according to the frequency of that item. [9] performs Neural Input Search (NIS) method to search for the embedding sizes for the sparse categorical features, and designs the multi-size embedding framework to leverage the model capacity more efficiently. [88] proposes a discovering framework to automatically find the interaction architecture of the prediction model.

2.4 Vision Transformer

Transformer [14] architecture stems from natural language processing (NLP) applications first, with the renowned technique utilizing Self-Attention to exploit information from sequential data. Though intuitively, transformer model seems inept to the special inductive bias of space correlation for images-oriented tasks, it has proved itself of capability on vision tasks just as good as CNN [19]. The main point of Vision Transformer is that they encode the images by partitioning it into sequences of patches, projecting them into token embeddings, and feeding them to transformer encoders [19]. ViT outperforms convolutional nets if given sufficient training data on various image classification benchmarks.

Since then, ViT has been developed to various different variants first on data efficiency towards

training like DeiT [41] and T2T-ViT [89] are proposed to enhance ViT's training data efficiency, by leveraging teacher-student and better crafted architectures respectively. Swin-Transformer Then modifications are made to the general structure of ViT to tackle other popular downstream computer vision tasks, including object detection [90–93], semantic segmentation [94, 95], image enhancement [96, 97], image generation [98], video understanding [99], and 3D point cloud processing [100].

2.5 Minimax Optimization

Following [101], the minimax problem has been studied for decades due to its wide applicability. Simultaneous gradient descent (SimGD) or gradient descent ascent (GDA) [52, 102–104] is one of the simplest minimax algorithms, conducting gradient descent over variable x and gradient ascent over variable y. However, the dynamics of SimGD or GDA can converge to limit cycles or even diverge [50–52]. To address this issue, Optimistic gradient descent ascent (OGDA) simply modifies the dynamics of GDA and shows more stable performance [53–58]. OGDA attracts more attention because of its empirical success in training GANs. [62] theoretically studies OGDA by analyzing its continuous time dynamic and proposes Anchored simultaneous gradient descent that shows good performance. Follow-the-Ridge [105] also addresses the limit cycling problem by introducing second-order information into the dynamic of GDA. Lately, K-Beam [106] stabilizes the convergence of GDA by duplicating variable y, yielding strong performance. At each iteration, it performs gradient ascent independently on K copies of y and greedily chooses the copy that leads to a large function value f, then it updates x based on the selected copies.

2.6 Learning To Optimize

As a special instance of meta-learning, L2O has been studied in multiple contexts, with continuous optimization being one of its main playgrounds so far. The first L2O framework is introduced in [4], where both the optimizee's gradients and loss function values are formulated as the input features for an RNN optimizer. Due to the enormous number of parameters, a coordinate-wise design of RNN optimizer is adopted, where all optimization coordinates share the same updating

strategy. [60] uses the gradient history and objective values as observations and step vectors as actions in their reinforcement learning framework. [59] leverages RNN to train a meta-optimizer to optimize black-box functions. Two effective training tricks, random scaling and objective convexifying, are presented in [107]. [108] presents an optimizer of multi-level hierarchical RNN architecture augmented with additional architectural features. [109] introduces a Jacobian regularization to L2O and enhances the domain adaptation performance of optimizees. [110] proposes several improved training techniques to stabilize L2O training and ameliorate performance.

The above works address continuous minimization problems using single optimizer models. One exception, [111], extends L2O to solving Bayesian swarm optimization. The author presents a novel architecture where multiple LSTMs jointly learn iterative update formulas for the swarm of particles, coordinated by attention mechanisms. We also notice that two recent efforts [112, 113] introduce L2O to adversarial training, a renowned application of minimax optimization. However, both of them merely utilize L2O to solve the inner minimization of their minimax problems (i.e., generating attacks), while the outer maximization is still solved analytically. Neither of the two directly solves the full minimax optimization.

3. METHODOLOGY: UNIFIED COMPRESSION FOR RECOMMENDATION SYSTEMS*

3.1 Method Overview

For an arbitrary NN-based recommendation model $f(\cdot)$ in an RS, the terminology of "compression" refers to reduce the power consumption with respect to all the computation operations for the inference. The learning parameters of $f(\mathcal{W})$ can be expressed as $\mathcal{W} \coloneqq \left\{\boldsymbol{W}^{(l)} | l \in [L]\right\}$, where L is the number of layers in the $f(\mathcal{W})$. For an arbitrary layer l, the layer's weight is denoted as $\boldsymbol{W}^{(l)} \in \mathbb{R}^{\mathrm{d}_{\text{out}} \times \mathrm{d}_{\text{in}}}$. The recommendation inference on the layer l is executed as

$$\boldsymbol{x}^{(l+1)} = \sigma\left(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}\right),$$

where $\boldsymbol{x}^{(l)} \in \mathbb{R}^{\mathrm{d}_{\text{in}}}$ is the input of l-th layer, and $\boldsymbol{b}^{(l)}$ is the corresponding learning bias term. Since the RM consists of two components: feature embedding sub-model and neural network prediction sub-model, the "compression" can be interpreted as shrinking the dimension of embedding feature $\boldsymbol{v}$ and reducing the number of parameters in $\mathcal{W}$ and $\boldsymbol{b}$.

3.1.1 Perspective Of Model Parameters

It is to reduce the input or output neurons for the specific layer l. For example, the i-th neuron of $\boldsymbol{x}^{(l+1)}$ and j-th neuron of $\boldsymbol{x}^{(l)}$ are chosen to remove, then the weight matrix $\boldsymbol{W}^{(l)}$ will be reshaped to $(\mathrm{d}_{\text{out}} - 1) \times (\mathrm{d}_{\text{in}} - 1)$.

3.1.2 Perspective Of Feature Embedding Vectors

The reduction of d_{in} would result in the elimination of the input features. For example, $\boldsymbol{v}^{(1)}$ is defined as the concatenation of a series of embedding vectors, $\boldsymbol{v}^{(1)} = [\boldsymbol{e}_1; \boldsymbol{e}_2; \cdots; \boldsymbol{e}_n]$, where $[\cdot;\cdot;\cdot]$ represents the concatenation operator, and n denotes the number of embedding feature vectors in the RS. Without loss of generality, for the first embedding vector $\boldsymbol{e}_1 \in \mathbb{R}^{\mathrm{d}_1}$, the zero-out of first

* This chapter is reprinted with permission from "UMEC: Unified model and embedding compression for efficient recommendation systems [5]" by Shen, J., Wang, H., Gui, S., Tan, J., Wang, Z., Liu, J. (2020, September). in International Conference on Learning Representations, Copyright 2020 held by the authors.

d_1 columns in $\boldsymbol{W}^{(1)}$ will result in the elimination of the whole embedding vectors for $\boldsymbol{e}_1$. On the other hand, the compression on $f_e(\mathcal{W}_e)$ would reduce the feature embedding vectors' dimensions.

3.1.3 A Compression Example: Group Lasso For Weight Pruning

We first consider a simple example with an RS model consisting of $f_e(\mathcal{W}_e)$ and $f_p(\mathcal{W}_p)$. For this RS, to reduce the computation consumption, we present a straightforward model compression method with Group Lasso. The method considers both the feature embedding selection and network slimming. We only need to apply the sparse regularization terms to the weight matrices. The optimization problem can be expressed as

$$\min_{\mathcal{W}} \ell(\mathcal{W}) + \lambda_1 \sum_i \left\| \boldsymbol{W}^{(1)}_{.,g_i} \right\|_1 + \sum_{l=2}^{L} \lambda_l \sum_i \left\| \boldsymbol{W}^{(l)}_{.,i} \right\|_1 , \tag{3.1}$$

where $\ell(\mathcal{W})$ is the original objective function . The λs are hyper-parameters to control the importance of these regularization terms. $g_.$ denotes the feature groups for the input layer. $\| \cdot \|_1$ indicates the $\mathcal{L}_1$ norm. The optimization problem (3.1) restricts the group-wise sparsity for the input layer and the neuron-wise sparsity (or group size = 1) for the rest model layers while training with the original objective function $\ell(\mathcal{W})$. The optimization problem can be solved with the projected gradient descent method. However, the optimization problem (3.1) has its own limitation. The hyper-parameters λs only have the latent impact on the model size and we could not directly affect the model size. It is not friendly when the users would like to restrict the model size to a specific number. On the other hand, the hyper-parameters also make the optimization problem hard to tune, requiring multiple arguments searching rounds. To avoid these shortages, we are supposed to have new approaches to handling the compression of RS.

3.2 Resource-Constrained RS Model Compression

In this section, we propose a novel unified model and embedding compression method on RS and treat it as a constrained optimization problem. We discuss how to solve the problem using a gradient-based optimization algorithm.

3.2.1 Computation Resource Function For RS Model

To avoid the hyper-parameter tuning in the optimization of problem (3.1), we define the resource function to measure the resource consumption for a specific RS model structure, which can be used to guide the compression of structure. For example, if considering the computation Flops, the resource of an MLP can be defined as

$$R_{\text{Flops}} = \sum_{l=1}^{L} \left(2 \times \mathrm{d}_{\text{in}}^{(l)} \times \mathrm{d}_{\text{out}}^{(l)} + \mathrm{d}_{\text{out}}^{(l)}\right),$$

where $\mathrm{d}_{\text{in}}^{(l)}$ and $\mathrm{d}_{\text{out}}^{(l)}$ indicate the number of input and output neurons. For a pruned layer l, the input and output dimension are restricted by the number of pruned neurons, annotated as $s^{(l)}$ and $s^{(l+1)}$.

Therefore, we can define a resource consumption function in terms of s from each layer as

$$R_{\text{Flops}}(\boldsymbol{s}) \triangleq \sum_{l=1}^{L} \left(2 \times \left(\mathrm{d}_{\text{in}}^{(l)} - s^{(l)}\right) \times \left(\mathrm{d}_{\text{out}}^{(l)} - s^{(l+1)}\right) + \left(\mathrm{d}_{\text{out}}^{(l)} - s^{(l+1)}\right)\right).$$

For simplicity, the resource function R_{Flops} only takes the s as the number of neurons. In a more general case, s can be substituted with the product between the number of pruned groups and the size of each pruned group.

With the definition of the resource function, we can formulate the following optimization problem,

$$\min_{\mathcal{W}} \ell(\mathcal{W}), \quad \text{s.t. } R_{\text{Flops}}(\mathcal{W}) \leq R_{\text{budget}}, \tag{3.2}$$

where R_{budget} denotes the upper bound of the overall model computation Flops set by users. Since we consider the group-wise sparsity for each layer, we utilize the set of variables $\boldsymbol{s} \coloneqq \{s^{(1)}, s^{(2)}, \ldots, s^{(L)}\}$ to control the model's Flops. The $s^{(l)}$ defines the lower bound of the l-th layer's zero group number for the pruned model. Therefore, we reformulate the optimization prob-

lem (3.2) as

$$\min_{\mathcal{W},\boldsymbol{s}} \ \ell(\mathcal{W}) \tag{3.3a}$$

$$\text{s.t. } R_{\text{Flops}}(\boldsymbol{s}) \leq R_{\text{budget}} \tag{3.3b}$$

$$\sum_i \mathbb{I}\left(\left\|\boldsymbol{W}_{.,i}^{(l)}\right\|_2^2 = 0\right) \geq s^{(l)},\ \forall\, l = 2, \ldots, L, \tag{3.3c}$$

$$\sum_i \mathbb{I}\left(\left\|\boldsymbol{W}_{.,g_i}^{(1)}\right\|_2^2 = 0\right) \geq s^{(1)}, \tag{3.3d}$$

where $\mathbb{I}(\cdot)$ is an indicator function with the output $\{0, 1\}$. When the condition is satisfied, the indicator function results in a value of 1, otherwise, it would be 0. $\sum_i \mathbb{I}\left(\left\|W_{.,i}^{(l)}\right\|_2^2 = 0\right)$ denotes the zero group number for the l-th layer. The resource constraint (5.2) bounds the resource consumption of the recommendation model. The inequalities in both (5.1c) and (3.3d) regularize the Flops consumption for each layer.

3.3 Optimization

To address the optimization problem (5.1), we adopt the alternating direction method of multipliers (ADMM) for the reformulation. In details, the optimization problem contains non-continuous indicator function in constraint (5.1c, 3.3d), and non-convex constraint (5.2), which makes the problem difficult to solve. Therefore, we first reformulate the inequality constraints as soft regularizations and introduce minimax optimization with dual variables. Then we tackle the non-differentiable objective function using self-defined numerical differentiation. At last, we summarize all the optimization details into a gradient-based optimization formulation.

3.3.1 Minimax Optimization Reformulation

Let $\|\boldsymbol{W}_{.,g}\|_{s,2}$ be the bottom-$(s, 2)$ "norm", which denotes the Frobenius-norm of the sub-matrix (group) composed of bottom-s groups, by sorting from top to bottom in terms of norm of each

group. The equivalent formulation of the sparsity constraint (5.1c, 3.3d) can be

$$\|\boldsymbol{W}_{\cdot,g}\|_{s,2}^2 = 0 \Leftrightarrow \sum_i \mathbb{I}\left(\|\boldsymbol{W}_{\cdot,g_i}\|_2^2 = 0\right) \geq s. \tag{3.4}$$

The above equivalence (5.3) is proved by [114], where the LHS is called DC (difference of convex functions) representation of the $\mathcal{L}_0$-constraint. With the transformation, we can reformulate the problem in Eq. (5.1), into this minimax optimization problem:

$$\min_{\mathcal{W},\boldsymbol{s}} \max_{\boldsymbol{y},z\geq 0} \ell(\mathcal{W}) + \underbrace{y^{(1)} \left\|\boldsymbol{W}_{\cdot,g}^{(1)}\right\|_{\lceil s^{(1)}\rceil,2}^2 + \sum_{l=2}^{L} y^{(l)} \left\|\boldsymbol{W}_{\cdot,\cdot}^{(l)}\right\|_{\lceil s^{(l)}\rceil,2}^2}_{\text{sparsity loss: } \mathcal{S}(\boldsymbol{y},\boldsymbol{s},\mathcal{W})} + \underbrace{z\left(R_{\text{Flops}}(\boldsymbol{s}) - R_{\text{budget}}\right)}_{\text{resource loss}}, \tag{3.5}$$

where y and z are dual variables. The operator $\lceil\cdot\rceil$ indicates the ceiling function. In the problem (5.5), the importance of regularization terms are tuned automatically based on the minimax optimization. The sparsity loss $\mathcal{S}(\boldsymbol{y},\boldsymbol{s},\mathcal{W}) \coloneqq y^{(1)} \left\|\boldsymbol{W}_{\cdot,g}^{(1)}\right\|_{\lceil s^{(1)}\rceil,2}^2 + \sum_{l=2}^{L} y^{(l)} \left\|\boldsymbol{W}_{\cdot,\cdot}^{(l)}\right\|_{\lceil s^{(l)}\rceil,2}^2$ and the resource loss $z\left(R_{\text{Flops}}(\boldsymbol{s}) - R_{\text{budget}}\right)$ are introduced to substitute the original constraints (5.2, 5.1c, 3.3d). Therefore, the optimal solution to the new problem (5.5) can serve as the optimal for the original problem (5.1).

3.3.2 Update Rule For All Learnable Variables

To solve the optimization problem (5.5), we utilize the ADMM strategy to solve a series of sub-problems as follows.

3.3.2.1 Update $\mathcal{W}$

The optimization problem on $\mathcal{W}$ can be solved with proximal-SGD [115]. The definition of the proximal operator is as follows:

$$\text{Prox}_{\eta\mathcal{S}(\boldsymbol{y},\boldsymbol{s},\mathcal{W})}(\bar{\mathcal{W}}) = \arg\min_{\mathcal{W}} \frac{1}{2}\left\|\mathcal{W} - \bar{\mathcal{W}}\right\|^2 + \eta\mathcal{S}\left(\boldsymbol{y},\boldsymbol{s},\mathcal{W}\right)$$

where $\bar{\mathcal{W}}$ is the direct stochastic gradient descent update of $\mathcal{W}$. The sub-problem above has the closed-form solution for $\mathcal{W}^* := \left\{\boldsymbol{W}^{(1)*}, \boldsymbol{W}^{(2)*}, \ldots, \boldsymbol{W}^{(L)*}\right\}$:

$$\boldsymbol{W}_i^* = \begin{cases} \bar{\boldsymbol{W}}_i, & \text{if } \left\|\bar{\boldsymbol{W}}_i\right\| >= \left\|\bar{\boldsymbol{W}}_{\text{least-}\lceil s \rceil}\right\|, \\ \frac{1}{1+2\eta y}\bar{\boldsymbol{W}}_i, & \text{otherwise,} \end{cases}$$

where $\boldsymbol{W}_i$ denotes the i-th sub-matrix, and $\boldsymbol{W}$ is the weight for an arbitrary layer.

3.3.2.2 Update Dual Variables

For the sub-optimization problem, we have

$$\max_{\boldsymbol{y}, z \geq 0} \; y^{(1)} \left\|\boldsymbol{W}_{\cdot,g}^{(1)}\right\|_{\lceil s^{(1)} \rceil,2}^2 + \sum_{l=2}^{L} y^{(l)} \left\|\boldsymbol{W}_{\cdot,\cdot}^{(l)}\right\|_{\lceil s^{(l)} \rceil,2}^2 + z\left(R_{\text{Flops}}(\boldsymbol{s}) - R_{\text{budget}}\right),$$

and we can adopt the gradient ascent method as the update rule.

