Toward Efficient Deep Learning with Sparse Neural Networks

Namhoon Lee

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2020

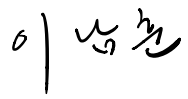
To my family and friends, for their endless support and encouragement.

Declaration

I hereby declare that, this thesis (and the work presented in it) is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy, and that this thesis is entirely my own work based on publications in collaboration with the co-authors where due acknowledgement is made.

August 2020

Namhoon Lee



Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor Professor Philip H. S. Torr for providing the opportunity to pursue a PhD and having faith in me at all times. His highest standards in scientific research and everlasting enthusiasm have been truly inspirational, without which this thesis would have not been possible. Thank you Phil for everything you have done for me. I am forever in your debt.

I have been extremely fortunate to have received opportunities to work with collaborators outside Oxford as well. I am sincerely grateful to Professor Stephen Gould for supporting my visiting research at Australian National University and providing invaluable ideas and breakthroughs to the challenging problems in my research project. I am equally thankful for Professor Martin Jaggi for hosting me as a research intern at EPFL, welcoming me into his group of great minds, and opening my eyes to interesting concepts in optimization. Besides their feedback on my research projects, I am appreciative for their support and encouragement for my career plans.

I would also have to deeply thank Dr. Thalaiyasingam Ajanthan for his countless hours of guidance throughout my PhD. His scientific rigor and strong work ethics, amongst many, have tremendously influenced my attitude towards how to research, which have been vital for making this thesis possible.

I am grateful for all the joy, advice, and support I received from a lot of people I have met at Oxford. I truly enjoyed my time thanks to Anurag and Qizhu who have made beautiful memories in and out of the lab with a full of kindness and laughter. I also thank Jack who first embraced me into Oxford and taught me every skill I needed to conduct scientific research during my early years of PhD with patience and encouragement. I am fortunate to have been surrounded by all the great minds in the Torr Vision Group. Their wisdom and support contributed

to this thesis immensely, and I thank them all wholeheartedly. I am also grateful for my transfer and confirmation report examiners Professors Pawan Kumar and Victor Prisacariu for their insightful feedback. I would also like to thank Joanna, Cassandra, and Maddy for all their help on administrative stuff that could have easily got me bogged down into a pile of unrecognisable documents.

Finally, I am grateful to have been funded by the Graduate Study Abroad Scholarship from the Korean Government and the Light Senior Scholarship from St Catherine's College.

Abstract

Despite the tremendous success that deep learning has achieved in recent years, it remains challenging to deal with the excessive computational and memory cost involved in executing deep learning based applications. To address the challenge, this thesis focuses on studying sparse neural networks, particularly around their construction, initialization, and large-scale training aspects, as an attempt to take a step toward efficient deep learning.

Firstly, this thesis addresses the problem of finding sparse neural networks by pruning. Network pruning is an effective methodology to sparsify neural networks, and yet, existing approaches often introduce hyperparameters that either need to be tuned with expert knowledge or are based on ad-hoc intuitions, and typically entails iterative training steps. Alternatively, this thesis begins with proposing an efficient pruning method that is applied to a neural network prior to training in a single shot. The obtained sparse neural network using this method, once trained, exhibit state-of-the-art performance on various image classification tasks.

Albeit efficient, it remains unclear exactly why this approach of pruning at initialization can be effective. This thesis then extends this method by developing a new perspective, from which the problem of finding trainable sparse neural networks is approached based on network initialization. Being a key to the success of finding and training sparse neural networks, this thesis proposes a sufficient initialization condition that can be easily satisfied with a simple optimization step and, once achieved, accelerates training sparse neural networks quite significantly.

While sparse neural networks can be obtained by pruning at initialization, there has been little study concerning the subsequent training of these sparse networks. This thesis lastly concentrates on studying

data parallelism – a straightforward approach to speed up neural network training by parallelizing it using a distributed computing system – under the influence of sparsity. To this end, the effects of data parallelism and sparsity are first measured accurately based on extensive experiments which are accompanied by metaparameter search. Then, this thesis establishes theoretical results that precisely account for these effects, which have only been addressed partially and empirically and thus remained as debatable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of this integrated thesis, which is structured and summarized as follows: Section 1.1 presents a brief introduction to the science of deep learning, which will provide a context that this thesis is based on; Section 1.2 discusses some of the critical challenges in deep learning, with a particular focus on specific issues regarding efficiency; Section 1.3 proposes a summary of approaches that this thesis takes to address those challenges; Sections 1.4 and 1.5 provide an outline and main contributions of the thesis, respectively; Lastly, a list of publications constituting the main chapters is displayed in Section 1.6.

1.1 Deep learning: a brief introduction

When programmable computers were first conceived, people wondered whether such machines might become intelligent [Lovelace, 1842]. Today, artificial intelligence (AI) is everywhere affecting every aspect of modern society and changing people's daily lives; from personalized search results and news feeds on social media platforms, to language translation and text completion on mobile phones, to vision and robotics applications such as autonomous vehicles, and further to medical science including diagnostics, drug discovery, and so forth. AI is becoming increasingly more powerful and prevalent in society today [LeCun et al., 2015; Goodfellow et al., 2016].

These highly intelligent systems are often powered by machine learning technologies, and *deep learning* has been adopted as a primary method of choice to develop solutions for such needs quite rapidly and dominantly in recent years, which has brought about major breakthroughs in many scientific fields of research. Here are some examples:

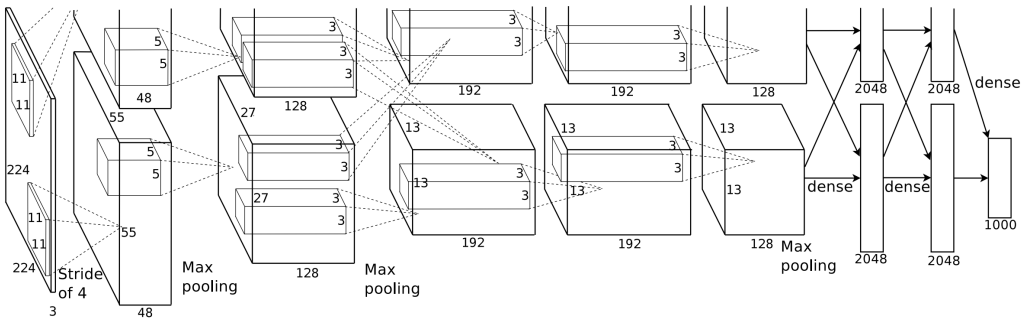


Figure 1.1: The original AlexNet model architecture used to win the ImageNet Large Scale Visual Recognition Challenge. AlexNet is considered one of the most influential works in modern computer vision, employing CNNs and GPUs to accelerate deep learning. Figure from Krizhevsky et al. [2012].

Computers recognizing from images. Deep learning can learn useful patterns or representations from images, and the learned representations can be used to train a classifier to identify an object or person. In contrast to traditional machine learning approaches, this can be done without hard-coding knowledge or even without requiring labels for the images, by scaling up the amount of data and the size of the neural network model as well as by using massive computing resources. For example, Le et al. [2012] developed a deep sparse autoencoder to perform unsupervised feature learning from 10 million unlabeled images obtained from the web across 1,000 machines (16,000 cores), which resulted in a system that can recognize a face of human, cat, or human body. Krizhevsky et al. [2012] developed a model called AlexNet (see Figure 1.1) based on a convolutional neural network architecture and won the image recognition challenge (ImageNet) by a large margin to the runner-up.

Gameplay. Deep neural networks, combined with advanced search trees, can perform human-level game plays. AlphaGo, a computer program which was developed by researchers at DeepMind for example, defeated a professional human Go player in 2016 [Silver et al., 2016]. Rather than taking an impractical, brute-force approach, AlphaGo makes use of neural networks that take a description of the Go board as an input and process it through a number of different neural network layers containing millions of neuron-like connections. Over time, AlphaGo improved and became increasingly stronger and better at learning and decision-making.

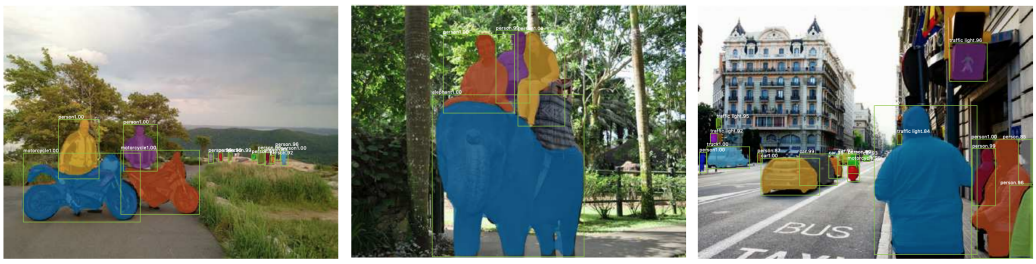


Figure 1.2: Deep learning can be used to detect objects and predict segmentaion masks (instance segmentation). Masks are shown in color, and bounding box, category, with confidence scores. Such algorithms can be deployed to intelligent systems such as for autonomous driving to recognize and track objects in the scene. Figure from He et al. [2017a].

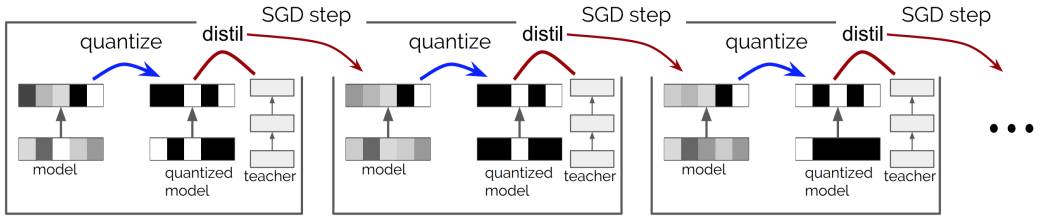


Figure 1.3: Quantized distillation [Polino et al., 2018] as an example of "Tiny AI". A smaller student network is trained by distiling knowledge of a larger teacher network, while its parameters are quantized for further compression.

Robotics. Deep learning approaches are driving rapid progress in new developments in various robotics applications. Object recognition and tracking from images, self-localization in environments, semantic scene understanding, and further, controls for autonomous driving systems, for example, are mainly driven by the latest deep learning approaches [Sivaraman and Trivedi, 2013; Zhou and Tuzel, 2018; Badrinarayanan et al., 2017; Sallab et al., 2017] (see Figure 1.2). For other areas of robotics including industrial robotics, intelligent transportation systems, manufacturing, etc. deep learning are becoming increasingly more popular.

Natural language processing. Deep learning based language modeling has been making remarkable strides in ability to form natural language sentences, to solve questions, and to complete text passages. Recent language models that achieved truly impressive results on language tasks such as BERT [Devlin et al., 2018] or GPT [Radford et al., 2018; 2019] are based on the architecture known as Transformer [Vaswani et al., 2017] which is based on deep attention models.

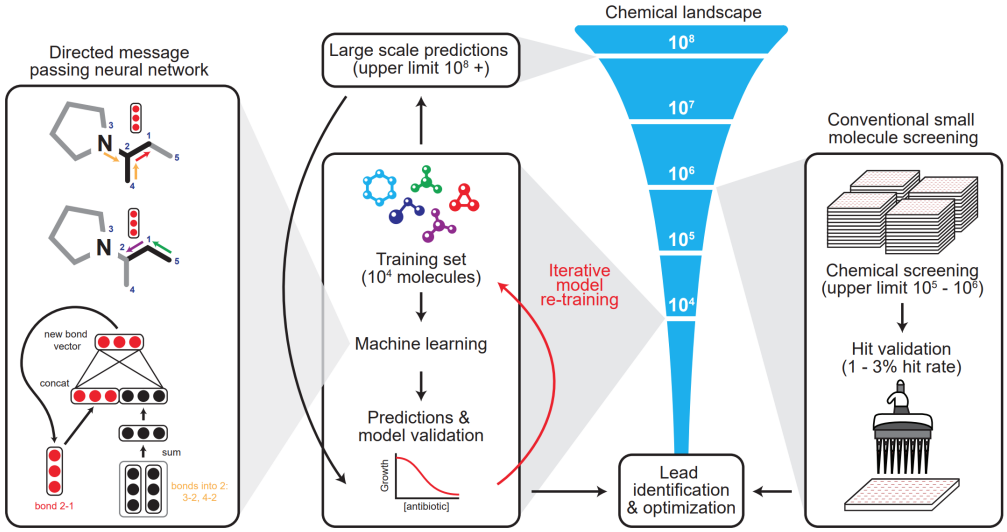


Figure 1.4: Use of deep learning for antibiotic discovery [Stokes et al., 2020]. Deep neural networks can learn a molecular representation using a collection of diverse molecules for those that inhibited the growth of a certain bacteria (*e.g.*, E. coli). Such a representation can be used to identify potential lead compounds against the bacteria more efficiently, without having to perform the expensive and time consuming process of screening large chemical libraries as in a conventional way.

Tiny AI. A key element for the success of AI applications is deployability: programs often have to run on limited resources environments such as mobile phones. The MIT Technological Review lists the "tiny AI" as one of the breakthroughs in 2020 [Hao, 2020]. In deep learning, the size of neural network models can be shrinked through a diverse set of compression techniques (*e.g.*, knowledge distillation as in Hinton et al. [2015] which can also be combined with quantization technique; see Figure 1.3) without losing performance.

Medical science. Deep learning can also be used in various fields of medical science. For example, a new type of antibiotic based on newly discovered molecules has been recently developed using a deep learning approach [Stokes et al., 2020]. While it is expensive to find out a new drug, deep learning can expedite the evaluation process effectively (see Figure 1.4).

Basically, deep learning is considered a branch of machine learning methods based on artificial neural networks, which are inspired by information processing in biological brains [Schmidhuber, 2015; LeCun et al., 2015; Marblestone et al.,

2016]. The term "deep" stems from the contrast that traditional neural network models only consist of a few hidden layers (usually less than three) whereas deep models can often have more than thousands of layers, and such, it usually refers to the scale of modern neural network models.

While traditional machine learning methods require a feature extraction mechanism that is designed manually with some prior knowledge in order to recognize meaningful patterns from data, deep learning can address this problem of representation learning [Bengio et al., 2013] with structured neural network models and extract features at various levels.

1.2 Challenges in deep learning

While deep learning has achieved tremendous success in various areas, it is extremely challenging to make it *efficient*. This is partly because neural network models are highly overparameterized; the number of parameters in a modern neural network model is easily more than millions, if not billions or trillions. For example, a recent language model (called GPT-3) capable of achieving state-of-the-art results on a set of benchmark and unique natural language processing tasks has about 175 billion parameters [Brown et al., 2020]. Such large models entail a significant processing cost and energy at use, potentially posing a serious challenge to resource constrained environments such as embedded systems in mobile devices having to perform various tasks real-time.

In fact, there have been made a lot of efforts to design neural network architectures more efficient, while the performance remains the same or even better; *e.g.*, for the task of image classification in computer vision, the deep network architecture has evolved to have less number of parameters, for instance from VGG [Simonyan and Zisserman, 2015] to ResNets [He et al., 2016]. This is usually done by better understanding the role of network's components and then designing new types of network architectures based on it; *e.g.*, small convolutional kernels, residual skip connection, etc.

Another way to address overparameterization in deep networks is to utilize sparsity. *Sparse neural networks*, with a relatively high sparsity level in weight matrices for instance, have only a small fraction of non-zero parameters in the network. This way, the computations required to run a large network model (*e.g.*, floating point operations per second or FLOPs) can be significantly reduced along with its memory requirements.

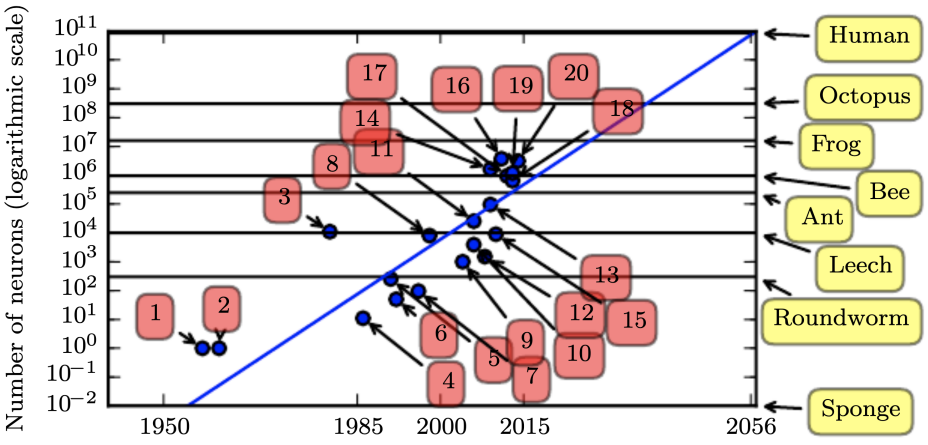


Figure 1.5: Increasing neural network size over time: from 1 Perceptron [Rosenblatt, 1958], to 20 GoogLeNet [Szegedy et al., 2015]. The number of neurons in artificial neural networks are rapidly increasing. Figure from Goodfellow et al. [2016].

Nonetheless, there remain many aspects of deep learning that makes it challenging to utilize sparse neural networks. The main focus of this thesis is precisely on this matter, and the thesis addresses the following questions: How can deep learning be more efficient with sparse neural networks? What are the characteristics of sparse neural networks? Are there any weaknesses in using sparse neural networks? How can we mitigate them, if any?

This section briefly discusses some major challenges to efficient deep learning: Deep neural network models are too large (Section 1.2.1); Training sparse neural networks can be difficult (Section 1.2.2); The effects of large-scale training of neural networks (with and without sparsity) are quite unknown as of yet (Section 1.2.3).

1.2.1 Large models

As described earlier, a modern neural network model can be extremely large. See Figure 1.5 for the trend of increasing network models.

To secure the efficiency of deep network models, researchers have come up with various different tactics, and one effective methodology is compressing the network model. This approach yields a model that is smaller in size, and thus, the amount of computations can be reduced quite dramatically.

However, existing network compression algorithms are often too complex, which makes them hard to be employed in practice. For example, many approaches introduce a lot of new hyperparameters that need to be tuned with some prior knowledge or expert experience; these hyperparameters are often based on ad-hoc

intuition lacking mathematical principles; some algorithms based on optimization approaches can suffer from convergence issues; and most notably, almost all existing algorithms require the network to be pretrained beforehand. In short, large neural network models are still presenting a significant challenge to be employed for efficient deep learning.

1.2.2 Training difficulty

Training neural networks can be difficult for various reasons. Typically the objective for a deep learning task is formulated as an empirical risk minimization problem, so as to achieve a low training loss. However, the objective function is non-convex in most cases, and hence, the training process can be stuck in local minima. In fact, stochastic gradient descent (SGD) [Robbins and Monro, 1951] is a de facto optimization algorithm and works well in practice. Nevertheless, due to its stochasticity, the initial point at which the optimization starts from is also quite crucial for convergence and the success of training. While different initialization methods have been developed in the past to address this issue on network training, the impact of the initialization process for a compressed form of neural networks on training performance in general is yet to be well-understood.

1.2.3 Complex scaling

Another challenging aspect of deep learning is that training neural networks is usually complex to scale in terms of an increasing amount of data. In other words, the effects of large-scale data for distributed optimization, is quite unknown as of yet. A massive amount of data is produced daily with a variety of devices, and hence, being able to scale according to the large amount of data is crucial for efficient deep learning. Therefore, it is necessary to have rigorous understanding of the effect of large-scale training.

1.3 Approach

In order to address the aforementioned issues, this thesis focuses on sparse neural networks, especially for their construction, initialization, and large-scale training. To this end, we adopt *network pruning* methodology to compress large neural network models. Network pruning is a subject studied over the last decades with many successful use cases. However, many existing pruning algorithms usually

require hyperparameter tuning and are weaved with training steps, while many of which are based on some heuristics. As a result, they are often non-scalable to different network models or data sets without making significant adjustments, resulting in limited availability.

We first notice that the pruning algorithm can be simpler and more efficient. We develop a new saliency metric to identify important parameters in the network based on which the sparsification process is performed. The saliency criterion is defined based on sensitivity to the network's loss function and can be measured prior to training.

Although the dependency on the actual parameter values at which the saliency is measured can be alleviated, and hence, the sparsification can be done on untrained neural networks, the distribution of remaining parameters after sparsification will change, losing desirable properties in proper weight initialization schemes in deep learning. Borrowing the concepts of dynamical isometry and a mean-field theory, a signal propagation perspective for sparsification is developed. Then a simple optimization step is used to restore an initialization condition sufficient for good signal propagation and trainability of the sparse network.

Now that sparse neural networks can be obtained more efficiently at random initialization prior to training, the subsequent training of these sparse models is studied further. Especially, the remaining part of the thesis focuses on their behavior for data parallelism. Data parallelism refers to utilizing a parallel computing system where the training data is distributed to multiple processors for computing gradients (or higher-order derivatives) in parallel such that the training process can be accelerated. We accurately measure the effects of data parallelism based on extensive metaparameter search, and then demystify the effects of data parallelism on neural network training theoretically based on the convergence properties of stochastic gradient methods.

1.4 Thesis outline

The remaining chapters of this thesis are outlined as follows:

Chapter 2. This chapter provides backgrounds on sparse neural networks and preliminary materials, which are used throughout the thesis. The review will be focused on the concepts of network compression, initialization, and large-scale distributed training.

Chapter 3. This chapter addresses the problem of large neural network models and existing pruning algorithms. We present an efficient method to prune neural networks that yields highly sparse neural networks. The method presented in this chapter will also be used for subsequent chapters.

Chapter 4. In the previous chapter, it is presented that pruning can be done prior to training. However, it was not entirely clear how pruning a randomly initialized neural network can be effective. This chapter studies this aspect by approaching from a signal propagation perspective to neural networks.

Chapter 5. Previous chapters discuss how to compress the network efficiently (Chapter 3) or initialize (Chapter 4). This chapter studies the stage after pruning, *i.e.*, the subsequent training phase of sparse neural networks after pruning. In particular, this chapter focuses on the effects of data parallelism on neural network training, firstly by measuring them accurately and then providing theoretical insights to the observed results.

Chapter 6. Finally, we conclude in Chapter 6, by summarizing the contributions presented in this thesis. We also discuss remaining challenges and future works.

1.5 Contributions

The main contribution of this thesis is on the study of sparse neural networks for their efficient construction, trainability ensuring initialization, and large-scale training. These are illustrated briefly in the following:

Chapter 3

This chapter presents an efficient network pruning algorithm. Existing prior arts have notable drawbacks; many approaches introduce hyperparameters in their algorithm that need to be tuned well (otherwise they can fail) based on some heuristics and differently for each and every different workload; almost all previous works require some training steps to a certain degree of convergence before pruning begins. In this work, we propose to prune neural

networks at initialization prior to training in a single-shot. Our approach does not require any additional hyperparameters, a large amount of data, or the pretraining requirement to perform pruning, and it is still applicable to all kinds of neural network architectures without any modification. The resulting sparse neural network can simply be trained in the standard way, and they achieve(d) state-of-the-art performance (at the time of its publication). In addition, we show that this approach enables visualization of what parameters are pruned away, and further, that the method can also be used to prevent memorization in deep learning.

Chapter 4

In this chapter, we suggest an initialization condition to accelerate the training process of sparse neural networks. In the previous chapter, we presented an efficient pruning algorithm that can be performed at random initialization prior to training. Albeit effective, it remains unclear exactly why pruning untrained neural networks can be effective. We first notice a critical impact of network initialization on pruning, and develop a signal propagation perspective to ensure reliable pruning results. The key insight is understanding the effect of initial weights on how signals, *i.e.*, inputs and outputs of networks, propagate during forward and backward phases. This allows us to develop a formal initialization condition to ensure faithful gradient measurements, and thereby, pruning results. Based on our signal propagation perspective, we suggest a simple, yet effective data-free optimization step that, once performed after pruning, accelerates the training of sparse neural networks quite significantly. The idea is validated through extensive experiments, and further, extended to different scenarios including pruning without labels, pruning non-standard arbitrarily-designed network architectures.

Chapter 5

Network compression can reduce computations required for deep learning, and pruning is a promising avenue to compress large neural networks. The previous chapters show that it is possible to obtain extremely sparse neural network models prior to training. However, little has been studied about various aspects of training these sparse neural networks. This chapter focuses on the effects of data parallelism and sparsity on neural network training. First, we accurately measure the effects of data parallelism and sparsity on neural

network training by performing extensive experiments with metaparameter search. Also, based on the empirical findings, we derive theoretical results to help understand the effects of data parallelism and sparsity. Our result describes accurately what has remained as empirical findings previously in the literature, and it applies to both the regular and sparse models.

1.6 Publications

The contributions in each chapter, as described the previous section, are based on the following publications:

Chapter 3

- **Namhoon Lee**, Thalaiyasingam Ajanthan, Philip H. S. Torr. "SNIP: Single-shot Network Pruning based on Connection Sensitivity". *International Conference on Learning Representations (ICLR)*, 2019.

Chapter 4

- **Namhoon Lee**, Thalaiyasingam Ajanthan, Stephen Gould, Philip H. S. Torr. "A Signal Propagation Perspective for Pruning Neural Networks at Initialization". *International Conference on Learning Representations (ICLR)*, 2020.

Chapter 5

- **Namhoon Lee**, Philip H. S. Torr, Martin Jaggi. "Data Parallelism in Training Sparse Neural Networks". *International Conference on Learning Representations (ICLR) Workshop on Practical Machine Learning*, 2020.

- **Namhoon Lee**, Thalaiyasingam Ajanthan, Philip H. S. Torr, Martin Jaggi. "Understanding the Effects of Data Parallelism and Sparsity on Neural Network Training". *Submitted to Neural Information Processing Systems (NeurIPS)*, 2020.

Chapter 2

Background

This chapter presents a brief review of existing works in the literature around the primary concepts studied in this thesis. Specifically, we focus on pruning (Section 2.1), initializing (Section 2.2), and large-scale training (Section 2.3) of deep neural networks. The aim is at providing the audience with a modest background to facilitate the understanding of materials in later chapters, rather than establishing a comprehensive review.

2.1 Pruning neural networks

2.1.1 Intuition

Despite the tremendous success of deep learning, the size of a typical modern neural network model is often too large; for instance, the number of parameters on a recent language model is more than a few hundred of billions [Brown et al., 2020] as illustrated in Section 1.2.

One way to address the issue is to induce sparsity. By making a large number of elements in a neural network be zero valued (*e.g.*, weights, biases, or neurons), the computational requirements of training and running a model could be reduced quite significantly, as lots of floating-point operations such as multiplication and addition associated with those zero valued elements can be avoided. With the support of sparse representation formats [Bell and Garland, 2009; Nagasaka et al., 2018; Yang et al., 2018; Li et al., 2018], it further allows a sparse model to take less memory space while generating less memory footprints at use and to communicate little at distributing.

Network pruning [Reed, 1993] is a promising avenue to realize such sparsity in neural networks. In essence, pruning is done by removing parameters (synapses)

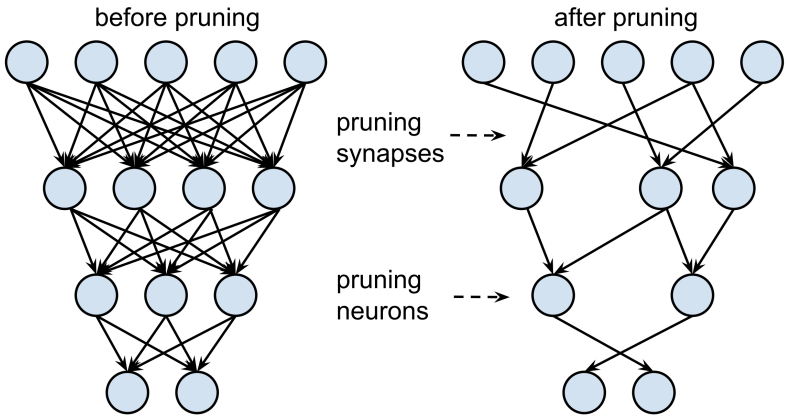


Figure 2.1: Illustration of pruning a neural network. After pruning some elements of a networksuch as synapses (parameters) or neurons (activations), the network becomes much simpler, potentially rendering increased computational efficiency. Figure from Han et al. [2015b].

or activations (neurons) based on a diverse set of principles (see Figure 2.1). By removing many of these elements, sparsity in neural networks can be achieved, and hence, computational and memory complexity of such neural network models can decrease quite effectively.

Not just for computational efficiency, network pruning can also be interpreted more generally and viewed as achieving an ultimate goal in machine learning, which is model selection: finding an optimal model that balances between complexity of the model and its generalization performance. A conventional wisdom in machine learning is that if the model is too complex, it can overfit the training data resulting in poor generalization performance [Vapnik, 1999]. By removing parts of networks or reducing the number of learnable parameters, network pruning can be used as a means to avoid overfitting and improving generalization; *i.e.*, finding a smallest model that performs the best. Network pruning therefore shares a similar goal in architecture search [Zoph and Le, 2017] as in searching for a better model in terms of efficiency, while the latter focuses more on optimality of performance.

2.1.2 History of network pruning

Network pruning is a research topic that has been studied for many decades, yet the basic idea remains pretty much the same, which is to address the following two issues: 1) a process of (re-)designing new architecture for a task that become increasingly more complex is prohibitively expensive, and 2) a system becomes

more complex with more number of parameters, posing a potential issue of overfitting data and degrading generalization performance. A rich body work has been produced to address these issues, and in fact, many of them can be categorized into two groups [Reed, 1993].

The first is those that augment the optimization objective with a sparsity inducing regularization term and decay some portion of parameters during backpropagation. The gist is to relate this regularization term with respect to some measure of network complexity. One obvious penalty term is the L1 regularizer. Different terms, but in similar spirit, have also been developed [Chauvin, 1989; Weigend et al., 1991]. Once the training is finished, the parameters of value close to zero can be thresholded out and set to be zero, and a retraining process may need to follow to compensate for any loss in performance.

Another group of approaches is based on a more straightforward strategy: define some saliency criterion to determine the importance of parameters, such that more important ones (based on the saliency criterion) are kept while the less important can be removed. A widely used saliency metric is the magnitude of parameters. That is, once the training is finished, delete parameters whose magnitudes are small in order. The network may need to be retrained to recover any drop in loss after pruning, and this process can be done iteratively based on some predefined pruning schedule.

The saliency can also be defined based on some kind of sensitivity measure. The sensitivity is usually defined as derivatives (potentially higher-order) of the loss function with respect to inputs. Precisely in this spirit, Mozer and Smolensky [1989] used sensitivity with respect to changes in hidden units (or neurons), and Karnin [1990] used instead the sensitivity with respect to parameters. Another seminal work is what is known as Optimal Brain Damage proposed in [LeCun et al., 1990] and its variants (*e.g.*, Hassibi et al. [1993]), where a parameter saliency is designed using the influence of small perturbations to the second order Taylor expansion of network error. We leave the audience to refer to Engelbrecht [2001] for more comprehensive review of sensitivity based pruning approaches.

2.1.3 Modern approach

Compared to neural networks used in early days, the size of modern neural networks has increased significantly (see Figure 2.2). This invoked a resurgence of interests in network pruning research in recent years as with the rise of deep learning. A seminal work is by Han et al. [2015b] in which the authors performed

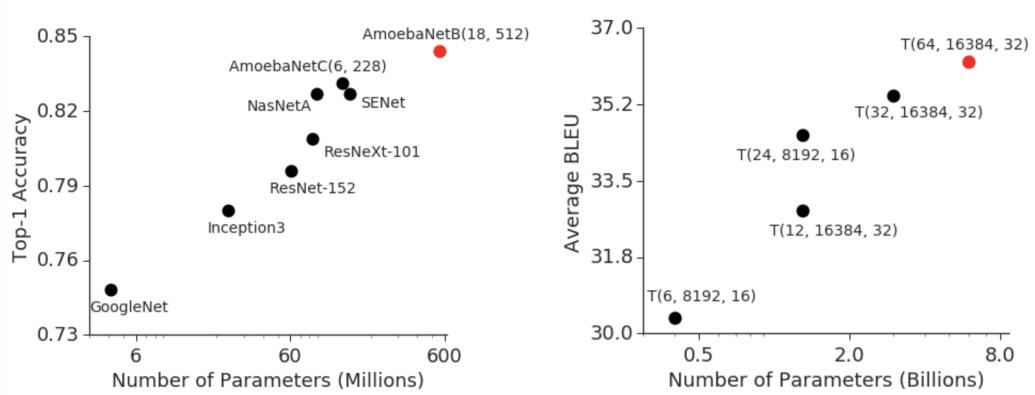


Figure 2.2: (left) Top-1 accuracy on ImageNet 2012 validation dataset [Deng et al., 2009] for representative state-of-the-art image classification models in recent years. (right) Average improvement in BLEU scores with increasing model size. Each point, T(L, H, A), depicts the performance of a Transformer with L encoder and L decoder layers, a feed-forward hidden dimension of H and A attention heads. There are strong correlations between the performance of a model and its size in both cases. Figure from Huang et al. [2019].

pruning based on the magnitude of parameters in an iterative fashion. Combined with quantization and a coding scheme, it is demonstrated further in Han et al. [2015a] that pruning can lead to a significant rate of compression of modern neural network models.

While the magnitude based pruning remains one of the most popular approaches [Gale et al., 2019], researchers have proposed various novel pruning strategies since then. For example, multiple Bayesian approaches have been proposed [Neklyudov et al., 2017; Louizos et al., 2018; Dai et al., 2018]. Approaches based on reinforcement learning have been proposed as well; for instance, Lin et al. [2017] performs pruning based on input data adaptively at runtime, while He et al. [2018] proposes to automate pruning by determining a sparsity ratio based on a policy gradient method.

Pruning can also be done dynamically [Bellec et al., 2018; Mostafa and Wang, 2019; Lin et al., 2020]. The term "dynamic" refers to their algorithm characteristic that it can re-enable previously removed parameters during the pruning – (re-)training cycle. These approaches can start from scratch without requiring a pretraining step.

Pruning can also be done in a structured way such that a group of elements can be pruned together rather than independently of each other. Potentially, this enables sparsity to be implemented more efficiently in practice. For example, all

parameters in a channel [He et al., 2017b] or a filter [Li et al., 2017] in a convolutional neural network can be pruned altogether, such that floating-point operations can be reduced without having to realize sparsity support matrix multiplications.

Another important aspect that this thesis studies is *when* to perform pruning. So far, existing approaches perform pruning either during or after the training process, meaning that parameters of neural networks have to be trained to some degree prior to (complete) pruning. Chapter 3 presents an efficient pruning algorithm that performs pruning on untrained neural networks at random initialization prior to training, which is considered the first pruning at initialization approach in the literature.

2.1.4 Sparse models in practice

It has been questioned whether sparse neural networks can realize computational benefits in practice; for example, in deep learning, a majority of computations comes from matrix multiplications, and a naive implementation of sparse matrix computations might not lead to a direct increase in computational efficiency.

Despite some doubts, many recent works have shown that sparsity can indeed improve deep learning efficiency quite significantly when implemented properly. For example, Kalchbrenner et al. [2018] applied pruning to a text-to-speech model (sparse WaveRNN) and demonstrated that it is possible to sample high-fidelity audio on a mobile CPU in real time. He et al. [2020] built and deployed a real-time neural text-to-speech system on CPU servers based on unstructured and blockwise sparsification methods, which brings about 160 times faster speed for real-time performance. Also, Elsen et al. [2020] developed efficient sparse kernels for convolutional neural networks that are implemented in existing software libraries and showed that sparse neural networks can outperform strong dense baselines, but with much less FLOPs.

2.2 Initializing neural networks

2.2.1 Intuition

It has been known that initialization of neural networks (*i.e.*, setting initial values of weights or other parameters) can have a critical impact on the success of their training. In essence, this has to do with avoiding vanishing or exploding gradients during backpropagation. If the weights are not properly scaled such that the output

of activation function (*e.g.*, sigmoid) is produced to be either too large or small, then the resulting gradient computed at the backward pass will likely saturate. This will cause the training to slow down, potentially too slow to ever converge.

2.2.2 Methods

The standard approach is random initialization. As discussed earlier, the goal is to set the initial parameters of the network such that the activations are neither too big nor too small, and the corresponding gradients do not explode nor vanish.

To achieve this, the initial weight values are drawn from a normal distribution with zero mean and a standard deviation, uniformly at random. In doing so, it is required that the distribution of the outputs of each neuron have a standard deviation of approximately 1. For the first layer, this can be achieved by normalizing input data. For intermediate layers, this can be achieved by making sure that the inputs to the activation function (*i.e.*, pre-activations) have a standard deviation of 1. For fully-connected layers, this can be done by setting the standard deviation for the normal distribution to be 1/(number of inputs to the neuron, or fan-in) [LeCun et al., 1998]. There are other standard initialization methods for different types of network architectures such as Glorot and Bengio [2010]; He et al. [2015], where the normalizing factor differs based on architecture specifications.

As with the development of new deep network architectures, initialization method as it was previously may not remain effective without modification. For example, an additive residual skip connection, which is what made ResNets [He et al., 2016] the most popular modern deep network architecture for image classification, will not maintain the distribution of pre-activations at each layer simply by having the same initialization methods as before. For this reason, normalization techniques such as batch normalization [Ioffe and Szegedy, 2015] have been utilized besides the initialization scheme. Nonetheless, Zhang et al. [2019b] proposed fixed-update initialization (Fixup), an initialization method that rescales the standard initialization of residual branches by adjusting for the network architecture. As a result, Fixup enables residual networks without normalization to achieve state-of-the-art performance in image classification and machine translation.

On the contrary to the aforementioned approach, [Xiao et al., 2018] proposed a different random initialization method that does not require any architectural tricks. They demonstrated that simply by using an appropriate initialization scheme, it is possible to train vanilla CNNs with 10,000 layers or more. They basically extended the initialization condition based on dynamical isometry, which is for the

equilibration of singular values of the input-output Jacobian matrix [Saxe et al., 2014], and a mean field theory for signal propagation [Poole et al., 2016], such that it can be applied to convolutional neural networks. In fact, these conditions require that the convolution operator be an orthogonal transformation in the sense that it is norm-preserving. This thesis in Chapter 4 further extends this to develop a new initialization condition for sparse neural networks.

2.2.3 Initialization for sparse neural networks

So far we have discussed initialization methods for the standard (densely parameterized) neural networks. In the context of pruning and sparse neural networks, the initialization process has not received much attention because initializing a network is simply thought as a preprocessing step that has to precede pruning; *i.e.*, initialization is done as the same as before for the standard networks. Also, iterative pruning approaches that perform fine-tuning to recover any loss in performance due to pruning gave an illusion that training a sparse network that is initialized may not be so effective.

This conventional approach of pruning after training recently started to raise the following question: what would happen if a sparse neural network obtained by pruning is (re-)initialized and trained? Recently, Frankle and Carbin [2019] has claimed inheriting the original random weights (corresponding to the sparse topology obtained by pruning) is important to find what is called winning lottery tickets, which can achieve the same performance as the dense network once trained (within the same number of training iterations). They basically argue that re-initializing the obtained sparse network should not be performed, if one is to win the lottery. On the contrary, Liu et al. [2019] demonstrated that this is not the case, at least for structured pruning, meaning that what is important is the trainable sparse topology itself, and therefore, once trainable sparse network is obtained, it is not necessary to restore the original random set of weights used before, and re-initializing should not damage the performance of the sparse network. Gale et al. [2019] also reported that they were not able to replicate the phenomenon for the lottery ticket hypothesis.

While these seemingly contradicting results could easily mislead, the lottery ticket hypothesis, at least in its originally claimed form, can be regarded as a special case of sparse network found reversely to establish the hypothesis; the algorithm to find the winning lottery tickets is not deterministic anyway, and the search process

has to run over and over again until it finally finds one that can be used to satisfy the hypothesis.

In fact, this thesis addresses further this issue of initializing sparse neural networks. Precisely, while Chapter 3 presents an algorithm to find trainable sparse neural networks that does not require restoring of the original random weights as in Liu et al. [2019], Chapter 4 demonstrates that the initialization for sparse neural networks is indeed important (yet not as in the way claimed in Frankle and Carbin [2019]) and can be done differently to improve the trainability of sparse neural networks.

2.3 Large-scale training of neural networks

2.3.1 Intuition

As with ever-growing neural network models and data sets, the amount of computations required for deep learning increased significantly, and it becomes challenging to train such large models in a reasonable amount of time. While there exist a variety of different techniques to accelerate neural network training (*e.g.*, batch normalization [Ioffe and Szegedy, 2015]), one most straightforward approach is by some form of *parallelism*, which is to parallelize training over multiple processing units using a distributed computing system [Dean et al., 2012; Das et al., 2016; Anil et al., 2018; Shallue et al., 2019].

Model parallelism enables distributed training by splitting a large model into small ones and computing the training updates (*e.g.*, gradient computations) – using the same training examples – across different processing units. Early works in model parallelism develop systems that can utilize computing clusters with thousands of CPU cores [Dean et al., 2012] or optimize computations in neural network training to train large models [Das et al., 2016]. Methods of ensembling and distillation can also be used in conjunction with model parallelism; for example, Shazeer et al. [2017] develop a mixture-of-experts layer and distribute the standard layers of the model as well as the gating network; also, Anil et al. [2018] propose codistillation, an online distillation variant to train multiple copies of a model in parallel to improve accuracy of a model and speed up training. While most algorithms for model parallelism are architecture and task-specific, Huang et al. [2019] develop a pipeline parallelism library called GPipe that provides flexibility of scaling any networks by pipelining different sub-sequences of layers on separate accelerators.

Recently, the trend of scaling up neural network training has focused on *data parallelism* as with recent hardware developments [Shallue et al., 2019; McCandlish et al., 2018]. As opposed to model parallelism, data parallelism distributes different training data (and the same model) across multiple processing units in a distributed system, such that the gradient (or higher-order derivatives) computations can be performed over different training examples which are then aggregated back in the central server to finally update the shared model. Without any need for (re-)designing network architectures, data parallelism is among the simplest approaches to speed up neural network training. Also, for a synchronized distributed training, data parallelism is equivalent to increasing minibatch size on a single computing unit, and as long as the training objective can be decomposed as a sum over training examples, data parallelism is model-agnostic and applicable to any neural network architecture.

2.3.2 Large-batch training

Neural networks are typically trained with a relatively small minibatch of examples in practice. The practical limits on compute resources to support large-batch training has been one main reason, but also it used to be believed that using a large batch degrades the generalization performance of the solution model. LeCun et al. [1998] suggest that using the smallest batch size of 1 can achieve a better solution because the noise in the gradient estimate using such a small minibatch can help escape from local minima. Similarly, Keskar et al. [2017] support the idea of using small batches by arguing that large-batch training tends to converge to sharp minimizers which lead to poorer generalization.

However, other studies show counter evidence that using large batch training does not necessarily lead to poor generalization, at least not as a sole reason. For example, Hoffer et al. [2017] claim that the generalization gap stems from the relatively small number of updates rather than the size of minibatch, and thus, can be removed by training longer.

Other works also demonstrated that large-batch training may not lose generalization performance by adapting the training regime in different ways; rather, it can indeed improve the training speed quite significantly. For example, Goyal et al. [2017] propose a different learning rate schedule and warmup scheme and train ResNet-50 on ImageNet using a large minibatch of 8192 images, within 1 hour across 256 GPUs. Smith et al. [2018] increase the batch size during training unsteady of decaying the learning rate and reduce the training time to be only 30

Team	Hardware	Software	Minibatch size	Time	Accuracy
He et al. [2016]	Tesla P100 × 8	Caffe	256	29 hours	75.3%
Goyal et al. [2017]	Tesla P100 × 256	Caffe2	8,192	1 hour	76.3%
You et al. [2018]	Xeon 8160 × 1600	Intel Caffe	16,000	31 minutes	75.3%
Akiba et al. [2017]	Tesla P100 × 1024	Chainer	32,768	15 minutes	74.9%
Jia et al. [2018]	Tesla P40 × 2048	TensorFlow	64,000	6.6 minutes	75.8%

Table 2.1: Results reproduced from [Akiba et al., 2017; Jia et al., 2018].

minutes. Others have reduced this training even further [Akiba et al., 2017; You et al., 2018; Jia et al., 2018] as shown in Table 2.1.

While the debate about whether small- or large-batch training continues, it is worth noting that previous results were bounded by limited resources in practice. A recent work by Shallue et al. [2019] conducted extensive experiments over different workloads of data set, model, and optimization algorithm, to measure the effects of batch size on neural network training. As one of the most comprehensive works to this date, they find no evidence that large-batch training, or effectively data parallelism, degrades generalization performance. Interestingly, McCandlish et al. [2018] show that a gradient noise scale can be used to predict the largest useful batch size across many domains and applications. Besides supervised learning, batch sizes of over a million timesteps have been used in reinforcement learning [OpenAI, 2018]. Chapter 5 in this thesis discusses further the effects of large-batch training on neural networks in conjuction with the influence of sparsity.

Chapter 3

SNIP: Single-shot Network Pruning based on Connection Sensitivity

Namhoon Lee
University of Oxford

Thalaiyasingam Ajanthan
University of Oxford

Philip H. S. Torr
University of Oxford

Abstract

Pruning large neural networks while maintaining their performance is often desirable due to the reduced space and time complexity. In existing methods, pruning is done within an iterative optimization procedure with either heuristically designed pruning schedules or additional hyperparameters, undermining their utility. In this work, we present a new approach that prunes a given network once at initialization prior to training. To achieve this, we introduce a saliency criterion based on connection sensitivity that identifies structurally important connections in the network for the given task. This eliminates the need for both pretraining and the complex pruning schedule while making it robust to architecture variations. After pruning, the sparse network is trained in the standard way. Our method obtains extremely sparse networks with virtually the same accuracy as the reference network on the MNIST,

CIFAR-10, and Tiny-ImageNet classification tasks and is broadly applicable to various architectures including convolutional, residual and recurrent networks. Unlike existing methods, our approach enables us to demonstrate that the retained connections are indeed relevant to the given task.

3.1 Introduction

Despite the success of deep neural networks in machine learning, they are often found to be highly overparametrized making them computationally expensive with excessive memory requirements. Pruning such large networks with minimal loss in performance is appealing for real-time applications, especially on resource-limited devices. In addition, compressed neural networks utilize the model capacity efficiently, and this interpretation can be used to derive better generalization bounds for neural networks [Arora et al., 2018].

In network pruning, given a large reference neural network, the goal is to learn a much smaller subnetwork that mimics the performance of the reference network. The majority of existing methods in the literature attempt to find a subset of weights from the pretrained reference network either based on a saliency criterion [Mozer and Smolensky, 1989; LeCun et al., 1990; Han et al., 2015b] or utilizing sparsity enforcing penalties [Chauvin, 1989; Carreira-Perpiñán and Idelbayev, 2018]. Unfortunately, since pruning is included as a part of an iterative optimization procedure, all these methods require many expensive *p*rune – retrain cycles and heuristic design choices with additional hyperparameters, making them non-trivial to extend to new architectures and tasks.

In this work, we introduce a saliency criterion that identifies connections in the network that are important to the given task in a data-dependent way before training. Specifically, we discover important connections based on their influence on the loss function at a variance scaling initialization, which we call connection sensitivity. Given the desired sparsity level, redundant connections are pruned once prior to training (*i.e.*, single-shot), and then the sparse pruned network is trained in the standard way. Our approach has several attractive properties:

- *Simplicity.* Since the network is pruned once prior to training, there is no need for pretraining and complex pruning schedules. Our method has no additional hyperparameters and once pruned, training of the sparse network is performed in the standard way.

- *Versatility.* Since our saliency criterion chooses structurally important connections, it is robust to architecture variations. Therefore our method can be applied to various architectures including convolutional, residual and recurrent networks with no modifications.

- *Interpretability.* Our method determines important connections with a mini-batch of data at single-shot. By varying this mini-batch used for pruning, our method enables us to verify that the retained connections are indeed essential for the given task.

We evaluate our method on MNIST, CIFAR-10, and Tiny-ImageNet classification datasets with widely varying architectures. Despite being the simplest, our method obtains extremely sparse networks with virtually the same accuracy as the existing baselines across all tested architectures. Furthermore, we investigate the relevance of the retained connections as well as the effect of the network initialization and the dataset on the saliency score.

3.2 Related work

Classical methods. Essentially, early works in network pruning can be categorized into two groups [Reed, 1993]: 1) those that utilize sparsity enforcing penalties; and 2) methods that prune the network based on some saliency criterion. The methods from the former category [Chauvin, 1989; Weigend et al., 1991; Ishikawa, 1996] augment the loss function with some sparsity enforcing penalty terms (*e.g.*, L_0 or L_1 norm), so that back-propagation effectively penalizes the magnitude of the weights during training. Then weights below a certain threshold may be removed. On the other hand, classical saliency criteria include the sensitivity of the loss with respect to the neurons [Mozer and Smolensky, 1989] or the weights [Karnin, 1990] and Hessian of the loss with respect to the weights [LeCun et al., 1990; Hassibi et al., 1993]. Since these criteria are heavily dependent on the scale of the weights and are designed to be incorporated within the learning process, these methods are prohibitively slow requiring many iterations of pruning and learning steps. Our approach identifies redundant weights from an architectural point of view and prunes them once at the beginning before training.

Modern advances. In recent years, the increased space and time complexities as well as the risk of overfitting in deep neural networks prompted a surge of further investigation in network pruning. While Hessian based approaches employ

the diagonal approximation due to its computational simplicity, impressive results (*i.e.*, extreme sparsity without loss in accuracy) are achieved using magnitude of the weights as the criterion [Han et al., 2015b]. This made them the de facto standard method for network pruning and led to various implementations [Guo et al., 2016; Carreira-Perpiñán and Idelbayev, 2018]. The magnitude criterion is also extended to recurrent neural networks [Narang et al., 2017], yet with heavily tuned hyperparameter setting. Unlike our approach, the main drawbacks of magnitude based approaches are the reliance on pretraining and the expensive prune – retrain cycles. Furthermore, since pruning and learning steps are intertwined, they often require highly heuristic design choices which make them non-trivial to be extended to new architectures and different tasks. Meanwhile, Bayesian methods are also applied to network pruning [Ullrich et al., 2017; Molchanov et al., 2017a] where the former extends the soft weight sharing in Nowlan and Hinton [1992] to obtain a sparse and compressed network, and the latter uses variational inference to learn the dropout rate which can then be used to prune the network. Unlike the above methods, our approach is simple and easily adaptable to any given architecture or task without modifying the pruning procedure.

Network compression in general. Apart from weight pruning, there are approaches focused on structured simplification such as pruning filters [Li et al., 2017; Molchanov et al., 2017b], structured sparsity with regularizers [Wen et al., 2016], low-rank approximation [Jaderberg et al., 2014], matrix and tensor factorization [Novikov et al., 2015], and sparsification using expander graphs [Prabhu et al., 2018] or Erdős-Rényi random graph [Mocanu et al., 2018]. In addition, there is a large body of work on compressing the representation of weights. A non-exhaustive list includes quantization [Gong et al., 2014], reduced precision [Gupta et al., 2015] and binary weights [Hubara et al., 2016]. In this work, we focus on weight pruning that is free from structural constraints and amenable to further compression schemes.

3.3 Neural network pruning

The main hypothesis behind the neural network pruning literature is that neural networks are usually overparametrized, and comparable performance can be obtained by a much smaller network [Reed, 1993] while improving generalization [Arora et al., 2018]. To this end, the objective is to learn a sparse network while

maintaining the accuracy of the standard reference network. Let us first formulate neural network pruning as an optimization problem.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, and a desired sparsity level κ (*i.e.*, the number of non-zero weights) neural network pruning can be written as the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \ , \\ \text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa \ . \end{aligned} \tag{3.1}$$

Here, $\ell(\cdot)$ is the standard loss function (*e.g.*, cross-entropy loss), $\mathbf{w}$ is the set of parameters of the neural network, m is the total number of parameters and $\|\cdot\|_0$ is the standard L_0 norm.

The conventional approach to optimize the above problem is by adding sparsity enforcing penalty terms [Chauvin, 1989; Weigend et al., 1991; Ishikawa, 1996]. Recently, Carreira-Perpiñán and Idelbayev [2018] attempts to minimize the above constrained optimization problem using the stochastic version of projected gradient descent (where the projection is accomplished by pruning). However, these methods often turn out to be inferior to saliency based methods in terms of resulting sparsity and require heavily tuned hyperparameter settings to obtain comparable results.

On the other hand, saliency based methods treat the above problem as selectively removing redundant parameters (or connections) in the neural network. In order to do so, one has to come up with a good criterion to identify such redundant connections. Popular criteria include magnitude of the weights, *i.e.*, weights below a certain threshold are redundant [Han et al., 2015b; Guo et al., 2016] and Hessian of the loss with respect to the weights, *i.e.*, the higher the value of Hessian, the higher the importance of the parameters [LeCun et al., 1990; Hassibi et al., 1993], defined as follows:

$$s_j = \begin{cases} |w_j| \ , & \text{for magnitude based} \\ \frac{w_j^2 H_{jj}}{2} \quad \text{or} \quad \frac{w_j^2}{2H_{jj}^{-1}} & \text{for Hessian based} \ . \end{cases} \tag{3.2}$$

Here, for connection j, s_j is the saliency score, w_j is the weight, and H_{jj} is the value of the Hessian matrix, where the Hessian $\mathbf{H} = \partial^2 L / \partial \mathbf{w}^2 \in \mathbb{R}^{m \times m}$. Considering Hessian based methods, the Hessian matrix is neither diagonal nor positive definite in general, approximate at best, and intractable to compute for large networks.

Despite being popular, both of these criteria depend on the scale of the weights and in turn require pretraining and are very sensitive to the architectural choices. For instance, different normalization layers affect the scale of the weights in a different way, and this would non-trivially affect the saliency score. Furthermore, pruning and the optimization steps are alternated many times throughout training, resulting in highly expensive *prune – retrain cycles*. Such an exorbitant requirement hinders the use of pruning methods in large-scale applications and raises questions about the credibility of the existing pruning criteria.

In this work, we design a criterion which directly measures the connection importance in a data-dependent manner. This alleviates the dependency on the weights and enables us to prune the network once at the beginning, and then the training can be performed on the sparse pruned network. Therefore, our method eliminates the need for the expensive prune – retrain cycles, and in theory, it can be an order of magnitude faster than the standard neural network training as it can be implemented using software libraries that support sparse matrix computations.

3.4 Single-shot network pruning based on connection sensitivity

Given a neural network and a dataset, our goal is to design a method that can selectively prune redundant connections for the given task in a data-dependent way even before training. To this end, we first introduce a criterion to identify important connections and then discuss its benefits.

3.4.1 Connection sensitivity: architectural perspective

Since we intend to measure the importance (or sensitivity) of each connection independently of its weight, we introduce auxiliary indicator variables $\mathbf{c} \in \{0,1\}^m$ representing the connectivity of parameters $\mathbf{w}$.[1] Now, given the sparsity level κ, Equation 3.1 can be correspondingly modified as:

$$\begin{aligned} \min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \,, \\ \text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \,, \\ \mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa \,, \end{aligned} \tag{3.3}$$

[1] Multiplicative coefficients (similar to $\mathbf{c}$) were also used for subset regression in Breiman [1995].

where $\odot$ denotes the Hadamard product. Compared to Equation 3.1, we have doubled the number of learnable parameters in the network and directly optimizing the above problem is even more difficult. However, the idea here is that since we have separated the weight of the connection ($\mathbf{w}$) from whether the connection is present or not ($\mathbf{c}$), we may be able to determine the importance of each connection by measuring its effect on the loss function.

For instance, the value of c_j indicates whether the connection j is active ($c_j = 1$) in the network or pruned ($c_j = 0$). Therefore, to measure the effect of connection j on the loss, one can try to measure the difference in loss when $c_j = 1$ and $c_j = 0$, keeping everything else constant. Precisely, the effect of removing connection j can be measured by,

$$\Delta L_j(\mathbf{w}; \mathcal{D}) = L(\mathbf{1} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{1} - \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D}) \ , \tag{3.4}$$

where $\mathbf{e}_j$ is the indicator vector of element j (*i.e.*, zeros everywhere except at the index j where it is one) and $\mathbf{1}$ is the vector of dimension m.

Note that computing ΔL_j for each $j \in \{1 \dots m\}$ is prohibitively expensive as it requires $m + 1$ (usually in the order of millions) forward passes over the dataset. In fact, since $\mathbf{c}$ is binary, L is not differentiable with respect to $\mathbf{c}$, and it is easy to see that ΔL_j attempts to measure the influence of connection j on the loss function in this discrete setting. Therefore, by relaxing the binary constraint on the indicator variables $\mathbf{c}$, ΔL_j can be approximated by the derivative of L with respect to c_j, which we denote $g_j(\mathbf{w}; \mathcal{D})$. Hence, the effect of connection j on the loss can be written as:

$$\Delta L_j(\mathbf{w}; \mathcal{D}) \approx g_j(\mathbf{w}; \mathcal{D}) = \frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j}\bigg|_{\mathbf{c}=\mathbf{1}} = \lim_{\delta \to 0} \frac{L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{c} - \delta\, \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})}{\delta}\bigg|_{\mathbf{c}=\mathbf{1}} . \tag{3.5}$$

In fact, $\partial L / \partial c_j$ is an infinitesimal version of ΔL_j, that measures the rate of change of L with respect to an infinitesimal change in c_j from $1 \to 1 - \delta$. This can be computed efficiently in one forward-backward pass using automatic differentiation, for all j at once. Notice, this formulation can be viewed as perturbing the weight w_j by a multiplicative factor δ and measuring the change in loss. This approximation is similar in spirit to Koh and Liang [2017] where they try to measure the influence of a datapoint to the loss function. Here we measure the influence of connections. Furthermore, $\partial L / \partial c_j$ is not to be confused with the gradient with respect to the weights ($\partial L / \partial w_j$), where the change in loss is measured with respect to an additive change in weight w_j.

Indeed, $\partial L/\partial c_j$ can be decomposed as $\partial L/\partial c_j = (\partial L/\partial w_j)w_j$ by the chain rule [2], and this, when taken as absolute value, is equivalent to the change in the loss in Equation 3.4 via the first order approximation of the loss. Pruning based on this quantity therefore means removing a parameter that has the least impact on the first order approximation.

Notably, our interest is to discover important (or sensitive) connections in the architecture, so that we can prune unimportant ones in single-shot, disentangling the pruning process from the iterative optimization cycles. To this end, we take the magnitude of the derivatives g_j as the saliency criterion. Note that if the magnitude of the derivative is high (regardless of the sign), it essentially means that the connection c_j has a considerable effect on the loss (either positive or negative), and it has to be preserved to allow learning on w_j. Based on this hypothesis, we define connection sensitivity as the normalized magnitude of the derivatives:

$$s_j = \frac{|g_j(\mathbf{w}; \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}; \mathcal{D})|} \,. \tag{3.6}$$

Once the sensitivity is computed, only the top-κ connections are retained, where κ denotes the desired number of non-zero weights. Precisely, the indicator variables $\mathbf{c}$ are set as follows:

$$c_j = \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0] \,, \quad \forall\, j \in \{1 \dots m\} \,, \tag{3.7}$$

where $\tilde{s}_\kappa$ is the κ-th largest element in the vector $\mathbf{s}$ and $\mathbb{1}[\cdot]$ is the indicator function. Here, for exactly κ connections to be retained, ties can be broken arbitrarily.

We would like to clarify that the above criterion (Equation 3.6) is different from the criteria used in early works by Mozer and Smolensky [1989] or Karnin [1990] which do not entirely capture the connection sensitivity. The fundamental idea behind them is to identify elements (*e.g.* weights or neurons) that least degrade the performance when removed. This means that their saliency criteria (*i.e.* $-\partial L/\partial \mathbf{w}$ or $-\partial L/\partial \boldsymbol{\alpha}$; $\boldsymbol{\alpha}$ refers to the connectivity of neurons), in fact, depend on the loss value before pruning, which in turn, require the network to be pre-trained and iterative optimization cycles to ensure minimal loss in performance. They also suffer from the same drawbacks as the magnitude and Hessian based methods as discussed in Section 3.3. In contrast, our saliency criterion (Equation 3.6) is designed to measure the sensitivity as to how much influence elements have on the loss function regardless of whether it is positive or negative. This criterion

[2] We will relate this further to a signal propagation perspective developed to understand pruning at initialization in Chapter 4.

Algorithm 1 SNIP: Single-shot Network Pruning based on Connection Sensitivity

Require: Loss function L, training dataset $\mathcal{D}$, sparsity level κ ▷ Refer Equation 3.3
Ensure: $\|\mathbf{w}^*\|_0 \leq \kappa$

$\mathbf{w} \leftarrow$ VarianceScalingInitialization ▷ Refer Section 3.4.2
$\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^b \sim \mathcal{D}$ ▷ Sample a mini-batch of training data
$s_j \leftarrow \frac{\left|g_j(\mathbf{w}; \mathcal{D}^b)\right|}{\sum_{k=1}^m \left|g_k(\mathbf{w}; \mathcal{D}^b)\right|}, \quad \forall j \in \{1 \dots m\}$ ▷ Connection sensitivity
$\tilde{\mathbf{s}} \leftarrow \text{SortDescending}(\mathbf{s})$
$c_j \leftarrow \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0], \quad \forall\, j \in \{1 \dots m\}$ ▷ Pruning: choose top-κ connections
$\mathbf{w}^* \leftarrow \arg\min_{\mathbf{w} \in \mathbb{R}^m} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})$ ▷ Regular training
$\mathbf{w}^* \leftarrow \mathbf{c} \odot \mathbf{w}^*$

alleviates the dependency on the value of the loss, eliminating the need for pre-training. These fundamental differences enable the network to be pruned at single-shot prior to training, which we discuss further in the next section.

3.4.2 Single-shot pruning at initialization

Note that the saliency measure defined in Equation 3.6 depends on the value of weights $\mathbf{w}$ used to evaluate the derivative as well as the dataset $\mathcal{D}$ and the loss function L. In this section, we discuss the effect of each of them and show that it can be used to prune the network in single-shot with initial weights $\mathbf{w}$.

Firstly, in order to minimize the impact of weights on the derivatives $\partial L / \partial c_j$, we need to choose these weights carefully. For instance, if the weights are too large, the activations after the non-linear function (*e.g.*, sigmoid) will be saturated, which would result in uninformative gradients. Therefore, the weights should be within a sensible range. In particular, there is a body of work on neural network initialization [Goodfellow et al., 2016] that ensures the gradients to be in a reasonable range, and our saliency measure can be used to prune neural networks at any such initialization.

Furthermore, we are interested in making our saliency measure robust to architecture variations. Note that initializing neural networks is a random process, typically done using normal distribution. However, if the initial weights have a fixed variance, the signal passing through each layer no longer guarantees to have the same variance, as noted by LeCun et al. [1998]. This would make the gradient and in turn our saliency measure, to be dependent on the architectural characteristics. Thus, we advocate the use of variance scaling methods (*e.g.*, Glorot and Bengio [2010]) to initialize the weights, such that the variance remains the same

throughout the network. By ensuring this, we empirically show that our saliency measure computed at initialization is robust to variations in the architecture.

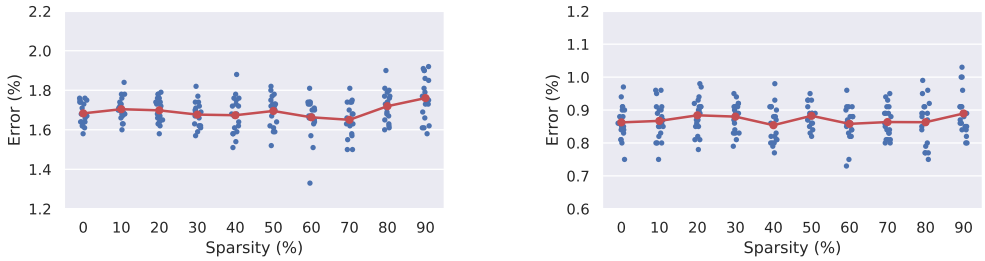
Next, since the dataset and the loss function defines the task at hand, by relying on both of them, our saliency criterion in fact discovers the connections in the network that are important to the given task. However, the practitioner needs to make a choice on whether to use the whole training set, or a mini-batch or the validation set to compute the connection saliency. Moreover, in case there are memory limitations (*e.g.*, large model or dataset), one can accumulate the saliency measure over multiple batches or take an exponential moving average. In our experiments, we show that using only one mini-batch of a reasonable number of training examples can lead to effective pruning.