3.3.2.3 Update s

The optimization on $\boldsymbol{s}$ relies on both sparsity and resource loss. However, both of them are non-differentiable due to the ceiling function $\lceil s \rceil$. Straight-through estimator (STE) [116] is an effective approach for optimization through the non-differentiable functions. The main idea is to adopt some simple proxy to be the derivative of the non-differentiable formulation. Thus, the back-propagation can be applied as what happened in the differentiable case. For $\|\boldsymbol{W}\|_{s,2}^2$, we can apply the numerical differentiation $\|\boldsymbol{W}\|_{s+1,2}^2 - \|\boldsymbol{W}\|_{s,2}^2$ as the proxy derivative of $\|\boldsymbol{W}\|_{s,2}^2$ with respect to s:

$$\frac{\tilde{\partial}\left\|\boldsymbol{W}\right\|_{s,2}^2}{\tilde{\partial} s} = \boldsymbol{W}^2{}_{\text{least-min(Dim}(\boldsymbol{W}),\, s+1)},$$

where $\boldsymbol{W}^2$ is the column-group wise $\mathcal{L}_2$ format of $\boldsymbol{W}$, Dim($\boldsymbol{W}$) is the total number of column groups of $\boldsymbol{W}$, and $\boldsymbol{W}_{\text{least}-j}^2$ is the j-th least column group in $\boldsymbol{W}^2$.

For a general resource consumption function (e.g., Flops), the non-differentiable part of R is

the ceiling function $\lceil s \rceil$, for which we can use a common STE [116]: $\frac{\tilde{\partial} \lceil s \rceil}{\tilde{\partial} s} = 1$.

$$\frac{\tilde{\partial} \lceil s \rceil}{\tilde{\partial} s} = 1.$$

Now we can wrap up all the update rules together to be a unified optimization algorithm 2. In this algorithm, we only show an example for the input layer to be column-wise grouped. For a more general case, the column-group wise sparsity can be applied to an arbitrary layer in the RS model.

We wrap up all the update rules together to be a unified optimization algorithm, as described in Algorithm 2. In this algorithm, we only show an example that the input layer is column-wise grouped. For a more general case, the column-wise group sparsity can be applied to an arbitrary layer in the RS model.

Algorithm 1: Gradient-based algorithm to solve problem (5.5) for **UMEC**.

Input: Resource budget R_{budget}, learning rates $\eta_1, \eta_2, \eta_3, \eta_4$, number of total iterations τ.
Result: DNN pruned weights $\mathcal{W}^*$.

Initialize $t = 1, \mathcal{W}^1$; `// random or a pre-trained dense model`

for $t \leftarrow 1$ **to** τ **do**

$\mathcal{W}^{t+1} = \text{Prox}_{\eta_1 \mathcal{S}(\boldsymbol{y}^t, \boldsymbol{s}^t, \mathcal{W}^t)} \left(\mathcal{W}^t - \eta_1 \hat{\nabla}_{\mathcal{W}} \ell \left(\mathcal{W}^t \right) \right)$; `// Proximal-SGD`

$\boldsymbol{s}^{t+1} = \boldsymbol{s}^t - \eta_2 \left(\tilde{\nabla}_{\boldsymbol{s}} \mathcal{S} \left(\boldsymbol{y}^t, \boldsymbol{s}^t, \mathcal{W}^{t+1} \right) + \tilde{\nabla}_{\boldsymbol{s}} z^t \left(R_{\text{Flops}} \left(\boldsymbol{s}^t \right) - R_{\text{budget}} \right) \right)$; `// Gradient (STE) Descent`

$y^{(1)^{t+1}} = y^{(1)^t} + \eta_3 \left(\left\| \boldsymbol{W}_{\cdot,g}^{(1)^{t+1}} \right\|^2_{\lceil \boldsymbol{s}^{(1)^{t+1}} \rceil, 2} \right)$; `// Gradient Ascent`

$y^{(l)^{t+1}} = y^{(l)^t} + \eta_3 \left(\left\| \boldsymbol{W}^{(l)^{t+1}} \right\|^2_{\lceil \boldsymbol{s}^{(l)^{t+1}} \rceil, 2} \right), \forall\, l = 2, \cdots, L$

$z^{t+1} = z^t + \eta_4 \left(R_{\text{Flops}} \left(\boldsymbol{s}^t \right) - R_{\text{budget}} \right)$; `// Gradient Ascent`

end

$\mathcal{W}^* = \mathcal{W}$

3.3.3 Prune And Finetune

Solving the minimax optimization problem (5.5) will make some channel weights extremely close but not exactly equal to zeros. In order to extract the slimmed network, for each $\boldsymbol{W}^{(l)}$, we discard the $s^{(l)}$ groups with smallest $\mathcal{L}_2$ norms $\|\boldsymbol{W}^{(l)}_{:,g_i}\|_2$ (extremely close to 0) after we have solved problem (5.5). Then we finetune the explicitly pruned network to further improve the performance.

4. EXPERIMENTS: UNIFIED COMPRESSION FOR RECOMMENDATION SYSTEMS *

To demonstrate the effectiveness of **UMEC**, we run experiments on a standard public RM over the real-world recommendation dataset. And we would like to answer the following questions: *Whether our unified optimization framework that jointly considers input feature selection and prediction model compression has superiority over previous state-of-the-art RS compression methods which solve the problem from only one aspect? If yes, why?*

4.1 Experimental Setup

4.1.1 Dataset And Model

We conduct all our experiments on the state-of-the-art ads CTR prediction model named DLRM [1], as illustrated in Figure 1.1. Following [1], we use the Criteo AI Labs Ad Kaggle[1] and Terabyte[2] datasets for our experiments. The Criteo AI Labs Ad Kaggle dataset contains approximately 45 million click log samples collected over seven days. The Terabyte dataset consists of approximately 4.4 billion click log samples collected over 24 days and we perform uniform sampling with 12.5% sampling rate from the raw data following the sampling scheme in [1]. Both datasets contain samples that have 13 continuous and 26 categorical input features. Following the official setting, for both datasets we split the data of the last day into validation and testing sets, and use data from the rest days as the training set.

4.1.2 Baseline Methods

The latency of an ads CTR prediction model mainly comes from two sources: feature embedding and prediction model inference. Feature embedding layers map categorical input values into a vector space. Both the number of input categorical values and the output feature space dimension can affect the latency of feature embedding stage. The inference latency of the predication model

* This chapter is reprinted with permission from "UMEC: Unified model and embedding compression for efficient recommendation systems" [5] by Shen, J., Wang, H., Gui, S., Tan, J., Wang, Z., Liu, J. (2020, September). in International Conference on Learning Representations, Copyright 2020 held by the authors.

[1] https://www.kaggle.com/c/criteo-display-ad-challenge

[2] https://labs.criteo.com/2013/12/download-terabyte-click-logs/

mainly depends on the prediction model size. Following the above analyses, methods to accelerate ads CTR prediction models mainly fall into three categories:

- Prediction model compression: compressing the prediction model with traditional model compression methods, such as [6, 7, 81].
- Input feature selection: reducing the number of categorical input features by filtering out non-essential ones.
- Embedding dimension reduction: shrinking the dimension of embedding layers' output space. Two recent works [8, 9] fall into this category.

Our unified optimization framework jointly considers input feature selection and model compression. We will further show that our method can be generalized to considering input feature dimension-reduction as well. So we compare our method with baselines from all three categories to show the superiority of our method over those baselines tackling the RS compression problem from only one aspect.

4.1.3 General Training And Evaluation Details

We use the following settings for our experiments on the Criteo Ad Kaggle dataset: we use SGD optimizer with learning rate 0.1 to optimize the BCE loss ($\ell(\mathcal{W})$ in our method); we set training batch size to 128, initial feature embedding dimension to 16, and use the three-layer MLP prediction model with hidden dimensions 512 and 256.

For our method, we choose Adam optimizer [117] for $\boldsymbol{s}$ and SGD optimizer for $\mathcal{W}$, $\boldsymbol{y}$, and z. We set learning rates $\eta_1, \eta_2, \eta_3, \eta_4$ in Algorithm 2 to be $[0.1, 0.05, 0.1, 2.0]$ respectively, and set τ to be 306,969 which equals to 1 epoch. In our experiments, we initialize the model weights from a dense model that has been pre-trained for 1 epoch using $\ell(\mathcal{W})$ to facilitate training. After the training process in Algorithm 2, we zero out the redundant neurons (as described in the last paragraph in Section 3) and extract the sub-network as the pruned model. We then perform fine-tuning for 2 epochs. For a fair comparison, we also fine-tune 2 epochs for all baseline methods. Following [1], the best accuracy on the validation set is reported.

For experiments on the Criteo Ad Terabyte dataset, the training settings generally follow those mentioned above with several exceptions: batch size equal to 4096 , τ equal to 157,655, the hidden dimensions of the three-layer MLP prediction model equal to 256 and 128.

4.1.4 Evaluation Metrics

We evaluate the performance with the following two metrics: *Accuracy* and *Compression Ratio (CR)*. **Accuracy:** CTR prediction accuracy on the validation set. **Compression ratio (CR):** the ratio between the Flops that has been deducted and original model Flops. More formally, CR $= 1 - p_f(\mathcal{W})/p_f(\mathcal{W}_0)$, where $p_f(\cdot)$ calculates the Flops of a (sub-)model. $\mathcal{W}$ and $\mathcal{W}_0$ are the pruned and original dense prediction models, respectively. The larger CR, the more compact model we will obtain.

4.2 Prediction Model Compression For RS Model

We design and conduct extensive experiments to examine the compression performance among our method and several state-of-the-arts model compression methods. The baseline methods include handcrafted structures, one-shot magnitude pruning (MP) [6], and two recent state-of-the-art methods, namely Taylor pruning (TP) [7] and energy-constrained model compression (ECC) [81].

In Figure 4.1 and 4.2, we demonstrate the best accuracy on validation set under different compression ratios (CRs) on the Criteo AI Labs Ad Kaggle and Terabyte datasets. Each curve shows the CR-Accuracy trade-off of one compression method. The doted magenta line represents the accuracy achieved by the original dense DLRM model.

Several observations can be drawn from this comparison figure. First, our method continuously outperforms all other methods with a margin of at least 0.01%. Second, only our method can successfully compress the original dense model to the ideal budget while still manages to maintain the performance without any degradation within a certain CR scope.

These results show that our joint model compression and input selection method can achieve much better balance of efficiency vs accuracy than traditional model compression methods.

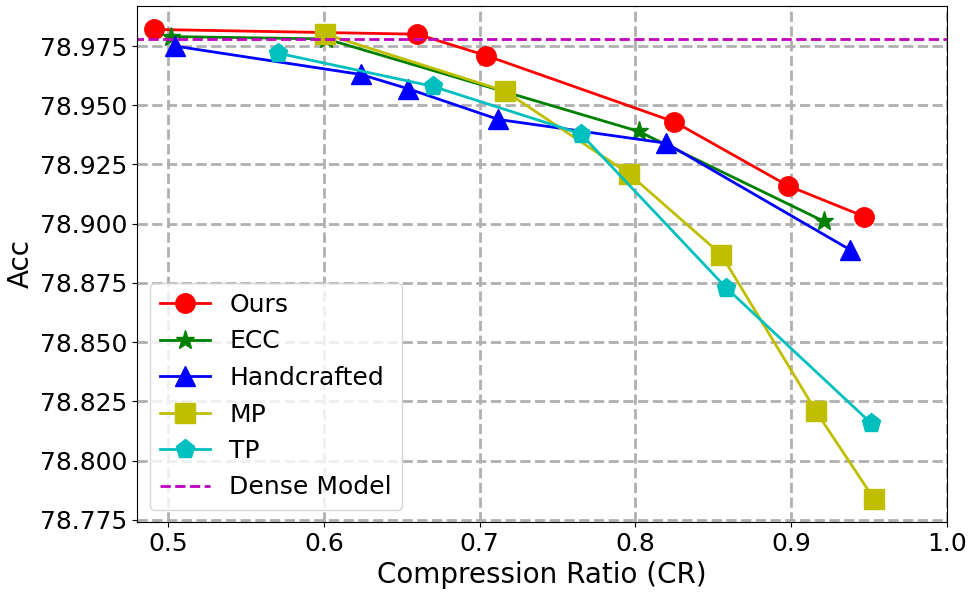


Figure 4.1: Results on RS prediction model compression by different methods on the Criteo AI Labs Ad Kaggle dataset. Reprinted from [2].

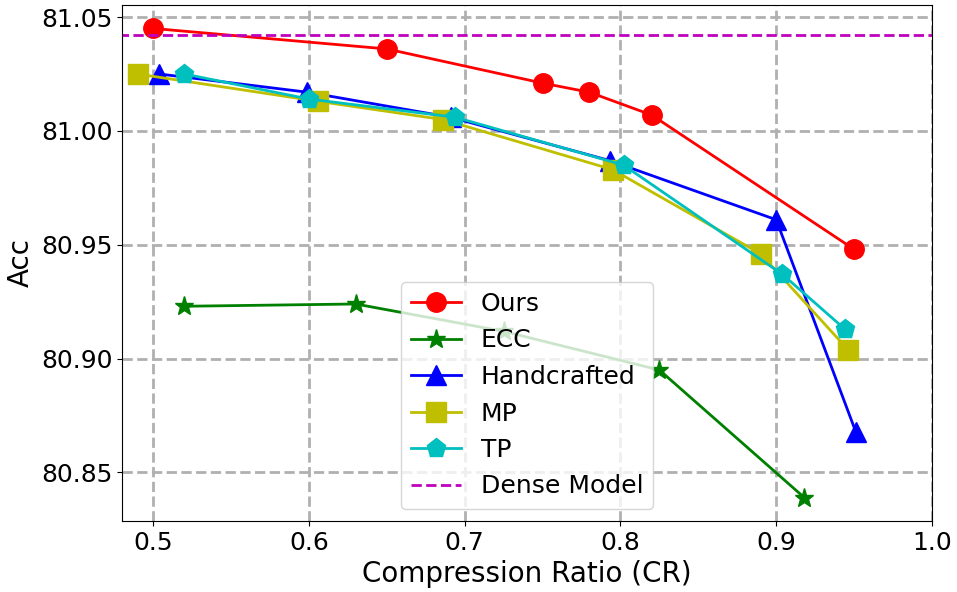


Figure 4.2: Results on RS prediction model compression by different methods on the Criteo AI Labs Ad Kaggle dataset. Reprinted from [2].

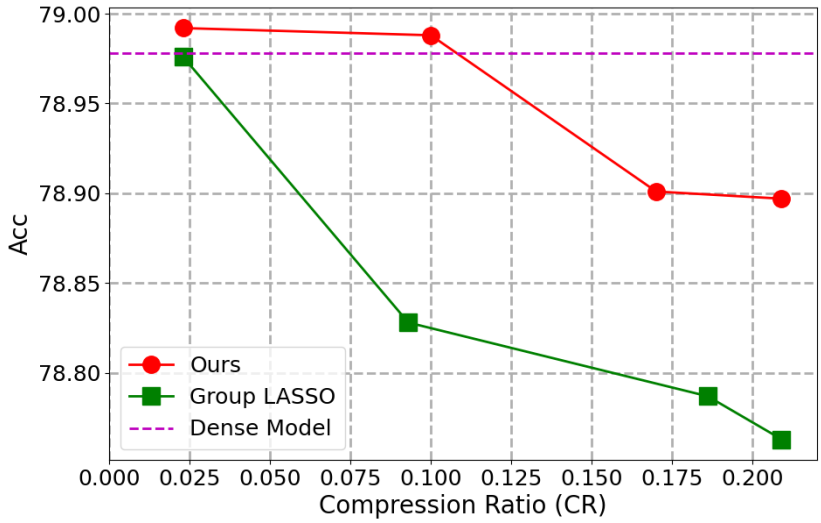


Figure 4.3: Results on RS input feature selection by different methods on the Criteo AI Labs Ad Kaggle dataset. Reprinted from [2].

4.3 Input Feature Selection For RS Model

We evaluate the effectiveness of our proposed method on input feature selection task. The input feature selection task can be easily realized using our proposed framework, by keeping the resource constraint for the features groups in input layer (*i.e.*, Eq. (5.1c)) and ignoring constraint for weights of hidden layers(*i.e.*, Eq. (3.3d)). We use Group Lasso as a self-designed baseline. The loss function of Group Lasso is shown in Eq. (3.1). Since we are only considering input feature selection in this sub-section, the sparsity regularization on hidden layers *(e.g.*, the second term in $\mathcal{S}(\boldsymbol{y}, \boldsymbol{s}, \mathcal{W}))$ is ignored. We tune hyper-parameter λ_1 among 0.0005, 0.001, 0.005, 0.01 for the Criteo AI Labs Ad Kaggle dataset, and 0.001, 0.003, 0.005 for the Terabyte dataset, in order to get different compression ratios. After Group Lasso training, all input features whose $\mathcal{L}_2$ norms are below a threshold th will be removed, and the Flops for the new model are calculated accordingly. We empirically find that setting th to 0.01 on the Kaggle dataset and 0.001 on the Terabyte dataset can achieve good performance. For both our method and Group Lasso, we train and fine-tune for 1 epoch and 2 epochs, respectively.