Finally, in contrast to the previous approaches, our criterion for finding redundant connections is simple and directly based on the sensitivity of the connections. This allows us to effectively identify and prune redundant connections in a single step even before training. Then, training can be performed on the resulting pruned (sparse) network. We name our method SNIP for Single-shot Network Pruning, and the complete algorithm is given in Algorithm 1.

3.5 Experiments

We evaluate our method, SNIP, on MNIST, CIFAR-10 and Tiny-ImageNet classification tasks with a variety of network architectures. Our results show that SNIP yields extremely sparse models with minimal or no loss in accuracy across all tested architectures, while being much simpler than other state-of-the-art alternatives. We also provide clear evidence that our method prunes genuinely explainable connections rather than performing blind pruning.

Experiment setup For brevity, we define the sparsity level to be $\bar{\kappa} = (m - \kappa)/m \cdot 100\,(\%)$, where m is the total number of parameters and κ is the desired number of non-zero weights. For a given sparsity level $\bar{\kappa}$, the sensitivity scores are computed using a batch of 100 and 128 examples for MNIST and CIFAR experiments, respectively. After pruning, the pruned network is trained in the standard way. Specifically, we train the models using SGD with momentum of 0.9, batch size of 100 for MNIST and 128 for CIFAR experiments and the weight decay rate of 0.0005, unless stated otherwise. The initial learning rate is set to 0.1 and decayed by 0.1 at every 25k or 30k iterations for MNIST and CIFAR, respectively. Our algorithm requires no other hyperparameters or complex learning/pruning



(a) LeNet-300-100 (b) LeNet-5-Caffe

Figure 3.1: Test errors of LeNets pruned at varying sparsity levels $\bar{\kappa}$, where $\bar{\kappa} = 0$ refers to the reference network trained without pruning. Our approach performs as good as the reference network across varying sparsity levels on both the models.

schedules as in most pruning algorithms. We spare 10% of the training data as a validation set and used only 90% for training. For CIFAR experiments, we use the standard data augmentation (*i.e.*, random horizontal flip and translation up to 4 pixels) for both the reference and sparse models. The code can be found here: https://github.com/namhoonlee/snip-public.

3.5.1 Pruning LeNets with varying levels of sparsity

We first test our approach on two standard networks for pruning, LeNet-300-100 and LeNet-5-Caffe. LeNet-300-100 consists of three fully-connected (fc) layers with 267k parameters and LeNet-5-Caffe consists of two convolutional (conv) layers and two fc layers with 431k parameters. We prune the LeNets for different sparsity levels $\bar{\kappa}$ and report the performance in error on the MNIST image classification task. We run the experiment 20 times for each $\bar{\kappa}$ by changing random seeds for dataset and network initialization. The results are reported in Figure 3.1.

The pruned sparse LeNet-300-100 achieves performances similar to the reference ($\bar{\kappa} = 0$), only with negligible loss at $\bar{\kappa} = 90$. For LeNet-5-Caffe, the performance degradation is nearly invisible. Note that our saliency measure does not require the network to be pre-trained and is computed at random initialization. Despite such simplicity, our approach prunes LeNets quickly (single-shot) and effectively (minimal accuracy loss) at varying sparsity levels.

3.5.2 Comparisons to existing approaches

What happens if we increase the target sparsity to an extreme level? For example, would a model with only 1% of the total parameters still be trainable and perform

Method	Criterion	LeNet-300-100 $\bar{\kappa}$ (%)	LeNet-300-100 err. (%)	LeNet-5-Caffe $\bar{\kappa}$ (%)	LeNet-5-Caffe err. (%)	Pretrain	# Prune	Additional hyperparam.	Augment objective	Arch. constraints
Ref.	–	–	1.7	–	0.9	–	–	–	–	–
LWC	Magnitude	91.7	**1.6**	91.7	**0.8**	✓	many	✓	✗	✓
DNS	Magnitude	98.2	2.0	99.1	0.9	✓	many	✓	✗	✓
LC	Magnitude	99.0	3.2	99.0	1.1	✓	many	✓	✓	✗
SWS	Bayesian	95.6	1.9	99.5	1.0	✓	soft	✓	✓	✗
SVD	Bayesian	98.5	1.9	99.6	**0.8**	✓	soft	✓	✓	✗
OBD	Hessian	92.0	2.0	92.0	2.7	✓	many	✓	✗	✗
L-OBS	Hessian	98.5	2.0	99.0	2.1	✓	many	✓	✗	✓
SNIP (ours)	Connection sensitivity	95.0 98.0	**1.6** 2.4	98.0 99.0	**0.8** 1.1	✗	**1**	✗	✗	✗

Table 3.1: Pruning results on LeNets and comparisons to other approaches. Here, "many" refers to an arbitrary number often in the order of total learning steps, and "soft" refers to soft pruning in Bayesian based methods. Our approach is capable of pruning up to 98% for LeNet-300-100 and 99% for LeNet-5-Caffe with marginal increases in error from the reference network. Notably, our approach is considerably simpler than other approaches, with no requirements such as pretraining, additional hyperparameters, augmented training objective or architecture dependent constraints.

well? We test our approach for extreme sparsity levels (*e.g.*, up to 99% sparsity on LeNet-5-Caffe) and compare with various pruning algorithms as follows: LWC [Han et al., 2015b], DNS [Guo et al., 2016], LC [Carreira-Perpiñán and Idelbayev, 2018], SWS [Ullrich et al., 2017], SVD [Molchanov et al., 2017a], OBD [LeCun et al., 1990], L-OBS [Dong et al., 2017]. The results are summarized in Table 3.1.

We achieve errors that are comparable to the reference model, degrading approximately 0.7% and 0.3% while pruning 98% and 99% of the parameters in LeNet-300-100 and LeNet-5-Caffe respectively. For slightly relaxed sparsities (*i.e.*, 95% for LeNet-300-100 and 98% for LeNet-5-Caffe), the sparse models pruned by SNIP record better performances than the dense reference network. Considering 99% sparsity, our method efficiently finds 1% of the connections even before training, that are sufficient to learn as good as the reference network. Moreover, SNIP is competitive to other methods, yet it is unparalleled in terms of algorithm simplicity.

To be more specific, we enumerate some key points and non-trivial aspects of other algorithms and highlight the benefit of our approach. First of all, the aforementioned methods require networks to be fully trained (if not partly) before pruning. These approaches typically perform many pruning operations even if the network is well pretrained, and require additional hyperparameters (*e.g.*, pruning frequency in Guo et al. [2016], annealing schedule in Carreira-Perpiñán and Idelbayev [2018]). Some methods augment the training objective to handle

Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m → 507k	14.12 → 14.99	+0.87
	AlexNet-b	90.0	8.5m → 849k	13.92 → 14.50	+0.58
	VGG-C	95.0	10.5m → 526k	6.82 → 7.27	+0.45
	VGG-D	95.0	15.2m → 762k	6.76 → 7.09	+0.33
	VGG-like	97.0	15.0m → 449k	8.26 → 8.00	**−0.26**
Residual	WRN-16-8	95.0	10.0m → 548k	6.21 → 6.63	+0.42
	WRN-16-10	95.0	17.1m → 856k	5.91 → 6.43	+0.52
	WRN-22-8	95.0	17.2m → 858k	6.14 → 5.85	**−0.29**
Recurrent	LSTM-s	95.0	137k → 6.8k	1.88 → 1.57	**−0.31**
	LSTM-b	95.0	535k → 26.8k	1.15 → 1.35	+0.20
	GRU-s	95.0	104k → 5.2k	1.87 → 2.41	+0.54
	GRU-b	95.0	404k → 20.2k	1.71 → 1.52	**−0.19**

Table 3.2: Pruning results of the proposed approach on various modern architectures (before → after). AlexNets, VGGs and WRNs are evaluated on CIFAR-10, and LSTMs and GRUs are evaluated on the sequential MNIST classification task. The approach is generally applicable regardless of architecture types and models and results in a significant amount of reduction in the number of parameters with minimal or no loss in performance.

pruning together with training, increasing the complexity of the algorithm (*e.g.*, augmented Lagrangian in Carreira-Perpiñán and Idelbayev [2018], variational inference in Molchanov et al. [2017a]). Furthermore, there are approaches designed to include architecture dependent constraints (*e.g.*, layer-wise pruning schemes in Dong et al. [2017]).

Compared to the above approaches, ours seems to cost almost nothing; it requires no pretraining or additional hyperparameters, and is applied only once at initialization. This means that one can easily plug-in SNIP as a preprocessor before training neural networks. Since SNIP prunes the network at the beginning, we could potentially expedite the training phase by training only the survived parameters (*e.g.*, reduced expected FLOPs in Louizos et al. [2018]). Notice that this is not possible for the aforementioned approaches as they obtain the maximum sparsity at the end of the process.

3.5.3 Various modern architectures

In this section we show that our approach is generally applicable to more complex modern network architectures including deep convolutional, residual and recurrent ones. Specifically, our method is applied to the following models:

- AlexNet-s and AlexNet-b: Models similar to Krizhevsky et al. [2012] in terms of the number of layers and size of kernels. We set the size of fc layers to 512 (AlexNet-s) and to 1024 (AlexNet-b) to adapt for CIFAR-10 and use strides of 2 for all conv layers instead of using pooling layers.

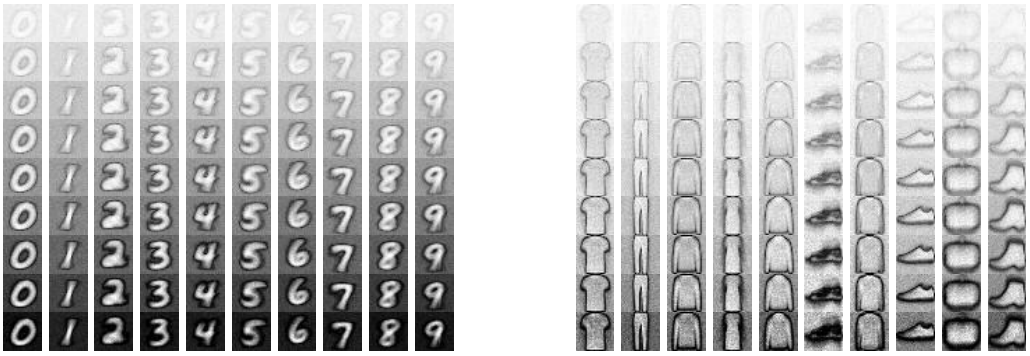
- VGG-C, VGG-D and VGG-like: Models similar to the original VGG models described in Simonyan and Zisserman [2015]. VGG-like [Zagoruyko, 2015] is a popular variant adapted for CIFAR-10 which has one less fc layers. For all VGG models, we set the size of fc layers to 512, remove dropout layers to avoid any effect on sparsification and use batch normalization instead.

- WRN-16-8, WRN-16-10 and WRN-22-8: Same models as in Zagoruyko and Komodakis [2016].

- LSTM-s, LSTM-b, GRU-s, GRU-b: One layer RNN networks with either LSTM [Zaremba et al., 2014] or GRU [Cho et al., 2014] cells. We develop two unit sizes for each cell type, 128 and 256 for $\{\cdot\}$-s and $\{\cdot\}$-b, respectively. The model is adapted for the sequential MNIST classification task, similar to Le et al. [2015]. Instead of processing pixel-by-pixel, however, we perform row-by-row processing (*i.e.*, the RNN cell receives each row at a time).

The results are summarized in Table 3.2. Overall, our approach prunes a substantial amount of parameters in a variety of network models with minimal or no loss in accuracy ($< 1\%$). Our pruning procedure does not need to be modified for specific architectural variations (*e.g.*, recurrent connections), indicating that it is indeed versatile and scalable. Note that prior art that use a saliency criterion based on the weights (*i.e.*, magnitude or Hessian based) would require considerable adjustments in their pruning schedules as per changes in the model.

We note of a few challenges in directly comparing against others: different network specifications, learning policies, datasets and tasks. Nonetheless, we provide a few comparison points that we found in the literature. On CIFAR-10, SVD prunes 97.9% of the connections in VGG-like with no loss in accuracy (ours: 97% sparsity) while SWS obtained 93.4% sparsity on WRN-16-4 but with a non-negligible loss in accuracy of 2%. There are a couple of works attempting to prune RNNs (*e.g.*, GRU in Narang et al. [2017] and LSTM in See et al. [2016]). Even though these methods are specifically designed for RNNs, none of them are able to obtain extreme sparsity without substantial loss in accuracy reflecting the challenges of pruning RNNs. To the best of our knowledge, we are the first to demonstrate



(a) MNIST (b) Fashion-MNIST

Figure 3.2: Visualizations of pruned parameters of the first layer in LeNet-300-100; the parameters are reshaped to be visualized as an image. Each column represents the visualizations for a particular class obtained using a batch of 100 examples with varying levels of sparsity $\bar{\kappa}$, from 10 (top) to 90 (bottom). Bright pixels indicate that the parameters connected to these region had high importance scores (s) and survived from pruning. As the sparsity increases, the parameters connected to the discriminative part of the image for classification survive and the irrelevant parts get pruned.

on convolutional, residual and recurrent networks for extreme sparsities without requiring additional hyperparameters or modifying the pruning procedure.

3.5.4 Understanding which connections are being pruned

So far we have shown that our approach can prune a variety of deep neural network architectures for extreme sparsities without losing much on accuracy. However, it is not clear yet which connections are actually being pruned away or whether we are pruning the right (*i.e.*, unimportant) ones. What if we could actually peep through our approach into this inspection?

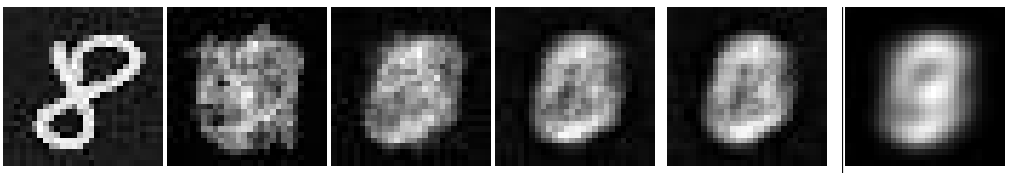
Consider the first layer in LeNet-300-100 parameterized by $\mathbf{w}_{l=1} \in \mathbb{R}^{784\times300}$. This is a layer fully connected to the input where input images are of size $28 \times 28 = 784$. In order to understand which connections are retained, we can visualize the binary connectivity mask for this layer $\mathbf{c}_{l=1}$, by averaging across columns and then reshaping the vector into 2D matrix (*i.e.*, $\mathbf{c}_{l=1} \in \{0,1\}^{784\times300} \rightarrow \mathbb{R}^{784} \rightarrow \mathbb{R}^{28\times28}$). Recall that our method computes $\mathbf{c}$ using a mini-batch of examples. In this experiment, we curate the mini-batch of examples of the same class and see which weights are retained for that mini-batch of data. We repeat this experiment for all classes (*i.e.*, digits for MNIST and fashion items for Fashion-MNIST) with varying sparsity levels $\bar{\kappa}$. The results are displayed in Figure 3.2 (see Appendix 3.A for more results).

The results are significant; important connections seem to reconstruct either the complete image (MNIST) or silhouettes (Fashion-MNIST) of input class. When we use a batch of examples of the digit 0 (*i.e.*, the first column of MNIST results), for example, the parameters connected to the foreground of the digit 0 survive from pruning while the majority of background is removed. Also, one can easily determine the identity of items from Fashion-MNIST results. This clearly indicates that our method indeed prunes the *unimportant* connections in performing the classification task, receiving signals only from the most discriminative part of the input. This stands in stark contrast to other pruning methods from which carrying out such inspection is not straightforward.

3.5.5 Effects of data and weight initialization

Recall that our connection saliency measure depends on the network weights $\mathbf{w}$ as well as the given data $\mathcal{D}$ (Section 3.4.2). We study the effect of each of these in this section.

Effect of data. Our connection saliency measure depends on a mini-batch of train examples $\mathcal{D}^b$ (see Algorithm 1). To study the effect of data, we vary the batch size used to compute the saliency ($|\mathcal{D}^b|$) and check which connections are being pruned as well as how much performance change this results in on the corresponding sparse network. We test with LeNet-300-100 to visualize the remaining parameters, and set the sparsity level $\bar{\kappa} = 90$. Note that the batch size used for training remains the same as 100 for all cases. The results are displayed in Figure 3.3.



$\lvert\mathcal{D}^b\rvert = 1$	$\lvert\mathcal{D}^b\rvert = 10$	$\lvert\mathcal{D}^b\rvert = 100$	$\lvert\mathcal{D}^b\rvert = 1000$	$\lvert\mathcal{D}^b\rvert = 10000$	train set
(1.94%)	(1.72%)	(1.64%)	(1.56%)	(1.40%)	–

Figure 3.3: The effect of different batch sizes: (top-row) survived parameters in the first layer of LeNet-300-100 from pruning visualized as images; (bottom-row) the performance in errors of the pruned networks. For $|\mathcal{D}^b| = 1$, the sampled example was 8; our pruning precisely retains the valid connections. As $|\mathcal{D}^b|$ increases, survived parameters get close to the average of all examples in the train set (last column), and the error decreases.

Effect of initialization. Our approach prunes a network at a stochastic initialization as discussed. We study the effect of the following initialization methods: 1) RN (random normal), 2) TN (truncated random normal), 3) VS-X (a variance scaling method using Glorot and Bengio [2010]), and 4) VS-H (a variance scaling method He et al. [2015]). We test on LeNets and RNNs on MNIST and run 20 sets of experiments by varying the seed for initialization. We set the sparsity level $\bar{\kappa} = 90$, and train with Adam optimizer [Kingma and Ba, 2015] with learning rate of 0.001 without weight decay. Note that for training VS-X initialization is used in all the cases. The results are reported in Figure 3.3.

For all models, VS-H achieves the best performance. The differences between initializers are marginal on LeNets, however, variance scaling methods indeed turns out to be essential for complex RNN models. This effect is significant especially for GRU where without variance scaling initialization, the pruned networks are unable to achieve good accuracies, even with different optimizers. Overall, initializing with a variance scaling method seems crucial to making our saliency measure reliable and model-agnostic.

Init.	LeNet-300-100	LeNet-5-Caffe	LSTM-s	GRU-s
RN	$1.90 \pm (0.09)$	$0.89 \pm (0.04)$	$2.93 \pm (0.20)$	$47.61 \pm (20.49)$
TN	$1.96 \pm (0.11)$	$0.87 \pm (0.05)$	$3.03 \pm (0.17)$	$46.48 \pm (22.25)$
VS-X	$1.91 \pm (0.10)$	$0.88 \pm (0.07)$	$1.48 \pm (0.09)$	$\mathbf{1.80} \pm (0.10)$
VS-H	$\mathbf{1.88} \pm (0.10)$	$\mathbf{0.85} \pm (0.05)$	$\mathbf{1.47} \pm (0.08)$	$\mathbf{1.80} \pm (0.14)$

Table 3.3: The effect of initialization on our saliency score. We report the classification errors ($\pm$std). Variance scaling initialization (VS-X, VS-H) improves the performance, especially for RNNs.

3.5.6 Fitting random labels

To further explore the use cases of SNIP, we run the experiment introduced in Zhang et al. [2017] and check whether the sparse network obtained by SNIP memorizes the dataset. Specifically, we train LeNet-5-Caffe for both the reference model and pruned model (with $\bar{\kappa} = 99$) on MNIST with either true or randomly shuffled labels. To compute the connection sensitivity, always true labels are used. The results are plotted in Figure 3.4.

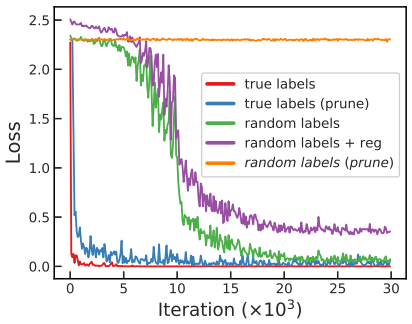


Figure 3.4: The sparse model pruned by SNIP does not fit the random labels.

Given true labels, both the reference (red) and pruned (blue) models quickly reach to almost zero training loss. However, the reference model provided with random labels (green) also reaches to very low training loss, even with an explicit L2 regularizer (purple), indicating that neural networks have enough capacity to memorize completely random data. In contrast, the model pruned by SNIP (orange) fails to fit the random labels (high training error). The potential explanation is that the pruned network does not have sufficient capacity to fit the random labels, but it is able to classify MNIST with true labels, reinforcing the significance of our saliency criterion. It is possible that a similar experiment can be done with other pruning methods [Molchanov et al., 2017a], however, being simple, SNIP enables such exploration much easier. We provide a further analysis on the effect of varying $\bar{\kappa}$ in Appendix 3.B.

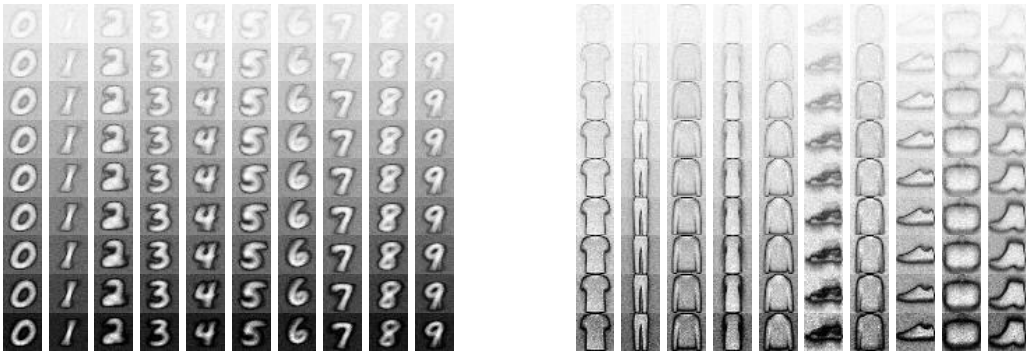
3.6 Discussion and future work

In this work, we have presented a new approach, SNIP, that is simple, versatile and interpretable; it prunes irrelevant connections for a given task at single-shot prior to training and is applicable to a variety of neural network models without modifications. While SNIP results in extremely sparse models, we find that our connection sensitivity measure itself is noteworthy in that it diagnoses important connections in the network from a purely untrained network. We believe that this opens up new possibilities beyond pruning in the topics of understanding of neural network architectures, multi-task transfer learning and structural regularization, to name a few. In addition to these potential directions, we intend to explore the generalization capabilities of sparse networks.

Acknowledgements

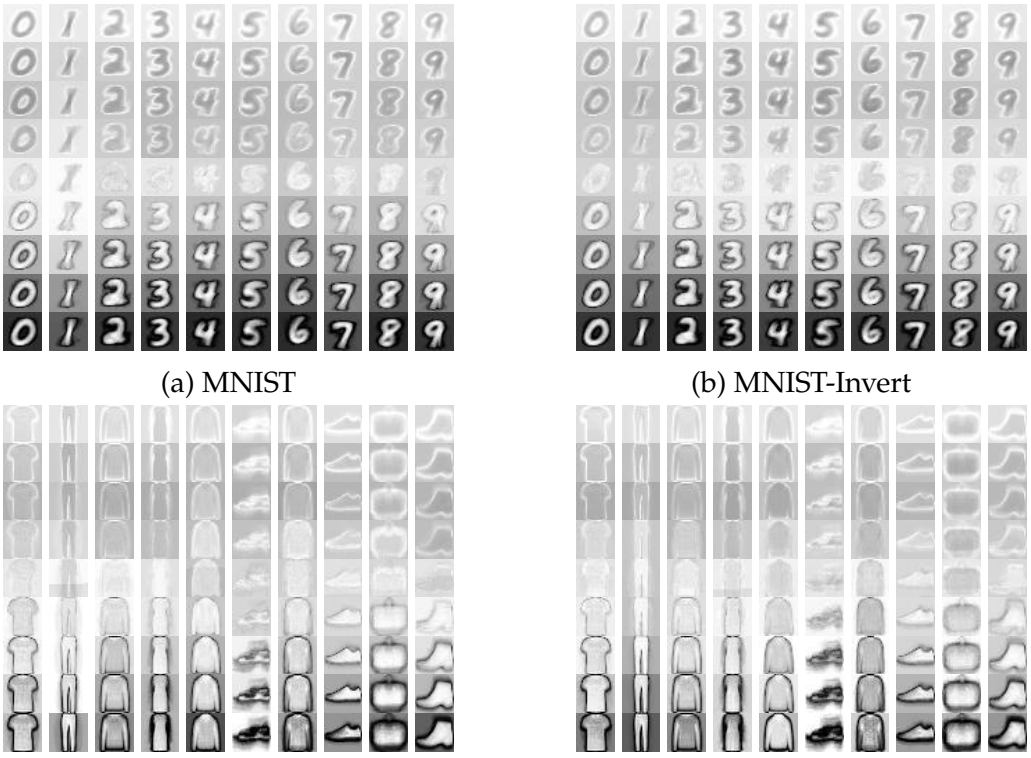
This work was supported by the Korean Government Graduate Scholarship, the ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1 and EPSRC/MURI grant EP/N019474/1. We would also like to acknowledge the Royal Academy of Engineering and FiveAI.

3.A Visualizing pruned parameters on (inverted) (fashion-)mnist



(a) MNIST-Invert (b) Fashion-MNIST-Invert

Figure 3.5: Results of pruning with SNIP on inverted (Fashion-)MNIST (*i.e.*, dark and bright regions are swapped). Notably, even if the data is inverted, the results are the same as the ones on the original (Fashion-)MNIST in Figure 3.2.



(a) MNIST (b) MNIST-Invert

(c) Fashion-MNIST (d) Fashion-MNIST-Invert

Figure 3.6: Results of pruning with $\partial L/\partial \mathbf{w}$ on the original and inverted (Fashion-)MNIST. Notably, compared to the case of using SNIP (Figures 3.2 and 3.5), the results are different: Firstly, the results on the original (Fashion-)MNIST (*i.e.*, (a) and (c) above) are not the same as the ones using SNIP (*i.e.*, (a) and (b) in Figure 3.2). Moreover, the pruning patterns are inconsistent with different sparsity levels, either intra-class or inter-class. Furthermore, using $\partial L/\partial \mathbf{w}$ results in different pruning patterns between the original and inverted data in some cases (*e.g.*, the 2^{nd} columns between (c) and (d)).

3.B Fitting random labels: varying sparsity levels

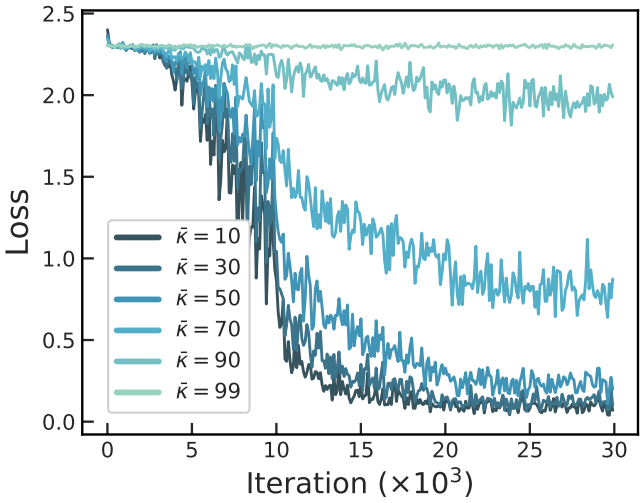


Figure 3.7: The effect of varying sparsity levels ($\bar{\kappa}$). The lower $\bar{\kappa}$ becomes, the lower training loss is recorded, meaning that a network with more parameters is more vulnerable to fitting random labels. Recall, however, that all pruned models are able to learn to perform the classification task without losing much accuracy (see Figure 3.1). This potentially indicates that the pruned network does not have sufficient capacity to fit the random labels, but it is capable of performing the classification.

3.C Tiny-imagenet

Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m → 507k	62.52 → 65.27	+2.75
	AlexNet-b	90.0	8.5m → 849k	62.76 → 65.54	+2.78
	VGG-C	95.0	10.5m → 526k	56.49 → 57.48	+0.99
	VGG-D	95.0	15.2m → 762k	56.85 → 57.00	+0.15
	VGG-like	95.0	15.0m → 749k	54.86 → 55.73	+0.87

Table 3.4: Pruning results of SNIP on Tiny-ImageNet (before → after). Tiny-ImageNet[3] is a subset of the full ImageNet: there are 200 classes in total, each class has 500 and 50 images for training and validation respectively, and each image has the spatial resolution of 64×64. Compared to CIFAR-10, the resolution is doubled, and to deal with this, the stride of the first convolution in all architectures is doubled, following the standard practice for this dataset. In general, the Tiny-ImageNet classification task is considered much more complex than MNIST or CIFAR-10. Even on Tiny-ImageNet, however, SNIP is still able to prune a large amount of parameters with minimal loss in performance. AlexNet models lose more accuracies than VGGs, which may be attributed to the fact that the first convolution stride for AlexNet is set to be 4 (by its design of no pooling) which is too large and could lead to high loss of information when pruned.

3.D Architecture details

Module	Weight	Stride	Bias	BatchNorm	ReLU
Conv	$[11, 11, 3, 96]$	$[2, 2]$	$[96]$	✓	✓
Conv	$[5, 5, 96, 256]$	$[2, 2]$	$[256]$	✓	✓
Conv	$[3, 3, 256, 384]$	$[2, 2]$	$[384]$	✓	✓
Conv	$[3, 3, 384, 384]$	$[2, 2]$	$[384]$	✓	✓
Conv	$[3, 3, 384, 256]$	$[2, 2]$	$[256]$	✓	✓
Linear	$[256, 1024 \times k]$	–	$[1024 \times k]$	✓	✓
Linear	$[1024 \times k, 1024 \times k]$	–	$[1024 \times k]$	✓	✓
Linear	$[1024 \times k, c]$	–	$[c]$	✗	✗

Table 3.5: AlexNet-s ($k = 1$) and AlexNet-b ($k = 2$). In the last layer, c denotes the number of possible classes: $c = 10$ for CIFAR-10 and $c = 200$ for Tiny-ImageNet. The strides in the first convolution layer for Tiny-ImageNet are set $[4, 4]$ instead of $[2, 2]$ to deal with the increase in the image resolution.

[3] https://tiny-imagenet.herokuapp.com/

Module	Weight	Stride	Bias	BatchNorm	ReLU
Conv	$[3, 3, 3, 64]$	$[1, 1]$	$[64]$	✓	✓
Conv	$[3, 3, 64, 64]$	$[1, 1]$	$[64]$	✓	✓
Pool	–	$[2, 2]$	–	✗	✗
Conv	$[3, 3, 64, 128]$	$[1, 1]$	$[128]$	✓	✓
Conv	$[3, 3, 128, 128]$	$[1, 1]$	$[128]$	✓	✓
Pool	–	$[2, 2]$	–	✗	✗
Conv	$[3, 3, 128, 256]$	$[1, 1]$	$[256]$	✓	✓
Conv	$[3, 3, 256, 256]$	$[1, 1]$	$[256]$	✓	✓
Conv	$[1/3/3, 1/3/3, 256, 256]$	$[1, 1]$	$[256]$	✓	✓
Pool	–	$[2, 2]$	–	✗	✗
Conv	$[3, 3, 256, 512]$	$[1, 1]$	$[512]$	✓	✓
Conv	$[3, 3, 512, 512]$	$[1, 1]$	$[512]$	✓	✓
Conv	$[1/3/3, 1/3/3, 512, 512]$	$[1, 1]$	$[512]$	✓	✓
Pool	–	$[2, 2]$	–	✗	✗
Conv	$[3, 3, 512, 512]$	$[1, 1]$	$[512]$	✓	✓
Conv	$[3, 3, 512, 512]$	$[1, 1]$	$[512]$	✓	✓
Conv	$[1/3/3, 1/3/3, 512, 512]$	$[1, 1]$	$[512]$	✓	✓
Pool	–	$[2, 2]$	–	✗	✗
Linear	$[512, 512]$	–	$[512]$	✓	✓
Linear	$[512, 512]$	–	$[512]$	✓	✓
Linear	$[512, c]$	–	$[c]$	✗	✗

Table 3.6: VGG-C/D/like. In the last layer, c denotes the number of possible classes: $c = 10$ for CIFAR-10 and $c = 200$ for Tiny-ImageNet. The strides in the first convolution layer for Tiny-ImageNet are set $[2, 2]$ instead of $[1, 1]$ to deal with the increase in the image resolution. The second Linear layer is only used in VGG-C/D.

Chapter 4

A Signal Propagation Perspective for Pruning Neural Networks at Initialization

Namhoon Lee
University of Oxford

Thalaiyasingam Ajanthan
Australian National University

Stephen Gould
Australian National University

Philip H. S. Torr
University of Oxford

Abstract

Network pruning is a promising avenue for compressing deep neural networks. A typical approach to pruning starts by training a model and then removing redundant parameters while minimizing the impact on what is learned. Alternatively, a recent approach shows that pruning can be done at initialization prior to training, based on a saliency criterion called connection sensitivity. However, it remains unclear exactly why pruning an untrained, randomly initialized neural network is effective. In this work, by noting connection sensitivity as a form of gradient, we formally characterize initialization conditions to ensure

reliable connection sensitivity measurements, which in turn yields effective pruning results. Moreover, we analyze the signal propagation properties of the resulting pruned networks and introduce a simple, data-free method to improve their trainability. Our modifications to the existing pruning at initialization method lead to improved results on all tested network models for image classification tasks. Furthermore, we empirically study the effect of supervision for pruning and demonstrate that our signal propagation perspective, combined with unsupervised pruning, can be useful in various scenarios where pruning is applied to non-standard arbitrarily-designed architectures.

4.1 Introduction

Deep learning has made great strides in machine learning and been applied to various fields from computer vision and natural language processing, to health care and playing games [LeCun et al., 2015]. Despite the immense success, however, it remains challenging to deal with the excessive computational and memory requirements of large neural network models. To this end, lightweight models are often preferred, and *network pruning*, a technique to reduce parameters in a network, has been widely employed to compress deep neural networks [Han et al., 2015a]. Nonetheless, designing pruning algorithms has been often purely based on ad-hoc intuition lacking rigorous underpinning, partly because pruning was typically carried out after training the model as a post-processing step or interwoven with the training procedure, without adequate tools to analyze.

Recently, Lee et al. [2019] have shown that pruning can be done on randomly initialized neural networks in a single-shot prior to training (*i.e.*, *pruning at initialization*). They empirically showed that as long as the initial random weights are drawn from appropriately scaled Gaussians (*e.g.*, Glorot and Bengio [2010]), their pruning criterion called connection sensitivity can be used to prune deep neural networks, often to an extreme level of sparsity while maintaining good accuracy once trained. However, it remains unclear as to why pruning at initialization is effective, how it should be understood theoretically and whether it can be extended further.

In this work, we first look into the effect of initialization on pruning, and find that initial weights have critical impact on connection sensitivity, and therefore,

pruning results. Deeper investigation shows that connection sensitivity is determined by an interplay between gradients and weights. Therefore when the initial weights are not chosen appropriately, the propagation of input signals into layers of these random weights can result in saturating error signals (*i.e.*, gradients) under backpropagation, and hence unreliable connection sensitivity, potentially leading to a catastrophic pruning failure.

This result leads us to develop a signal propagation perspective for pruning at initialization, and to provide a formal characterization of how a network needs to be initialized for reliable connection sensitivity measurements and in turn effective pruning. Precisely, we show that a sufficient condition to ensure faithful[1] connection sensitivity is *layerwise dynamical isometry*, which is defined as all singular values of the layerwise Jacobians being concentrated around 1. Our signal propagation perspective is inspired by the recent literature on dynamical isometry and mean field theory [Saxe et al., 2014; Poole et al., 2016; Schoenholz et al., 2017; Pennington et al., 2017], in which the general signal propagation in neural networks is studied. We extend this result to understanding and improving pruning at initialization.

Moreover, we study signal propagation in the pruned sparse networks and its effect on trainability. We find that pruning neural networks can indeed break dynamical isometry, and hence, hinders signal propagation and degrades the training performance of the resulting sparse network. In order to address this issue, we propose a simple, yet effective data-free method to recover the layerwise orthogonality given the sparse topology, which in turn improves the training performance of the compressed network significantly. Our analysis further reveals that in addition to signal propagation, the choice of pruning method and sparsity level can influence trainability in sparse neural networks.

Perfect layerwise dynamical isometry cannot always be ensured in the modern networks that have components such as ReLU nonlinearities [Pennington et al., 2017] and/or batch normalization [Yang et al., 2019]. Even in such cases, however, our experiments on various modern architectures (including convolutional and residual neural networks) indicate that connection sensitivity computed based on layerwise dynamical isometry is robust and consistently outperforms pruning based on other initialization schemes. This indicates that the signal propagation

[1] The term faithful is used to describe signals propagating in a network isometrically with minimal amplification or attenuation, and borrowed from Saxe et al. [2014], the first work to introduce dynamical isometry.

perspective is not only important to theoretically understand pruning at initialization, but also it improves the results of pruning for a range of networks of practical interest.

Furthermore, this signal propagation perspective for pruning poses another important question: how informative is the error signal computed on randomly initialized networks, or can we prune neural networks even without supervision? To understand this, we compute connection sensitivity scores with different unsupervised surrogate losses and evaluate the pruning results. Interestingly, our results indicate that we can in fact prune networks in an unsupervised manner to extreme sparsity levels without compromising accuracy, and it often compares competitively to pruning with supervision. Moreover, we test if pruning at initialization can be extended to obtain architectures that yield better performance than standard pre-designed architectures with the same number of parameters. In fact, this process, which we call *neural architecture sculpting*, compares favorably against hand-designed architectures, taking network pruning one step further towards neural architecture search.

4.2 Preliminaries

Pruning at initialization. The principle behind conventional approaches for network pruning is to find unnecessary parameters, such that by eliminating them the complexity of the model is reduced while minimizing the impact on what is learned [Reed, 1993]. Naturally, a typical pruning algorithm starts *after* convergence to a minimum or training to some degree. This pretraining requirement has been left unattended until Lee et al. [2019] recently showed that pruning can be performed on untrained networks at initiailzation prior to training. They proposed a method called SNIP which relies on a new saliency criterion, namely *connection sensitivity*, defined as follows:

$$s_j(\mathbf{w};\mathcal{D}) = \frac{|g_j(\mathbf{w};\mathcal{D})|}{\sum_{k=1}^{m}|g_k(\mathbf{w};\mathcal{D})|}\,, \qquad \text{where} \quad g_j(\mathbf{w};\mathcal{D}) = \frac{\partial L(\mathbf{c}\odot\mathbf{w};\mathcal{D})}{\partial c_j}\bigg|_{\mathbf{c}=\mathbf{1}}\,. \tag{4.1}$$

Here, s_j is the saliency of the parameter j, $\mathbf{w}\in\mathbb{R}^m$ is the network parameters, $\mathbf{c}\in\{0,1\}^m$ is the auxiliary indicator variables representing the connectivity of network parameters, m is the total number of parameters in the network, and $\mathcal{D}$ is a given dataset. Also, g_j is the derivative of the loss L with respect to c_j, which turns out to be an infinitesimal approximation of the change in the loss with respect to removing the parameter j. Designed to be computed at initialization, pruning

is performed by keeping top-κ (where κ denotes a desired sparsity level) salient parameters based on the above sensitivity scores.

Dynamical isometry and mean field theory. The success of training deep neural networks is due in large part to the initial weights [Hinton and Salakhutdinov, 2006; Glorot and Bengio, 2010; Pascanu et al., 2013]. In essence, the principle behind these random weight initializations is to have the mean squared singular value of a network's input-output Jacobian close to 1, so that on average, an error vector will preserve its norm under backpropagation; however, this is not sufficient to prevent amplification or attenuation of an error vector on worst case. A stronger condition that having as many singular values as possible near 1 is called *dynamical isometry* [Saxe et al., 2014]. Under this condition, error signals backpropagate isometrically through the network, approximately preserving its norm and all angles between error vectors. Alongside dynamical isometry, *mean field theory* is used to develop a theoretical understanding of signal propagation in neural networks with random parameters [Poole et al., 2016]. Precisely, the mean field approximation states that preactivations of wide, untrained neural networks can be captured as a Gaussian distribution. Recent works revealed a maximum depth through which signals can propagate at initialization, and verified that networks are trainable when signals can travel all the way through them [Schoenholz et al., 2017; Yang and Schoenholz, 2017; Xiao et al., 2018].

4.3 Signal propagation perspective to pruning random networks

Problem setup. Consider a fully-connected, feed-forward neural network with weight matrices $\mathbf{W}^l \in \mathbb{R}^{N \times N}$, biases $\mathbf{b}^l \in \mathbb{R}^N$, pre-activations $\mathbf{h}^l \in \mathbb{R}^N$, and post-activations $\mathbf{x}^l \in \mathbb{R}^N$, for $l \in \{1 \dots K\}$ up to K layers. Now, the feed-forward dynamics of a network can be written as,

$$\mathbf{x}^l = \phi(\mathbf{h}^l) \,, \qquad \mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l \,, \tag{4.2}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is an elementwise nonlinearity, and the input is denoted by $\mathbf{x}^0$. Given the network configuration, the parameters are initialized by sampling from a probability distribution, typically a zero mean Gaussian with scaled variance [LeCun et al., 1998; Glorot and Bengio, 2010].

4.3.1 Effect of initialization on pruning

It is observed in Lee et al. [2019] that pruning results tend to improve when initial weights are drawn from a scaled Gaussian, or so-called variance scaling initialization [LeCun et al., 1998; Glorot and Bengio, 2010; He et al., 2015]. As we wish to better understand the role of these random initial weights in pruning, we will examine the effect of varying initialization on the pruning results.

In essence, variance scaling schemes introduce normalization factors to adjust the variance σ of the weight sampling distribution, which can be summarized as $\sigma \to \frac{\alpha}{\psi_l}\sigma$, where ψ_l is a layerwise scalar that depends on an architecture specification such as the number of output neurons in the previous layer (*e.g.*, fan-in), and α is a global scalar throughout the network. Notice in case of a network with layers of the same width, the variance can be controlled by a single scalar $\gamma = \frac{\alpha}{\psi}$ as $\psi_l = \psi$ for all layers l. In particular, we take both linear and tanh multilayer perceptron networks (MLP) of layers $K = 7$ and width $N = 100$ on MNIST with $\sigma = 1$ as the default, similar to Saxe et al. [2014]. We initialize these networks with different γ, compute the connection sensitivity, prune it, and then visualize layerwise the resulting sparsity patterns $\mathbf{c}$ as well as the corresponding connection sensitivity used for pruning in Figure 4.1.

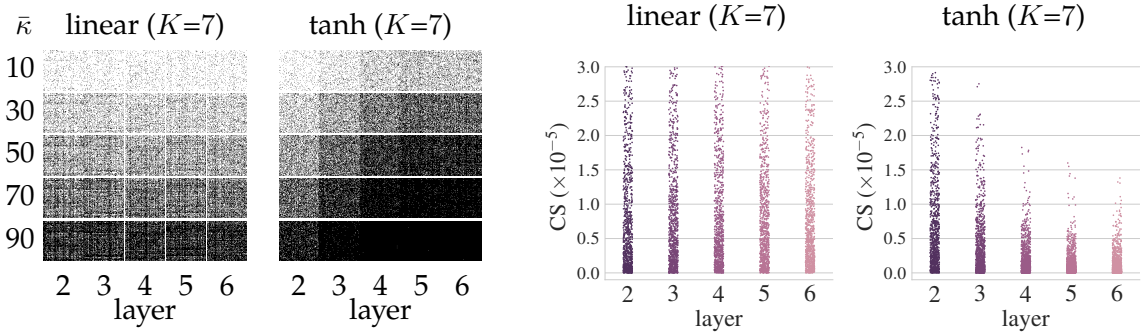


Figure 4.1: (left) layerwise sparsity patterns $c \in \{0, 1\}^{100\times100}$ obtained as a result of pruning for the sparsity level $\bar{\kappa} = \{10, .., 90\}$%. Here, black(0)/white(1) pixels refer to pruned/retained parameters; (right) connection sensitivities (cs) measured for the parameters in each layer. All networks are initialized with $\gamma = 1.0$. Unlike the linear case, the sparsity pattern for the tanh network is non-uniform over different layers. When pruning for a high sparsity level (*e.g.*, $\bar{\kappa} = 90\%$), this becomes critical and leads to poor learning capability as there are only a few parameters left in later layers. This is explained by the connection sensitivity plot which shows that for the nonlinear network parameters in later layers have saturating, lower connection sensitivities than those in earlier layers.

It is seen in the sparsity patterns that for the tanh network, unlike the linear case, more parameters tend to be pruned in the later layers than the earlier layers. As a result, this limits the learning capability of the subnetwork critically when a high sparsity level is requested; *e.g.*, for $\bar{\kappa} = 90\%$, only a few parameters in later layers are retained after pruning. This is explained by the connection sensitivity plot. The sensitivity of parameters in the nonlinear network tends to decrease towards the later layers, and therefore, choosing the top-κ parameters globally based on the sensitivity scores results in a subnetwork in which retained parameters are distributed highly non-uniformly and sparsely towards the end of the network. This result implies that the initial weights have a crucial effect on the connection sensitivity, and from there, the pruning results.

4.3.2 Gradient signal in connection sensitivity

We posit that the unreliability of connection sensitivity observed in Figure 4.1 is due to poor signal propagation: an initialization that projects the input signal to be strongly amplified or attenuated in the forward pass will saturate the error signal under backpropagation (*i.e.*, gradients), and hence will result in poorly calibrated connection sensitivity scores across layers, which will eventually lead to poor pruning results, potentially with complete disconnection of signal paths (*e.g.*, entire layer).

Precisely, we give the relationship between the connection sensitivity and the gradients as follows. From Equation 3.6, connection sensitivity is a normalized magnitude of gradients with respect to the connectivity parameters $\mathbf{c}$. Here, we use the vectorized notation where $\mathbf{w}$ denotes all learnable parameters and $\mathbf{c}$ denotes the corresponding connectivity parameters. From chain rule, we can write:

$$\left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial \mathbf{c}}\right|_{\mathbf{c}=\mathbf{1}} = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial (\mathbf{c} \odot \mathbf{w})}\right|_{\mathbf{c}=\mathbf{1}} \odot \mathbf{w} = \frac{\partial L(\mathbf{w}; \mathcal{D})}{\partial \mathbf{w}} \odot \mathbf{w} \,. \tag{4.3}$$

Therefore, $\partial L/\partial \mathbf{c}$ is the gradients $\partial L/\partial \mathbf{w}$ amplified (or attenuated) by the corresponding weights $\mathbf{w}$, *i.e.*, $\partial L/\partial c_j = \partial L/\partial w_j \, w_j$ for all $j \in \{1 \dots m\}$. Considering $\partial L/\partial c_j$ for a given j, since w_j does not depend on any other layers or signal propagation, the only term that depends on signal propagation in the network is the gradient term $\partial L/\partial w_j$. Hence, a necessary condition to ensure faithful $\partial L/\partial \mathbf{c}$ (and connection sensitivity) is that the gradients $\partial L/\partial \mathbf{w}$ need to be faithful. In the following section, we formalize this from a signal propagation perspective, and characterize an initial condition that ensures reliable connection sensitivity measurement.

4.3.3 Layerwise dynamical isometry

4.3.3.1 Gradients in terms of Jacobians

From the feed-forward dynamics of a network in Equation 4.2, the network's input-output Jacobian corresponding to a given input $\mathbf{x}^0$ can be written, by the chain rule of differentiation, as:

$$\mathbf{J}^{0,K} = \frac{\partial \mathbf{x}^K}{\partial \mathbf{x}^0} = \prod_{l=1}^{K} \mathbf{D}^l \mathbf{W}^l \,, \tag{4.4}$$

where $\mathbf{D}^l \in \mathbb{R}^{N \times N}$ is a diagonal matrix with entries $\mathbf{D}^l_{ij} = \phi'(h^l_i)\delta_{ij}$, with ϕ' denoting the derivative of nonlinearity ϕ, and $\delta_{ij} = \mathbb{1}[i = j]$ is the Kronecker delta. Here, we will use $\mathbf{J}^{k,l}$ to denote the Jacobian from layer k to layer l. Now, we give the relationship between gradients and Jacobians:

Proposition 4.3.1. Let $\epsilon = \partial L / \partial \mathbf{x}^K$ denote the error signal and $\mathbf{x}^0$ denote the input signal. Then,

1. the gradients satisfy:
$$\mathbf{g}^T_{\mathbf{w}^l} = \epsilon \, \mathbf{J}^{l,K} \mathbf{D}^l \otimes \mathbf{x}^{l-1} \,, \tag{4.5}$$
where $\mathbf{J}^{l,K} = \partial \mathbf{x}^K / \partial \mathbf{x}^l$ is the Jacobian from layer l to the output and $\otimes$ is the Kronecker product.

2. additionally, for linear networks, *i.e.*, when ϕ is the identity:
$$\mathbf{g}^T_{\mathbf{w}^l} = \epsilon \, \mathbf{J}^{l,K} \otimes \left(\mathbf{J}^{0,l-1} \mathbf{x}^0 + \mathbf{a} \right) \,, \tag{4.6}$$
where $\mathbf{J}^{0,l-1} = \partial \mathbf{x}^{l-1} / \partial \mathbf{x}^0$ is the Jacobian from the input to layer $l-1$ and $\mathbf{a} \in \mathbb{R}^N$ is a constant term that does not depend on $\mathbf{x}^0$.

Proof. This can be proved by an algebraic manipulation of the chain rule while using the feed-forward dynamics in Equation 4.2. We provide the full derivation in Appendix 4.A. □

Notice that the gradient at layer l constitutes both the backward propagation of the error signal ϵ up to layer l and the forward propagation of the input signal $\mathbf{x}^0$ up to layer $l-1$. Moreover, especially in the linear case, the signal propagation in both directions is governed by the corresponding Jacobians. We believe that this interpretation of gradients is useful as it sheds light on how signal propagation affects the gradients. To this end, we next analyze the conditions on the Jacobians, which would guarantee faithful signal propagation in the network, and consequently, faithful gradients.

4.3.3.2 Ensuring faithful gradients

Here, we first consider the layerwise signal propagation which would be useful to derive properties on the initialization to ensure faithful gradients. To this end, let us consider the layerwise Jacobian:

$$\mathbf{J}^{l-1,l} = \frac{\partial \mathbf{x}^l}{\partial \mathbf{x}^{l-1}} = \mathbf{D}^l \mathbf{W}^l \,. \tag{4.7}$$

Note that it is sufficient to have *layerwise dynamical isometry* in order to ensure faithful signal propagation in the network.

Definition 4.3.1. (*Layerwise dynamical isometry*) Let $\mathbf{J}^{l-1,l} = \frac{\partial \mathbf{x}^l}{\partial \mathbf{x}^{l-1}} \in \mathbb{R}^{N_l \times N_{l-1}}$ be the Jacobian matrix of layer l. The network is said to satisfy layerwise dynamical isometry if the singular values of $\mathbf{J}^{l-1,l}$ are concentrated near 1 for all layers, *i.e.*, for a given $\epsilon > 0$, the singular value σ_j satisfies $|1 - \sigma_j| \leq \epsilon$ for all j.

This would guarantee that the signal from layer l to $l-1$ (or vice versa) is propagated without amplification or attenuation in any of its dimension. From Proposition 4.3.1 and Equation 4.7, by induction, it is easy to show that if the layerwise signal propagation is faithful, the error and input signals will faithfully propagate throughout the network, resulting in faithful gradients.

For linear networks, $\mathbf{J}^{l-1,l} = W^l$. Therefore, one can initialize the weight matrix to be *orthogonal* such that $(\mathbf{W}^l)^T \mathbf{W}^l = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix of dimension N. In this case, all singular values of $\mathbf{W}^l$ are exactly 1 (*i.e.*, exact dynamical isometry), and such an initialization guarantees faithful gradients. While a linear network is of little practical use, we note that it helps to develop theoretical analysis and provides intuition as to why dynamical isometry is a useful measure.

For nonlinear networks, the diagonal matrix $\mathbf{D}^l$ needs to be accounted for as it depends on the pre-activations $\mathbf{h}^l$ at layer l. In this case, it is important to have the pre-activations $\mathbf{h}^l$ fall into the linear region of the nonlinear function ϕ. Precisely, mean-field theory assumes that for large-N limit, the empirical distribution of the pre-activations $\mathbf{h}^l$ converges to a Gaussian with zero mean and variance q^l, where the variance follows a recursion relation [Poole et al., 2016]. Therefore, to achieve layerwise dynamical isometry, the idea becomes to find a fixed point q^* such that $\mathbf{h}^l \sim \mathcal{N}(0, q^*)$ for all $l \in \{1 \dots K\}$. Such a fixed point makes $\mathbf{D}^l = \mathbf{D}$ for all layers, and therefore, the pre-activations are placed in the linear region of the nonlinearity.[2] Then, given the nonlinearity, one can find a rescaling such that

[2] Dynamical isometry can hold for antisymmetric sigmoidal activation functions (*e.g.*, tanh) as shown in Pennington et al. [2017]. A recent work by Tarnowski et al. [2019] have also shown that dynamical isometry is achievable irrespective of the activation function in ResNets.

$(\mathbf{DW}^l)^T(\mathbf{DW}^l) = (\mathbf{W}^l)^T\mathbf{W}^l/\sigma_w^2 = \mathbf{I}$. The procedure for finding the rescaling σ_w^2 for various nonlinearities are discussed in Pennington et al. [2017; 2018]. Also, this easily extends to convolutional neural networks using the initialization method in Xiao et al. [2018].

We note that dynamical isometry is in fact a weaker condition than layerwise dynamical isometry. However, in practice, the initialization suggested in the existing works [Pennington et al., 2017; Xiao et al., 2018], *i.e.*, orthogonal initialization for weight matrices in each layer with rescaling based on mean field theory, satisfy layerwise dynamical isometry, even though this term was not mentioned.

Now, recall from Section 4.3.1 that a network is pruned with a global threshold based on connection sensitivity, and from Section 4.3.2 that the connection sensitivity is the gradients scaled by the weights. This in turn implies that the connection sensitivity scores across layers are required to be of the same scale. To this end, we require the gradients to be faithful and the weights to be in the same scale for all the layers. Notice, this condition is trivially satisfied when the layerwise dynamical isometry is ensured, as each layer is initialized identically (*i.e.*, orthogonal initialization) and the gradients are guaranteed to be faithful.

Finally, we verify the failure of pruning cases presented in Section 4.3.1 based on the signal propagation perspective. Specifically, we measure the singular value distribution of the input-output Jacobian ($\mathbf{J}^{0,K}$) for the 7-layer tanh MLP network, and the results are reported in Table 4.1. Note that while connection sensitivity based pruning is robust to moderate changes in the Jacobian singular values, it failed catastrophically when the condition number of the Jacobian is very large ($> 1\text{e}{+11}$). In fact, these failure cases correspond to the completely disconnected networks, as a consequence of pruning with unreliable connection sensitivity resulted from poorly conditioned initial Jacobians. As we will show subsequently, these findings extend to modern architectures, and layerwise dynamical isometry yields well-conditioned Jacobians and in turn the best pruning results.

4.4 Signal propagation in sparse neural networks

So far, we have shown empirically and theoretically that layerwise dynamical isometry can improve the process of pruning at initialization. One remaining question to address is the following: how well do signals propagate in the pruned sparse networks? In this section, we first examine the effect of sparsity on signal propagation after pruning. We find that indeed pruning can break dynamical isometry,

Table 4.1: Jacobian singular values and resulting sparse networks for the 7-layer tanh MLP network considered in section 4.3.1. SG, CN, and Sparsity refer to Scaled Gaussian, Condition Number (*i.e.*, $s_{\max}/s_{\min}$, where $s_{\max}$ and $s_{\min}$ are the maximum and minimum Jacobian singular values), and a ratio of pruned prameters to the total number of parameters, respectively. SG (γ=10^{-2}) is equivalent to the variance scaling initialization as in LeCun et al. [1998]; Glorot and Bengio [2010]. The failure cases correspond to unreliable connection sensitivity resulted from poorly conditioned initial Jacobians.

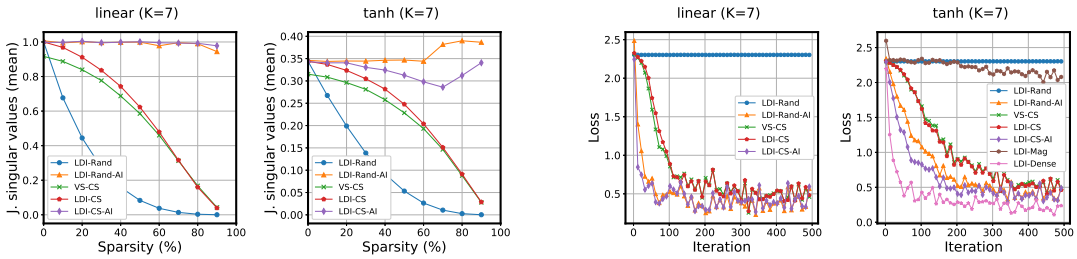
	Jacobian singular values			Sparsity in pruned network (across layers)							
Initialization	Mean	Std	CN	1	2	3	4	5	6	7	**Error**
SG (γ=10^{-4})	2.46e−07	9.90e−08	4.66e+00	0.97	0.80	0.80	0.80	0.80	0.81	0.48	2.66
SG (γ=10^{-3})	5.74e−04	2.45e−04	8.54e+00	0.97	0.80	0.80	0.80	0.80	0.81	0.48	2.67
SG (γ=10^{-2})	4.49e−01	2.51e−01	5.14e+01	0.96	0.80	0.80	0.80	0.81	0.81	0.49	2.67
SG (γ=10^{-1})	2.30e+01	2.56e+01	2.92e+04	0.96	0.81	0.82	0.82	0.82	0.80	0.45	2.61
SG (γ=10^{0})	1.03e+03	2.61e+03	3.34e+11	0.85	0.88	0.99	1.00	1.00	1.00	0.91	90.2
SG (γ=10^{1})	3.67e+04	2.64e+05	inf	0.84	0.95	1.00	1.00	1.00	1.00	1.00	90.2

degrading trainability of sparse networks. Then we follow up to present a simple, but effective data-free method to recover approximate dynamical isometry on sparse networks.

Setup. The overall process is summarized as follows: Step 1. Initialize a network with a variance scaling (VS) or layerwise dynamical isometry (LDI) satisfying orthogonal initialization. Step 2. Prune at initialization for a sparsity level $\bar{\kappa}$ based on connection sensitivity (CS); we also test random (Rand) and magnitude (Mag) based pruning for comparison. Step 3. (optional) Enforce approximate dynamical isometry, if specified. Step 4. Train the pruned sparse network using SGD. We measure signal propagation (*e.g.*, Jacobian singular values) on the sparse network right before Step 4, and observe training behavior during Step 4. Different methods are named as {A}-{B}-{C}, where A, B, C stand for initialization scheme, pruning method, (optional) approximate isometry, respectively. We perform this on 7-layer linear and tanh MLP networks as before [3].

Effect of pruning on signal propagation and trainability. Let us first check signal propagation measurements in the pruned networks (see Figure 4.2a). In general, Jacobian singular values decrease continuously as the sparsity level increases (except for {·}-{·}-AI which we will explain later), indicating that the more parameters are removed, the less faithful a network is likely to be with regard to propagating signals. Also, notice that the singular values drop more rapidly with random pruning compared to connection sensitivity based pruning methods (*i.e.*,

[3]We conduct the same experiments for ReLU and Leaky-ReLU activation functions (see Appendix 4.C).



(a) Signal propagation (b) Training behavior

Figure 4.2: (a) Signal propagation (mean Jacobian singular values) in sparse networks pruned for varying sparsity levels $\bar{\kappa}$, and (b) training behavior of the sparse network at $\bar{\kappa} = 90\%$. Signal propagation, pruning scheme, and overparameterization affect trainability of sparse neural networks. We train using SGD with the initial learning rate of 0.1 decayed by $1/10$ at every 20k iterations. All results are the average over 10 runs. We provide other singular value statistics (max, min, std), accuracy plot, and extended training results for random and magnitude pruning in Appendix 4.C.

{·}-Rand vs. {·}-CS). This means that pruning using connection sensitivity is more robust to destruction of dynamical isometry and preserve better signal propagation in the sparse network than random pruning. We further note that, albeit marginal, layerwise dynamical isometry allows better signal propagation than variance scaling initialization, with relatively higher mean singular values and much lower standard deviations especially in the low sparsity regime (see Appendix 4.C).

Now, we look into the relation between signal propagation and trainability of the sparse networks. Figure 4.2b shows training behavior of the pruned networks ($\bar{\kappa} = 90\%$) obtained by different methods. We can see a clear correlation between signal propagation capability of a network and its training performance; *i.e.*, the better a network propagates signals, the faster it converges during training. For instance, compare the trainability of a network before and after pruning. That is, compared to LDI-Dense ($\bar{\kappa} = 0$), LDI-{CS, Mag, Rand} decrease the loss much slowly; random pruning starts to decrease the loss around 4k iteration, and finally reaches to close to zero loss around 10k iterations (see Appendix 4.C), which is more than an order of magnitude slower than a network pruned by connection sensitivity. Recall that the pruned networks have much smaller singular values.

Enforcing approximate dynamical isometry. The observation above indicates that the better signal propagation is ensured on sparse networks, the better their training performs. This motivates us to think of the following: what if we can *repair* the broken isometry, before we start training the pruned network, such that

we can achieve trainability comparable to that of the dense network? Precisely, we consider the following:

$$\min_{\mathbf{W}^l} \|(\mathbf{C}^l \odot \mathbf{W}^l)^T (\mathbf{C}^l \odot \mathbf{W}^l) - \mathbf{I}^l\|_F \,, \tag{4.8}$$

where $\mathbf{C}^l$, $\mathbf{W}^l$, $\mathbf{I}^l$ are the sparse mask obtained by pruning, the corresponding weights, the identity matrix at layer l, respectively, and $\|\cdot\|_F$ is the Frobenius norm. We optimize this for all layers identically using gradient descent. Given the sparsity topology $\mathbf{C}^l$ and initial weights $\mathbf{W}^l$, this data-free method attempts to find an optimal $\mathbf{W}^*$ such that the combination of the sparse topology and the weights to be layerwise orthogonal, potentially to the full rank capacity. This simple method (*i.e.*, {·}-{·}-AI, where AI is named for Approximate Isometry) turns out to be highly effective. The results are provided in Figure 4.2, and we summarize our key findings below:

- *Signal propagation* (LDI-{CS, Rand} vs. LDI-{CS, Rand}-AI). The decreased singular values (by pruning $\bar{\kappa} > 0$) bounce up dramatically and become close to the level before pruning. This means that orthogonality enforced by Equation 4.8 is achieved in the sparse topology of the pruned network (*i.e.*, *approximate dynamical isometry*), and therefore, signal propagation on the sparse network is likely to behave similarly to the dense network. As expected, the training performance increased significantly (*e.g.*, compare LDI-CS with LDI-CS-AI for trainability). This works more dramatically for random pruning; *i.e.*, even for randomly pruned sparse networks, training speed increases significantly, implying the benefit of ensuring signal propagation.

- *Structure* (LDI-Rand-AI vs. LDI-CS-AI). Even if the approximate dynamical isometry is enforced identically, the network pruned using connection sensitivity shows better trainability than the randomly pruned network. This potentially means that the sparse topology obtained by different pruning methods also matters, in addition to signal propagation characteristics.

- *Overparameterization* (LDI-Dense vs. LDI-{CS, Rand}-AI). Even though the singular values are restored to a level close to before pruning with approximate isometry, the non-pruned dense network converges faster than pruned networks. We hypothesize that in addition to signal propagation, overparameterization helps in optimization taking less time to find a minimum.

Table 4.2: Pruning results for various neural networks on different datasets. All networks are pruned at initialization for the sparsity $\bar{\kappa} = 90\%$ based on connection sensitivity scores as in Lee et al. [2019]. We report orthogonality scores (OS) and generalization errors (Error) on CIFAR-10 (VGG16, ResNets) and Tiny-ImageNet (WRN16); all results are the average over 5 runs. The **first** and second best results are highlighted in each column of errors. The orthogonal initialization with enforced approximate isometry method (*i.e.*, LDI-AI) achieves the best results across all tested architectures.