In Figure 4.3 and 4.4, the performance on the holdout validation set at the end of the compress-

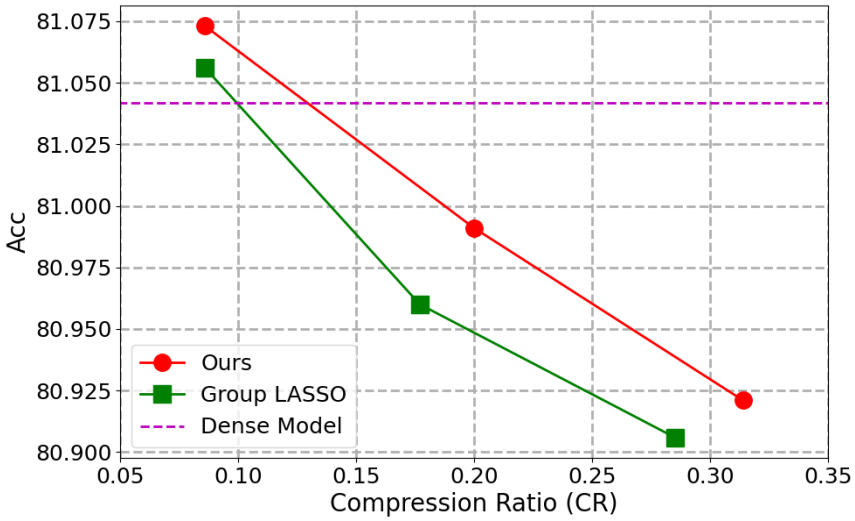


Figure 4.4: Results on RS input feature selection by different methods on the Criteo AI Labs Ad Terabyte dataset. Reprinted from [2].

ing process is reported. We demonstrate the effectiveness of our method in input feature selection task, by showing its superior performance under all the compress ratios. Another observation is that the pruned model of both methods will fail to recover the performance of the original dense model if too many input features are removed. For example, at a 0.21 compression ratio on the Kaggle dataset, which corresponds to removing 9 out of the total 27 input features, both methods have a considerable drop of accuracy. This indicates that reducing the number of input features will cause considerable harm to the model's performance.

4.4 Embedding Dimension Reduction For RS Model

We show that our method can also be applied to embedding dimension reduction. In this new scenario, $\mathcal{W}$ in Eq. (5.1) is embedding layer weights, instead of previous prediction model weights. Eq. (3.3d) is removed (*i.e.*, no longer to prune the input dimension). The rest parts of the objective function remain the same.

We compare our method with the state-of-the-art embedding dimension reduction method named mixed dimension embedding layers (MDEL) [8]. Following [8], we use size of the embedding layers to help define compression ratio. More specifically, in this setting, CR $= 1 -$

$p_s(\mathcal{W})/p_s(\mathcal{W}_0)$, where $p_s(\cdot)$ calculates the number of parameters in a (sub-)model. $\mathcal{W}$ and $\mathcal{W}_0$ are the pruned and original dense embedding sub-models, respectively. To evaluate performance change after pruning, we define $\Delta_{acc} = \text{Acc}_{\mathcal{W}} - \text{Acc}_{\mathcal{W}_0}$, where $\text{Acc}_{\mathcal{W}}$ and $\text{Acc}_{\mathcal{W}_0}$ are CTR prediction accuracy of a pruned and original dense model, respectively.

Method	CR	Δ_{acc}
MDEL	0.5	0.10%
Ours	0.5	0.24%
	0.6	0.16%

Table 4.1: Results on embedding dimension reduction. Reprinted from [2].

MDEL only provides results on the Criteo AI Labs Ad Kaggle dataset, so we conduct the comparison on this single dataset. Experiment results are shown in Table 4.1. As we can see, both methods have slight improvement in accuracy at around 50% CR, while the accuracy improvement of our method is larger than that of MDEL. Also, our method can still achieve better accuracy improvements than MDEL with even 10% larger CR. These results show that our ADMM based compression algorithm is better than MDEL.

4.5 Sparsity Analyses Of `UMEC`

Below we present how the sparsity of the prediction model evolves during the pruning phase in Section 4.2. As a recall, the aforementioned prediction model consists of one input layer and two hidden layers, with 27 input features, 512 and 256 neurons respectively. We take the Criteo AI Labs Ad Kaggle dataset as an example. As shown in Figure 4.5, when $R_{\text{budget}} = 0.5 \times p_f(\mathcal{W}_0)$, the pruned input features/neurons converge to 0, 102 and 250 respectively. During training, the last hidden layer gets the most percentage of neurons removed and the input layer gets the least percentage. We can come to several conclusions from this result. First, such phenomenon aligns with the observation in Section 4.3 and serves as a piece of evidence that we need enough input features to adequately represent the raw input information and contribute to the downstream classification

task. Second, to perform the relatively easier task, i.e., binary classification, the requirement of neurons from the last hidden layer falls far below the original design. Thus considerable resource consumption can be saved here. Third, it demonstrates that our resource-constrained unified optimization plays a smart role in globally finding the optimal resource allocations across all layers.

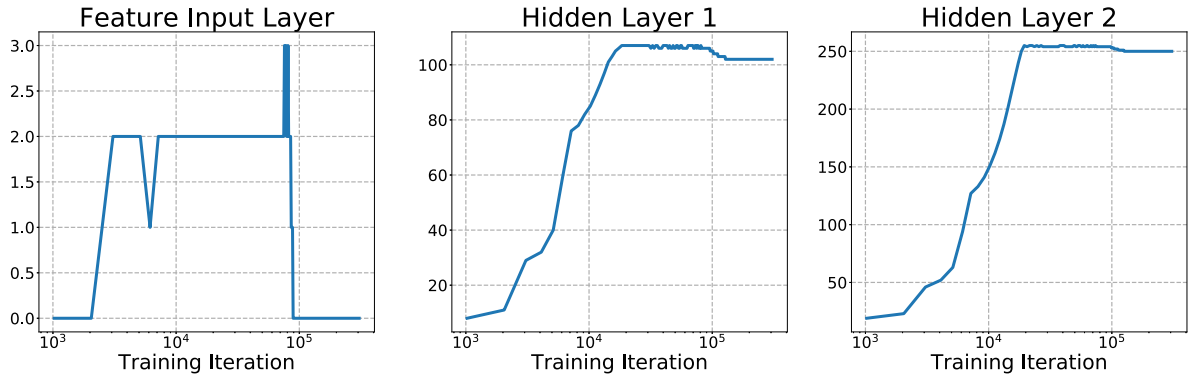


Figure 4.5: The number of pruned features/neurons for each layer during the training process using **UMEC** with $R_{\text{budget}} = 0.5 \times p_f(\mathcal{W}_0)$ on the Criteo AI Labs Ad Kaggle dataset, where $p_f(\mathcal{W}_0)$ denotes the Flops of the original dense model. For the input layer, the mentioned dense model has 27 features in total. For the two hidden layers, it has 512 and 256 neurons respectively. The convergency of the binary cross entropy (BCE) loss during training is shown in Figure 4.6 . Reprinted from [2].

4.6 Cascaded Pipeline As A Baseline

We provide experimental results under a cascaded pipeline using **UMEC**, where the compression processes for the input feature and the prediction model are carried out sequentially. Two scenarios are demonstrated: a) Conducting input feature selection first, then prediction model compression, of which the results are shown in Figure 4.7; b) Conducting embedding dimension reduction first, then prediction model compression, of which the results are shown in Figure 4.8. We set CR values of first stage in cascaded pipeline as 0.116, 0.116, 0.302, 0.302, 0.302 respectively for a), and 0.1, 0.1, 0.1 for b). Then we try our best to set the final CR values of the whole cascaded pipeline to be similar to the CR values used in our joint framework respectively for fair comparisons in

Figure 4.6: The convergency of the binary cross entropy (BCE) loss during training using **UMEC** with $R_{\text{budget}} = 0.5 \times p_f(\mathcal{W}_0)$, where $p_f(\mathcal{W}_0)$ denotes the Flops of the original dense model. Reprinted from [2].

both figures. As an example, we conduct the experimental comparisons on the Criteo AI Labs Ad Kaggle dataset.

Each curve shows the CR-Accuracy trade-off of one compression method. Comparing the results of the cascaded pipeline with the joint framework, Figure 4.7 and Figuire 4.8 both support the superiority of the joint framework.

4.7 Energy Consumption And Inference Latency Of The Network

We evaluate the energy cost and latency of all models on a GTX 2080 Ti GPU. Following [81], we use the `nvidia-smi` utility[3] to monitor the energy consumption. We follow the settings in [118] to measure inference latency on the aforementioned real-device. All experiments are implemented with PyTorch. As shown in Table 4.2, our method achieves the best accuracy with the least energy consumption and latency among all compared compression methods.

[3] `https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf`

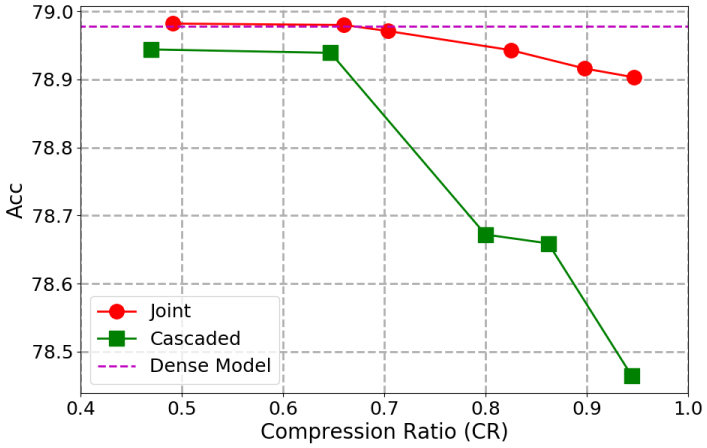


Figure 4.7: Results on the pipeline with a cascade of input feature selection and prediction model compression, and comparison with the corresponding jointly optimized framework. Reprinted from [2].

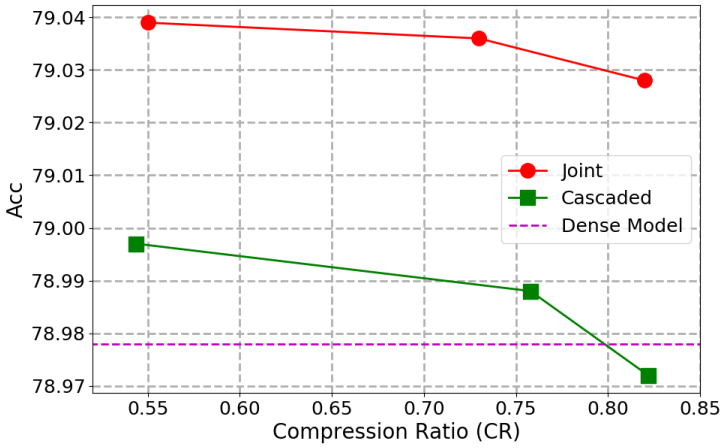


Figure 4.8: Results on the pipeline with a cascade of embedding dimension reduction and prediction model compression, and comparison with the corresponding jointly optimized framework. Reprinted from [2].

Method	Acc (%)	Energy (10^{-3}J)	Latency (ms)
Ours	**78.971**	**8.81**	**0.123**
ECC	78.939	9.05	0.132
Handcrafted	78.963	9.32	0.140
MP	78.956	9.16	0.136
TP	78.958	9.23	0.137
Dense Model	78.978	9.49	0.144

Table 4.2: Real-device energy consumption and latency of different methods. Best values among all compression methods are shown in bold. Reprinted from [2].

5. METHODOLOGY:UNIFIED COMPRESSION FOR VISION TRANSFORMER *

5.1 Preliminary

5.1.1 Vision Transformer (ViT) Architecture

To unfold the unified algorithm in the following sections, here we first introduce the notations. There are totally L transformer blocks. In each block l of the ViT, there are two constituents, namely the Multi-head Self-Attention module(MSA) and the MLP module. Uniformly, each MSA for transformer blcok l has H attention heads originally.

In the Multi-head Self-Attention module, $W_Q^{(l)}$, $W_K^{(l)}$, $W_V^{(l)}$ are the weights of the three linear projection matrix in block l that uses the block input X^l to calculate attention matrices: Q^l, K^l, V^l. The weights of the projection module that follows self-attention calculation is denoted as $W^{(l,1)}$, represents the first linear projection module in block l. The MLP module consists of two linear projection modules $W^{(l,2)}$ and $W^{(l,3)}$.

5.1.2 Compression Targets

The main parameters that can be potentially compressed in a ViT block are $W_Q^{(l)}$, $W_K^{(l)}$, $W_V^{(l)}$ and $W^{(l,1)}$, $W^{(l,2)}$, $W^{(l,3)}$. Our goal is to prune the head number and head dimensions simultaneously inside each layer, associated with the layer level skipping, solved in a unified framework. Currently, we do not extend the scope to reducing other dimensions such as input patch number or token size. However, our framework can also pack these parts together easily.

For head number and head dimensions pruning, instead of going into details of QKV computation, we innovate to use $\{W^{(l,1)}\}_{1\leq l\leq L}$ to be the proxy pruning targets. Pruning on these linear layers is equivalent to the pruning of head number and head dimension. We also add $\{W^{(l,3)}\}_{1\leq l\leq L}$ as our pruning targets, since these linear layers do not have dimension alignment issues with other parts, they can be freely pruned, while the output of $\{W^{(l,2)}\}$ should match with the dimension of

* This chapter is reprinted with permission from "Unified visual transformer compression" [3] by Yu, S., Chen, T., Shen, J., Yuan, H., Tan, J., Yang, S., ... Wang, Z. (2022). in International Conference on Learning Representations, Copyright 2022 held by the authors.

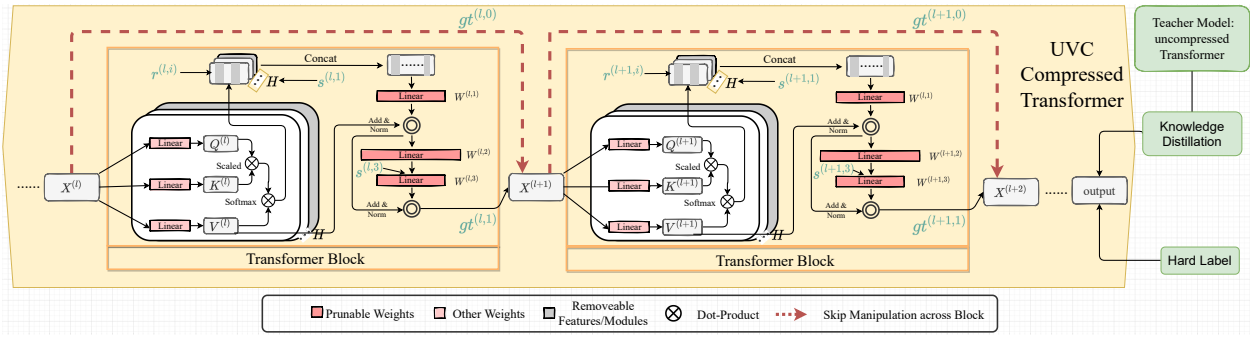


Figure 5.1: The overall framework of UVC. We seamlessly integrate Pruning within a block: In a transformer block, we targeting on pruning Self-Attention head numbers($s^{(l,1)}$), neuron numbers within a Self-Attention head($r^{l,i}$) and the hidden size of MLP module($s^{(l,3)}$) as well. Reprinted from [3].

block input. We do not prune $W_Q^{(l)}$, $W_K^{(l)}$, $W_V^{(l)}$ inside each head, since Q^l, K^l, V^l should be of the same shape for computing self-attention.

5.2 Resource-Constrained End-to-End ViT Compression

We target a principled constrained optimization framework jointly optimizing all weights and compression hyperparameters, i.e., structural pruning of linear projection modules in a ViT block. The full framework is illustrated in Figure 5.1.

Alternatively, the two strategies could be considered as enforcing **mixed-level group sparsity**: the head dimension level, the head number level, and the block number level. The rationale is: when enforced with the same pruning ratios, models that are under finer-grained sparsity (i.e., pruning in smaller groups) is unfriendly to latency, while models that are under coarser-grained sparsity (i.e., pruning in larger groups) is unfriendly to accuracy. The mixed-level group sparsity can hence more flexibly trade-off between latency and accuracy.

5.2.1 Pruning Within A Block

To compress each linear projection module on width, we first denote $s^{(l,3)}$ as the number of columns to be pruned for weights $W^{(l,3)}$. Compression for weights $W^{(l,1)}$ is more complicated as it is decided by two degrees of freedom. As the input tensor to be multiplied with $W^{(l,1)}$ is the direct output of attention heads. Hence, we denote $s^{(l,1)}$ to be the attention head numbers that

need to be pruned, and $r^{(l,i)}$ the number of output neurons to be pruned for each attention head i. Figure 5.2 illustrates the two sparsity levels: the head dimension level as controlled by $r^{(l,i)}$, and the head number level as controlled by $s^{(l,1)}$. They give more flexibility to transformer compression by selecting the optimal architecture in a more delicate and multi-grained way. Let us emphasize again that those variables above are not manually picked, but rather optimized globally.