	VGG16		ResNet32		ResNet56		ResNet110		WRN16	
Initialization	OS	Error	OS	Error	OS	Error	OS	Error	OS	Error
VS-L	13.72	8.16	4.50	11.96	4.64	10.43	4.65	9.13	11.99	45.08
VS-G	13.60	8.18	4.55	11.89	4.67	10.60	4.67	9.17	11.50	44.56
VS-H	15.44	8.36	4.41	12.21	4.44	10.63	4.39	9.08	13.49	46.62
LDI	13.33	8.11	4.43	11.55	4.51	10.08	4.57	8.88	11.28	44.20
LDI-AI	6.43	**7.99**	2.62	**11.47**	2.79	**9.85**	2.92	**8.78**	6.62	**44.12**

While being simple and data free (thus fast), our signal propagation perspective not only can be used to improve trainability of sparse neural networks, but also to complement a common explanation for decreased trainability of compressed networks which is often attributed merely to a reduced capacity. Our results also extend to the case of convolutional neural network (see Figure 4.8 in Appendix 4.C).

4.5 Validation and extensions

In this section, we aim to demonstrate the efficacy of our signal propagation perspective on a wide variety of settings. We first evaluate the idea of employing layerwise dynamical isometry on various modern neural networks. In addition, we further study the role of supervision under the pruning at initialization regime, extending it to unsupervised pruning. Our results show that indeed, pruning can be approached from the signal propagation perspective at varying scale, bringing forth the notion of neural architecture sculpting. The experiment settings used to generate the presented results are detailed in Appendix 4.B. The code can be found here: https://github.com/namhoonlee/spp-public.

4.5.1 Evaluation on various neural networks and datasets

Here, we verify that our signal propagation perspective for pruning neural networks at initialization is indeed valid, by evaluating further on various modern neural networks and datasets. To this end, we provide orthogonality scores (OS) and generalization errors of the sparse networks obtained by different methods

Table 4.3: Pruning results for VGG16 and ResNet32 with different activation functions on CIFAR-10. We report generalization errors (avg. over 5 runs), and the **first** and second best results are highlighted.

	VGG16			ResNet32		
Initialization	tanh	l-relu	selu	tanh	l-relu	selu
VS-L	9.07	7.78	8.70	13.41	12.04	12.26
VS-G	9.06	7.84	8.82	13.44	12.02	12.32
VS-H	9.99	8.43	9.09	**13.12**	11.66	12.21
LDI	8.76	7.53	8.21	13.22	11.58	11.98
LDI-AI	**8.72**	**7.47**	**8.20**	13.14	**11.51**	**11.68**

Table 4.4: Unsupervised pruning results for K-layer MLP networks on MNIST. All networks are pruned for sparsity $\bar{\kappa} = 90\%$ at orthogonal initialization. We report generalization errors (avg. over 10 runs).

Loss	**Superv.**	**K=3**	**K=5**	**K=7**
GT	✔	2.46	2.43	2.61
Pred. (raw)	✗	3.31	3.38	3.60
Pred. (softmax)	✗	3.11	3.37	3.56
Unif.	✗	2.77	2.77	2.94

and show that layerwise dynamical isometry with enforced approximate isometry results in the best performance; here, we define OS as $\frac{1}{l}\sum_l \|(\mathbf{C}^l \odot \mathbf{W}^l)^T(\mathbf{C}^l \odot \mathbf{W}^l) - \mathbf{I}^l\|_F$, which can be used to indicate how close are the weight matrices in each layer of the pruned network to being orthogonal. All results are the average of 5 runs, and we do not optimize anything specific for a particular case (see Appendix 4.B for experiment settings). The results are presented in Table 4.2.

The **best** pruning results are achieved when the approximate dynamical isometry is enforced on the pruned sparse network (*i.e.*, LDI-AI), across all tested architectures. Also, the second best results are achieved with the orthogonal initialization that satisfies layerwise dynamical isometry (*i.e.*, LDI). Looking closely, it is evident that there exists a high correlation between the orthogonality scores and the performance of pruned networks; *i.e.*, the network initialized to have the lowest orthogonality scores achieves the best generalization errors after training. Note that the orthogonality scores being close to 0, by definition, states how faithful a network will be with regard to letting signals propagate without being amplified or attenuated. Therefore, the fact that a pruned network with the lowest orthogonality scores tends to yield good generalization errors further validates that our signal propagation perspective is indeed effective for pruning at initialization. Moreover, we test for other nonlinear activation functions (tanh, leaky-relu, selu), and found that the orthogonal initialization consistently outperforms variance scaling methods (see Table 4.3).

4.5.2 Pruning without supervision

So far, we have shown that pruning random networks can be approached from a signal propagation perspective by ensuring faithful connection sensitivity. Another

factor that constitutes connection sensitivity is the loss term. At a glance, it is not obvious how informative the supervised loss measured on a random network will be for connection sensitivity. In this section, we look into the effect of supervision, by simply replacing the loss computed using ground-truth labels with different unsupervised surrogate losses as follows: replacing the target distribution using ground-truth labels with uniform distribution (Unif.), and using the averaged output prediction of the network (Pred.; softmax/raw). The results for MLP networks are in Table 4.4. Even though unsupervised pruning results are not as good as the supervised case, the results are still interesting, especially for the uniform case, in that there was no supervision given. We thus experiment further for the uniform case on other networks, and obtain the following results: 8.25, 11.69, 11.01, 8.82 errors (%) for VGG16, ResNet32, ResNet56, ResNet110, respectively. Surprisingly, the results are often competitive to that of pruning with supervision (*i.e.*, compare to LDI results in Table 4.2). Notably, previous pruning algorithms assume the existence of supervision a priori. Being the first demonstration, along with the signal propagation perspective, this unsupervised pruning strategy can be useful in scenarios where there are no labels or only weak supervision is available.

To demonstrate further, we also conducted *transfer of sparsity* experiments such as transferring a pruned network from one task to another (MNIST $\leftrightarrow$ Fashion-MNIST). Table 4.5 shows that, while pruning results may degrade if sparsity is transferred, or done without supervision, less impact is caused for unsupervised pruning when transferred to a different task (*i.e.*, 0.52 to 0.14 on MNIST, and 1.11 to -0.78 on F-MNIST). This indicates that inductive bias exists in data, affecting transfer and unsupervised pruning, and potentially, that "universal" sparse topology might be obtainable if universal data distribution is known (*e.g.*, extremely large dataset in practice). This may help in situations where different tasks from unknown data distribution are to be performed (*e.g.*, continual learning). We also tested two other unsupervised losses, but none performed as well as uniform loss (*e.g.*, Jacobian norms $\|J\|_1$: 5.03, $\|J\|_2$: 3.00 vs. Unif.: 2.94), implying that if pruning is to be unsupervised, the uniform loss would better be

Table 4.5: Transfer of sparsity experiment results for LeNet. We prune for $\bar{\kappa} = 97\%$ at orthogonal initialization, and report gen. errors (average over 10 runs).

	Dataset		Error		Error
Category	prune	train&test	sup. $\rightarrow$ unsup.	(Δ)	rand
Standard	MNIST	MNIST	2.42 $\rightarrow$ 2.94	+0.52	15.56
Transfer	F-MNIST	MNIST	2.66 $\rightarrow$ 2.80	**+0.14**	18.03
Standard	F-MNIST	F-MNIST	11.90 $\rightarrow$ 13.01	+1.11	24.72
Transfer	MNIST	F-MNIST	14.17 $\rightarrow$ 13.39	**-0.78**	24.89

used, because other unsupervised losses depend on the input data (thus can suffer from inductive bias). Random pruning degrades significantly at high sparsity for all cases.

4.5.3 Neural architecture sculpting

We have shown that pruning at initialization, even when no supervision is provided, can be effective based on the signal propagation perspective. This begs the question of whether pruning needs to be limited to pre-shaped architectures or not. In other words, what if pruning is applied to an arbitrarily bulky network and is treated as *sculpting* an architecture? In order to find out, we conduct the following experiments: we take a popular pre-designed architecture (ResNet20 in He et al. [2016]) as a base network, and consider a range of variants that are originally bigger than the base model, but pruned to have the same number of parameters as the base dense network. Specifically, we consider the following equivalents: **(1)** the same number of residual blocks, but with larger widths; **(2)** a reduced number of residual blocks with larger widths; **(3)** a larger residual block and the same width (see Table 4.6 in Appendix 4.B for details). The results are presented in Figure 4.3.

Overall, the sparse equivalents record lower errors than the dense base model. Notice that some models are extremely sparse (*e.g.*, Equivalent 1 pruned for $\bar{\kappa} = 98.4\%$). While all networks have the same number of parameters, discovered sparse equivalents outperform the dense reference network. This result is well aligned with recent findings in Kalchbrenner et al. [2018]: large sparse networks can outperform their small dense counterpart, while enjoying increased computational and memory efficiency via a dedicated implementation for sparsity in practice. Also, it seems that pruning wider networks tends to be more effective in producing a better model than pruning deeper ones (*e.g.*, Equivalent 1 vs. Equivalent 3). We further note that unlike existing prior works, the sparse networks are discovered by sculpting arbitrarily-designed architecture, without pretraining nor supervision.

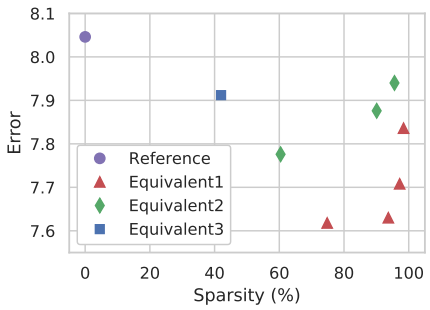


Figure 4.3: Neural architecture sculpting results on CIFAR-10. We report generalization errors (avg. over 5 runs). All networks have the same number of parameters (269k) and trained identically.

4.6 Discussion and future work

In this work, we have approached the problem of pruning neural networks at initialization from a signal propagation perspective. Based on observing the effect of varying the initialization, we found that initial weights have a critical impact on connection sensitivity measurements and hence pruning results. This led us to conduct theoretical analysis based on dynamical isometry and a mean field theory, and formally characterize a sufficient condition to ensure faithful signal propagation in a given network. Moreover, our analysis on compressed neural networks revealed that signal propagation characteristics of a sparse network highly correlates with its trainability, and also that pruning can break dynamical isometry ensured on a network at initialization, resulting in degradation of trainability of the compressed network. To address this, we introduced a simple, yet effective data-free method to recover the orthogonality and enhance trainability of the compressed network. Finally, throughout a range of validation and extension experiments, we verified that our signal propagation perspective is effective for understanding, improving, and extending the task of pruning at initialization across various settings. We believe that our results on the increased trainability of compressed networks can take us one step towards finding "winning lottery ticket" (*i.e.*, a set of initial weights that given a sparse topology can quickly reach to a generalization performance that is comparable to the uncompressed network, once trained) suggested in Frankle and Carbin [2019].

We point out, however, that there remains several aspects to consider. While pruning on enforced isometry produces trainable sparse networks, the two-stage orthogonalization process (*i.e.*, prune first and enforce the orthogonality later) can be suboptimal especially at a high sparsity level. Also, network weights change during training, which can affect signal propagation characteristics, and therefore, dynamical isometry may not continue to hold over the course of training. We hypothesize that a potential key to successful neural network compression is to address the complex interplay between optimization and signal propagation, and it might be immensely beneficial if an optimization naturally takes place in the space of isometry. We believe that our signal propagation perspective provides a means to formulate this as an optimization problem by maximizing the trainability of sparse networks while pruning, and we intend to explore this direction as a future work.

Acknowledgments

This work was supported by the ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1, EPSRC/MURI grant EP/N019474/1 and the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016). We would also like to acknowledge the Royal Academy of Engineering and FiveAI, and thank Richard Hartley, Puneet Dokania and Amartya Sanyal for helpful discussions.

4.A Gradients in terms of Jacobians

Proposition 4.A.1. Let $\epsilon = \partial L/\partial \mathbf{x}^K$ denote the error signal and $\mathbf{x}^0$ denote the input signal. Then,

1. the gradients satisfy:
$$\mathbf{g}_{\mathbf{w}^l}^T = \epsilon\, \mathbf{J}^{l,K} \mathbf{D}^l \otimes \mathbf{x}^{l-1} \,, \tag{4.9}$$
where $\mathbf{J}^{l,K} = \partial \mathbf{x}^K / \partial \mathbf{x}^l$ is the Jacobian from layer l to the output and $\otimes$ is the Kronecker product.

2. additionally, for linear networks, *i.e.*, when ϕ is the identity:
$$\mathbf{g}_{\mathbf{w}^l}^T = \epsilon\, \mathbf{J}^{l,K} \otimes \left(\mathbf{J}^{0,l-1}\mathbf{x}^0 + \mathbf{a}\right) \,, \tag{4.10}$$
where $\mathbf{J}^{0,l-1} = \partial \mathbf{x}^{l-1} / \partial \mathbf{x}^0$ is the Jacobian from the input to layer $l-1$ and $\mathbf{a} \in \mathbb{R}^N$ is the constant term that does not depend on $\mathbf{x}^0$.

Proof. The proof is based on a simple algebraic manipulation of the chain rule. The gradient of the loss with respect to the weight matrix $\mathbf{W}^l$ can be written as:

$$\mathbf{g}_{\mathbf{w}^l} = \frac{\partial L}{\partial \mathbf{W}^l} = \frac{\partial L}{\partial \mathbf{x}^K} \frac{\partial \mathbf{x}^K}{\partial \mathbf{x}^l} \frac{\partial \mathbf{x}^l}{\partial \mathbf{W}^l} \,. \tag{4.11}$$

Here, the gradient $\partial \mathbf{y}/\partial \mathbf{x}$ is represented as a matrix of dimension $\mathbf{y}$-size $\times$ $\mathbf{x}$-size. For gradients with respect to matrices, their vectorized from is used. Notice,

$$\frac{\partial \mathbf{x}^l}{\partial \mathbf{W}^l} = \frac{\partial \mathbf{x}^l}{\partial \mathbf{h}^l} \frac{\partial \mathbf{h}^l}{\partial \mathbf{W}^l} = \mathbf{D}^l \frac{\partial \mathbf{h}^l}{\partial \mathbf{W}^l} \,. \tag{4.12}$$

Considering the feed-forward dynamics for a particular neuron i,

$$\begin{aligned} h_i^l &= \sum_j W_{ij}^l x_j^{l-1} + b_i^l \,, \\ \frac{\partial h_i^l}{\partial W_{ij}^l} &= x_j^{l-1} \,. \end{aligned} \tag{4.13}$$

Therefore, using the Kronecker product, we can compactly write:

$$\frac{\partial \mathbf{x}^l}{\partial \mathbf{W}^l} = (\mathbf{D}^l)^T \otimes (\mathbf{x}^{l-1})^T \,. \tag{4.14}$$

Now, Equation 4.11 can be written as:

$$\begin{aligned} \mathbf{g}_{\mathbf{w}^l} &= (\epsilon\, \mathbf{J}^{l,K} \mathbf{D}^l)^T \otimes (\mathbf{x}^{l-1})^T \,, \\ \mathbf{g}_{\mathbf{w}^l}^T &= \epsilon\, \mathbf{J}^{l,K} \mathbf{D}^l \otimes \mathbf{x}^{l-1} \,. \end{aligned} \tag{4.15}$$

Here, $\mathbf{A}^T \otimes \mathbf{B}^T = (\mathbf{A} \otimes \mathbf{B})^T$ is used. Moreover, for linear networks $\mathbf{D}^l = \mathbf{I}$ and $\mathbf{x}^l = \mathbf{h}^l$ for all $l \in \{1 \dots K\}$. Therefore, $\mathbf{x}^{l-1}$ can be written as:

$$\begin{aligned}\mathbf{x}^{l-1} &= \phi(\mathbf{W}^{l-1}\phi(\mathbf{W}^{l-2} \dots \phi(\mathbf{W}^1\mathbf{x}^0 + \mathbf{b}^1) \dots + \mathbf{b}^{l-2}) + \mathbf{b}^{l-1}) \,, \\ &= \mathbf{W}^{l-1}(\mathbf{W}^{l-2} \dots (\mathbf{W}^1\mathbf{x}^0 + \mathbf{b}^1) \dots + \mathbf{b}^{l-2}) + \mathbf{b}^{l-1} \,, \\ &= \prod_{k=1}^{l-1} \mathbf{W}^k \mathbf{x}^0 + \prod_{k=2}^{l-1} \mathbf{W}^k \mathbf{b}^1 + \dots + \mathbf{b}^{l-1} \,, \\ &= \mathbf{J}^{0,l-1}\mathbf{x}^0 + \mathbf{a} \,,\end{aligned} \tag{4.16}$$

where $\mathbf{a}$ is the constant term that does not depend on $\mathbf{x}^0$. Hence, the proof is complete. □

4.B Experiment settings

Pruning at initialization. By default, we perform pruning at initialization based on connection sensitivity scores as in Lee et al. [2019]. When computing connection sensitivity, we always use all examples in the training set to prevent stochasticity by a particular mini-batch. Unless stated otherwise, we set the default sparsity level to be $\bar{\kappa} = 90\%$ (*i.e.*, 90% of the entire parameters in a network is pruned away). For all tested architectures, pruning for such level of sparsity does not lead to a large accuracy drop. Additionally, we perform either random pruning (at initialization) or a magnitude based pruning (at pretrained) for comparison purposes. Random pruning refers to pruning parameters randomly and globally for a given sparsity level. For the magnitude based pruning, we first train a model and simply prune parameters globally in a single-shot based on the magnitude of the pretrained parameters (*i.e.*, keep the large weights while pruning small ones). For initialization methods, we follow either variance scaling initialization schemes (*i.e.*, VS-L, VS-G, VS-H, as in LeCun et al. [1998]; Glorot and Bengio [2010]; He et al. [2015], respectively) or (convolutional) orthogonal initialization schemes [Saxe et al., 2014; Xiao et al., 2018].

Training and evaluation. Throughout experiments, we evaluate pruning results on MNIST, CIFAR-10, and Tiny-ImageNet image classification tasks. For training of the pruned sparse networks, we use SGD with momentum and train up to 80k (for MNIST) or 100k (for CIFAR-10 and Tiny-ImageNet) iterations. The initial learning rate is set to be 0.1 and is decayed by $1/10$ at every 20k (MNIST) or 25k (CIFAR-10 and Tiny-ImageNet). The mini-batch size is set to be 100, 128, 200

for MNIST, CIFAR-10, Tiny-ImageNet, respectively. We do not optimize anything specific for a particular case, and follow the standard training procedure. For all experiments, we use 10% of training set for the validation set, which corresponds to 5400, 5000, 9000 images for MNIST, CIFAR-10, Tiny-IamgeNet, respectively. We evaluate at every 1k iteration, and record the lowest test error. All results are the average of either 10 (for MNIST) or 5 (for CIFAR-10 and Tiny-ImageNet) runs.

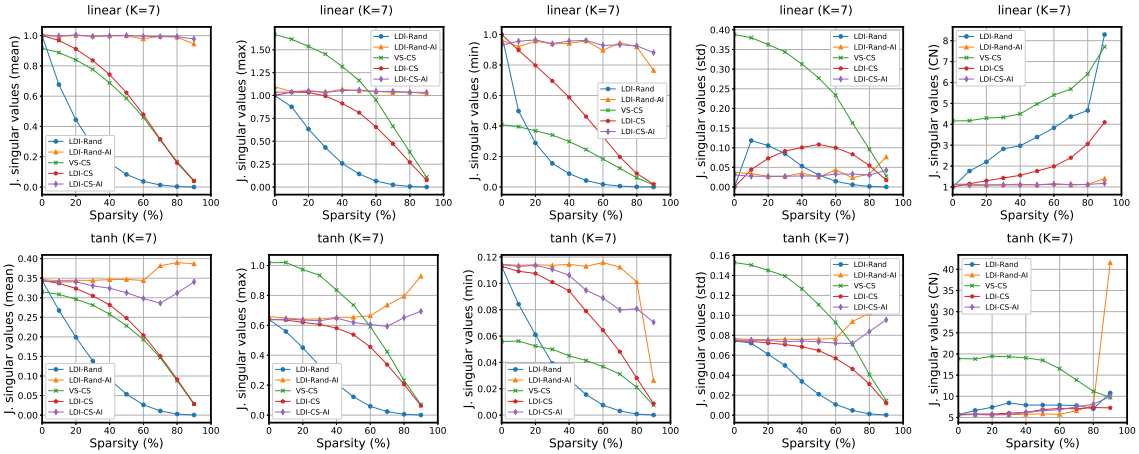
Signal propagation and approximate dynamical isometry. We use the entire training set when computing Jacobian singular values of a network. In order to enforce approximate dynamical isometry when specified, given a pruned sparse network, we optimize for the objective in Equation 4.8 using gradient descent. The learning rate is set to be 0.1 and we perform 10k gradient update steps (although it usually reaches to convergence far before). This process is data-free and thus fast; *e.g.*, depending on the size of the network and the number of update steps, it can take less than a few seconds on a modern computer.

Neural architecture sculpting. We provide the model details in Table 4.6.

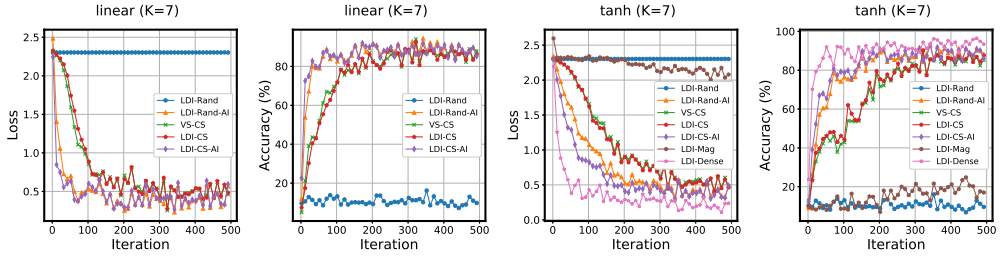
Table 4.6: All models (Equivalents 1,2,3) are initially bigger than the base network (ResNet20), by either being wider or deeper, but pruned to have the same number of parameters as the base network (269k). The widening factor (k) refers to the filter multiplier; *e.g.*, for the basic filter size of 16, the widening factor of k=2 will result in 32 filters. The block size refers to the number of residual blocks in each block layer; all models have three block layers. More/less number of residual blocks means the network is deeper/shallower. The reported generalization errors are averages over 5 runs. We find that the technique of *architecture sculpting*, pruning randomly initialized neural networks based on our signal propagation perspective even in the absence of ground-truth supervision, can be used to find models of superior performance under the same parameter budget.

Model category	Shape	Widening (k)	Block size	Init.	GT	Sparsity	Error
Base	ResNet20 [He et al., 2016]	1	3	VS-H	✔	0.0	8.046
Equivalent 1	wider	2	3	LDI	✗	74.8	7.618
	wider	4	3	LDI	✗	93.7	7.630
	wider	6	3	LDI	✗	97.2	7.708
	wider	8	3	LDI	✗	98.4	7.836
Equivalent 2	wider & shallower	2	2	LDI	✗	60.4	7.776
	wider & shallower	4	2	LDI	✗	90.1	7.876
	wider & shallower	6	2	LDI	✗	95.6	7.940
Equivalent 3	deeper	1	5	LDI	✗	42.0	7.912

4.C Signal propagation in sparse networks: additional results

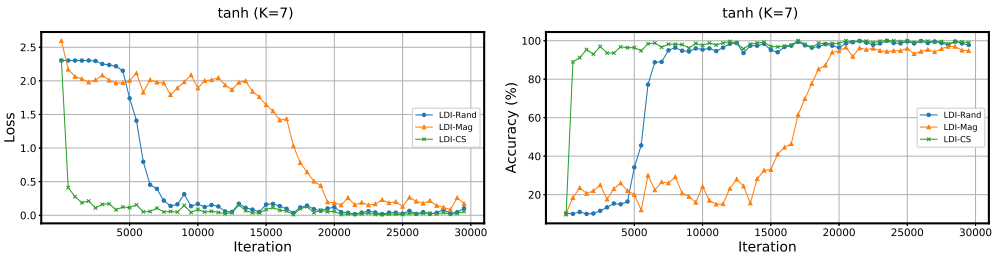


(a) Signal propagation (all statistics)



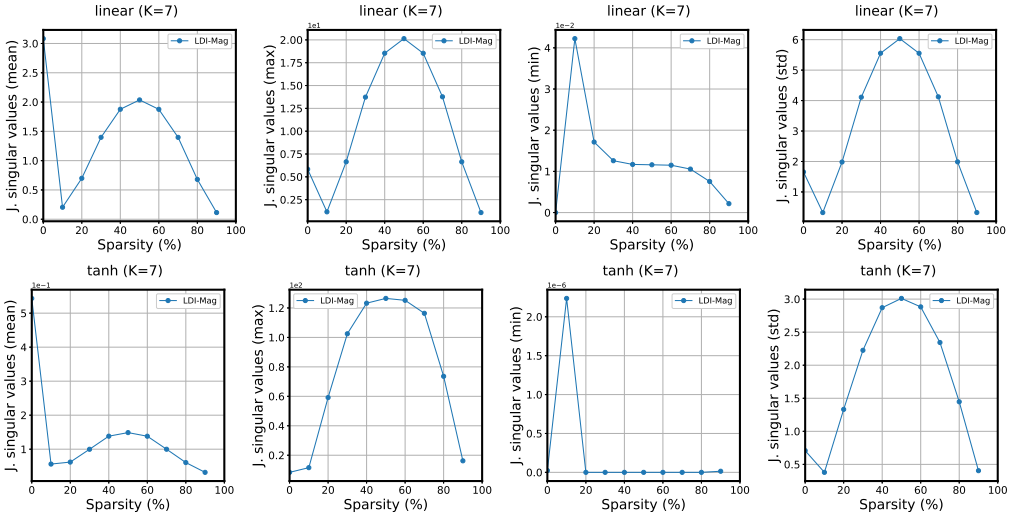
(b) Training behavior (loss and accuracy)

Figure 4.4: Full results for (a) signal propagation (all signular value statistics), and (b) training behavior (including accuracy) for 7-layer linear and tanh MLP networks. Methods are named as {initialization method}-{pruning method}; initialization method could be either a variance scaling (VS) or layerwise dynamical isometry (LDI); pruning method could be either random (Rand), magnitude based (Mag), connection sensitivity based pruning (CS), or unpruned (Dense); if approximate dynamical isometry is enforced on the pruned network, we add the suffix AI to the naming scheme; for example, the proposed method is named as LDI-CS-AI, since the network is first initialized to satisfy layerwise dynamical isometry (LDI), then pruned based on the connection sensitivity scores (CS), and lastly enforced to satisfy approximate dynamical isometry (AI), before the pruned sparse network is started to train. We provide results of LDI-Rand, LDI-Rand-AI, VS-CS, LDI-CS, LDI-CS-AI on the linear case for both singular value statistics and training log. We also plot results of LDI-Mag and LDI-Dense on the tanh case for trainability; the training results of non-pruned (LDI-Dense) and magnitude (LDI-Mag) pruning are only reported for the tanh case, because the learning rate had to be lowered for the linear case (otherwise it explodes), which makes the comparison not entirely fair (see Figures 4.6 and 4.5 for additional results). As a result, we find that as sparsity increases the signal propagation scores (indicated as Jacobian singular values) decreases breaking dynamical isometry (a), and yet, enforcing approximate isometry restores the signal propagation scores (find {·}-{·}-AI), which in turn increases the training speed of the pruned network quite significantly (b).



(a) Training behavior

Figure 4.5: Extended training log (*i.e.*, Loss and Accuracy) for random (Rand) and magnitude (Mag) pruning. The sparse networks obtained by random or magnitude pruning take a much longer time to train than that obtained by pruning based on connection sensitivity. All methods are pruned at the layerwise orthogonal initialization, and trained the same way as before.



(a) Signal propagation (all statistics; magnitude based pruning)

Figure 4.6: Signal propagation measurments (all signular value statistics) for the magnitude based pruning (Mag) on the 7-layer linear and tanh MLP networks. As described in the experiment settings in Appendix 4.B, the magnitude based pruning is performed on a pretrained model. Notice that unlike other cases where pruning is done at initialization (*i.e.*, using either random or connection sensitivity based pruning methods), the singular value distribution changes abruptly when pruned (*i.e.*, note of the sharp change of singular values from 0 to 10% sparsity). Also, the singular values are not concentrated (note of high standard deviations), which explains rather inferior trainability compared to other methods. We conjecture that naively pruning based on the magnitude of parameters in a single-shot, without pruning gradually or employing some sophisticated tricks such as layerwise thresholding, can lead to a failure of training compressed networks.

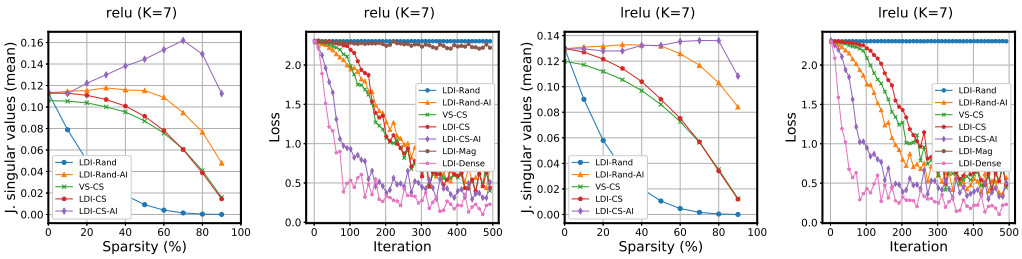


Figure 4.7: Signal propagation and training behavior for ReLU and Leaky-ReLU activation functions. They resemble those of the tanh case as in Figure 4.2, and hence the conclusion holds about the same.

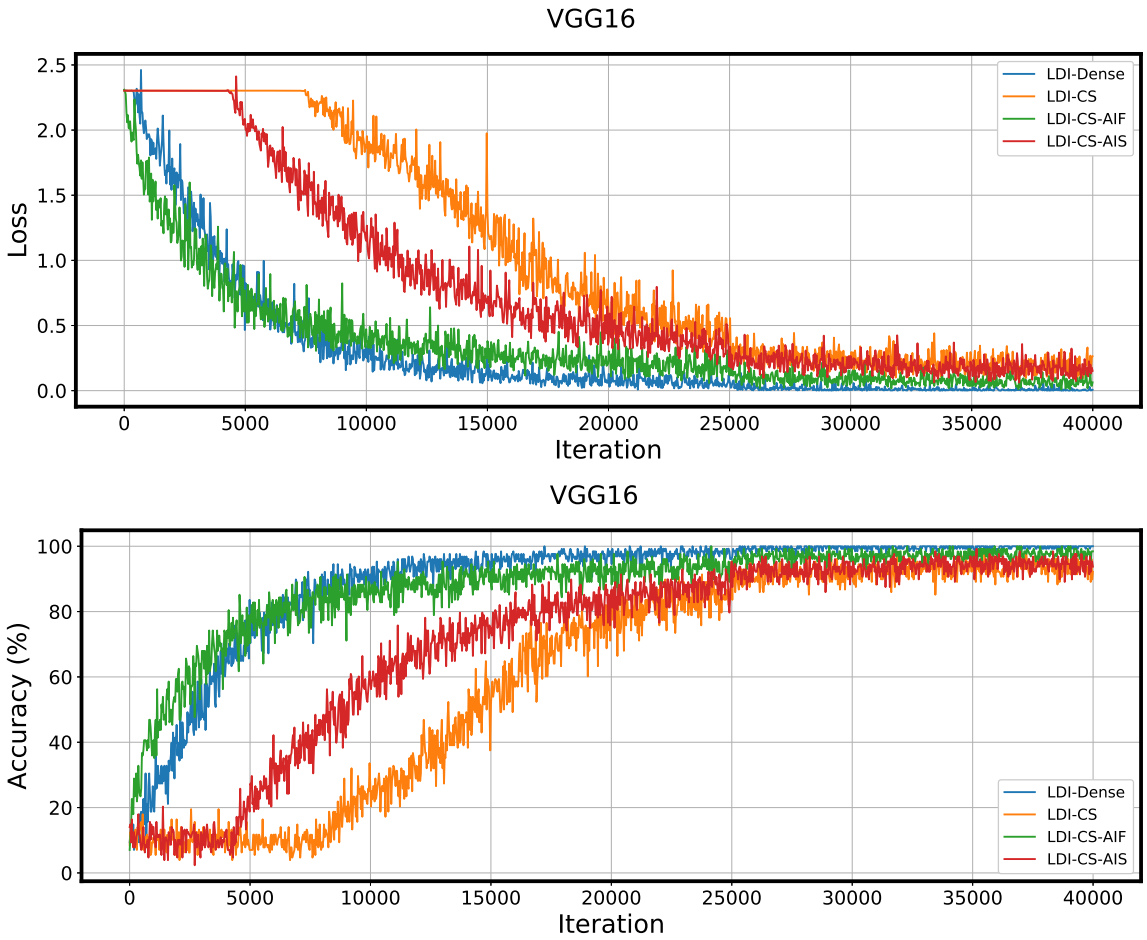


Figure 4.8: Training performance (loss and accuracy) by different methods for VGG16 on CIFAR-10. To examine the effect of initialization in isolation on the trainability of sparse neural networks, we remove batch normalization (BN) layers for this experiment, as BN tends to improve training speed as well as generalization performance. As a result, enforcing approximate isometry (LDI-CS-AIF) improves the training speed quite dramatically compared to the pruned network without isometry (LDI-CS). We also find that even compared to the non-pruned dense network (LDI-Dense) which is ensured layerwise dynamical isometry, LDI-CS-AIF trains faster in the early training phase. This result is quite promising and more encouraging than the previous case of MLP (see Figures 4.2 and 4.7), as it potentially indicates that an underparameterized network (by connection sensitivity pruning) can even outperform an overparameterized network, at least in the early phase of neural network training. Furthermore, we add results of using the spectral norm in enforcing approximate isometry in Equation 4.8 (LDI-CS-AIS), and find that it also trains faster than the case of broken isometry (LDI-CS), yet not as much as the case of using the Frobenius norm (LDI-CS-AIF).