5.2.2 The Constraints

We next formulate weight sparsity constraints for the purpose of pruning. As discussed in Sec 5.1, the target of the proposed method is to prune the head number and head dimension simultaneously, which can actually be modeled as a two-level group sparsity problem when choosing $\{W^{(l,1)}\}_{1\leq l\leq L}$ as proxy compression targets. Specifically, the input dimension of $\{W^{(l,1)}\}_{1\leq l\leq L}$ is equivalent to the sum of the dimensions of all heads. Then we can put a two-level group sparsity regularization on $\{W^{(l,1)}\}_{1\leq l\leq L}$ input dimension to compress head number and head dimension at same time, as shown in 5.1a and 5.1b. $r^{(l,i)}$ corresponding to the pruned size of i_{th} head. $s^{(l,1)}$ means the pruned number of heads. For $\{W^{(l,3)}\}_{1\leq l\leq L}$, it is our compression target, we just perform standard one level group sparsity regularization on its input dimension, as shown in 5.1c.

$$\sum_{j} \mathbb{I}\left(\left\|\boldsymbol{W}_{g_{ij},\cdot}^{(l,1)}\right\|_2^2 = 0\right) \geq r^{(l,i)}, \tag{5.1a}$$

$$\sum_{i} \mathbb{I}\left(\left\|\boldsymbol{W}_{g_{i},\cdot}^{(l,1)}\right\|_2^2 = 0\right) \geq s^{(l,1)}, \tag{5.1b}$$

$$\sum_{i} \mathbb{I}\left(\left\|\boldsymbol{W}_{i,\cdot}^{(l,3)}\right\|_2^2 = 0\right) \geq s^{(l,3)}, \tag{5.1c}$$

$$\forall\, l = 1, 2, ..., L, \forall\, i = 1, 2, ..., H \tag{5.1d}$$

where $\boldsymbol{W}_{i,\cdot}$ denotes the i-th column of $\boldsymbol{W}$; $\boldsymbol{W}_{g_i,\cdot}$ denotes the i-th grouped column matrix of $\boldsymbol{W}$, i.e. the i-th head; and $\boldsymbol{W}_{g_{ij},\cdot}$ the j-th column of $\boldsymbol{W}_{g_i,\cdot}$, which is the j-th column of the i-th head. In other words, in Eqn. 5.1b, column matrices are grouped by attention heads. Hence among the original

H heads in total at block l, at least $s^{(l,1)}$ heads should be discarded. Similarly, Eqn. 5.1c demands that at least $s^{(l,3)}$ input neurons should be pruned at this linear projection module. Furthermore, Eqn. 5.1a requests that in the i-th attention head at block l, $r(l,i)$ of the output units should be set to zeros.

Our method is formulated as a resource constrained compression framework, given a target resource budget, it will compress the model until the budget is reached.

Given a backbone architecture, the FLOPs is the function of $\boldsymbol{s}$, $\boldsymbol{r}$ and $\boldsymbol{gt}$, denoted as $\mathcal{R}_{\text{Flops}}(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{gt})$. We have the constraint as:

$$\mathcal{R}_{\text{Flops}}(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{gt}) \leq \mathcal{R}_{\text{budget}}, \tag{5.2}$$

where $\boldsymbol{s} = \{s^{(l,1)}, s^{(l,3)}\}_{1 \leq l \leq L}$ and $\boldsymbol{r} = \{r^{(l,i)}\}_{1 \leq l \leq L, 1 \leq i \leq H}$. $\mathcal{R}_{\text{budget}}$ is the resource budget. We present detailed flops computation equations in terms of $\boldsymbol{s}$, $\boldsymbol{r}$ and $\boldsymbol{gt}$ in Appendix.

Those inequalities can be further rewritten into equation forms to facilitate optimization. As an example, we follow [114] to reformulate Eqn. 5.1b as:

$$\sum_i \mathbb{I}\left(\|\boldsymbol{W}_{\cdot,g_i}\|_2^2 = 0\right) \geq s \Leftrightarrow \|\boldsymbol{W}_{\cdot,g}\|_{s,2}^2 = 0. \tag{5.3}$$

where $\|\boldsymbol{W}_{\cdot,g}\|_{s,2}^2$ denotes the Frobenius norm of the sub-matrix of $\boldsymbol{W}$ consisting of s groups of $\boldsymbol{W}$ with smallest group norms. Eqn. 5.1a and Eqn. 5.1c have same conversions.

5.2.3 The Objective

The target objective could be written as:

$$\min_{\boldsymbol{W}, \boldsymbol{gt}} \ \mathcal{L}(\boldsymbol{W}, \boldsymbol{gt}) = \ell(\boldsymbol{W}, \boldsymbol{gt}) \tag{5.4}$$

where $\boldsymbol{W}_t$ denotes the weights from teacher model, i.e., the uncompressed transformer model. We choose the simper ℓ_2 norm as its implementation, as its performance was found to be comparable to the more traditional K-L divergence [119].

5.2.4 The Final Unified Formulation

Summarizing all above, we arrive at our unified optimization as a mini-max formulation, by leveraging the primal-dual method [120]:

$$\min_{\boldsymbol{W},\boldsymbol{s},\boldsymbol{r},\boldsymbol{gt}} \max_{\boldsymbol{p},\boldsymbol{y},z\geq 0} \mathcal{L}_{\text{pruning}} = \min_{\boldsymbol{W},\boldsymbol{s},\boldsymbol{r},\boldsymbol{gt}} \max_{\boldsymbol{p},\boldsymbol{y},z\geq 0} \mathcal{L}(\boldsymbol{W},\boldsymbol{gt}) + \underbrace{z\left(R_{\text{Flops}}(\boldsymbol{s},\boldsymbol{r},\boldsymbol{gt}) - R_{\text{budget}}\right)}_{\text{resource loss}} + \underbrace{\sum_{l=1}^{L}\left(y^{(l,1)}\left\|\boldsymbol{W}_{\cdot,g}^{(l,1)}\right\|_{\lceil s^{(l,1)}\rceil,2}^{2} + y^{(l,3)}\left\|\boldsymbol{W}_{\cdot,\cdot}^{(l,3)}\right\|_{\lceil s^{(l,3)}\rceil,2}^{2}\right) + \sum_{l=1}^{L}\sum_{i=1}^{H} p^{(l,i)}\left\|\boldsymbol{W}_{\cdot,g_{i\cdot}}^{(l,1)}\right\|_{\lceil r^{(l,i)}\rceil,2}^{2}}_{\text{sparsity loss: } \boldsymbol{S}(\boldsymbol{y},\boldsymbol{s},\boldsymbol{p},\boldsymbol{r},\boldsymbol{gt},\boldsymbol{W})} \tag{5.5}$$

Eventually, we use the solution of the above mini-max problem, i.e. $\boldsymbol{s}$ and $\boldsymbol{r}$ to determine the compression ratio of each layer. Here, we select the pruned groups of the parameters by directly ranking columns by their norms, and remove those with the smallest magnitudes.

5.3 Solving The Unified Optimization

The general updating policy follows the idea of primal-dual algorithm.

5.3.1 Updating policy

5.3.1.1 Updating Weights

Different from other pruning methods in CNN that uses fixed pretrained weights to select the pruned channels/groups with certain pre-defined metrics [23, 121], here our subproblem could be considered as following a dynamic pruning criteria that will be updated along. Specifically we solve the following subproblem:

$$\text{Prox}_{\eta_1 \boldsymbol{S}(\boldsymbol{y},\boldsymbol{s},\boldsymbol{p},\boldsymbol{r},\boldsymbol{W})}(\bar{\boldsymbol{W}}) = \arg\min_{\boldsymbol{W}} \frac{1}{2}\left\|\boldsymbol{W} - \bar{\boldsymbol{W}}\right\|^2 + \eta_1 \boldsymbol{S}\left(\boldsymbol{y},\boldsymbol{s},\boldsymbol{p},\boldsymbol{r},\boldsymbol{W}\right), \tag{5.6}$$

where $\bar{\boldsymbol{W}} = \boldsymbol{W}^t - \eta_1 \hat{\nabla}_{\boldsymbol{W}} \ell\left(\boldsymbol{W}^t\right)$. The solution admits a bi-level projection [122]:

$$\boldsymbol{W}_{\cdot,g_{ij}}^{(l,1)*} = \begin{cases} \bar{\boldsymbol{W}}_{\cdot,g_{ij}}^{(l,1)}, & \text{if } \left\|\bar{\boldsymbol{W}}_{\cdot,g_{ij}}^{(l,1)}\right\|_2^2 \geq \left\|\boldsymbol{W}_{\cdot,g_{i\text{least-}\lceil r^{(l,i)}\rceil}}^{(l,1)}\right\|_2^2, \\ \frac{1}{1+2\eta_1 p^{(l,i)}}\bar{\boldsymbol{W}}_i, & \text{otherwise,} \end{cases}$$

$$\boldsymbol{W}^{(l,1)*}_{.,g_i} = \begin{cases} \bar{\boldsymbol{W}}^{(l,1)}_{.,g_i}, & \text{if } \left\| \bar{\boldsymbol{W}}^{(l,1)}_{.,g_i} \right\|_2^2 \geq \left\| \boldsymbol{W}^{(l,1)}_{.,g_{\text{least-}\lceil s^{(l,1)} \rceil}} \right\|_2^2, \\ \frac{1}{1+2\eta_1 y^{(l,1)}} \bar{\boldsymbol{W}}_{.,g_i}, & \text{otherwise,} \end{cases}$$

$$\boldsymbol{W}^{(l,3)*}_{.,i} = \begin{cases} \bar{\boldsymbol{W}}^{(l,3)}_{.,i}, & \text{if } \left\| \bar{\boldsymbol{W}}^{(l,3)}_{.,i} \right\|_2^2 \geq \left\| \boldsymbol{W}^{(l,3)}_{.,\text{least-}\lceil s^{(l,3)} \rceil} \right\|_2^2, \\ \frac{1}{1+2\eta_1 y^{(l,3)}} \bar{\boldsymbol{W}}_{.,i}, & \text{otherwise,} \end{cases}$$

where least-j denotes the index of the (group) columns of $\boldsymbol{W}$ that have j-th least (group) norm.

5.3.1.2 Updating t

$(\boldsymbol{gt}^{(l,0)}, \boldsymbol{gt}^{(l,1)})$ are used to generate a binomial categorical distribution to decide whether pass through block l or directly skip it. As the two variables are discrete, we apply the renowned `Gumbel-Softmax` (GSM) trick [123] to obtain differentiable and polarized sampling. For $\boldsymbol{gt}^l = (\boldsymbol{gt}^{(l,0)}, \boldsymbol{gt}^{(l,1)})$, given i.i.d Gumbel noise g drawn from $Gumbel(0,1)$ distribution, a soft categorical sample can be drawn by

$$G^l = GSM(\boldsymbol{gt}^l) = Softmax((log(\boldsymbol{gt}^l) + g)/\tau) \in \mathbb{R}^2, \tag{5.7}$$

where $G^{(l,1)}$ refers to the continuous possibility to preserve current block l while $G^{(l,0)}$ to drop it.

Directly applying the chain rule on $\mathcal{L}(\boldsymbol{W}, \boldsymbol{gt})$ w.r.t $\boldsymbol{gt}$ can now calculate $\tilde{\nabla}_{\boldsymbol{gt}} \mathcal{L}(\boldsymbol{W}^t, \boldsymbol{gt}^t)$.

Also, $\boldsymbol{gt}$ participates in the calculation of FLOPs, by deciding whether to pass certain blocks. Since passing or skipping a block is a dynamic choice during training, we estimate the FLOPs of the l-th block $\mathcal{R}_{\text{Flops}}(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{gt})$ with skip gating by using its expectation. To be specific,

$$\mathcal{R}_{\text{Flops}_l}(\boldsymbol{s}_l, \boldsymbol{r}_l, \boldsymbol{gt}_l) = \mathbb{E}[\mathcal{R}_{\text{Flops}_l}(\boldsymbol{s}_l, \boldsymbol{r}_l, \boldsymbol{gt}_l)|\boldsymbol{s}_l, \boldsymbol{r}_l] \tag{5.8a}$$

$$= G^{(l,0)}\mathcal{R}_{\text{Flops}_l}(Identity) + G^{(l,1)}\mathcal{R}_{\text{Flops}_l}(\boldsymbol{s}_l, \boldsymbol{r}_l) \tag{5.8b}$$

$$= G^{(l,1)}\mathcal{R}_{\text{Flops}_l}(\boldsymbol{s}_l, \boldsymbol{r}_l) \tag{5.8c}$$

Hence, updating policy for $\boldsymbol{gt}$ is formulated as:

$$\boldsymbol{gt}^{t+1} = \boldsymbol{gt}^t - \eta_4 \left(\tilde{\nabla}_{\boldsymbol{gt}}\mathcal{L}\left(\boldsymbol{W}, \boldsymbol{gt}^t\right) + \tilde{\nabla}_{\boldsymbol{gt}} z \left(R_{\text{Flops}}\left(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{gt}^t\right) - R_{\text{budget}}\right)\right) \tag{5.9}$$

$$= \boldsymbol{gt}^t - \eta_4 \left(\tilde{\nabla}_{\boldsymbol{gt}}\mathcal{L}\left(\boldsymbol{W}, \boldsymbol{gt}^t\right) + \tilde{\nabla}_{\boldsymbol{gt}} z R_{\text{Flops}}(\boldsymbol{s}, \boldsymbol{r}) GSM(\boldsymbol{gt}^t)\right) \tag{5.10}$$

5.3.1.3 Updating s and r

Similar to the updating policy of $\boldsymbol{gt}$, one gradient term w.r.t. $\boldsymbol{s}$ and $\boldsymbol{r}$ are $\tilde{\nabla}_{\boldsymbol{s}} z \left(R_{\text{Flops}}\left(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{gt}\right) - R_{\text{budget}}\right)$, $\tilde{\nabla}_{\boldsymbol{r}} z \left(R_{\text{Flops}}\left(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{gt}\right) - R_{\text{budget}}\right)$ respectively.

The other gradient term is calculated on the unified formulation Eqn. 5.5. Refer to that, $\boldsymbol{s}$ and $\boldsymbol{r}$ are floating-point numbers during the optimization process. In pratice, ceiling functions are operated on them to determine the integer number that should be pruned for each layer. However, the ceiling function $\lceil . \rceil$ is non-differentiable. To solve this problem, we implement Straight-through estimator(STE) [116] to provide a proxy of gradient when performing the backward pass. We set $\frac{\tilde{\partial}\lceil s \rceil}{\tilde{\partial} s} = 1$.

As for $\|\boldsymbol{W}_{.,g}\|_{s,2}^2$ term in the sparsity loss, we use $\|\boldsymbol{W}_{.,g}\|_{s+1,2}^2 - \|\boldsymbol{W}_{.,g}\|_{s,2}^2$ as the proxy of partial derivative of $\|\boldsymbol{W}_{.,g}\|_{s,2}^2$ with respect to s:

$$\frac{\tilde{\partial}\|\boldsymbol{W}_{.,g}\|_{s,2}^2}{\tilde{\partial} s} = \left\|\boldsymbol{W}_{.,g_{\text{least-min}\{\text{Dim}(\boldsymbol{W}), s+1\}}}\right\|_2^2, \tag{5.11}$$

where $\text{Dim}(\boldsymbol{W})$ is the number of column groups of $\boldsymbol{W}$. The other two terms in the sparsity

loss can be processed similarly.

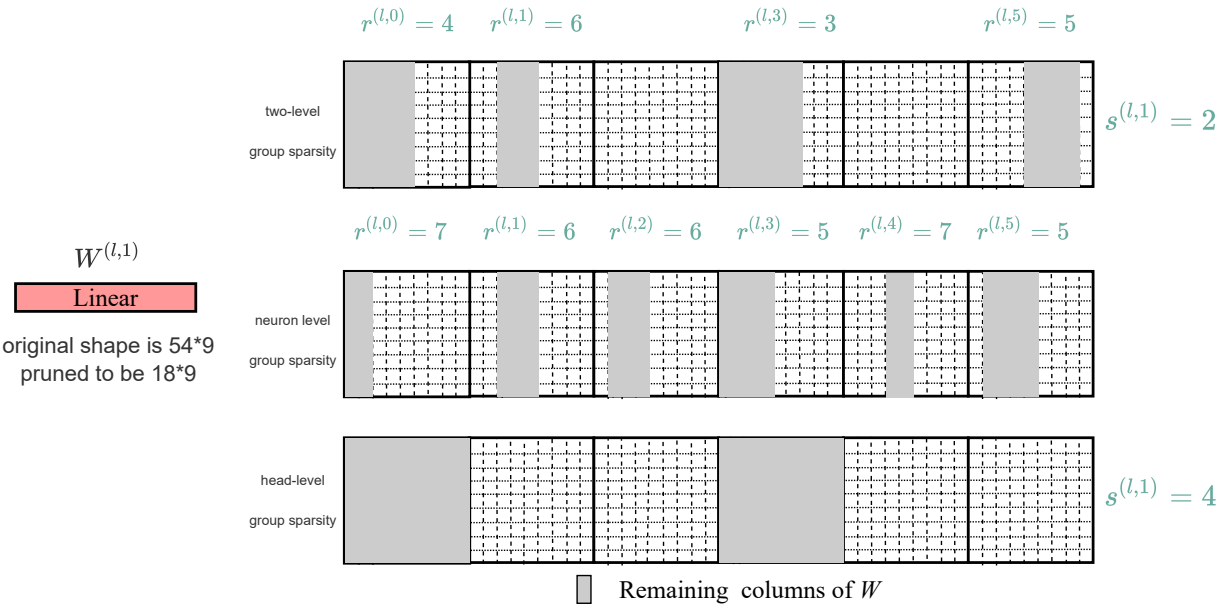


Figure 5.2: The two sparsity levels for pruning within a block: the head dimension level as controlled by $r^{(l,i)}$, and the head number level as controlled by $s^{(l,1)}$. When reaching same pruning ratio, neuron level sparsity will not remove any head, which is usually not friendly to latency; while head level sparsity will only remove head, which is usually not friendly to accuracy. Reprinted from [3].

5.3.2 Main Algorithm

The general updating policy follows the idea of primal-dual algorithm. The full algorithm is outlined in Algorithm 2.

Algorithm 2: Gradient-based algorithm to solve problem (5.5) for Unified ViT Compression.

Input: Resource budget R_{budget}, learning rates $\eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6$, number of total iterations τ.

Result: Transformer pruned weights $\boldsymbol{W}^*$.

Initialize $t = 1, \boldsymbol{W}^1$; `// random or a pre-trained dense model`

for $t \leftarrow 1$ **to** τ **do**

$$\boldsymbol{W}^{t+1} = \text{Prox}_{\eta_1 \boldsymbol{S}(\boldsymbol{y}^t, \boldsymbol{s}^t, \boldsymbol{p}^t, \boldsymbol{r}^t, \boldsymbol{W}^t)} \left(\boldsymbol{W}^t - \eta_1 \hat{\nabla}_{\boldsymbol{W}} \mathcal{L} \left(\boldsymbol{W}^t, \boldsymbol{gt} \right) \right) ; \quad \texttt{// Proximal-SGD}$$

$$\boldsymbol{s}^{t+1} = \boldsymbol{s}^t - \eta_2 \left(\tilde{\nabla}_{\boldsymbol{s}} \boldsymbol{S} \left(\boldsymbol{y}^t, \boldsymbol{s}^t, \boldsymbol{p}^t, \boldsymbol{r}^t, \boldsymbol{gt}^t, \boldsymbol{W}^{t+1} \right) + \tilde{\nabla}_{\boldsymbol{s}} z^t \left(R_{\text{Flops}} \left(\boldsymbol{s}^t, \boldsymbol{r}^t, \boldsymbol{gt}^t \right) - R_{\text{budget}} \right) \right) ;$$

`// Gradient (STE) Descent`

$$\boldsymbol{r}^{t+1} = \boldsymbol{r}^t - \eta_3 \left(\tilde{\nabla}_{\boldsymbol{r}} \boldsymbol{S} \left(\boldsymbol{y}^t, \boldsymbol{s}^{t+1}, \boldsymbol{p}^t, \boldsymbol{r}^t, \boldsymbol{gt}^t, \boldsymbol{W}^{t+1} \right) + \tilde{\nabla}_{\boldsymbol{r}} z^t \left(R_{\text{Flops}} \left(\boldsymbol{s}^{t+1}, \boldsymbol{r}^t, \boldsymbol{gt}^t \right) - R_{\text{budget}} \right) \right) ;$$

`// Gradient (STE) Descent`

$$\boldsymbol{gt}^{t+1} = \boldsymbol{gt}^t - \eta_4 \left(\tilde{\nabla}_{\boldsymbol{gt}} \mathcal{L} \left(\boldsymbol{W}^{t+1}, \boldsymbol{gt}^t \right) + \tilde{\nabla}_{\boldsymbol{gt}} z^t \left(R_{\text{Flops}} \left(\boldsymbol{s}^{t+1}, \boldsymbol{r}^{t+1}, \boldsymbol{gt}^t \right) - R_{\text{budget}} \right) \right) ;$$

`// Gradient Descent`

$$z^{t+1} = z^t + \eta_7 \left(R_{\text{Flops}} \left(\boldsymbol{s}^{t+1}, \boldsymbol{r}^{t+1}, \boldsymbol{gt}^{t+1} \right) - R_{\text{budget}} \right) ; \quad \texttt{// Gradient Ascent}$$

$$y^{(l,1)^{t+1}} = y^{(l,1)^t} + \eta_5 \left(\left\| \boldsymbol{W}_{:,g}^{(l,1)^{t+1}} \right\|^2_{\left\lceil s^{(l,1)^{t+1}} \right\rceil, 2} \right) ; \quad \texttt{// Gradient Ascent}$$

$$y^{(l,3)^{t+1}} = y^{(l,3)^t} + \eta_5 \left(\left\| \boldsymbol{W}_{:,:}^{(l,3)^{t+1}} \right\|^2_{\left\lceil s^{(l,3)^{t+1}} \right\rceil, 2} \right), \ \forall\ l = 1, \cdots, L$$

$$p^{(l,i)^{t+1}} = p^{(l,i)^t} + \eta_6 \left(\left\| \boldsymbol{W}_{:,g_i:}^{(l,1)^{t+1}} \right\|^2_{\left\lceil r^{(l,i)^{t+1}} \right\rceil, 2} \right), \ \forall\ i = 1, \cdots, H, \ \forall\ l = 1, \cdots, L$$

end

$\boldsymbol{W}^* = \boldsymbol{W}$

6. EXPERIMENTS:UNIFIED COMPRESSION FOR VISION TRANSFORMER *

6.1 Datasets And Benchmarks

We conduct experiments for image classification on ImageNet [12]. We implement UVC on DeiT [41], which has basically the identical architecture compared with ViT [19] except for an extra distillation token. Experiment has been conducted on DeiT-Tiny. We measure all resource consumptions (including the UVC resource constraints) in terms of inference FLOPs.

6.2 Training Settings

The whole process of our method consists of two steps.

- Step 1: UVC training. We firstly conduct the primal-dual algorithm to the pretrained DeiT model to produce the compressed model under the given resource budget.
- Step 2: Post training. When we have the sparse model, we finetune it for another round of training to regain its accuracy loss during compression.

In the two steps mentioned above, we mainly follow the training settings of DeiT [41] except for relatively smaller learning rate which benefits finetuning of converged models.

Numerically, the learning rate for parameter z is always changing during the primal-dual algorithm process. Thurs, we propose to use a dynamic learning rate for the parameter z that controls the budget constraint. We use a four-step schedule of $\{1, 5, 9, 13, 17\}$ in practice.

6.3 Baseline Methods

We adopt several latest compression methods specifically developed for ViTs (all proposed only in the past four months), which fall under two categories:

* This chapter is reprinted with permission from "Unified visual transformer compression" [3] by Yu, S., Chen, T., Shen, J., Yuan, H., Tan, J., Yang, S., ... Wang, Z. (2022). in International Conference on Learning Representations, Copyright 2022 held by the authors.

- *Category 1: Input Patch Reduction*, including (i) **HVT** [124]: which designs efficient architectures by reducing the spatial dimensions using max pooling hierarchically. (ii) **PatchSlimming** [40]: which identifies the effective patches in the last layer and then use them to guide the patch selection process of previous layersk.

- *Category 2: Model Weight Pruning*, including (iii) **SViTE** [38]: which jointly optimizes model parameters and explores sparse connectivity throughout training, ending up with one final sparse network. SViTE belongs to the most competitive accuracy-efficiency trade-off achieved so far, for ViT pruning. We choose its structured variant to be fair with UVC.

6.4 Main Results

The full comparison result are listed in Tab. 6.1. Firstly, we notice that most existing methods cannot save beyond 50% FLOPs without sacrificing too much accuracy. In comparison, UVC can easily go with **larger compression rates** (up to $\geq$ 60% FLOPS saving) without compromising as much, showing stronger promise for resource-constrained ViT applications. For example, when compressing DeiT-Tiny (with distillation token), UVC can trim the model down to a compelling $\geq$50% of the original FLOPs while losing only 0.4% accuracy. Compared with the latest SViTE [38] that can only save up to around 30% FLOPs, we observe UVC to **significantly outperform** at DeiT-Tiny, obtaining less accuracy drops (0.9%, versus 2.1%), at much aggressive FLOPs savings (50.7%, versus 23.7%).

Secondly, On other models, we observe UVC to generally save more FLOPs, yet also sacrificing more accuracies. Moreover, as we explained in Section 3.1, those input token reduction methods represent an orthogonal direction to the model weight sparsification way that UVC is pursuing. UVC can also be seamlessly extended to include token reduction into the joint optimization - a future work that we would pursue. UVC also outperformed recent strong competitors such as HVT [124] on DeiT-Tiny.

Model	Method	Top-1 Acc. (%)	FLOPs(G)	FLOPs remained(%)
DeiT-Tiny	Baseline	72.2	1.3	100
	SViTE	70.12(-2.08)	0.99	76.31
	PatchSlimming	72.0 (-0.2)	0.7	53.8
	UVC	71.8 (-0.4)	0.69	53.1
	HVT	69.7 (-2.5)	0.64	49.23
	UVC	71.3 (-0.9)	0.64	49.23
	UVC	70.6 (-1.6)	0.51	39.12

Table 6.1: Comparison of the vision transformers compressed by UVC with different benchmarks on ImageNet. FLOPs remained denotes the remained ratio of FLOPs to the full-model FLOPs. Reprinted from [3].

7. METHODOLOGY: LEARNING A MINIMAX OPTIMIZER *

7.1 Main Framework: Twin Learnable Optimizers (Twin-L2O)

The minimax objective in Section 2 is solved by alternative gradient descent, of which the effectiveness is questionable. As we discussed in Section 1.2, We would like to resort to learning to optimize (L2O) technique to solve the minimax objective, and explore to what extent the meta-learned optimizers can be adapted to a special class of functions and outperform general-purpose optimizers for minimax problems.

The main L2O framework we proposed is named *Twin-L2O*, where we use two learnable optimizers to alternate between min and max updates. See Figure 7.1. Our design adopts the basic idea of [4] to use Long Short-Term Memory (LSTM) to model learnable *optimizers*, for solving target problems known as *optimizees*. At each step, LSTM outputs the update of the optimizee variables. The LSTM inputs are typically the current zero-order or first-order information of the optimizee [4, 125], plus the historic optimization trajectory information.

In Twin-L2O, two LSTMs separately update x and y and record historical trajectory information of their own variables respectively. Formally, we consider the minimax problem $\min_x \max_y f(x, y)$. We use two LSTM optimizers, LSTM-Min and LSTM-Max, to updates the min variable x and the max variable y respectively. LSTM-Min is parameterized by $\phi^{\min}$ and LSTM-Max is parameterized by $\phi^{\max}$. At each iteration t, Twin-L2O updates x and y in turns and yields the following rule:

$$\begin{aligned} x_{t+1} =& x_t + \Delta x_t, \text{where } (\Delta x_t, h_{t+1}^{\min}) = \text{LSTM-Min}\Big([\nabla_x f(x_t, y_t), \nabla_y f(x_t, y_t)], h_t^{\min}, \phi^{\min}\Big), \\ y_{t+1} =& y_t + \Delta y_t, \text{where } (\Delta y_t, h_{t+1}^{\max}) = \text{LSTM-Max}\Big([\nabla_y f(x_{t+1}, y_t), \nabla_x f(x_{t+1}, y_t)], h_t^{\max}, \phi^{\max}\Big), \end{aligned} \tag{7.1}$$

where $h_t^{\min}$ and $h_t^{\max}$ are the historical trajectory information of LSTM-Min and LSTM-Max at time step t.

* This chapter is reprinted with permission from "Learning a minimax optimizer: A pilot study " [2] by Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., Wang, Z. (2020, September) in International Conference on Learning Representations, Copyright 2020 held by the authors.

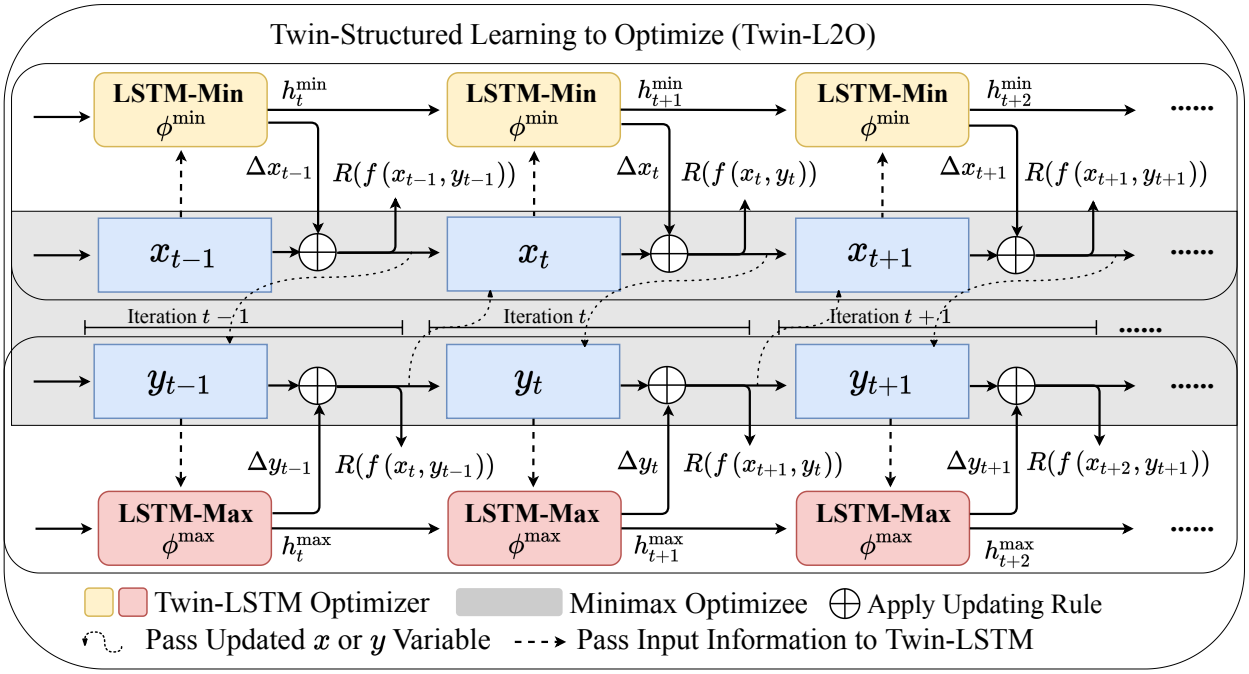


Figure 7.1: Architecture of Twin-L2O. We let LSTM-Min and LSTM-Max, parameterized by ϕ^{min} and ϕ^{max}, update x and y respectively. As shown by curved dashed lines, Twin-LSTM keeps being updated about the latest variable values of x and y when computing input information and the reward. When constructing the computational graph and training the Twin-LSTM, the solid lines allow gradients to flow while the dashed lines do not pass any gradient [4]. Reprinted from [5].

This formulation is inspired by the SimGD/GDA-style algorithms [52, 102–104] that conduct simultaneous/alternative gradient descent over x and ascent over y.

The next question is to design the L2O reward. To train the LSTM optimizers, the loss function is often to penalize some type of cost, accumulated along the optimization trajectory for a horizon of T steps (also known as the unrolling length for LSTM [126])

$$\mathcal{L}(\phi^{\min}, \phi^{\max}) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t R(f, x_t, y_t) \right], \tag{7.2}$$

w_t is chosen to be all 1 following the basic setting in [4], that might be tuned for better performance in future work.

As a key design option, $R(f)$ represents the reward to guide the L2O training. In existing L2O methods for continuous minimization [4, 107], $R(f)$ is usually simply set to $R(f, x_t)) = f(x_t)$

to encourage fast decrease of objective values over time. To extend this existing reward to the minimax scenario, we cannot directly penalize the overall objective function value either way, since the min and max objectives are entangled. Also, different from pure minimization problems, the Twin-L2O updates (7.1) consist of two alternating steps governed by two different LSTM optimizers: each accounts for its own subproblem goal (min or max updates), but the two also have to collaborate to explore/exploit the minimax landscape. We specifically design the following reward that implicitly addresses the above issue by setting a new reward function:

$$\mathcal{L}(\phi^{\min}, \phi^{\max}) = \mathbb{E}_f \left[\sum_{t=1}^{T} \{[f(x_t, y_{t-1}) - f(x_t, y_t)] + [f(x_t, y_{t-1}) - f(x_{t-1}, y_{t-1})]\} \right]. \quad (7.3)$$

7.1.1 Analysis Of The Reward Design

In Eqn. 7.3, the first and second terms always characterize two consecutive min and max updates. In more details, the value of $f(x_t, y_t) - f(x_t, y_{t-1})$ solely reflects how effectively the t-step max update increases the objective f , while $f(x_t, y_{t-1}) - f(x_{t-1}, y_{t-1})$ reflects the effectiveness of t-step min update in decreasing the objective f. Our goal is then to maximize the weighted accumulated sum for $f(x_t, y_t) - f(x_t, y_{t-1})$, while minimizing the weighted accumulated sum for $f(x_t, y_t) - f(x_t, y_{t-1}), t = 1, 2, ..., T$. Combining the two sub-goals together (with a sign change to turn max into min) yields our reward. One may also alternatively interpret Eqn. 7.3 as penalizing the loss change from $f(x_t, y_t)$ along both x and y updating directions, which would encourage yielding stationary points.

We term the reward function in Eqn. 7.3 as an *objective-based* reward, since it penalizes the objective change from $f(x_t, y_t)$ along both x and y updating directions. It is naturally inherited and extends the reward functions prevailing in most prior L2O works for minimization [4, 60], whose default reward is to minimize a weighted sum of the past function values.