Chapter 5

Understanding the Effects of Data Parallelism and Sparsity on Neural Network Training

Namhoon Lee
University of Oxford

Thalaiyasingam Ajanthan
Australian National University

Philip H. S. Torr
University of Oxford

Martin Jaggi
EPFL

Abstract

Network pruning is an effective methodology to compress large neural networks, and sparse neural networks obtained by pruning can benefit from their reduced memory and computational costs at use. Notably, recent studies have found that it is possible to find a trainable sparse neural network even at random initialization prior to training. While this approach of pruning at initialization turned out to be highly effective, there has been little study concerning the subsequent training of these sparse neural networks. In this work, we focus on studying the effects of data parallelism and sparsity on neural network training. For

data parallelism, this usually means processing training data in parallel using distributed systems, or equivalently increasing batch size, so that the training process can be accelerated. To this end, we first measure the effects for different study cases of batch size and sparsity level while tuning all metaparameters involved in the optimization. As a result, we find across various workloads of data set, network model, and optimization algorithms that there exists a general scaling trend in the relationship between batch size and number of training steps to convergence for the effect of data parallelism, irrespective of sparsity levels. Also, the effect of data parallelism in training sparse networks turns out to be no worse, or can be even better when the training is done by a momentum based optimizer, than that in training densely parameterized networks, despite the general difficulty of training sparse networks. We further provide theoretical insights based on the convergence properties of stochastic gradient methods and a smoothness analysis, so as to precisely illustrate our empirical findings and hence to develop a better account of the effects of data parallelism and sparsity on neural network training.

5.1 Introduction

Deep learning has been of great interest in recent years, bringing about tremendous success in science and technology [LeCun et al., 2015]. Behind its success lie powerful, yet often extremely large neural network models however, and they entail a significant amount of processing cost at use. As such cost can be critical to resource constrained environments, a diverse set of principles and approaches have been developed to compress large neural network models, and network pruning – a technique to remove redundant parameters in an overly parameterized neural network – has been widely employed [Reed, 1993; Han et al., 2015b].

While there exist various approaches to pruning, recent studies discovered that pruning can be done on a randomly initialized neural network prior to training, and the sparse neural network obtained can be easily trained to achieve good performance [Lee et al., 2019; Wang et al., 2020]. By separating the pruning process from training entirely, this process of pruning at initialization not only realizes algorithmic efficiency, but also can save a significant amount of time and effort in finding a trainable sparse neural network.

However, there has been little study of the training aspects of sparse neural networks. For example, while Evci et al. [2019] examined the difficulty of training sparse neural networks by analyzing the energy landscape in optimization dynamics, Lee et al. [2020] suggested a signal propagation perspective to speed up the training of sparse neural networks. Despite their potential, various aspects of the optimization of sparse neural networks remain rather unknown as of yet.

In this work, we focus on studying the effects of data parallelism on training these sparse neural networks. By data parallelism, we refer to utilizing a parallel computing system where the training data is distributed to multiple processors for gradient computations, so that the training process can be accelerated. Being model-agnostic, it is applicable to training any neural networks in contrast to other forms of parallelism such as task or model parallelism. For the purpose of this work, we consider the simplest setting of synchronized distributed systems in which the degree of parallelism equates to the size of mini-batch used for training neural network on a regular single-node system. Hence, we study specifically the effect of changing batch size on neural network training for the effect of data parallelism.

As with the development of cost-effective parallel hardware, understanding the effects of data parallelism (or equivalently increasing batch size) on neural network training is crucial, and in fact, it has been an active research topic in recent years [Goyal et al., 2017; Hoffer et al., 2017; Smith et al., 2018; Chen et al., 2018; Golmant et al., 2018; Shallue et al., 2019]. While we realize that sparse networks can benefit from data parallelism in distributed or large-batch training due to their reduced number of parameters and cost, there has been a lack of investigation of the effects of data parallelism on sparse neural network training. In this regard, we first conduct extensive experiments to measure the effects of data parallelism on sparse neural network training while carefully tuning metaparameters independently for each and every study case. As a result, we find a general trend of effect of data parallelism in training sparse neural networks, across varying sparsity levels and different workloads of data set, model and optimization algorithms, that turns out to be no worse or can be better than that in training densely parameterized networks, despite the difficulty of training sparse networks. Based on the convergence properties of stochastic gradient methods as well as an analysis on Lipschitz smoothness of sparse networks, we further present theoretical results that describe precisely the effects of data parallelism and sparsity on neural network training to this end.

5.1.1 Contributions

We summarize our main contributions as follows:

- We conduct a substantial amount of experiments based on extensive metaparameter search, and measure the effects of data parallelism for training neural networks while taking sparsity into account. Our results confirm that there exists a universal trend in the relationship between batch size and steps-to-result for the effect of data parallelism irrespective of sparsity levels.
- We also find that sparsity in general reduces the training speed approximately by the same factor across different batch sizes. Interestingly, however, the critical batch size for training sparse neural networks can be significantly increased by using a momentum based SGD.
- We provide theoretical insights into the effects of data parallelism and sparsity in training neural networks, that are established based on the convergence properties of stochastic gradient methods. Our theory precisely describes our observations, and hence, allows us to develop a better account of the effect of data parallelism. Combined with empirical analysis of Lipschitz smoothness, it further provides guidance on understanding the general difficulty of training sparse neural networks.

In light of our contributions, we make a further note that it has been considered difficult to obtain such results or estimate a priori. Therefore, we will release all our data points measured along with our code used in this work to the public in the hope of facilitating future research.

5.2 Measuring the effects of data parallelism and sparsity

5.2.1 Setup

Experiment protocol. We closely follow the experiment settings in Shallue et al. [2019]. For a given workload (*i.e.*, data set, model, optimization algorithm) and study (*i.e.*, batch size, sparsity level), we measure the number of training steps to reach a pre-defined goal error (*e.g.*, error rate for image classification). In this process, we search for the best metaparameters with a budget of runs to record the lowest number of steps, namely *steps-to-result*, as our primary quantity of interest.

For each run of training, we evaluate the intermediate model checkpoints on the entire validation set of a given data set at every fixed iterations to check if they reached the goal error. We describe more details and the scale of our experiments in Appendix 5.B.

Workloads. We consider the workloads constructed as the combinations of the following data sets, models, and optimization algorithms: (data sets) MNIST, Fashion-MNIST, CIFAR-10; (models) Simple-CNN, ResNet-8; (optimization algorithms) SGD, Momentum, Nesterov with either a constant learning rate or a linear learning rate decay schedule. In the main paper, we present results for the following four workloads: (W1) MNIST, Simple-CNN, SGD with a constant learning rate, (W2) MNIST, Simple-CNN, Momentum with a constant learning rate, (W3) Fashion-MNIST, Simple-CNN, Momentum with a constant learning rate, and (W4) CIFAR-10, ResNet-8, Nesterov with a linear learning rate decay; results for other workloads of combinations of above are provided in Appendix 5.D. We use 10% of the training set as a validation set, and the goal validation errors are set to be 0.02, 0.12, 0.4 for MNIST, Fashion-MNIST, CIFAR-10, respectively. The network models are the same as [Shallue et al., 2019], except that we remove dropout to prevent any effect on sparsification.

Metaparameter search. We perform a quasi-random search to tune metaparameters efficiently. More precisely, we first generate Sobol low-discrepancy sequences in a unit hypercube and convert them into metaparameters of interest, while taking into account a predefined search space for each metaparameter. The generated values for each metaparameter is in length of the budget of trials, and the search space is designed based on preliminary experimental results.

Pruning. We use the connection sensitivity saliency criterion in Lee et al. [2019] to prune a given network model at initialization prior to training. Sparse neural networks can be obtained by many different ways, and yet, for the purpose of this work, they must not undergo any training beforehand so we can measure the effects of data parallelism in isolation on training neural networks from scratch.

5.2.2 Results

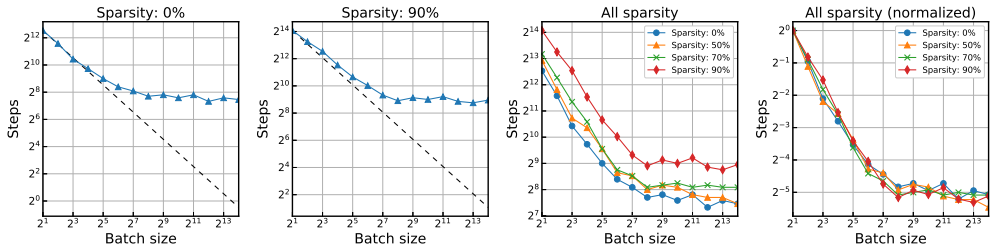
First of all, we observe in each and every sparsity level across different workloads a general trend in the relationship between batch size and steps-to-result for the effects of data parallelism (see the 1st and 2nd columns in Figure 5.1): Initially, we observe a period of *linear scaling* where doubling the batch size reduces the steps to achieve the goal error by half (*i.e.*, it aligns closely with the dashed line), followed

by a region of *diminishing returns* where the reduction in the required number of steps by increasing the batch size is less than the inverse proportional amount (*i.e.*, it starts to digress from the linear scaling region), which eventually arrives at a *maximal data parallelism* (*i.e.*, it hits a plateau) where increasing the batch size no longer decreases the steps to reach a goal error. This is observed across other workloads of different combinations of data set, network model, and optimization algorithm as well as different goal errors (see Appendix 5.D). We note that our observation is consistent with the results of regular network training presented in [Shallue et al., 2019]. We develop a theory of the effect of data parallelism that precisely accounts for this universal phenomenon in Section 5.3.1.

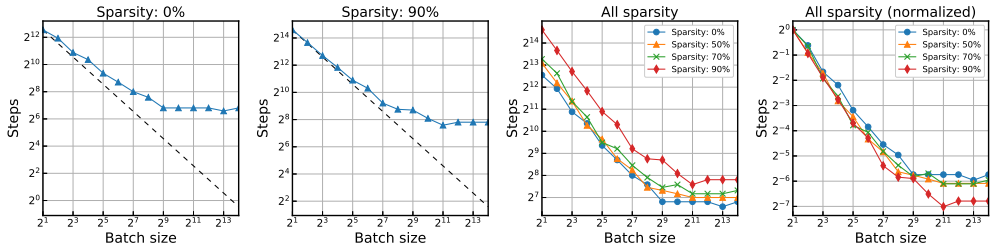
When we put the results for all sparsity levels together, the effect of data parallelism can be clearly visible and compared (see the 3rd column in Figure 5.1). In general, we find that the higher the sparsity level, the longer it takes for a sparse network to reach the given goal error. More precisely, a data parallelism curve for higher sparsity usually lies above than that for lower sparsity. For example, compared to the case of sparsity 0% (*i.e.*, the dense, over-parameterized network), 90% sparse network takes about $2-4$ times longer training time (or the number of training steps), consistently across different batch sizes (we provide more precise comparisons in Figure 5.12, Appendix 5.D.1). Recall that we tune all metaparameters independently for each and every study case of batch size and sparsity level without relying on a single predefined training rule, so as to find the best steps-to-result. Therefore, this result on the one hand corroborates the general difficulty of training sparse neural networks against the ease of training overly parameterized neural networks. We further find a potential cause of this difficulty based on our theory and a Lipschitz smoothness analysis in Section 5.3.2.

On the other hand, when we normalize the y-axis of each plot by dividing by the number of steps for the first batch size, we can see the phase transitions more clearly. As a result, we find that the regions of diminishing returns and maximal data parallelism appear no earlier when training sparse networks than the dense network (see the 4th column in Figure 5.1). This is quite surprising in that one could have easily guessed that the general optimization difficulty incurred by sparsification may influence the data parallelism too, at least to some degree; however, it turns out that the effects of data parallelism on sparse network training remain no worse than the dense case. Moreover, notice that in many cases the breakdown of linear scaling regime occurs even much later at larger batch sizes for a higher sparsity case; this is especially evident for Momentum and Nesterov

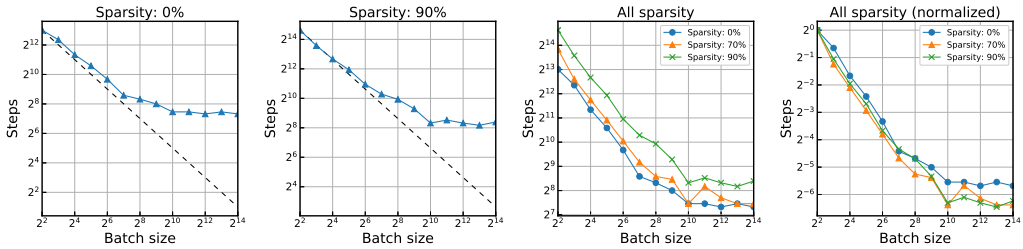
optimizers (*e.g.*, compare training 90% sparse network using Momentum against 0% dense network). In other words, for sparse networks, a *critical batch size* can be larger, and hence, when it comes to training sparse neural networks, one can increase the batch size (or design a parallel computing system for distributed optimization) more effectively, while better exploiting given resources. We find this result particularly promising since SGD with momentum is often the method of choice in practice. We further show that momentum optimizers being capable of exploiting large batch sizes hold the same across different sparsity levels by displaying all plots together in Figure 5.2. Overall, we believe that it is positively important to confirm the robustness of the data parallelism in sparse neural network training, which has been unknown thus far and difficult to estimate a priori.



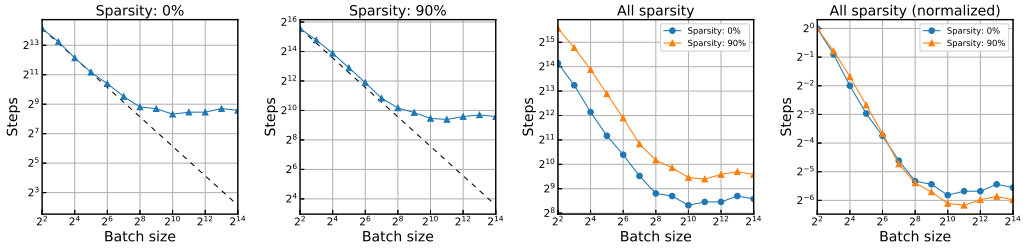
(a) MNIST, Simple-CNN, SGD



(b) MNIST, Simple-CNN, Momentum



(c) Fashion-MNIST, Simple-CNN, Momentum



(d) CIFAR-10, ResNet-8, Nesterov

Figure 5.1: Caption provided in the next page.

Figure 5.1: The effects of data parallelism and sparsity on neural network training for various workloads. Across all workloads and sparsity levels, the same scaling pattern is observed for the relationship between batch size and steps-to-result: it starts with the initial phase of *linear scaling* where increasing the batch size decreases the steps required to achieve the goal error inverse proportionally, and then there begins a region of *diminishing returns*, which is followed by the saturating phase of *maximal data parallelism* where increasing the batch size no longer speeds up the training process. Also, the effect of data parallelism in training sparse neural networks is no worse than that the dense, over-parameterized counterpart, despite the general difficulty of training sparse neural networks. When training using a *momentum* based SGD (*e.g.*, Momentum, Nesterov), the breakdown of the linear scaling regime often occurs much later at larger batch sizes for a network with higher sparsity; *i.e.*, a *critical batch size* can be even larger when training sparse neural networks with momentum. For example, in the case of workload {MNIST, Simple-CNN, Momentum}, the critical batch size for the sparsity 90% network is around 2^{11} whereas it is 2^9 for the sparsity 0% network (see the 4th column in row (b)). This potentially indicates that one can exploit large batch sizes more effectively when training sparse neural networks than densely parameterized neural networks. We supply more results in Appendix 5.D.

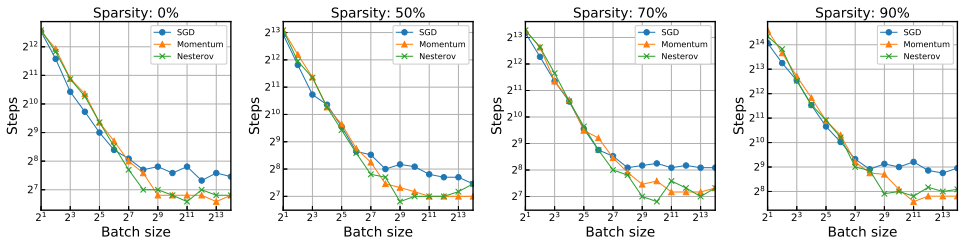


Figure 5.2: Comparing different optimization algorithms (SGD, Momentum, Nesterov) for the effects of data parallelism and sparsity. Across all sparsity levels, momentum based SGD optimizers (*i.e.*, Momentum, Nesterov) record lower steps-to-result in a large batch regime and have much bigger critical batch sizes than SGD without momentum. Identifying such patterns is crucial especially when training in resource constrained environments, as practitioners can potentially benefit from reducing the training time by deciding a critical batch size properly, while utilizing resources more effectively.

5.2.3 Metaparameter search

We perform metaparameter search to find the best metaparameter values that yield steps-to-result. In this section, we present and analyze the metaparameter search results used to measure the effects of data parallelism studied in the previous section. Specifically, we investigate the case of workload {MNIST, Simple-CNN, Momentum} where there are two metaparameters to tune (*i.e.*, learning rate and momentum) as this allows us to analyze all metaparameters easily in one figure. The results are presented in Figure 5.3, and other results can be found in Appendix 5.D.2.

First of all, our quasi-random search enables us to sample metaparameters efficiently, so that they are distributed evenly (without being cluttered) in a log-space and flexibly (rather than sitting in a grid with fixed spacing) within the search spaces. The search spaces are determined based on preliminary experiments and considered to be designed reasonably, in that the best metaparameters to yield lowest steps (marked by gold star ⋆) are located in the middle of the search ranges rather than sitting at the search boundaries, across different batch sizes and sparsity levels; *e.g.*, the best learning rate is located around 10^{-2} while the best momentum is found around a bit larger than 0.9. Also, the best learning rate increases with batch size, which aligns well with the classic result in learning theory that large batch training allows using bigger learning rates.

Interestingly, there are two distinguished regions (*i.e.*, complete (•) and incomplete (▲)) being separated by a seemingly linear boundary as per the relationship between learning rate and momentum. This indicates that the optimization is be-

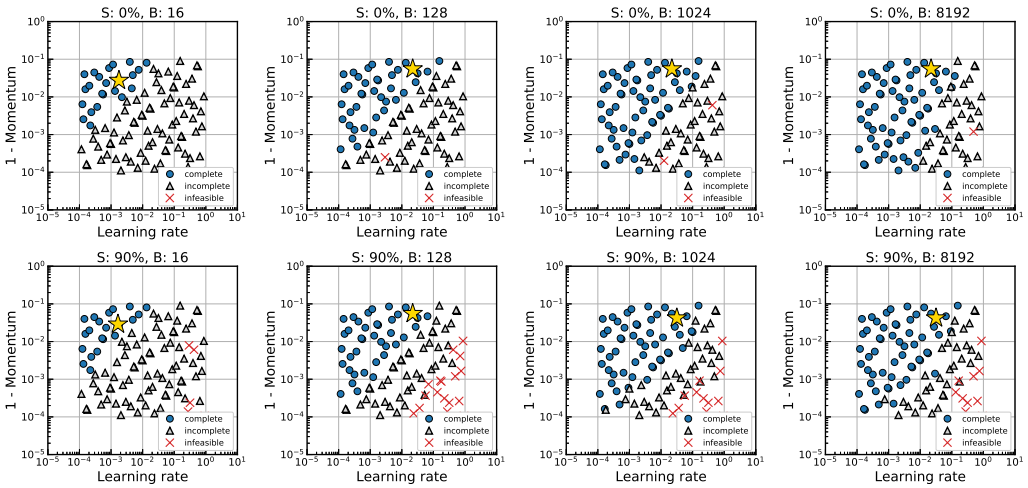


Figure 5.3: Visualizing all trials (100) for Simple-CNN on MNIST trained using Momentum optimizer with a constant learning rate. Sparsity level (S) and batch size (B) are denoted at the top of each plot. The best trial that records the lowest steps-to-result is marked by gold star (⋆). Complete/incomplete refer to the trials of goal reached/not reached given a maximum training step budget, while infeasible refers to the trial of divergence during training.

ing done by an interplay between these two metaparameters. More precisely, if one metaparameter is not chosen carefully with respect to the other (*e.g.*, increasing the learning rate for a fixed momentum), the optimizer may be stuck in a region spending time oscillating and eventually results in incomplete runs that did not reach a goal error within the maximum training iterations. Also, we can see that the successful region (filled with blue circles •) becomes larger as with increasing batch size, showing that large batch training reaches a given goal error in less number of iterations than small batch training, and hence, yields more complete runs.

We further present results for the workload {MNIST, Simple-CNN, SGD} in Figure 5.4. By isolating learning rate from momentum, it facilitates the search result analysis, and also allows us to study the effect of sparsity theoretically. More specifically, notice that for each batch size, the size of the range for successful learning rate $\bar{\eta}$ decreases by introducing sparsity. This supports our claim later in Section 5.3.2 that sparsity reduces the smoothness of gradients. We explain further in Appendix 5.C.

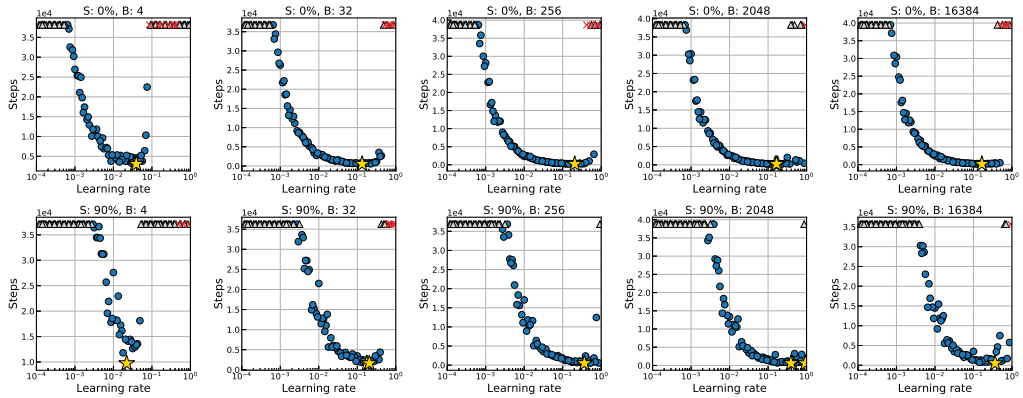


Figure 5.4: Visualizing metaparameter search results for the workload of {MNIST, Simple-CNN, SGD}. The metaparameter being tuned is the learning rate $\bar{\eta}$. We present results for different batch sizes (2^2, 2^5, 2^8, 2^{11}, 2^{14} – columns) and sparsity levels (0, 90% – rows). The blue circles (•) denote successful runs (*i.e.*, it reached the goal error), and the best trial that records the *lowest* steps to reach the goal (*i.e.*, steps-to-result) is marked by gold star (⋆); also, the grey triangles (▲) and red crosses (×) refer to incomplete and infeasible runs, respectively. We supply more results in Appendix 5.D.2.

5.3 Understanding the effects of data parallelism and sparsity

5.3.1 Convergence analysis as a tool to understand the general effects of data parallelism

We have empirically demonstrated across various workloads and sparsity levels that there seems to exist a general trend in the relationship between batch size and steps-to-result for training neural networks. While our results align well with previous findings [Shallue et al., 2019; Zhang et al., 2019a], we wish to gain theoretical insights into such global phenomenon and to develop a better account of the effects of data parallelism. To this end, we seek an answer from the convergence theory of stochastic gradient methods.

Let us begin with reviewing the convergence properties of stochastic gradient methods as the choice of numerical algorithms for solving optimization problems. Consider a generic optimization problem where the objective is to minimize some risk with the objective function $f : \mathbb{R}^m \to \mathbb{R}$, a prediction function $h : \mathbb{R}^{d_x} \times \mathbb{R}^m \to \mathbb{R}^{d_y}$, and a loss function $l : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ which yields the loss $l(h(\boldsymbol{x}; \mathbf{w}), \boldsymbol{y})$ given an input-output pair $(\boldsymbol{x}, \boldsymbol{y})$, where $\mathbf{w} \in \mathbb{R}^m$ is the parameters of the prediction model h, and d_x and d_y denote the dimensions of input $\boldsymbol{x}$ and output $\boldsymbol{y}$, respectively. A

generalized stochastic gradient method to solve this problem can be of the following form:

$$\mathbf{w}_{k+1} \coloneqq \mathbf{w}_k - \eta_k g(\mathbf{w}_k, \boldsymbol{\xi}_k)\,, \tag{5.1}$$

where η_k is a scalar learning rate, $g(\mathbf{w}_k, \boldsymbol{\xi}_k) \in \mathbb{R}^m$ is a stochastic vector (*e.g.*, gradient) with $\boldsymbol{\xi}_k$ denoting a random variable to realize data samples (either a single sample as in the prototypical stochastic gradient method [Robbins and Monro, 1951] or a set of samples as in the mini-batch version [Bottou, 1991]). Given an initial iterate $\mathbf{w}_1$, it finds a solution by performing the above update iteratively until convergence.

Under assumptions[1] on Lipschitz smoothness and variance of $g(\mathbf{w}_k, \boldsymbol{\xi}_k)$, the convergence rate result states that for such generic problem with nonconvex objective and optimization method with a fixed learning rate $\eta_k = \bar{\eta}$ for all k satisfying $0 < \bar{\eta} \leq \frac{\mu}{LM_G}$, the expected average squared norm of gradients of the objective function is guaranteed to satisfy the following inequality for all $K \in \mathbb{N}$ [Bottou et al., 2018]:

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\|\nabla f(\mathbf{w}_k)\|_2^2\right] \leq \frac{\bar{\eta}LM}{\mu} + \frac{2(f(\mathbf{w}_1) - f_\infty)}{K\mu\bar{\eta}}\,. \tag{5.2}$$

Here, $f(\mathbf{w}_1)$, f_∞, $\nabla f(\mathbf{w}_k)$ refer to the objective function's value at $\mathbf{w}_1$, lower bound, gradient at $\mathbf{w}_k$, respectively. Also, L is the Lipschitz constant of ∇f, and μ, M, M_G denote scalar bounds in the assumption on the second moment of $g(\mathbf{w}_k, \boldsymbol{\xi}_k)$. Note here that M is linked to batch size B as $M \propto 1/B$. In addition, if $g(\mathbf{w}_k, \boldsymbol{\xi}_k)$ is an unbiased estimate of $\nabla f(\mathbf{w}_k)$, which is the case for $\boldsymbol{\xi}_k$ being i.i.d. samples as in our experiments, then simply $\mu = 1$ [Bottou et al., 2018]. In essence, this result shows that the average squared gradient norm on the left-hand side is bounded above by asymptotically decreasing quantity as per K, indicating a sublinear convergence rate of the method. We note further that the convergence rate for the mini-batch stochastic optimization of nonconvex loss functions is studied previously [Ghadimi et al., 2016; Wang and Srebro, 2017], and yet, here we reconsider it to analyze the effects of data parallelism.

We now reformulate this result, such that it is translated into a form that matches our experiment settings and reveals the relationship between batch size and steps-to-result. To start with, let

$$\rho(\bar{\eta}, K) = \mathbb{E}\big[\min_{k \leq K} \|\nabla f(\mathbf{w}_k)\|^2\big] \tag{5.3}$$

[1] (simplified) (i) f is differentiable and satisfies $\|\nabla f(\mathbf{w}) - \nabla f(\bar{\mathbf{w}})\|_2 \leq L\|\mathbf{w} - \bar{\mathbf{w}}\|_2, \forall\{\mathbf{w}, \bar{\mathbf{w}}\} \subset \mathbb{R}^m$, and (ii) there exist scalars $M \geq 0$, $M_G \geq \mu^2 > 0$ such that $\mathbb{E}_{\boldsymbol{\xi}_k}\left[\|g(\mathbf{w}_k, \boldsymbol{\xi}_k)\|_2^2\right] \leq M + M_G\|\nabla f(\mathbf{w}_k)\|_2^2$.

where $\mathbf{w}_k$ is the set of weights produced after k steps with learning rate $\bar{\eta}$. Thus, $\rho(\bar{\eta}, K)$ is the expected minimum of the gradient-squared over the first K steps with learning rate $\bar{\eta}$. We consider here $\min_{k \leq K} \|\nabla f(\mathbf{w}_k)\|^2$ instead of $\sum_{k=1}^{K} \|\nabla f(\mathbf{w}_k)\|^2 / K$, since the former is what we are interested in and always less than the latter. Therefore, Eq. 5.2 can be rewritten as follows:

$$\rho(\bar{\eta}, K) \leq \frac{\bar{\eta} L M}{\mu} + \frac{2\Delta}{K \mu \bar{\eta}}\,, \tag{5.4}$$

where $\Delta = f(\mathbf{w}_1) - f_\infty$.

Next, for a given learning rate $\bar{\eta}$ and accuracy bound ϵ, we define

$$K^\star(\bar{\eta}, \epsilon) = \min(K) \text{ subject to } \rho(\bar{\eta}, K) < \epsilon\,, \tag{5.5}$$

as the number of steps that need to be taken with a given learning rate $\bar{\eta}$ to expect to reach a given bound ϵ. Further, let $K^\star(\epsilon) = \min_{\bar{\eta}} K^\star(\bar{\eta}, \epsilon)$, which is the smallest value of this required number of steps for all learning rates.

Now we present the following theorem.

Theorem 5.3.1. *Let $\epsilon > 0$ be given, then there is a critical batch size $2T$ and a constant C such that*

$$\begin{aligned} K^\star(\epsilon) &\leq C\frac{4}{B} && \text{if } B \leq 2T \\ &\leq C\Big(\frac{1}{T} + \frac{1}{B - T}\Big) && \text{if } B \geq 2T \end{aligned}$$

where B is the batch size, $C = \frac{2\Delta\beta L}{\epsilon^2 \mu^2}$, and $T = \frac{\beta}{\mu^2 \epsilon} - \frac{M_V}{2\mu^2}$.

Proof. First, we select ϵ. Then, we find values $\bar{\eta}$ and subsequently K such that $\rho(\bar{\eta}, K) < \epsilon$. This, by definition of $K^\star(\bar{\eta}, \epsilon)$ and $K^\star(\epsilon)$, means that $K \geq K^\star(\bar{\eta}, \epsilon) \geq K^\star(\epsilon)$. Therefore, an upper bound on K gives an upper bound on $K^\star(\epsilon)$.

Now, given ϵ, we choose $\bar{\eta}$ such that

$$\bar{\eta} = \frac{\epsilon \mu \gamma}{\beta L} \tag{5.6}$$

where γ is some parameter satisfying $0 < \gamma \leq \beta / (\epsilon M_G)$. With this choice, the learning rate condition in Bottou et al. [2018] is satisfied, and the theorem holds.

Next, we select K such that

$$K = \frac{2\Delta}{\epsilon \mu \bar{\eta} - \bar{\eta}^2 L M}\,. \tag{5.7}$$

We ignore for the moment that K is not an integer, and assume that this can be ensured by choosing $\bar{\eta}$ according to Eq. 5.6 appropriately. Note that the denominator is positive.