One may also design the following two rewards, which we name as *gradient-based* rewards:

$$\mathcal{L}(\phi^{\min}, \phi^{\max}) = \mathbb{E}_f \left[\sum_{t=1}^{T} \left\| \nabla_x f\left(x_t, y_t\right) \right\|^2 + \left\| \nabla_y f\left(x_t, y_t\right) \right\|^2 \right], \tag{7.4}$$

$$\mathcal{L}(\phi^{\min}, \phi^{\max}) = \mathbb{E}_f \left[\sum_{t=1}^{T} \left(\frac{f\left(x_t, y_{t-1}\right) - f\left(x_t, y_t\right)}{\left\| y_t - y_{t-1} \right\|} \right)^2 + \left(\frac{f\left(x_{t-1}, y_{t-1}\right) - f\left(x_t, y_t\right)}{\left\| x_t - x_{t-1} \right\|} \right)^2 \right]. \tag{7.5}$$

Eqn. 7.5 is the gradient-based Nikaido-Isoda function introduced by [127].

For minimax optimization, it is not immediately clear whether the objective-based or the gradient-based might work practically better. Intuitively by definition, the former is likely to lead towards a saddle point (defined in Eqn. 1.1) and the latter to a stationary point. They do not always coincide in general, e.g, a stationary point might not be a saddle point. But for all specific test problems we studied in this dissertation, a stationary point is also a saddle point.

We try several experiments on the challenging seesaw problem as a specific example, to provide a close comparison between the gradient-based in Eqn. 7.4 and the objective-based reward. We re-do Twin-L2O by only replacing Eqn. 7.3 with the gradient-based reward, and our observations are: a) the gradient-based reward solves the seesaw problem worse than the objective-based one; b) the minimization variable x diverges on testing problem instances; c) the maximization variable y will converge to a solution of precision magnitude 0.04 (for reference, y converges to have magnitude less than 0.01 when using the objective-based loss). We further identify one possible cause after analyzing the gradient behaviors. Note here the gradient-based reward could be expressed as:

$$||\nabla_x f(x,y)||^2 + \|\nabla_y f(x,y)\|^2 = a^2 b^2 \pi^2 y^2 \cos^2(a\pi x) + b^2 \sin^2(a\pi x) \tag{7.6}$$

Because $a, b \sim U[0.9, 1]$, the first term often dominates during training due to the π^2 multiplier, unless y is sufficiently close to zero. The imbalance could be a cause of instability. For example, this reward could sometimes penalize $\cos^2(a\pi x)$ to be close to zero, which is the opposite direction of the true solution $\sin^2(a\pi x) = 0$. Although this is just a very specific problem example, it reveals that the gradient-based loss may sometimes not work well as expected, due to the instability or asymmetry of min/max gradients.

Besides, we have also tried the second gradient-based reward in Eqn. 7.5, and find it ineffective. It is mainly because the denominator (consecutive variable differences) can become very small and the loss will then explode and break training.

Back to the objective-based reward used in this dissertation, we have not observed oscillation empirically from all experiments so far. Our hypodissertation is that the recurrent structure of the proposed Twin-L2O framework (shown in Eqn. 7.1) plays a role here. Although we use two LSTMs for the min and max updates respectively, the LSTM of one variable actually takes in the information of the other LSTM implicitly, because it takes the output of the other as input. When we penalize the objective function value of one LSTM update, all previous min and max updates can (in principle) be taken into account due to the effect of unrolled back-propagation, e.g., the min and max updates each take reference to not only its own, but also the other's higher-order past trajectory information. While this is a tentative explanation, we think more in-depth analysis of why oscillation may or may not happen in L2O could be a really interesting future work.

Another implicit intuition that leads us to prioritizing the use of objective-based over gradient-based is that, in classic minimization, objective change is summable (i.e., having a finite accumulation), but gradient change is not summable in general (unless with properties such as strong convexity). While summability is itself not a guarantee for good training/testing performance, lack of summability means the loss may have an overly large dynamic range.

To summarize, our function-based objective naturally extends previous L2O convention, works better than other alternatives, and observes no oscillation yet. However, we emphasize that there is no intention to claim the current reward in Eqn. 7.3 is the best choice for minimax L2O - it is one of plausible options. We do concur the gradient-based reward designs in Eqn. 7.4 and Eqn. 7.5 are a complicated yet interesting question, especially when considering more complicated minimax problems. Again, as this dissertation is intended only as the first work and pilot study towards understanding the profound challenges and rich possibilities for minimax L2O, we believe everything discussed and proposed here, including the loss function, has large room of improvement.

7.1.2 Rationale Of The Framework Selection

Another important design question is *to what extent learning the min and max updates should be (dis)entangled*: on the one hand, the two steps obviously interact with each other as they jointly explore the minimax landscape; on the other hand, min and max steps commonly have asymmetric difficulty levels, that have been leveraged by previous algorithms. For example, [106] demonstrates the failure of alternating gradient descent in minimax optimization due to the multiple solution discontinuity of the inner maximization, and addresses that by simultaneously tracking K candidate solutions for the max step, while the outer minimization remains to take one descent step. Besides the joint reward (7.3), the default Twin-L2O design leverages two independent LSTMs in Eqn. (7.1), each dedicatedly handling min or max updates. In comparison, we also consider two other more "entangled" ways: (a) fully entangling the two optimizers, i.e. using one LSTM to simultaneously generate min and max outputs; (b) weakly entangling the two optimizers, by using two LSTMs sharing weights, yet allowing either to maintain its own temporal hidden states. Our ablation experiments (see Section 8.1) find that the default decoupled design in Eqn. (7.1) seems to facilitate the L2O learning most.

7.2 Improving Generalizability Of Twin-L2O

Despite the empirical success of L2O, it is unfortunately impossible to ensure that any L2O algorithm always converges. Assuming the objective function type to keep unchanged, the testing instances' parameter distribution may differ from the one of training, and L2O can catastrophically fail. For the Twin-L2O, we discuss two remedies to partially fix this issue and boost its generalizability.

We first propose *curriculum L2O training scheme* as a practical L2O training technique such that Twin-L2O can be trained to work on a much wider coverage of problem parameters than its vanilla versions. That would empirically help the generalizability due to broader coverage by training instance, but would still inevitably fail when meeting unseen testing instances. We then present a preliminary exploration of the safeguard mechanism on minimax under a special case,

i.e., solving convex-concave problems. We demonstrate that with such strong assumptions, it is possible to theoretically establish the "perfect" convergence of Twin-L2O on *any unseen optimizee*.

7.2.1 Curriculum L2O Training

When it comes to general minimax problems, it is unlikely to exist an ideal theory to fully ensure Twin-L2O convergence *on all instances*. Therefore, we seek empirical L2O success *of as many instances as possible*. Specifically: *can we train Twin-L2O better, so that it can work on instances at a broader parameter range*?

We find a *curriculum learning* (**CL**) strategy [128] particularly useful. CL was first adopted to train neural networks by first focusing the training on an "easy" training subset (often adaptively selected), that is then gradually grown to the full set. It is known to be effective to stabilize training, especially when the training set is highly varied or noisy [129]. Since minimax optimization is notoriously unstable no matter via analytical or learned optimizers, we conjecture that the noisy minimax dynamics might challenge Twin-L2O by providing unreliable guidance and impede its training. Considering that our Twin-L2O is modeled using LSTMs, it is natural to think of whether CL can bring additional gains if applied to meta-training. Previously it was also found effective in L2O for minimization problems [110].

Specifically, in one epoch, we will rank all optimizee instances by their cumulative losses (7.2) from low to high, and only select the top C instances to count into the total reward. In that way, only the instances that exhibit "good training behaviors" (smaller gradients & more likely to get close to stationary points) will be initially used for updating the Twin-L2O. That prevents the learned optimizer being misled by random failures and outliers, which are commonly found in the early epochs of Twin-L2O training. We by default set the percentage C to start from 20%, then growing linearly every epoch until reaching 100% in the later training stage.

Up to our best knowledge, this is the first effort to incorporate CL with L2O training. We can this Twin-L2O trained with CL as **Enhanced Twin-L2O**: note that it is the same model structure, just trained in a different and better way. More details are provided below.

Initialize $\boldsymbol{u}^1 \in \mathbb{R}^n$, $C \in [0, \infty)$, $\alpha \in (0, \infty)$, $\{\lambda_\ell\} = \{1/(\ell+1)\}$, $k \leftarrow 2$, weights $\{\phi^k\}$
function SADDLEHALPERN(ε)
    $\boldsymbol{u}^2 \leftarrow \frac{1}{2}\left(\boldsymbol{u}^1 + J_{\alpha\partial f}(\boldsymbol{u}^1)\right)$.
    while $\|\boldsymbol{u}^k - J_{\alpha\partial f}(\boldsymbol{u}^k)\| > \varepsilon$ **do**
        $\boldsymbol{z}^{k+1} \leftarrow \mathrm{LSTM}(\boldsymbol{u}^k;\ \phi^k)$
                                ◁ *Apply L2O operator*
        if $E_{k+1}(\boldsymbol{u}^{k+1}) \leq \dfrac{C}{k+1}$
                                ◁ *Verify safeguard condition*
            $\boldsymbol{u}^{k+1} \leftarrow \boldsymbol{z}^{k+1}$
                                ◁ *Use L2O update*
        else
            $\boldsymbol{u}^{k+1} \leftarrow \lambda_k \boldsymbol{u}^1 + (1-\lambda_k) J_{\alpha\partial f}(\boldsymbol{u}^k)$
                                ◁ *Use fallback update*
        $k \leftarrow k+1$
    return $\boldsymbol{u}^k$
end function

Figure 7.2: Method for Safeguarded-Twin-L2O for Convex-Concave Saddle Point Problems. Adapted from [5].

In L2O framework, the reward for training the optimizer is defined as:

$$\mathcal{L}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t R\left(f\left(x_t\right)\right) \right] \tag{7.7}$$

where f is a distribution of functions. The **Enhanced Twin-L2O** using Curriculum Learning(CL) selects a portion of instances that demonstrate "good training behaviors" (smaller gradients & more likely to get close to stationary points) to be counted into the reward, with the portion C increasing linearly from 20% to 100% as the training epoch increases. In our experiments, the detailed scheme of C is:

$$C = min\{20 + \text{epoch\_index}, 100\}\% \tag{7.8}$$

where epoch_index denotes the index of epoch when training, starting from 0 and ending with 199

in our case. When applying CL, the actual reward becomes

$$\tilde{\mathcal{L}}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t \boldsymbol{q}(\boldsymbol{f}) R\left(f\left(x_t\right)\right) \right] \tag{7.9}$$

where $q(f) = 1$ if the value $m(f) = \sum_{t=1}^{T} w_t \left\|\nabla_y f\left(x_t, y_t\right)\right\|^2$ ranks top C of all sampled functions, and $q(f) = 0$ otherwise.

This process does not change the structure of Twin-L2O, and essentially acts as adding masks to those training instances that demonstrate poor behavior and ignoring them in the actual training phase. Combining this trick with the existing framework, the Twin-L2O can achieve a higher success rate when solving problems with a larger range of parameters.

7.2.2 Safeguard Twin-L2O: A Preliminary Theoretical Exploration

Most L2O methods have little or no convergence guarantees. Very recently, a safeguarding mechanism has been introduced to L2O for convex minimization problems with gradient and/or proximal oracles [61]. Conceptually, a safeguard is anything that identifies when a "bad" L2O update would occur and what "fallback" update to apply in place of that bad L2O update. In this section, we establish a safeguarding theory and algorithm, specifically for *learned convex-concave saddle point algorithms*. Here the safeguard takes the form of an energy inequality (c.f. Line 6 in Method 7.2).

In this section, we write $\boldsymbol{u} = (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^m \times \mathbb{R}^n$ and let $\alpha > 0$. We use the *resolvent*, defined by

$$J_{\alpha \partial f}(\boldsymbol{x}, \boldsymbol{y}) = (\mathrm{Id} + \alpha \partial f)^{-1}, \tag{7.10}$$

where we note $\partial f = (\partial_{\mathbf{x}} f, -\partial_{\mathbf{y}} f)$. For simple f (e.g., quadratic functions), a closed formula exists for $J_{\alpha \partial f}$. Otherwise, one may use an iterative method to approximate this quantity. In addition, define the residual operator

$$F(\boldsymbol{u}) := \frac{1}{2} \left(\boldsymbol{u} - J_{\alpha \partial f}(\boldsymbol{u})\right), \tag{7.11}$$

and, for each $k \in \mathbb{N}$, the energy $E_k : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ by

$$E_k(\boldsymbol{u}) := \|F(\boldsymbol{u})\|^2 - \frac{\lambda_k}{1-\lambda_k} \langle F(\boldsymbol{u}), \boldsymbol{u}^1 - \boldsymbol{u} \rangle , \tag{7.12}$$

where $\{\lambda_k\}$ is a sequence of step sizes. The full method is outlined in the **Method 7.2**, where the L2O update is denoted by $\mathrm{LSTM}(\boldsymbol{u}^k; \phi^k)$ and the fallback method is a Halpern iteration [130]. Our main result for minimax safeguarding theory is formally stated below:

Theorem 7.2.1. *If the sequence $\{\boldsymbol{u}^k\}$ is generated by Algorithm 7.2, then*

$$\|\boldsymbol{u}^k - J_{\alpha\partial f}(\boldsymbol{u}^k)\| \leq \frac{1}{2}\left(\frac{d_1}{k} + \sqrt{\frac{d_1^2}{k^2} + \frac{4C}{k}}\right), \quad \textit{for all } k \geq 2, \tag{7.13}$$

where $d_1 := \min\{\|\boldsymbol{u} - \boldsymbol{u}^1\| : 0 \in \partial f(\boldsymbol{u})\}$ is the distance from the initial iterate $\boldsymbol{u}^1$ to the set of saddle points and $C \geq 0$ is an arbitrary constant. In particular, this implies each limit point of $\{\boldsymbol{u}^k\}$ is a saddle point.

Our proof draws and integrates two sources of ideas: (1) the safeguarded L2O technique that has recently just been introduced to convex minimization [61]; and (2) Halpern iteration [64] that is adopted for analytical minimax optimization with favorable theoretical properties. The full proof is provided below. Note that this work is **not intended** as a theory innovation on (classical) minimax optimization. Instead, our aim is to extend the emerging idea of safeguarded L2O from convex minimization to convex-concave minimax problems of interest, and shows this idea to be helpful for minimax L2O too: see experiments section.

8. EXPERIMENTS: LEARNING A MINIMAX OPTIMIZER *

8.1 Ablation Study On The Design Of Twin-L2O

We first investigate the design choices for Twin-L2O that we discussed in Section 7.1. We mainly investigate two aspects: *(i)* whether to share the weights in the two LSTM solvers or not; *(ii)* whether to share the hidden states between the two LSTM solvers or not. That leads us to four options, denoted as (with self-explanatory names): Share-LSTM-Share-Hidden, Share-LSTM-Two-Hidden, Two-LSTM-Share-Hidden, and Two-LSTM-Two-Hidden. We use the seasaw problem, formulated as below, as the testbed for our ablation study (note that the ranges of a, b are picked only to make L2O easy to converge, while more will be investigated later):

$$\text{Seesaw: } \min_x \max_y -by\sin(a\pi x), a \sim U[0.9, 1], b \sim U[0.9, 1] \tag{Seesaw}$$

The Seesaw problem is nonconvex-concave, and is considered challenging [106] due to its non-differentiability arising from that the solutions of the state equation or the adjoint state equation are not unique [131]. The L2O training routine follows [4]: we use 128 optimizee instances for training; each of them has its parameters i.i.d. sampled, and variables x, y randomly initialized by i.i.d. sampling from $U[-0.5, 0.5]$. A validation set of 20 optimizees is used with parameters and variables sampled in the same way; and similarly we generate a hold-out testing set of another 100 instances. For each epoch, an L2O optimizer will update the optimizee parameters for 1000 iterations, with its unrolling length T = 10. When the next epoch starts, all x, y as well as LSTM hidden states are reset. We train the L2O solvers for 200 epochs, using Adam with a constant learning rate 10^{-4}. We pick the model checkpoint at the epoch when its validation performance reaches the peak. Figure 8.1 compares the convergence results of the four options, evaluated on the same testing set. We measure the ℓ_2 distances between the solved variables and their corresponding

* This chapter is reprinted with permission from "Learning a minimax optimizer: A pilot study " [2] by Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., Wang, Z. (2020, September) in International Conference on Learning Representations, Copyright 2020 held by the authors.

ground-truth solutions (or the closet one, if multiple exist). It is obvious that only the Two-LSTM-Two-Hidden can successfully converge to the correct solution $(x^*, y^*) = (0, 0)$, which is also the equilibrium. Our major observation from the above experiments is that for minimax L2O optimization, especially for asymmetric problems such as Seesaw, it would be a better choice to use decoupled two LSTM solvers and let them take care of their own trajectory information. We will hence stick to this option and use it as our default Twin-L2O.

All experiments in this and following sections are conducted using the GeForce GTX 1080 Ti GPUs.

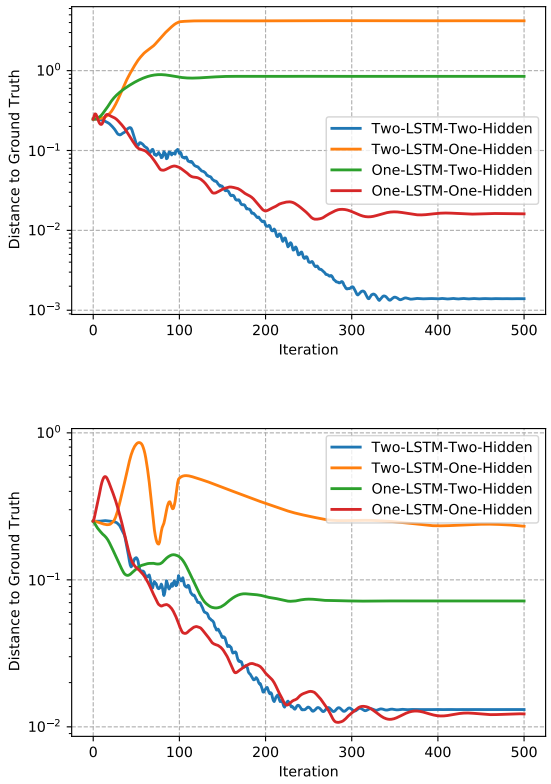


Figure 8.1: Convergence curves of x and y on the ablation study of Twin-L2O design options. Reprinted from [5].

8.2 Comparison With State-of-the-Art Analytical Optimizers

In this section, we apply Twin-L2O to two more test problems besides Seesaw:

- **Rotated Saddle**[1]: $\min_x \max_y ax^2 - by^2 + 2xy, a \sim U[0.9, 1], b \sim U[0.9, 1]$
- **Matrix Game**: $\min_{\mathbf{x}} \max_{\mathbf{y}} \mathbf{x}^T \mathbf{A}\mathbf{y}$, $\mathbf{A} \in \mathbb{R}^{5\times 5}$, $\mathbf{A}_{i,j} \sim$ Bernoulli$(0.5) \cdot U[-1, 1]$

On all three problems, we compare Twin-L2O with several state-of-the-art algorithms: Gradient Descent Ascent (GDA) [52], Optimistic Mirror Descent (OMD) [53] and GD with anchoring (GD-Anchoring) [62]. On Rotated Saddle and Seesaw we will compare with K-beam [106] in addition. For matrix game, we also compare it with the standard Halpern Iteration [64] that is designed for convex-concave minimax problems. For these analytical methods, all parameters are tuned with careful grid search. We train, validate and test Twin-L2O models following the protocol described in Section 8.1.

Figure 8.2, Figure 8.3 and Figure 8.4 plot the convergence curves of all methods, averaged across all testing problems (and each with 20 trials of random x, y initialization). Several observations are drawn below:

- L2O does not show superiority over well-tuned analytical algorithms on the simplest Rotated Saddle problem (and similarly Saddle). The problem is very gradient-friendly, and therefore OMD already achieves the best convergence speed as well as solution quality.
- On Matrix Game, Twin-L2O starts to show competitive edges over analytical solvers with faster convergence speed and higher-precision solutions.
- On the Seesaw problem, Twin-L2O largely outperforms all carefully-tuned analytical algorithms, achieving **one-magnitude higher-precision** solutions with comparable convergence speed. That shows us one *take-home message*: L2O can work for minimax optimization, and can contribute most significantly to those hard problems. That makes minimax L2O a

[1] We also test on the classical **Saddle** problem, but its behaviors and conclusions are almost identical to the Rotated Saddle. We hence report on Rotated Saddle due to the space limit.

highly meaningful complement to existing analytical minimax solvers. More analysis on comparing the actual computational costs (MAC numbers) are provided below.

8.2.1 Computational Cost Analysis

We analyze the number of the multiplier–accumulator operation (MAC) of Twin-L2O and K-beam [106] for a Seasaw problem testing instance with 20 trials of random x, y initialization, each trial lasting for 1000 iterations. As for K-beam, the numbers of MAC are 2.36M (Million), 3.8M, 8.11 M, 15.31M for K = 1, 2, 5, 10 respectively. For Twin-L2O, the total number of MAC is 3.86M.

We use K = 5 in K-beam for experiments in this dissertation whose number of MAC costs 2.1 times more than that of Twin-L2O, yet its solution quality in terms of both the converging speed and the precision fails to beat it.

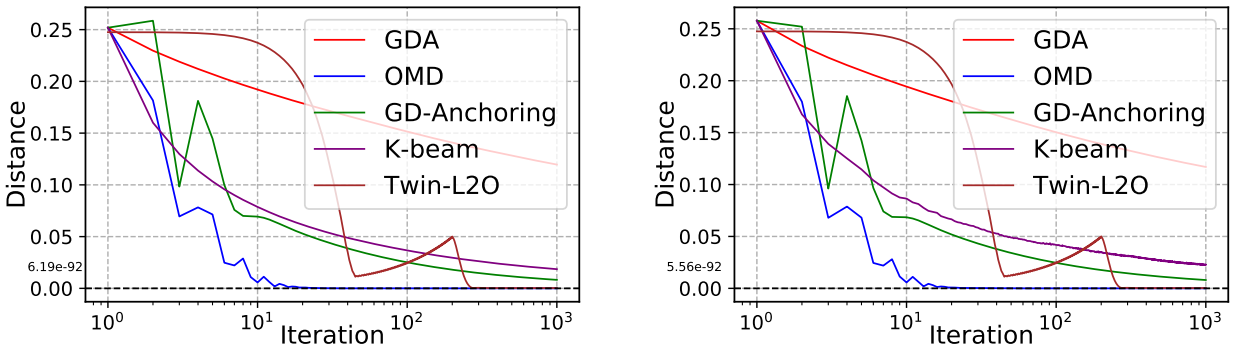


Figure 8.2: Convergence comparison of variable x (left) and y (right) between Twin-L2O and state-of-the-art analytical minimax optimizers (GDA, OMD, GD-Anchoring, and K-beam), for the rotated saddle problem. Reprinted from [5].

8.3 Enhanced Twin-L2O: Curriculum Learning Evaluation

We again use the Seesaw problem as an example in this section. Its two parameters a and b, i.e., the problem period and the scale, are sampled independently from two uniform distributions

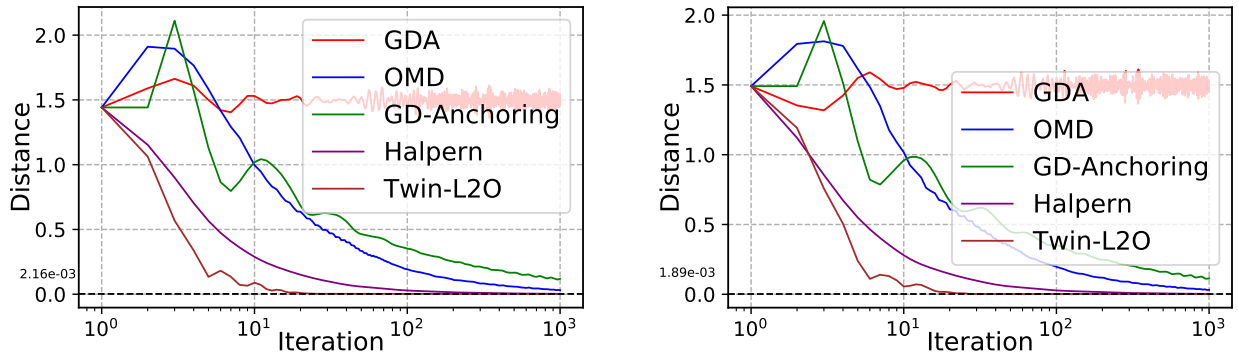


Figure 8.3: Convergence comparison of variable x (left) and y (right) between Twin-L2O and state-of-the-art analytical minimax optimizers (GDA, OMD, GD-Anchoring, and K-beam), for the matrix game problem. Reprinted from [5].

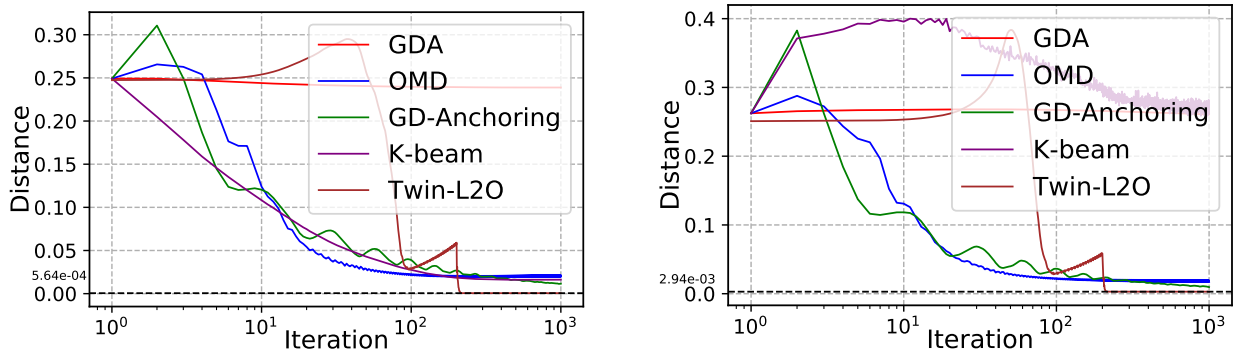


Figure 8.4: Convergence comparison of variable x (left) and y (right) between Twin-L2O and state-of-the-art analytical minimax optimizers (GDA, OMD, GD-Anchoring, and K-beam), for the seesaw problem. Reprinted from [5].

$U[L_{1a}, L_{2a}]$, U$[L_{1b}, L_{2b}]$. In Section 8.1, both are chosen as $U[0.9, 1]$ for the ease of L2O convergence. We now stretch both parameter ranges and test whether an L2O model can still solve the resultant broader range of problems. All other training protocols follow Section 8.1 identically.

Sections 8.1 and 8.2 evaluate the average solution distances over the testing set (100 instances), which worked fine in the small $[a, b]$ range then. However, when we extend the $[a, b]$ range, we find that the L2O behaviors can differ vastly across testing instances, i.e., some converging quickly while others suffering from heavy fluctuations or even divergence, which is an artifact of inefficient L2O training that leaves it unable to cover the full large problem range. That motivates

us to carefully re-design our evaluation metrics here, to reflect both the solution quality and its variation/stability.

For p-th testing instance, we record its solution distance l_2 D_t^p, at epoch $t = 1, 2, ...$ Given two thresholds ϵ_{acc} and ϵ_{std} (chosen by multi-fold validation; we use default $\epsilon_{\text{d}} = 2 \times 10^{-2}$ and $\epsilon_{\text{std}} = 10^{-4}$), we define two forms of success rate (SR):

$$\text{SR}_1 = \frac{\sum_{p=1}^{n} I(\text{d}(D^p) < \epsilon_{\text{acc}})}{n} \qquad \text{SR}_2 = \frac{\sum_{p=1}^{n} I(\text{Std}(D^p) < \epsilon_{\text{std}})}{n}$$

where $\text{d}(D^p) = \frac{\sum_{t=t_0}^{L} D_t^p}{L - t_0 + 1}$, $\text{Std}(Di) = \text{Std}(\{D_t^p\}_{t=t_0}^{L})$, $t_0 = 0.8L$; n = 100 is the number of testing instances; L = 1000 is the total iteration number that each instance (optimizee) is trained by L2O. Intuitively, SR_1 emphasizes the average solution precision from the last 20 iterations; and SR_2 measures how large solution variation is seen in the last 20 iterations.

Table 8.1 compares Twin-L2O and Enhanced Twin-L2O at multiple combinations to stretch the ranges of a and b, starting to the original $[0.9, 1] \times [0.9, 1]$, up to as large as $[0, 5] \times [0, 2]$: the parameter coverage increase by 1,000 times. Adding CL evidently helps Twin-L2O stay effective to train over a broader instance range, under both SR metrics. Vanilla Twin-L2O performs perfectly at $[0.9, 1] \times [0.9, 1]$, yet begins to drop at $[0, 1] \times [0.9, 1]$ (mainly showing higher instability, as indicated by lower SR_2), and hardly succeeds beyond $[0, 3.5] \times [0.9, 1]$. In contrast, Enhanced Twin-L2O obtains nontrivial results even at $[0, 5] \times [0, 1]$ (tens of times wider than the vanilla one).

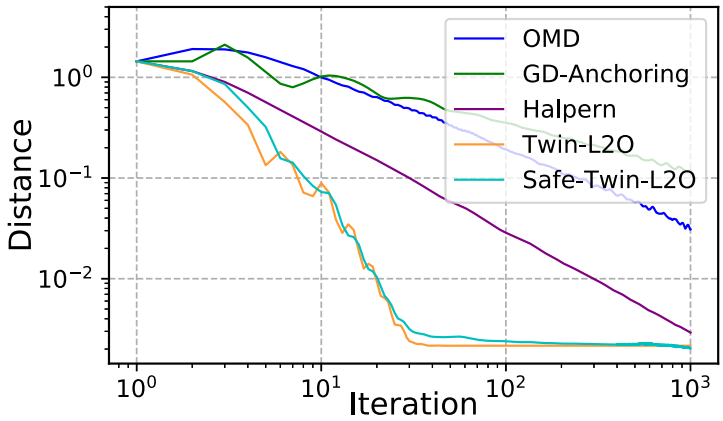
Settings \ Ranges	a b	$[0.9, 1.0]$ $[0.9, 1.0]$	$[0.0, 1.0]$ $[0.9, 1.0]$	$[0.0, 3.5]$ $[0.9, 1]$	$[0.0, 5.0]$ $[0.9, 1.0]$	$[0.9, 1.0]$ $[0.0, 1.0]$	$[0.0, 5.0]$ $[0.0, 1.0]$	$[0.0, 5.0]$ $[0.0, 2.0]$
SR_1 (Twin-L2O)		100.0%	96.6%	0.0%	0.0%	97.8%	11.6%	22.8%
SR_2 (Twin-L2O)		100.0%	89.6%	17.4%	16.0%	96.2%	31.4%	71.6%
SR_1 (Enhanced Twin-L2O)		100.0%	96.6%	64.8%	82.8%	98.0%	85.0%	62.6%
SR_2 (Enhanced Twin-L2O)		100.0%	96.0%	87.8%	91.2%	97.4%	86.2%	53.8%

Table 8.1: Success rate (SR) of different ranges of a,b on the Seesaw problem. Reprinted from [5].

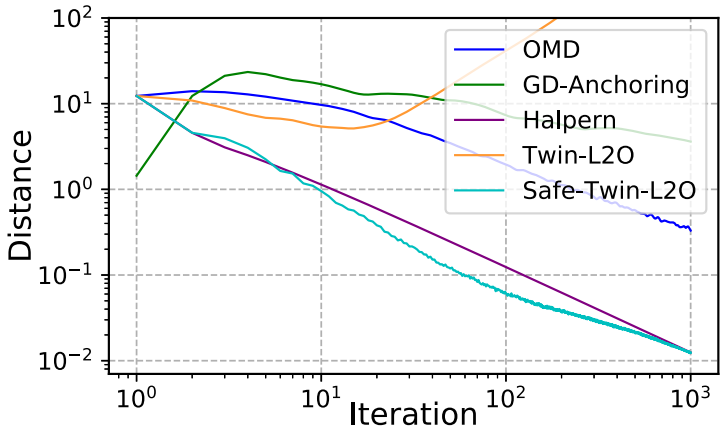
8.4 Safeguarded Twin-L2O Experiments

Here we use the matrix game as the example to evaluate the above established safeguard mechanism for convex-concave minimax optimization. We directly take a well-trained Twin-L2O model for matrix game in Section 8.2, where the matrix $\mathbf{A} \in \mathbb{R}^{5\times 5}$ and $\mathbf{A}_{i,j} \sim \text{Bernoulli}(0.5) \cdot U[-1, 1]$, and the coordinates of initial optimization variables $\mathbf{x}$ and $\mathbf{y}$ are independently sampled from $U[-1, 1]$. During testing, in addition to testing the Twin-L2O model on the testing data from this *seen* distribution, we also evaluate it on *unseen* data, whose A is now sampled from an intentionally very distinct distribution: $\mathbf{A}_{i,j} \sim \text{Bernoulli}(1.0) \cdot U[-8, 8]$. $\mathbf{x}$ and $\mathbf{y}$ are initialized in the same manner.

We compare Safeguarded Twin-L2O (denoted as **Safe-Twin-L2O**) with standard Halpern iteration [64] as the fallback update, when the L2O update is disapproved in Method 7.2. We also compare with OMD and GD-Anchoring on both seen and unseen testing data (GDA fails to converge in both cases, even we tune its hyperparameters to our best efforts). The results are shown in Figure 8.5. When tested on the aggressively varied unseen data, the vanilla Twin-L2O model fails and diverges, but Safe-Twin-L2O remains to converge successfully: even faster than Halpern iteration and OMD, and much better than GD-Anchoring.



(a) Seen Data



(b) Unseen Data

Figure 8.5: Evaluation of Safe-Twin-L2O. Reprinted from [5].

9. CONCLUSION*

In this dissertation, we discuss the unified compression under two use cases: the recommendation system and the vision transformer, and then we discuss how to utilize the learning to optimize technique to improve the minimax optimization.