This choice of K gives

$$\frac{\bar{\eta}LM}{\mu} + \frac{2\Delta}{K\mu\bar{\eta}} = \epsilon \,, \tag{5.8}$$

and from Eq 5.2 this gives

$$\rho(\bar{\eta}, K) \leq \epsilon \,, \tag{5.9}$$

and hence, by the argument above

$$K^\star(\epsilon) \leq K = \frac{2\Delta}{\epsilon\mu\bar{\eta} - \bar{\eta}^2 LM} \,. \tag{5.10}$$

Next we find a bound on K. Starting from Eq. 5.7 and substituting Eq. 5.6 gives

$$\begin{aligned} K &= \frac{2\Delta}{\epsilon\mu\bar{\eta} - \bar{\eta}^2 LM} \\ &\leq \frac{2\Delta B}{B\epsilon\mu\bar{\eta} - \bar{\eta}^2 L\beta} && \text{since } M \leq \beta/B \\ &= \frac{2\Delta\beta L}{\epsilon^2\mu^2}\frac{B}{B\gamma - \gamma^2} && \text{from } \bar{\eta} = \frac{\epsilon\mu\gamma}{L\beta} \\ &= \frac{2\Delta\beta L}{\epsilon^2\mu^2}\Big(\frac{1}{\gamma} + \frac{1}{B-\gamma}\Big) \,. \end{aligned}$$

We want to find the value of γ that gives the best bound, and minimizing this gives $\gamma = B/2$. As long as $\gamma = B/2$ lies within the range $0 < \gamma \leq \beta/(\epsilon M_G)$ required by Bottou et al. [2018], this is the minimum.

The condition that the optimal value of $\gamma = B/2$ lies in the region required for Eq. 5.2 to hold is

$$\begin{aligned} \bar{\eta} &\leq \frac{\mu}{LM_G} \\ \frac{\epsilon\mu\gamma}{\beta L} &\leq \frac{\mu}{L(\mu^2 + M_V/B)} && \text{since } M_G = \mu^2 + M_V/B \\ \frac{B\epsilon\mu}{2\beta L} &\leq \frac{\mu}{L(\mu^2 + M_V/B)} && \text{since } \gamma = B/2 \\ \frac{\epsilon}{2\beta} &\leq \frac{1}{B\mu^2 + M_V} \end{aligned}$$

and solving for B gives

$$B \leq \frac{2\beta - M_V\epsilon}{\mu^2\epsilon} = \frac{2\beta}{\mu^2\epsilon} - \frac{M_V}{\mu^2} \,. \tag{5.11}$$

Thus, we write

$$T = \frac{\beta}{\mu^2 \epsilon} - \frac{M_V}{2\mu^2} \tag{5.12}$$

and $2T$ is the critical batch size as introduced in the theorem. Note that T is the sum of a constant term plus one that is inversely proportional to ϵ.

Thus, for $B \leq 2T$, we set $\gamma = B/2$ and arrive at

$$K = \frac{2\Delta\beta L}{\epsilon^2 \mu^2} \frac{4}{B} .$$

Otherwise, for $B \geq 2T$, the minimum within the required interval occurs when $\gamma = T$. Thus, in this case,

$$K \leq \frac{2\Delta\beta L}{\epsilon^2 \mu^2} \Big(\frac{1}{T} + \frac{1}{B - T} \Big)$$

and the proof is complete [2]. □

Theorem 5.3.1 precisely illustrates the relationship between batch size B and steps-to-result $K^\star$. For example, when B is small being less than the critical batch size $2T$, the bound on $K^\star$ scales linearly fitting the linear scaling regime found in the experiments, whereas when B is large, $K^\star$ is bounded by a constant term, indicating the maximal data parallelism as $K^\star$ remains constant. The fact that our empirical observation fits well with the theorem despite the inequality indicates that our extensive metaparameter search is effective as well. Therefore, this result not only well accounts for the scaling trend observed in the experiments, but also describes it more precisely and generally. We note that the effect of data parallelism, which was previously limited to serve as empirical evidence to support large-batch training, is now theoretically verified. We will further relate our result to sparse networks via smoothness analysis in Section 5.3.2.

5.3.2 Attributing Lipschitz smoothness to the difficulty of training sparse neural networks

Another distinct phenomenon observed in our experiments is that the number of steps required to reach the same goal error for sparse networks is consistently higher than that for dense networks regardless of batch size (*i.e.*, a whole data parallelism curve shifts upwards when introducing sparsity). This indicates the

[2] Previously, we had an incomplete version of the theorem (which we provide in Appendix 5.A for reference). The current version is provided by Richard Hartley who is the external examiner of Namhoon's thesis.

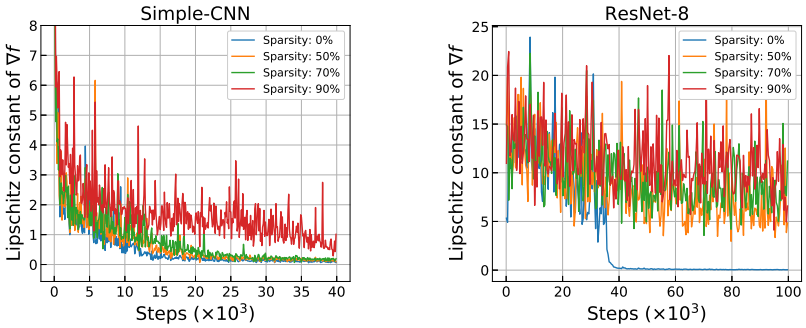


Figure 5.5: Lipschitz constant of ∇f measured locally over the course of training for networks with different sparsity levels. The more a network is pruned, the higher the Lipschitz constant becomes; *e.g.*, for 0, 50, 70, 90% sparsity levels, the average Lipschitz constants are 0.57, 0.72, 0.81, 1.76 for Simple-CNN and 3.87, 8.74, 9.54, 11.18 for ResNet-8, respectively. This indicates that pruning results in a network whose gradients are less smooth during training. We further provide additional training logs and explain how smoothness is measured in Appendix 5.C.

general difficulty of training sparse neural networks, and that sparsity degrades the training speed. In this section, we investigate what may cause this difficulty, and find a potential source of the problem by inspecting our theory of the effects of data parallelism to this end.

Let us begin with our result for the effect of data parallelism in Theorem 5.3.1. Notice that it is the coefficient C $(= 2\Delta\beta L/\epsilon^2\mu^2)$ that can shift a whole data parallelism curve vertically, by the same factor across different batch sizes. Taking a closer look, we realize that it is the Lipschitz constant L that can vary quite significantly by introducing sparsity and hence affect c_1; ϵ and μ are invariable, and Δ and β can change by sparsity in a relatively minor manner (we explain this in detail in Appendix 5.C). Specifically, L refers to the bound on the rate of change in ∇f and is by definition a function of f. Also, sparsity introduced by pruning changes the prediction function h which is linked to f via the loss function l. Therefore, we posit that a sparse neural network obtained by pruning will be *less smooth* (with a higher Lipschitz constant) than the non-pruned dense network. To verify our hypothesis, we empirically measure the Lipschitz constant for networks with different sparsity levels over the course of the entire training process. The results are presented in Figure 5.5.

As we can see, it turns out that the Lipschitz constant increases as with increasing sparsity level, and further, is consistently higher for sparse networks than for the dense network throughout training. This means that *pruning results in sparse networks whose gradient changes relatively too quickly* compared to the dense network;

in other words, the prediction function h becomes *less smooth* after pruning. This is potentially what hinders training progress, and as a result, sparse networks require more time (*i.e.*, steps-to-result) to reach the same goal error. We note that our findings of increased Lipschitz constant for sparse networks are consistent with the literature on over-parameterized networks such as Li and Liang [2018], which can be seen as the opposite of sparsity. The more input weights a neuron has, the less likely it is that a single parameter significantly changes the resulting activation pattern, and that wide layers exhibit convexity-like properties in the optimization landscape Du et al. [2019]. This even extends to non-smooth networks with ReLU activations, which are still shown to exhibit pseudo-smoothness in the overparameterized regime Li and Liang [2018]. We further show that our theory precisely explains the difficulty of training sparse networks due to decreased smoothness based on a quantitative analysis in Appendix 5.C.

5.4 Discussion and future work

In this work, we studied the effects of data parallelism and sparsity on neural network training. Despite the potential synergy between sparsity and data parallelism, little has been known about the behavior of sparse networks for this aspect. To this end, we first conducted rigorous experiments to accurately measure the effects, and further developed a theoretical analysis to precisely account for the general characteristics of data parallelism and sparsity in training neural networks. We find that there exists a universal trend in the relationship between batch size and number of training steps to convergence irrespective of sparsity levels, and that sparse neural networks can potentially benefit from large-batch training. While our findings render positive impacts to practitioners and theorists alike, there remain several challenges to address. First, our experiments are bounded by available computing resources; the cost of experiments increases exponentially for more complex workloads. Also, the lack of general convergence guarantees for existing momentum schemes in nonconvex settings hinders a further theoretical analysis. We hypothesize that ultimate understanding of the effects of data parallelism should be accompanied by a study of the generalization capability of optimization methods, however, these are beyond the scope of our current work, and we intend to explore this direction further as future work.

5.A The relationship between batch size and steps-to-result (outdated)

In this section, we provide an old result of the relationship between batch size and steps-to-result, which was previously included in the initial manuscript. This result is now considered outdated, but included in the thesis solely for the purpose of reference. For the complete result, the readers are referred to Theorem 5.3.1, which is provided by Richard Hartley who is the external examiner of Namhoon's thesis.

We start from the convergence rate result in Eq. 5.2. We first recognize that the expected average squared gradient norm on the left-hand side indicates the degree of convergence. Then, this quantity is directly related to the concept of goal error to reach in our experiments, and hence, reduces to be a small constant ϵ as soon as it is implemented as a pre-defined goal error for a given workload. Thus, it follows that

$$\epsilon \leq \frac{\bar{\eta} L M}{\mu} + \frac{2(f(\mathbf{w}_1) - f_\infty)}{K \mu \bar{\eta}} \,. \tag{5.13}$$

Notice that by fixing $\epsilon = \mathbb{E}\big[\frac{1}{K}\sum_{k=1}^{K} \|\nabla f(\mathbf{w}_k)\|_2^2\big]$, it effectively means that the training process has stopped, and therefore, K on the right-hand side will no longer contribute to decrease the bound of the quantity for a particular learning rate $\bar{\eta}$ and batch size B.

Also, we select the *optimal*[3] learning rate $\bar{\eta}^\star$, out of extensive metaparameter search, to record the *lowest* number of steps to reach the given goal error, which we denote as steps-to-result $K^\star$. Plugging in these yields the following:

$$\epsilon \leq \frac{\bar{\eta}^\star L M}{\mu} + \frac{2(f(\mathbf{w}_1) - f_\infty)}{K^\star \mu \bar{\eta}} \,. \tag{5.14}$$

Next, notice that the only factors that constitute the inequality in Eq. 5.2 come from the assumptions made to derive the convergence rate result, which are on the Lipschitz constant L and the variance bound M, and if they are assumed to be tight in the worst case, the inequality becomes tight. Then, after making algebraic manipulation and taking the first-order Taylor approximation while substituting $M = \beta/B$ since the variance bound M is related to batch size B as $M \propto 1/B$ [Bottou

[3]Here, *optimal* simply refers to the sense of yielding the *lowest* steps-to-result.

et al., 2018], we obtain the following result:

$$
\begin{aligned}
K^\star &\approx \frac{\Delta}{\bar{\eta}^\star \mu\epsilon - (\bar{\eta}^\star)^2 LM} \\
&\approx \frac{\Delta}{\bar{\eta}^\star \mu\epsilon} + \frac{\Delta LM}{\mu^2\epsilon^2} \\
&= \frac{\Delta L\beta}{\mu^2\epsilon^2 B} + \frac{\Delta}{\bar{\eta}^\star \mu\epsilon} \\
&= \frac{c_1}{B} + c_2\,, \qquad \text{where } c_1 = \frac{\Delta L\beta}{\mu^2\epsilon^2} \text{ and } c_2 = \frac{\Delta}{\bar{\eta}^\star \mu\epsilon}\,.
\end{aligned}
\tag{5.15}
$$

Here, $\Delta = 2(f(\mathbf{w}_1) - f_\infty)$, β is the initial variance bound at batch size $B = 1$ for a given workload. Notice that ϵ, Δ, L, β, μ $\bar{\eta}^\star$ all are constant or become fixed for a given workload. Therefore, this result precisely illustrates the relationship between batch size B and steps-to-result $K^\star$. Please refer to the main paper for the analysis of this result. We also note that the degree of metaparameter search quality is assumed to be the same across different batch sizes, and hence, the result can be reliably used to interpret the relationshp between batch size and steps-to-result.

5.B Scale of our experiments

For a given workload of {data set, network model, optimization algorithm} and for a study setting of {batch size, sparsity level}, we execute 100 training runs with different metaparameters to measure steps-to-result. At each run, we evaluate the intermediate models at every 16 (for MNIST) or 32 (for CIFAR-10) iterations, on the *entire* validation set, to check if it reached a goal error. This means that, in order to plot the results for the workload of {MNIST, Simple-CNN, SGD} for example, one would need to perform, 14 (batch sizes; 2^1 to 2^{14}) $\times$ 4 (sparsity levels; $0, 50, 70, 90\%$) $\times$ 100 (runs) $\times$ $40,000$ (max training iteration preset) / 16 (evaluation interval) $=$ $14,000,000$ number of evaluations. Assuming that evaluating the Simple-CNN model on the entire MNIST validation set takes only a *second* on a modern GPU, it will take $14,000,000$ (evaluations) $\times$ 1 (second per evaluation) / 3600 (second per hour) ≈ 3888 hours or 162 days.

Of course, there are multiple ways to reduce this cost; for instance, we may decide to stop as soon as the run hits the goal error without running until the max training iteration limit. Or, simply reducing any factor listed above that contributes to increasing the experiment cost (*e.g.*, number of batch sizes) can help to reduce the time, however, in exchange for the quality of experiments. We should also point

out that this is only for one workload where we assumed that the evaluation takes only a second. The cost can increase quite drastically if the workload becomes more complex and requires more time for evaluation and training (*e.g.*, CIFAR-10). Not to mention, we have tested for multiple workloads besides the above example, in order to confirm the generality of the findings in this work.

5.C More on Lipschitz smoothness analysis

We measure the local Lipschitz constant of ∇f based on a Hessian-free method as used in Zhang et al. [2020]. Precisely, the local smoothness (or Lipschitz constant of the gradient) at iteration k is estimated as in the following:

$$\hat{L}(\mathbf{w}_k) = \max_{\gamma \in \{\delta, 2\delta, .., 1\}} \frac{\|\nabla f(\mathbf{w}_k + \gamma \boldsymbol{d}) - \nabla f(\mathbf{w}_k)\|_2}{\|\gamma \boldsymbol{d}\|_2}, \tag{5.16}$$

where $\boldsymbol{d} = \mathbf{w}_{k+1} - \mathbf{w}_k$ and $\delta \in (0, 1)$ for which we set to be $\delta = 0.1$. The expected gradient ∇f is computed on the entire training set, and we measure $\hat{L}(\mathbf{w}_k)$ at every 100 iterations throughout training. This method searches the maximum bound on the smoothness along the direction between $\nabla f(\mathbf{w}_{k+1})$ and $\nabla f(\mathbf{w}_k)$ based on the intuition that the degree of deviation of the linearly approximated objective function is bounded by the variation of gradient between $\mathbf{w}_{k+1}$ and $\mathbf{w}_k$. Furthermore, while ReLU networks (*e.g.*, Simple-CNN, ResNet-8) can only be piecewise smooth, the smoothness can still be measured for the same reason that we measure gradient (*i.e.*, it only requires differentiability).

In addition to our main results on the effect of data parallelism, we provide in Figure 5.6 the training logs of the networks used for the Lipschitz smoothness analysis in Section 5.3.2, in order to show the correlation between the Lipschitz smoothness of a network and its training performance; *i.e.*, sparsity incurs low smoothness of gradients (high L; see Figure 5.5) and hence the poor training performance.

We also empirically measure the changes in Δ and β by introducing sparsity. Recall that these are the other elements in c_1 that can be affected by sparsity along with Lipschitz constant L. When we measure these quantities for Simple-CNN, we obtain the following results: $\Delta_s/\Delta_d \approx 4.68/4.66 \approx 1.00$, and $\beta_s/\beta_d \approx 107.39/197.06 \approx 0.54$; more precisely, Δ does not change much since neither $f(\mathbf{w}_1)$ or $f(\mathbf{w}_\infty)$ changes much, and β_s/β_d can be measured by the ratio of the variances of gradients between sparse and dense networks at $B = 1$. We have already provided L_s, L_d in Figure 5.5, which makes $L_s/L_d \approx 1.76/0.57 \approx 3.09$.

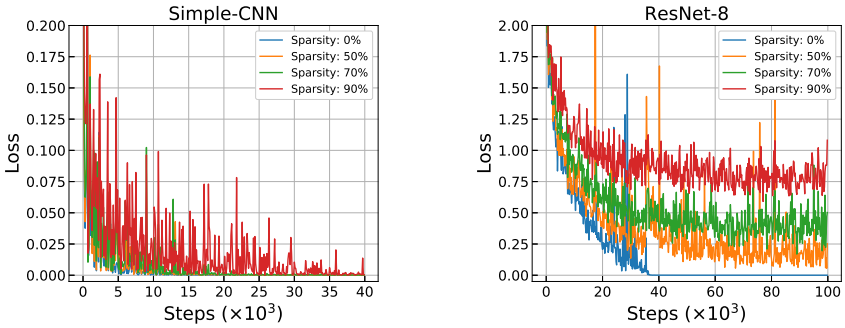


Figure 5.6: Training logs of the networks Simple-CNN and ResNet-8 used for the smoothness analysis in Section 5.3.2. The sparse networks that recorded high Lipschitz constants show worse training performance, indicating that low smoothness may be the potential cause of hampering the training of sparse neural networks.

Here, s and d denote sparse (90%) and dense, respectively. Notice that if we combine all the changes in Δ, β, L due to sparsity, and compute $c_{1,s}/c_{1,d}$, it becomes $1.00 \times 0.54 \times 3.09 \approx 1.67$. Importantly, $c_{1,s}/c_{1,d} > 1$ means that c_1 has increased by sparsity, and since the increase in Lipschitz constant L played a major role therein, these results indicate that the general difficulty of training sparse networks is indeed caused by reduced smoothness. We further note that this degree of change fits roughly the range of $k_s^\star/k_d^\star$ as shown in Figure 5.12.

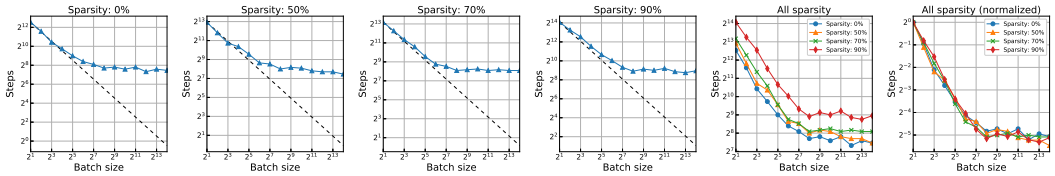
Evidence of increased Lipschitz constant for sparse networks can be found further in metaparameter search results such as in Figure 5.4. Notice that for each batch size, the size of the range for successful learning rate $\bar{\eta}$ decreases when switching from 0 to 90% sparsity level. This is potentially because the learning rate bound satisfying the convergence rate theory becomes $0 < \bar{\eta} \leq 1/L$ for a fixed batch size, and increased L due to sparsity shrinks the range of $\bar{\eta}$.

5.D Additional results

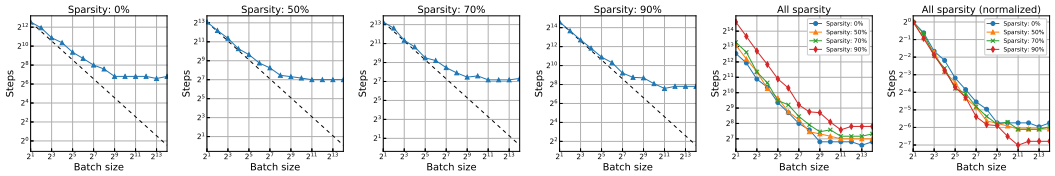
In this section, we provide additional experiemental results that are not included in the main paper. In Section 5.D.1, we supplement more results for the effects of data parallelism and sparsity in Figures 5.7, 5.8, 5.9, 5.10, 5.11. In Figure 5.12 we present the difference in ratio between sparse (90%) and dense networks across different batch sizes for all workloads presented in this work. This result shows how much increase in steps-to-result is induced by introducing sparsity, and therefore, is used to study the general difficulty of training sparse neural networks. In Section 5.D.2,

we provide metaparameter search results for a subset of workloads studied in this work.

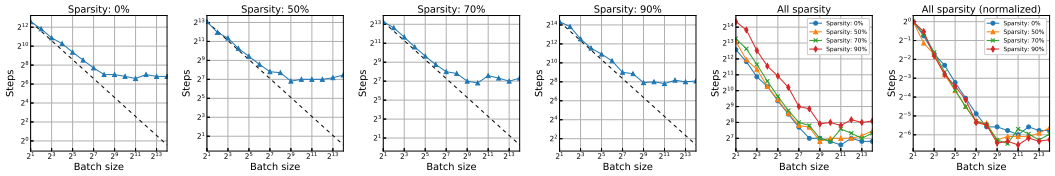
5.D.1 Effects of data parallelism and sparsity



(a) SGD

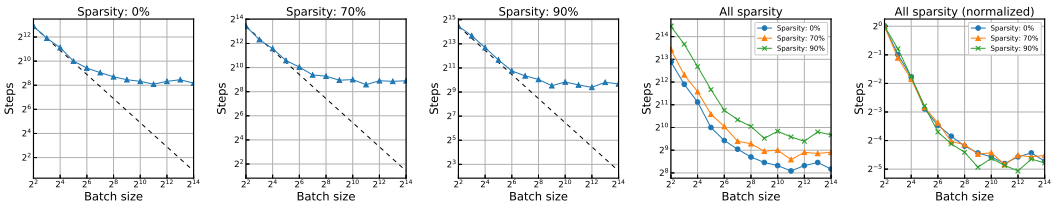


(b) Momentum

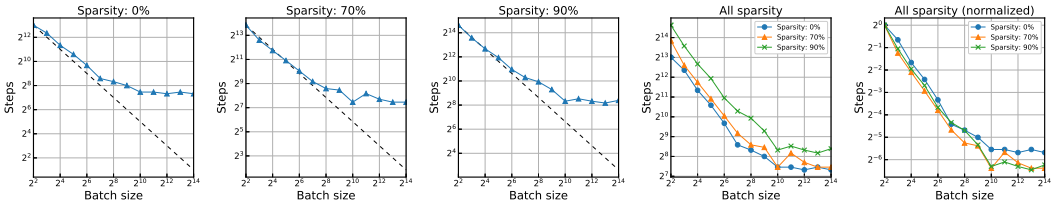


(c) Nesterov

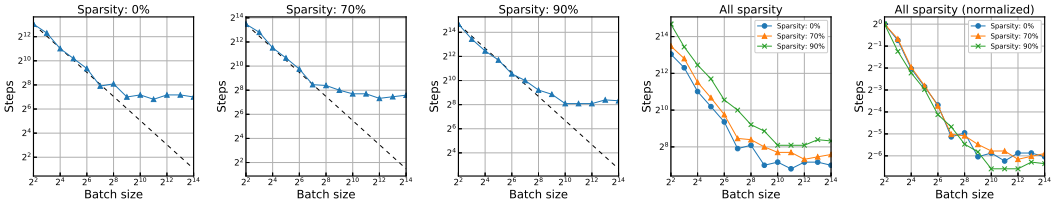
Figure 5.7: Results for the effects of data parallelism for the workloads of {MNIST, Simple-CNN, SGD/Momentum/Nesterov} with a constant learning rate.



(a) SGD

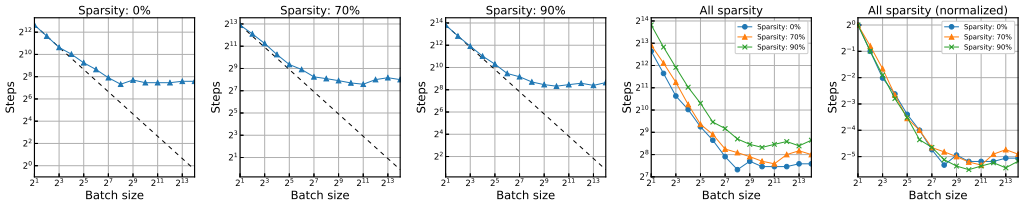


(b) Momentum

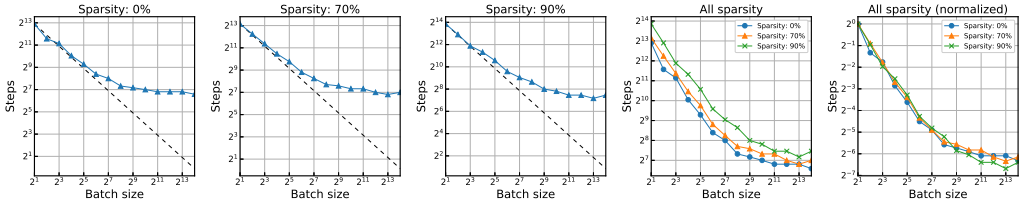


(c) Nesterov

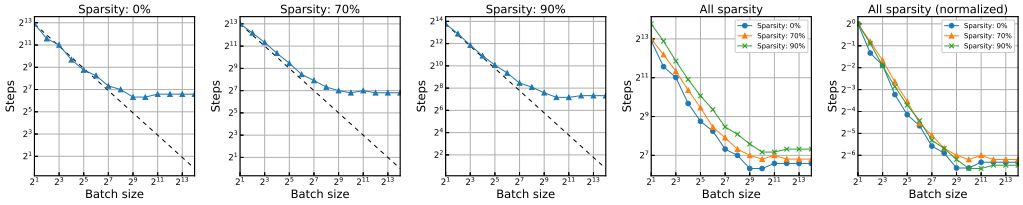
Figure 5.8: Results for the effects of data parallelism for the workloads of {Fashion-MNIST, Simple-CNN, SGD/Momentum/Nesterov} with a constant learning rate.



(a) SGD

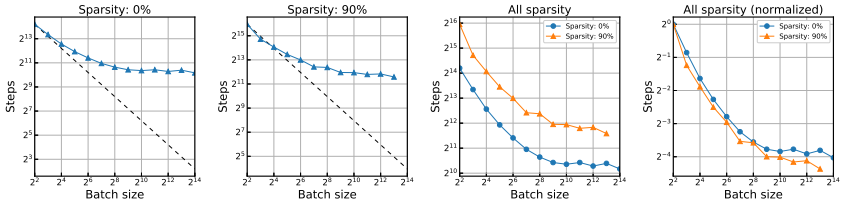


(b) Momentum

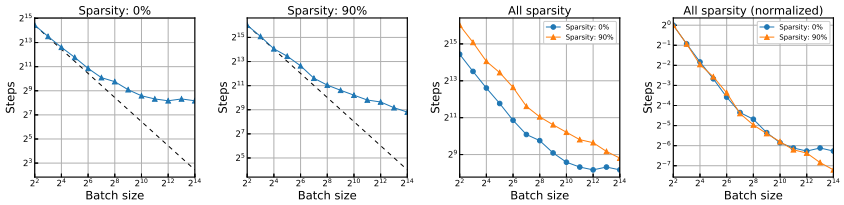


(c) Nesterov

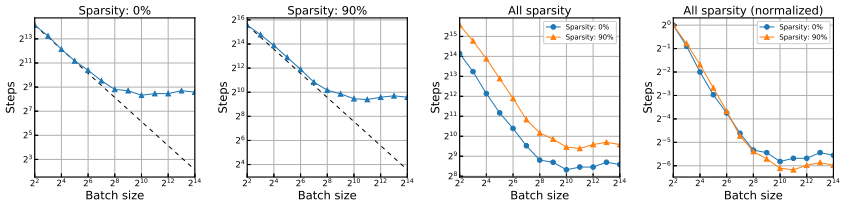
Figure 5.9: Results for the effects of data parallelism for the workloads of {Fashion-MNIST, Simple-CNN, SGD/Momentum/Nesterov} with a constant learning rate and the goal error of 0.14.



(a) SGD

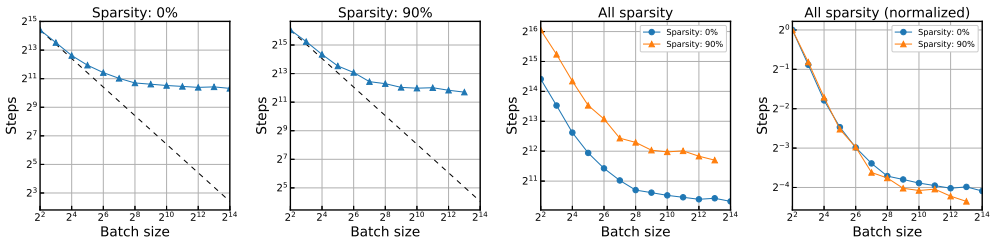


(b) Momentum

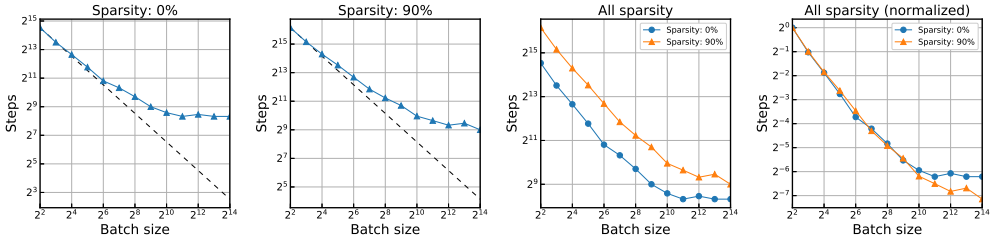


(c) Nesterov

Figure 5.10: Results for the effects of data parallelism for the workloads of {CIFAR-10, ResNet-8, SGD/Momentum/Nesterov} with a linear learning rate decay.

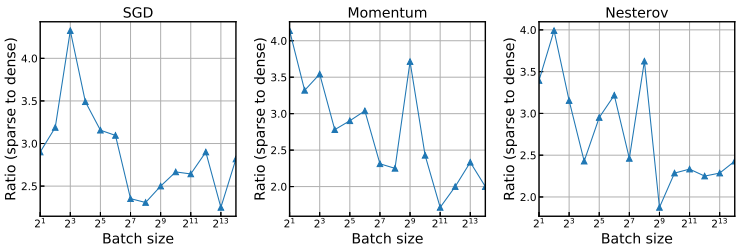


(a) SGD

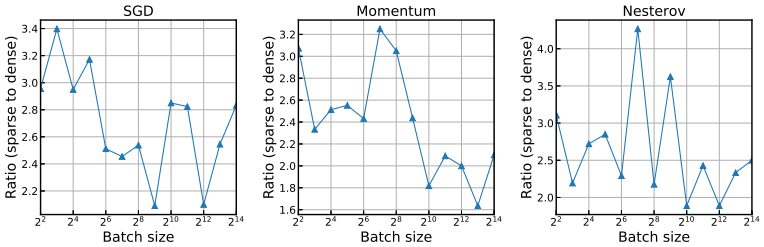


(b) Momentum

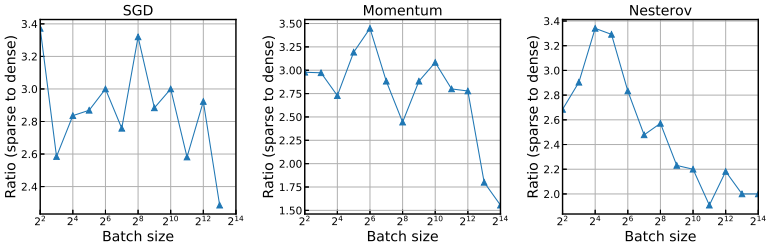
Figure 5.11: Results for the effects of data parallelism for the workloads of {CIFAR-10, ResNet-8, SGD/Momentum} with a constant learning rate.