First, we propose **UMEC** framework by integrating these two objectives into one unified constrained optimization problem, solved by the ADMM method. We conduct extensive experiments and demonstrate the effectiveness of our proposed **UMEC** method by observing its superior performance than other state-of-the-art baseline methods

Then, we propose UVC, a unified ViT compression framework that seamlessly assemble pruning techniques. A budget-constrained optimization framework is formulated for joint learning. Experiments demonstrate that UVC can aggressively trim down the prohibit computational costs in an end-to-end way. The future work will extend UVC to incorporating weight quantization as part of the end-to-end optimization as well.

Last, we study L2O for minimax optimization for the first time. We present the Twin-L2O model, and further improve its generalizability by introducing a theoretically grounded safeguarding framework (for convex-concave problems), as well as an empirical curriculum training strategy (for general problems). Extensive simulations endorse the promise of our algorithms. This pilot study suggests and paves the way for extending L2O beyond continuous minimization problems.

Our method has limitations. The entire L2O field faces challenges to scale up to larger-scale optimization [4], and our study has not yet made an exception. Despite very promising gains from challenging cases such as the Seasaw and Matrix Game problems, the current work only proves

* Part of this chapter is reprinted with permission from "Learning a minimax optimizer: A pilot study " [2] by Shen, J., Chen, X., Heaton, H., Chen, T., Liu, J., Yin, W., Wang, Z. (2020, September) in International Conference on Learning Representations, Copyright 2020 held by the authors. Part of this chapter is reprinted with permission from "UMEC: Unified model and embedding compression for efficient recommendation systems" [5] by Shen, J., Wang, H., Gui, S., Tan, J., Wang, Z., Liu, J. (2020, September). in International Conference on Learning Representations, Copyright 2020 held by the authors. Part of this chapter is reprinted with permission from "Unified visual transformer compression" [3] by Yu, S., Chen, T., Shen, J., Yuan, H., Tan, J., Yang, S., ... Wang, Z. (2022). in International Conference on Learning Representations, Copyright 2022 held by the authors.

the first concept of minimax L2O, on relatively basic and low-dimensional test problems. Our immediate next step is to scale up Twin-L2O, and to explore its potential in solving the minimax application problems of practical interest, such as adversarial training [112, 113] and GANs [132]. A potential idea might leverage the memory-efficient hierarchical RNN structure in [108].

REFERENCES

[1] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019.

[2] J. Shen, H. Wang, S. Gui, J. Tan, Z. Wang, and J. Liu, "Umec: Unified model and embedding compression for efficient recommendation systems," in *International Conference on Learning Representations*, 2021.

[3] S. Yu, T. Chen, J. Shen, H. Yuan, J. Tan, S. Yang, J. Liu, and Z. Wang, "Unified visual transformer compression," in *International Conference on Learning Representations*, 2022.

[4] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in neural information processing systems*, 2016.

[5] J. Shen, X. Chen, H. Heaton, T. Chen, J. Liu, W. Yin, and Z. Wang, "Learning a minimax optimizer: A pilot study," in *International Conference on Learning Representations*, 2020.

[6] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[7] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

[8] A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Zou, "Mixed dimension embeddings with application to memory-efficient recommendation systems," *arXiv preprint arXiv:1909.11810*, 2019.

[9] M. R. Joglekar, C. Li, M. Chen, T. Xu, X. Wang, J. K. Adams, P. Khaitan, J. Liu, and Q. V. Le, "Neural input search for large scale recommendation models," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2387–2397, 2020.

[10] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[16] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International Conference on Machine Learning*, pp. 4055–4064, PMLR, 2018.

[17] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[18] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *International Conference on Machine Learning*, pp. 1691–1703, PMLR, 2020.

[19] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[20] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, Z. Yang, Y. Zhang, and D. Tao, "A survey on vision transformer," 2021.

[21] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017.

[22] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[23] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, 2019.

[24] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *International Conference on Learning Representations*, 2018.

[25] H. Yang, S. Gui, Y. Zhu, and J. Liu, "Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2178–2188, 2020.

[26] Y. Zhao, X. Chen, Y. Wang, C. Li, H. You, Y. Fu, Y. Xie, Z. Wang, and Y. Lin, "Smartexchange: Trading higher-cost memory storage/access for lower-cost computation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 954–967, IEEE, 2020.

[27] P. Ganesh, Y. Chen, X. Lou, M. A. Khan, Y. Yang, D. Chen, M. Winslett, H. Sajjad, and P. Nakov, "Compressing large-scale transformer-based models: A case study on bert," *arXiv preprint arXiv:2002.11985*, 2020.

[28] M. A. Gordon, K. Duh, and N. Andrews, "Compressing bert: Studying the effects of weight pruning on transfer learning," *arXiv preprint arXiv:2002.08307*, 2020.

[29] F.-M. Guo, S. Liu, F. S. Mungall, X. Lin, and Y. Wang, "Reweighted proximal pruning for large-scale language representation," *arXiv preprint arXiv:1909.12486*, 2019.

[30] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?," *arXiv preprint arXiv:1905.10650*, 2019.

[31] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," *arXiv preprint arXiv:1909.11556*, 2019.

[32] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[33] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[34] W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu, "Ternarybert: Distillation-aware ultra-low bit bert," *arXiv preprint arXiv:2009.12812*, 2020.

[35] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "Binarybert: Pushing the limit of bert quantization," *arXiv preprint arXiv:2012.15701*, 2020.

[36] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "Dynabert: Dynamic bert with adaptive width and depth," *arXiv preprint arXiv:2004.04037*, 2020.

[37] M. Zhu, K. Han, Y. Tang, and Y. Wang, "Visual transformer pruning," *arXiv preprint arXiv:2104.08500*, 2021.

[38] T. Chen, Y. Cheng, Z. Gan, L. Yuan, L. Zhang, and Z. Wang, "Chasing sparsity in vision transformers: An end-to-end exploration," *arXiv preprint arXiv:2106.04533*, 2021.

[39] B. Pan, Y. Jiang, R. Panda, Z. Wang, R. Feris, and A. Oliva, "Ia-red^2: Interpretability-aware redundancy reduction for vision transformers," *arXiv preprint arXiv:2106.12620*, 2021.

[40] Y. Tang, K. Han, Y. Wang, C. Xu, J. Guo, C. Xu, and D. Tao, "Patch slimming for efficient vision transformers," *arXiv preprint arXiv:2106.02852*, 2021.

[41] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," *arXiv preprint arXiv:2012.12877*, 2020.

[42] D. Jia, K. Han, Y. Wang, Y. Tang, J. Guo, C. Zhang, and D. Tao, "Efficient vision transformers via fine-grained manifold distillation," *arXiv preprint arXiv:2107.01378*, 2021.

[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[44] A. Globerson and S. Roweis, "Nightmare at test time: robust learning by feature deletion," in *Proceedings of the 23rd international conference on Machine learning*, pp. 353–360, 2006.

[45] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," *arXiv preprint arXiv:1409.7495*, 2014.

[46] J. Shamma, *Cooperative control of distributed multi-agent systems*. John Wiley & Sons, 2008.

[47] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5262–5276, 2010.

[48] Z. Wu, Z. Wang, Z. Wang, and H. Jin, "Towards privacy-preserving visual recognition via adversarial training: A pilot study," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 606–624, 2018.

[49] Z. Wu, H. Wang, Z. Wang, H. Jin, and Z. Wang, "Privacy-preserving deep action recognition: An adversarial learning framework and a new dataset," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[50] M. Benaım and M. W. Hirsch, "Mixed equilibria and dynamical systems arising from fictitious play in perturbed games," *Games and Economic Behavior*, vol. 29, no. 1-2, pp. 36–72, 1999.

[51] P. Mertikopoulos, C. Papadimitriou, and G. Piliouras, "Cycles in adversarial regularized learning," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2703–2717, SIAM, 2018.

[52] T. Lin, C. Jin, and M. I. Jordan, "On gradient descent ascent for nonconvex-concave minimax problems," *arXiv preprint arXiv:1906.00331*, 2019.

[53] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng, "Training gans with optimism.," in *International Conference on Learning Representations (ICLR 2018)*, 2018.

[54] C. Daskalakis and I. Panageas, "The limit points of (optimistic) gradient descent in min-max optimization," in *Advances in Neural Information Processing Systems*, 2018.

[55] T. Liang and J. Stokes, "Interaction matters: A note on non-asymptotic local convergence of generative adversarial networks," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 907–915, 2019.

[56] P. Mertikopoulos, B. Lecouat, H. Zenati, C.-S. Foo, V. Chandrasekhar, and G. Piliouras, "Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile," *arXiv preprint arXiv:1807.02629*, 2018.

[57] G. Gidel, H. Berard, G. Vignoud, P. Vincent, and S. Lacoste-Julien, "A variational inequality perspective on generative adversarial networks," *arXiv preprint arXiv:1802.10551*, 2018.

[58] A. Mokhtari, A. Ozdaglar, and S. Pattathil, "A unified analysis of extra-gradient and optimistic gradient methods for saddle point problems: Proximal point approach," *arXiv preprint arXiv:1901.08511*, 2019.

[59] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. De Freitas, "Learning to learn without gradient descent by gradient descent," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 748–756, JMLR. org, 2017.

[60] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[61] H. Heaton, X. Chen, Z. Wang, and W. Yin, "Safeguarded learned convex optimization," *arXiv preprint arXiv:2003.01880*, 2020.

[62] E. K. Ryu, K. Yuan, and W. Yin, "Ode analysis of stochastic gradient methods with optimism and anchoring for minimax problems and gans," *arXiv:1905.10899*, 2019.

[63] T. Lin, C. Jin, M. Jordan, *et al.*, "Near-optimal algorithms for minimax optimization," *arXiv preprint arXiv:2002.02417*, 2020.

[64] J. Diakonikolas, "Halpern iteration for near-optimal and parameter-free monotone inclusion and strong solutions to variational inequalities," *arXiv preprint arXiv:2002.08872*, 2020.

[65] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[66] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, pp. 2074–2082, 2016.

[67] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[68] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin, "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions," in *International Conference on Machine Learning*, pp. 5363–5372, PMLR, 2018.

[69] T. Ajanthan, P. K. Dokania, R. Hartley, and P. H. Torr, "Proximal mean-field for neural network quantization," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4871–4880, 2019.

[70] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[71] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[72] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1365–1374, 2019.

[73] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2019.

[74] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin, "The lottery ticket hypothesis for pre-trained bert networks," *arXiv*, vol. abs/2007.12223, 2020.

[75] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, M. Carbin, and Z. Wang, "The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models," *arXiv preprint arXiv:2012.06908*, 2020.

[76] X. Chen, Z. Zhang, Y. Sui, and T. Chen, "Gans can play lottery tickets too," in *International Conference on Learning Representations*, 2021.

[77] T. Chen, Z. Zhang, S. Liu, S. Chang, and Z. Wang, "Long live the lottery: The existence of winning tickets in lifelong learning," in *International Conference on Learning Representations*, 2021.

[78] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," 2021.

[79] H. Ma, T. Chen, T.-K. Hu, C. You, X. Xie, and Z. Wang, "Good students play big lottery better," *arXiv preprint arXiv:2101.03255*, 2021.

[80] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

[81] H. Yang, Y. Zhu, and J. Liu, "Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11206–11215, 2019.

[82] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3296–3305, 2019.

[83] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[84] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198, 2016.

[85] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 346–353, 2019.

[86] Z. Huang, X. Xu, H. Zhu, and M. Zhou, "An efficient group recommendation model with multiattention-based neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[87] X. Mao, S. Mitra, and V. Swaminathan, "Feature selection for fm-based context-aware recommendation systems," in *2017 IEEE International Symposium on Multimedia (ISM)*, pp. 252–255, IEEE, 2017.

[88] Q. Song, D. Cheng, H. Zhou, J. Yang, Y. Tian, and X. Hu, "Towards automated neural interaction discovery for click-through rate prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 945–955, 2020.

[89] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," 2021.

[90] M. Zheng, P. Gao, X. Wang, H. Li, and H. Dong, "End-to-end object detection with adaptive clustering transformer," *arXiv preprint arXiv:2011.09315*, 2020.

[91] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, pp. 213–229, Springer, 2020.

[92] Z. Dai, B. Cai, Y. Lin, and J. Chen, "Up-detr: Unsupervised pre-training for object detection with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1601–1610, 2021.

[93] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," *arXiv preprint arXiv:2010.04159*, 2020.

[94] H. Wang, Y. Zhu, H. Adam, A. Yuille, and L.-C. Chen, "Max-deeplab: End-to-end panoptic segmentation with mask transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5463–5474, 2021.

[95] Y. Wang, Z. Xu, X. Wang, C. Shen, B. Cheng, H. Shen, and H. Xia, "End-to-end video instance segmentation with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8741–8750, 2021.

[96] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao, "Pre-trained image processing transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12299–12310, 2021.

[97] F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo, "Learning texture transformer network for image super-resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5791–5800, 2020.

[98] Y. Jiang, S. Chang, and Z. Wang, "Transgan: Two transformers can make one strong gan," *arXiv preprint arXiv:2102.07074*, 2021.

[99] G. Bertasius, H. Wang, and L. Torresani, "Is space-time attention all you need for video understanding?," *arXiv preprint arXiv:2102.05095*, 2021.

[100] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, "Point transformer," *arXiv preprint arXiv:2012.09164*, 2020.

[101] J. v. Neumann, "Zur theorie der gesellschaftsspiele," *Mathematische annalen*, vol. 100, no. 1, pp. 295–320, 1928.

[102] A. Nedić and A. Ozdaglar, "Subgradient methods for saddle-point problems," *Journal of optimization theory and applications*, vol. 142, no. 1, pp. 205–228, 2009.

[103] S. S. Du and W. Hu, "Linear convergence of the primal-dual gradient method for convex-concave saddle point problems without strong convexity," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 196–205, 2019.

[104] C. Jin, P. Netrapalli, and M. I. Jordan, "What is local optimality in nonconvex-nonconcave minimax optimization?," *arXiv preprint arXiv:1902.00618*, 2019.

[105] Y. Wang, G. Zhang, and J. Ba, "On solving minimax optimization locally: A follow-the-ridge approach," *arXiv preprint arXiv:1910.07512*, 2019.

[106] J. Hamm and Y.-K. Noh, "K-beam minimax: Efficient optimization for deep adversarial learning," *arXiv preprint arXiv:1805.11640*, 2018.

[107] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2247–2255, JMLR. org, 2017.

[108] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[109] C. Li, T. Chen, H. You, Z. Wang, and Y. Lin, "Halo: Hardware-aware learning to optimize," in *European Conference on Computer Vision*, pp. 500–518, Springer, 2020.

[110] T. Chen, W. Zhang, J. Zhou, S. Chang, S. Liu, L. Amini, and Z. Wang, "Training stronger baselines for learning to optimize," *arXiv preprint arXiv:2010.09089*, 2020.

[111] Y. Cao, T. Chen, Z. Wang, and Y. Shen, "Learning to optimize in swarms," in *Advances in Neural Information Processing Systems*, pp. 15018–15028, 2019.

[112] H. Jiang, Z. Chen, Y. Shi, B. Dai, and T. Zhao, "Learning to defense by learning to attack," *arXiv preprint arXiv:1811.01213*, 2018.

[113] Y. Xiong and C.-J. Hsieh, "Improved adversarial training via learned optimizer," 2020.

[114] K. Tono, A. Takeda, and J.-y. Gotoh, "Efficient dc algorithm for constrained sparse optimization," *arXiv preprint arXiv:1701.08498*, 2017.

[115] A. Nitanda, "Stochastic proximal gradient descent with acceleration techniques," in *Advances in Neural Information Processing Systems*, pp. 1574–1582, 2014.

[116] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[117] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[118] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, "Autogan-distiller: Searching to compress generative adversarial networks," *arXiv preprint arXiv:2006.08198*, 2020.

[119] T. Kim, J. Oh, N. Kim, S. Cho, and S.-Y. Yun, "Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation," *arXiv preprint arXiv:2105.08919*, 2021.

[120] N. Buchbinder and J. Naor, *The design of competitive online algorithms via a primal-dual approach*. Now Publishers Inc, 2009.

[121] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1529–1538, 2020.

[122] H. Yang, Y. Huang, L. Tran, J. Liu, and S. Huang, "On benefits of selection diversity via bilevel exclusive sparsity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5945–5954, 2016.

[123] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[124] Z. Pan, B. Zhuang, J. Liu, H. He, and J. Cai, "Scalable vision transformers with hierarchical pooling," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 377–386, 2021.

[125] Y. Lee and S. Choi, "Gradient-based meta-learning with learned layerwise metric and subspace," *arXiv preprint arXiv:1801.05558*, 2018.

[126] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *arXiv preprint arXiv:1808.03314*, 2018.

[127] A. Raghunathan, A. Cherian, and D. Jha, "Game theoretic optimization via gradient-based nikaido-isoda function," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 5291–5300, PMLR, 09–15 Jun 2019.

[128] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009.

[129] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," *arXiv preprint arXiv:1712.05055*, 2017.

[130] B. Halpern, "Fixed points of nonexpanding maps," *Bulletin of the American Mathematical Society*, vol. 73, no. 6, pp. 957–961, 1967.

[131] J. M. Danskin, "The theory of max-min, with applications," *SIAM Journal on Applied Mathematics*, vol. 14, no. 4, pp. 641–664, 1966.

[132] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, pp. 5767–5777, 2017.