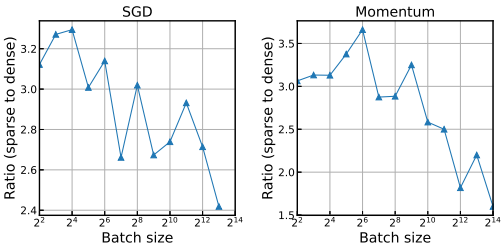
(a) MNIST, Simple-CNN, SGD/Momentum/Nesterov with a constant learning rate



(b) Fashion-MNIST, Simple-CNN, SGD/Momentum/Nesterov with a constant learning rate



(c) CIFAR-10, ResNet-8, SGD/Momentum/Nesterov with a linear learning rate decay



(d) CIFAR-10, ResNet-8, SGD/Momentum with a constant learning rate

Figure 5.12: Differences in ratio between (90%) sparse network's steps-to-result to dense network's, across different batch sizes for all workloads presented in this work. The difference ranges between (1.5, 4.5) overall. Note that the ratio difference > 1 indicates that it requires more number of training iterations (*i.e.*, steps-to-result) for sparse network compared to dense network. Also, the difference seems to decrease as batch size increases, especially for Momentum based optimizers. This potentially indicates that sparse neural networks can benefit from large batch training, despite the general difficulty therein.

5.D.2 Metaparameter search results

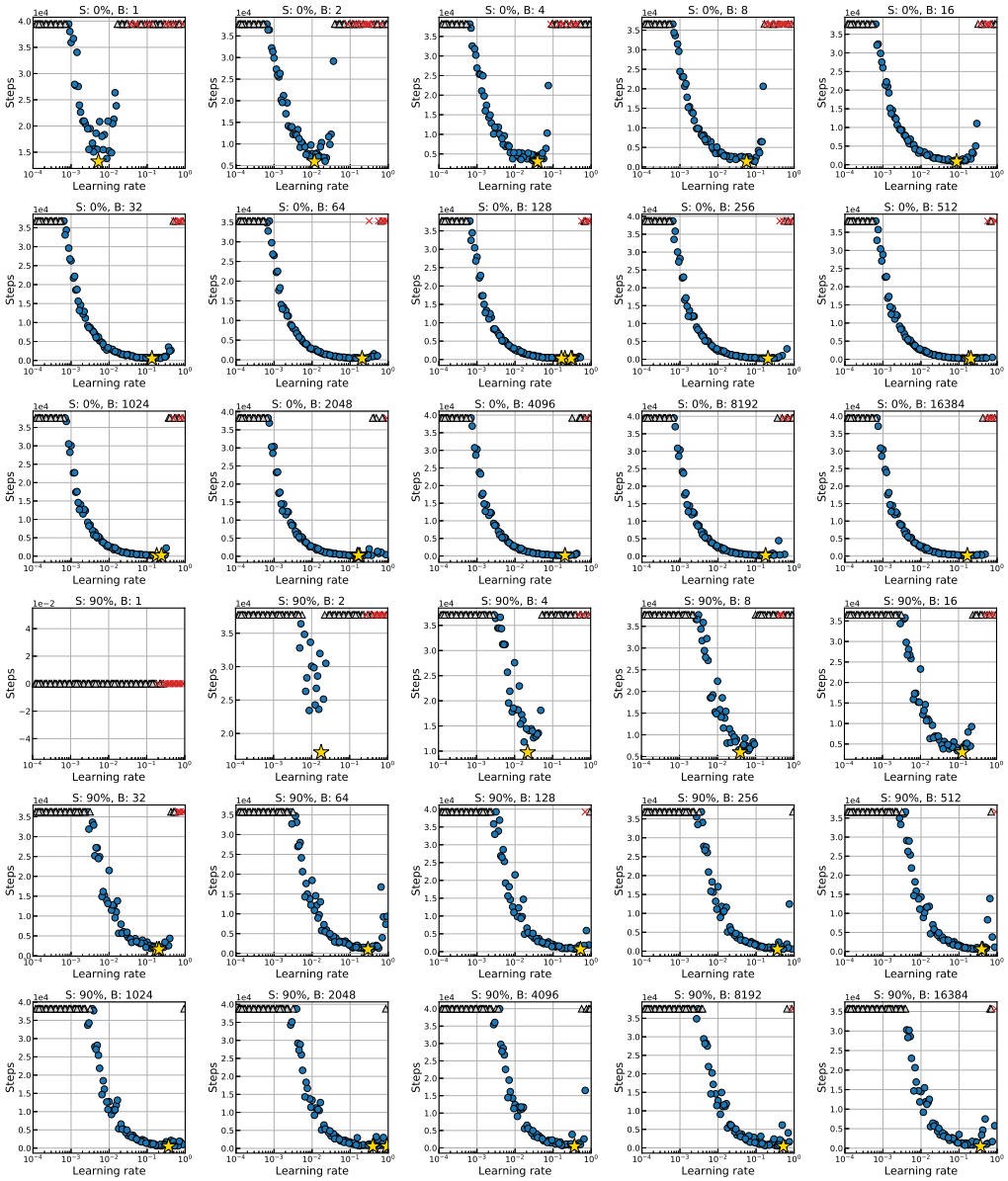


Figure 5.13: Meataparameter search results for the workloads of {MNIST, Simple-CNN, SGD} with a constant learning rate.

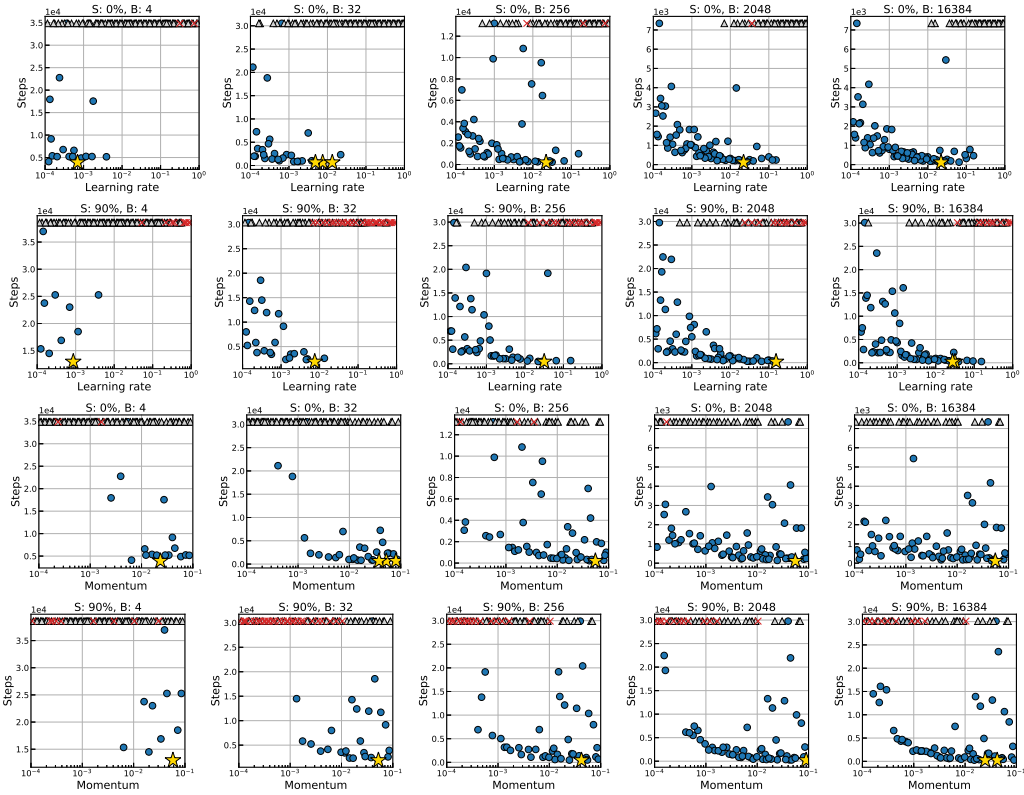


Figure 5.14: Meataparameter search results for the workloads of {MNIST, Simple-CNN, Momentum} with a constant learning rate.

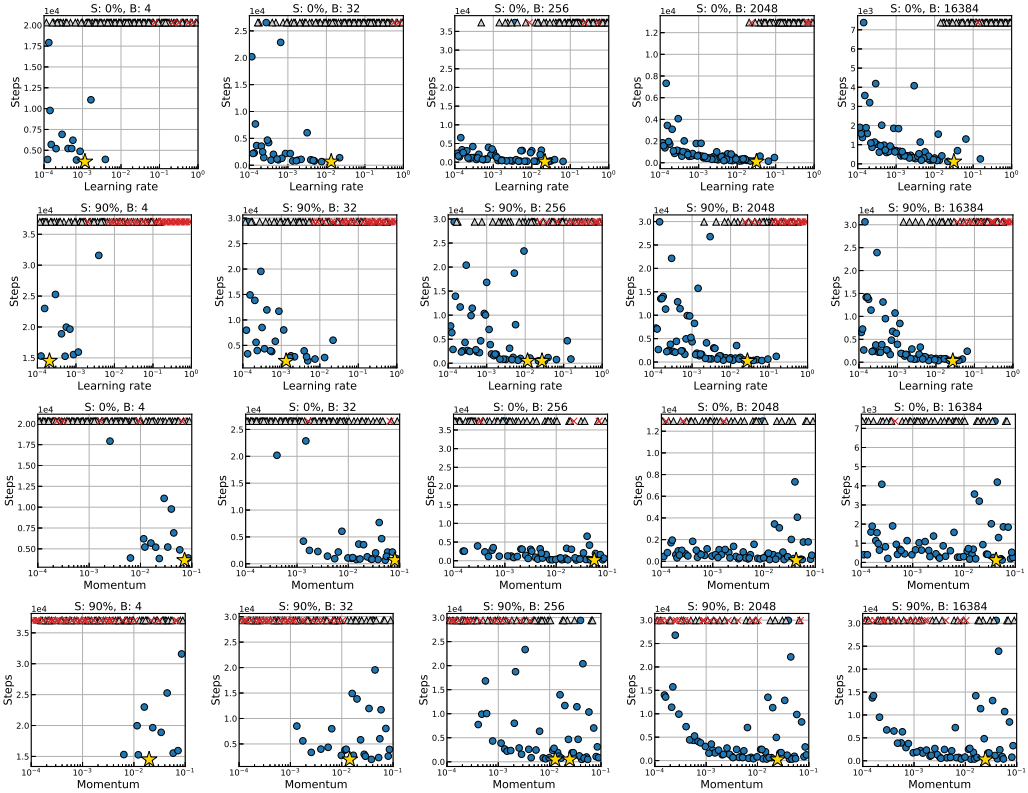


Figure 5.15: Meataparameter search results for the workloads of {MNIST, Simple-CNN, Nesterov} with a constant learning rate.

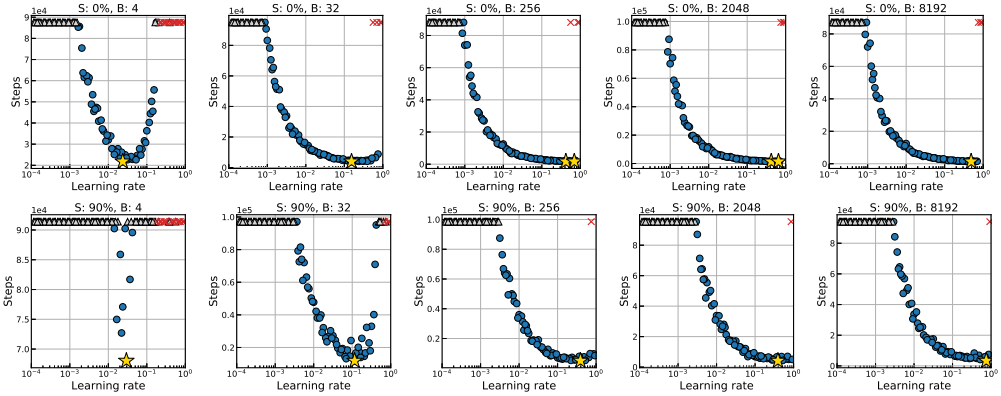


Figure 5.16: Meataparameter search results for the workloads of {CIFAR-10, ResNet-8, SGD} with a constant learning rate.

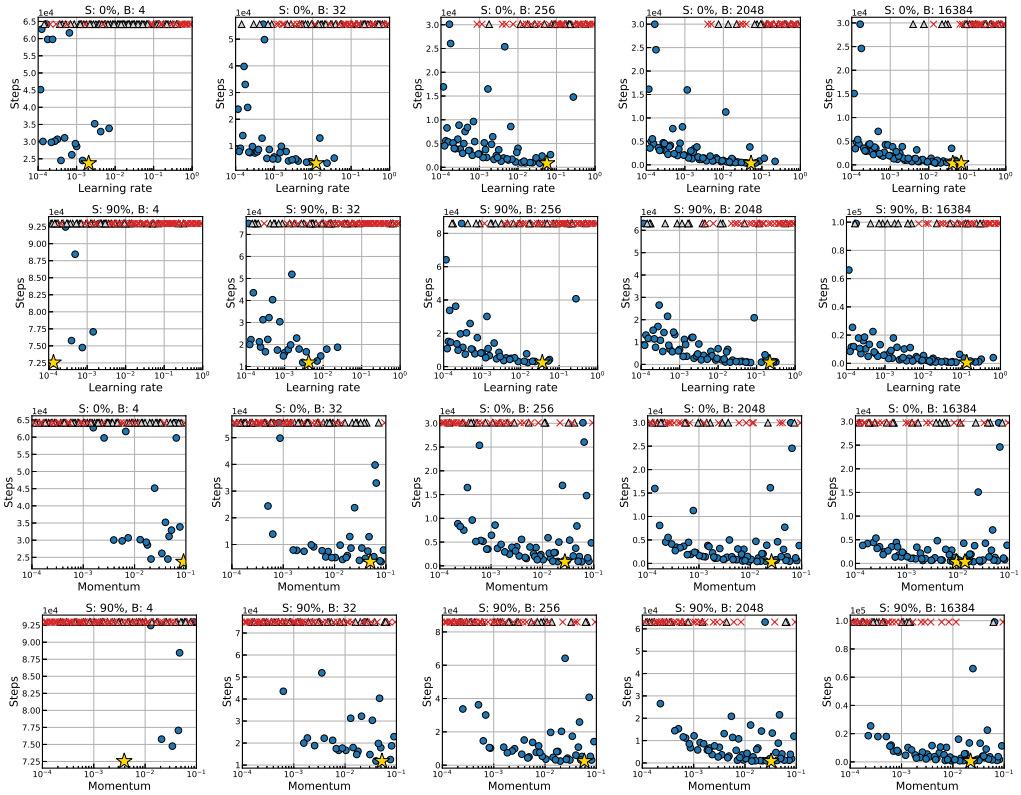


Figure 5.17: Meataparameter search results for the workloads of {CIFAR-10, ResNet-8, Momentum} with a constant learning rate.

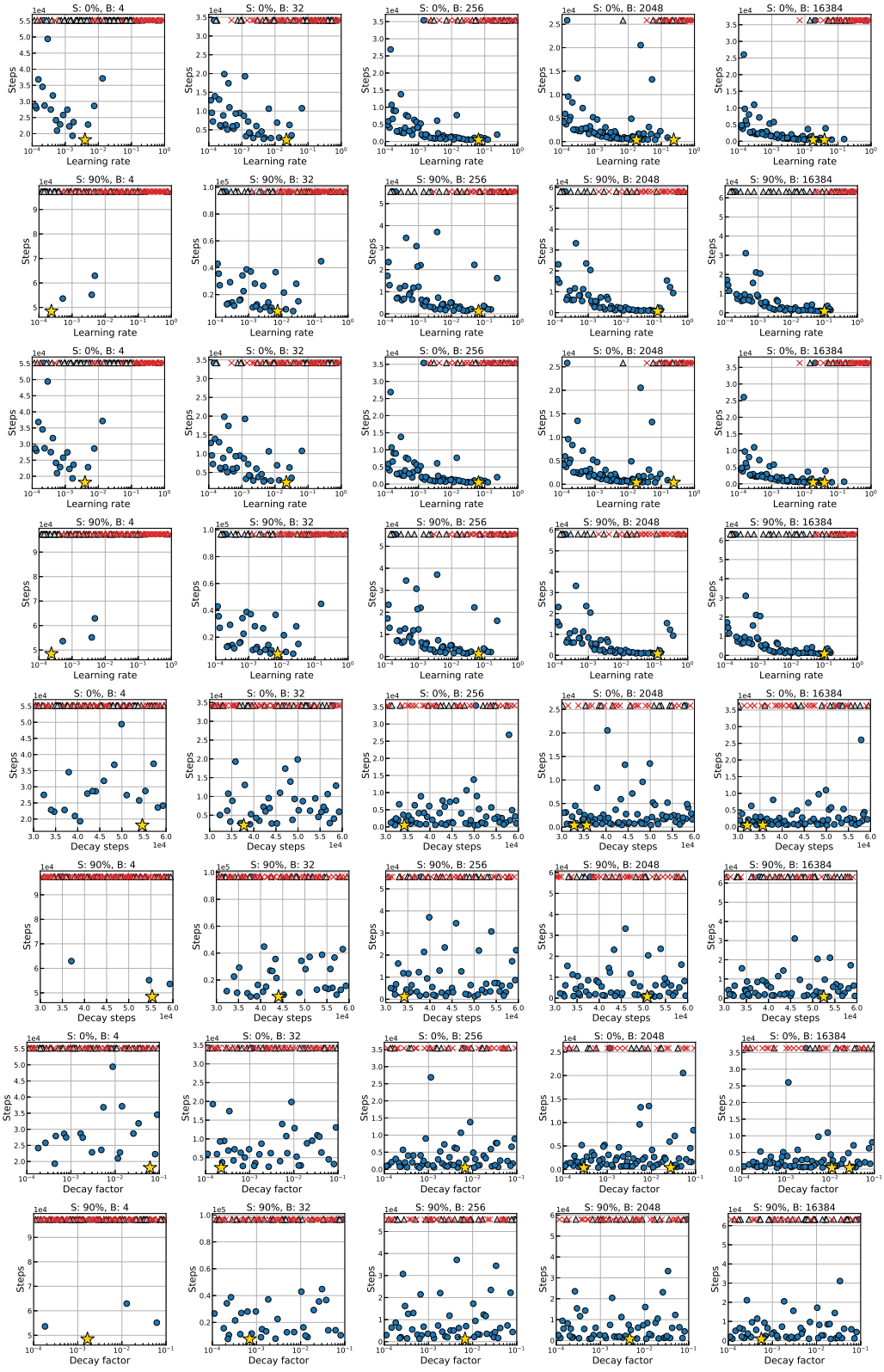


Figure 5.18: Meataparameter search results for the workloads of {CIFAR-10, ResNet-8, Nesterov} with a linear learning rate decay.

Chapter 6

Conclusion

So far, this integrated thesis presented papers covering different aspects of sparse neural networks, starting from their construction, to initialization, and further to large-scale training. Finally, this chapter conclude this thesis by summarizing the main contributions of each work presented in this thesis, and discussing remaining challenges and potential future directions.

6.1 Discussion of contributions

Chapter 3: SNIP: Single-shot network pruning based on connection sensitivity

In this chapter, we presented an efficient pruning algorithm, called SNIP, that can prune a large amount of parameters in a neural network model, prior to training in a single shot.

Network pruning has a decades-long history of research, and numerous variants have been developed to make large neural networks more computations and memory efficient. However, existing works are designed to have many hyperparameters that require some prior/expert knowledge often based on ad-hoc intuitions, undermining the utility of pruning. Moreover, they require some form of training steps, either as pre-training to convergence or as iterative training steps that are intertwined with the pruning procedure, which by posing additional workloads hinders pruning to be applied to neural networks generally. Considering the potential efficiency by pruning, these algorithmic weaknesses were quite critical.

Recently, network pruning research received a lot of attention largely due to Frankle and Carbin [2019] which claims that a deep neural network contains a small subnetwork that can be trained to achieve a similar performance of the

dense network. Although this general phenomenon is not new, and their hypothesis remains non-deterministic and statistically inexplicable, it is noteworthy that their subnetworks are subject to the initial random weights used to find the subnetwork, because it might indicate the effectiveness of pruning at initialization approach.

SNIP has its significance in the regard. This chapter has demonstrated that pruning can be done on a randomly initialized and untrained neural network for the first time in the literature. The algorithm does not introduce any additional hyperparameters, and the pruned sparse neural network by SNIP simply needs to be trained in the standard way. Also, the connection sensitivity criterion developed for SNIP is quite versatile in the sense that as a architecture-agnostic, single-shot measure it can be used to visualize which parameters are important to perform a given task, facilitating the analysis of the sparsity patterns obtained. At the time of submission, SNIP demonstrated its effectiveness by achieving state-of-the-art performance in accuracy as well as sparsity level of the pruned models.

Since then, it generated tremendous interest in network pruning, and many approaches started to shift toward pruning at initialization scheme.

Chapter 4: A signal propagation perspective for pruning neural networks at initialization

This chapter extended the pruning at initialization scheme presented in the previous chapter, by delving into the role of network initialization for pruning and training of neural networks. Although SNIP performs well, it remains unclear, at the time of submission, as to what it means to prune untrained neural networks and why it can be effective.

The investigation of the effect of initialization on pruning firstly revealed that initial weights can have critical impact on pruning results, based on empirical results on MLP networks with nonlinearity as well as the mathematical decomposition of the connection sensitivity criterion used for pruning as an interplay between gradients and scale of weights. This evidence is explained by the fact that the initial random weights are behind the success of training deep neural networks, which are sampled properly such that the error vectors preserve its norm under backpropagation.

We therefore developed a new signal propagation perspective for pruning at initialization: a sufficient initialization condition – layerwise dynamical isometry

– ensures the initial weights to be chosen appropriately so that the propagation of input signals into layers of random weights does not cause any saturating error signals or unreliable connection sensitivity measurements.

Importantly, we observed that pruning degrades Jacobian singular values of the pruned sparse network, an indicator of signal propagation. In order to address this issue, we proposed a simple, yet effective data free optimization step after pruning, which, by ensuring what we call approximate dynamical isometry, restores the degraded signal propagation properties back to the level of the original unpruned network, and in turn, significantly accelerates the speed of training sparse neural networks, even for randomly wired/pruned neural networks.

This result indicates that properly initializing sparse neural networks can be critical for the success of efficient network training under sparsity, which was unknown at the time of submission of this work.

Chapter 5: Understanding the effects of data parallelism and sparsity on neural network training

In this chapter, we established theoretical results that precisely account for two phenomena: the effects of data parallelism and sparsity on neural network training. Our results are significant, in that these phenomena, which have only been addressed partially and empirically and thus remained as debatable, are now theoretically verified with more accurate descriptions and applied to general non-convex objectives.

To be more specific, the effect of data parallelism has been understood based on empirical observations that the number of training steps required to reach a fixed goal (or, steps-to-result) decreases by increasing batch size, which is described with three different phases, namely perfect scaling, diminishing returns, and maximal data parallelism. Although this scaling trend is observed generally even in our experiments regardless of sparsity, the interpretation model of three distinctive regions was not precisely fitting the results. To address this issue, we developed theoretical insights that explain the relationship between batch size and steps-to-result more generally, based on the convergence properties of stochastic gradient methods, which applies to any non-convex objectives.

Furthermore, our results provide a new interpretation for why training sparse neural networks can be more difficult. We find that sparsity induced to neural networks decreases the Lipschitz smoothness, indicating that pruning results in

sparse networks whose gradient changes relatively too quickly compared to the dense network with more parameters. This means that the prediction function becomes less smooth after pruning potentially hindering the training process. Our theoretical results also support this empirical finding well, by showing that decreased Lipschitz smoothness will increase steps-to-result for a fixed batch size.

6.2 Remaining challenges and future directions

While important progress have been made, there remain many challenges and open questions to address. In this section, we discuss remaining challenges and potential future directions to explore.

Finding the smallest sparse neural network. While SNIP has been proven to be highly competitive to prior arts, it may not work very well for pruning an extreme number of parameters (*e.g.*, more than 99 percent). To be more specific, for an extremely high level of sparsity, SNIP could remove the entire parameters of a certain layer, resulting in a completely disconnected network that cannot be trained. This catastrophic failure is potentially because the saliency measured for a large number of parameters at a single-shot does not reflect some interplays between many weights. One straightforward approach to enhance SNIP would be to have an explicit mechanism to prevent the removal of entire layers such as setting a maximum number of parameters to remove for each layer. Ultimately, however, pruning can be viewed as an optimization problem that aims at finding the global minima constrained to parameters having as many zeros as possible. While there could be various ways to approach this probem, as neural networks are non-convex that are currently solved by stochastic algorithms, it essentially has the nature of iterative processing. Perhaps this explains why algorithms that prune small amounts of parameters gradually (rather than all at once) often achieve higher sparsity, although this would require some pruning – train schedule. To this end, exploiting meta-learning by which the algorithm becomes increasingly more aware of how pruning progresses could be useful.

Finding optimally trainable sparse neural networks. In Chapter 4, we observed that pruning can degrade signal propagation properties and slow down training. We attributed this to the broken dynamical isometry due to pruning, which can be restored back by performing a simple optimization process. Albeit effective,

this two-stage process of compression first and isometry enforcement next can be suboptimal. For example, in case the sparsity level is too high, the sparse topology obtained by pruning may not be capable to be compatible with remaining weights to satisfy exact dynamical isometry. In fact, this is a fundamental limitation of "prune once and for all" approach; in other words, a pruned parameter will never be revived in this scheme. Also, the network parameters keep on updating during training, which will break dynamic isometry ensured on the sparse network. To this end, understanding the complex interplay between optimisation and signal propagation could be a potential key to finding optimally trainable sparse neural networks, and whether keeping isometry properties throughout training (rather than only at initialization), along with the possibility of dynamic pruning, will be beneficial or not is an interesting open question.

Generalization characteristics of sparse neural networks. After all, what practitioners care about at the end is the generalization performance of efficient models. While tremendous progress has been made in the research of sparse neural networks, understanding their generalization characteristics has never been rigorously or theoretically studied. Understanding generalization of deep learning is an active research topic, and generalization bounds derived for the regular neural network models can be borrowed to study for sparse neural networks. While there remain technical challenges such as how to address the different algorithmic properties of pruning variants, one aspect is the same for all: the number of remaining parameters in the pruned sparse network. Focusing on this aspect, perhaps a generalization bound for randomly pruned model can be developed first to start with. More generally, this line of study can help shed light on the effect of overparameterization in deep learning.

Data parallelism in the wild. We have studied the effect of data parallelism under the influence of sparsity in Chapter 5. While our results render some significance to the research community, there are fundamental limitations that need addressing. Firstly, the trend observed on relatively small scale experiments may need to be confirmed for large-scale scenarios, even though performing experiments for data parallelism requires a tremendous amount of computing resources. Also, the results are developed for the constant learning rate scenario without momentum. In fact, the convergence rate of SGD with constant learning rate is already theoretically optimal, and therefore, a different learning rate schedule or accelerated gradient

methods (*e.g.*, momentum) will only be hoped to improve the constant terms in the convergence rate. Nonetheless, the effect of data parallelism needs better to be elucidated more precisely, potentially by extending our theoretical results to more practical setups including SGD with momentum.

Bibliography

Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *NeurIPS Workshop: Deep Learning at Supercomputer Scale*, 2017.

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. *ICLR*, 2018.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *ICML*, 2018.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *PAMI*, 2017.

Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. *Proceedings of the conference on high performance computing networking, storage and analysis*, 2009.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *ICLR*, 2018.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *PAMI*, 2013.

Léon Bottou. Stochastic gradient learning in neural networks. *Neuro Nîmes*, 1991.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Leo Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 1995.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev. "Learning-compression" algorithms for neural net pruning. *CVPR*, 2018.

Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. *NIPS*, 1989.

Lingjiao Chen, Hongyi Wang, Jinman Zhao, Dimitris Papailiopoulos, and Paraschos Koutris. The effect of network width on the performance of large-batch training. *NeurIPS*, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

Bin Dai, Chen Zhu, Baining Guo, and David Wipf. Compressing neural networks using the variational information bottleneck. *ICML*, 2018.

Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709*, 2016.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. 2012.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *NIPS*, 2017.

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *ICLR*, 2019.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. *CVPR*, 2020.

Andries Petrus Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *Neural Networks*, 2001.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *arXiv preprint arXiv:1906.10732*, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Saeed Ghadimi, Guanghui Lan, and Hongchao Zhang. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 155(1-2):267–305, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, 2010.

Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep learning. *MIT press Cambridge*, 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *NIPS*, 2016.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *ICML*, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *NIPS*, 2015b.

Karen Hao. Tiny AI. *MIT Technology Review*, 2020.

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. *Neural Networks*, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *ICCV*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *ICCV*, 2017a.

Qing He, Thilo Koehler, Antony D'Avirro, and Chetan Gupta. A highly efficient, real-time text-to-speech system deployed on cpus. *Facebook AI Blog*, 2020.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *ICCV*, 2017b.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. *ECCV*, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *NeurIPS*, 2017.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 2019.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *NIPS*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

Masumi Ishikawa. Structural learning with forgetting. *Neural Networks*, 1996.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *BMVC*, 2014.

Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. *ICML*, 2018.

Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *Neural Networks*, 1990.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *ICML*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. *ICML*, 2012.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, 2015.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. *NIPS*, 1990.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. *Neural Networks: Tricks of the Trade*, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. SNIP: Single-shot network pruning based on connection sensitivity. *ICLR*, 2019.

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. *ICLR*, 2020.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

Jiajia Li, Jimeng Sun, and Richard Vuduc. Hicoo: hierarchical storage of sparse tensors. *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. *NeurIPS*, pages 8157–8166, 2018.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *NeurIPS*, 2017.

Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. *ICLR*, 2020.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *ICLR*, 2019.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *ICLR*, 2018.

Ada Lovelace. Notes upon L. F. Menabrea's "Sketch of the analytical engine invented by Charles Babbage". 1842.

Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, 2016.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *ICML*, 2017a.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2017b.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *ICML*, 2019.

Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. *NIPS*, 1989.

Yusuke Nagasaka, Satoshi Matsuoka, Ariful Azad, and Aydın Buluç. High-performance sparse matrix-matrix products on intel knl and multicore architectures. *Proceedings of the 47th International Conference on Parallel Processing Companion*, 2018.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *ICLR*, 2017.

Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. *NeurIPS*, 2017.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. *NIPS*, 2015.

Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 1992.

OpenAI. Openai five. 2018.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML*, 2013.

Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *NeurIPS*, 2017.

Jeffrey Pennington, Samuel S Schoenholz, and Surya Ganguli. The emergence of spectral universality in deep networks. *AISTATS*, 2018.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *ICLR*, 2018.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *NeurIPS*, 2016.

Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. *ECCV*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Russell Reed. Pruning algorithms-a survey. *Neural Networks*, 1993.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 1958.

Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *ICLR*, 2014.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 2015.

Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *ICLR*, 2017.

Abigail See, Minh-Thang Luong, and Christopher D Manning. Compression of neural machine translation models via pruning. *CoNLL*, 2016.

Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *JMLR*, 2019.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ICLR*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

Sayanan Sivaraman and Mohan Manubhai Trivedi. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. *IEEE transactions on intelligent transportation systems*, 2013.

Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *ICLR*, 2018.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackerman, et al. A deep learning approach to antibiotic discovery. *Cell*, 2020.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CVPR*, 2015.

Wojciech Tarnowski, Piotr Warchoł, Stanisław Jastrzębski, Jacek Tabor, and Maciej A Nowak. Dynamical isometry is achieved in residual networks in a universal way for any activation function. *AISTATS*, 2019.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *ICLR*, 2017.

Vladimir N Vapnik. An overview of statistical learning theory. *Neural Networks*, 1999.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *ICLR*, 2020.

Weiran Wang and Nathan Srebro. Stochastic nonconvex optimization with large minibatches. *arXiv preprint arXiv:1709.08728*, 2017.

Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. Generalization by weight-elimination with application to forecasting. *NIPS*, 1991.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *NIPS*, 2016.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. *ICML*, 2018.

Carl Yang, Aydın Buluç, and John D Owens. Design principles for sparse matrix multiplication on the gpu. *European Conference on Parallel Processing*, 2018.

Ge Yang and Samuel Schoenholz. Mean field residual networks: On the edge of chaos. *NeurIPS*, 2017.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S Schoenholz. A mean field theory of batch normalization. *ICLR*, 2019.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. 2018.

Sergey Zagoruyko. 92.45% on cifar-10 in torch. *Torch Blog*, 2015.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *BMVC*, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2017.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *NeurIPS*, 2019a.

Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *ICLR*, 2019b.

Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *ICLR*, 2020.

Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CVPR*, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.