Data Parallelism in Training Sparse Neural Networks

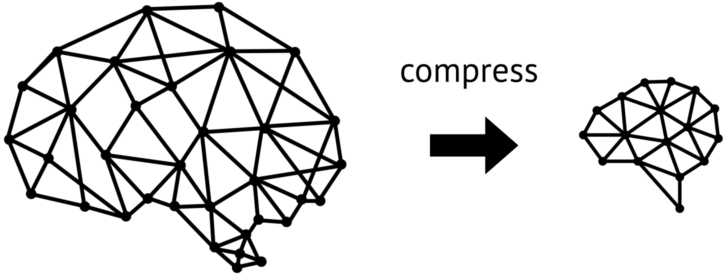
Namhoon Lee[1], Philip Torr[1], Martin Jaggi[2]

[1]University of Oxford, [2]EPFL

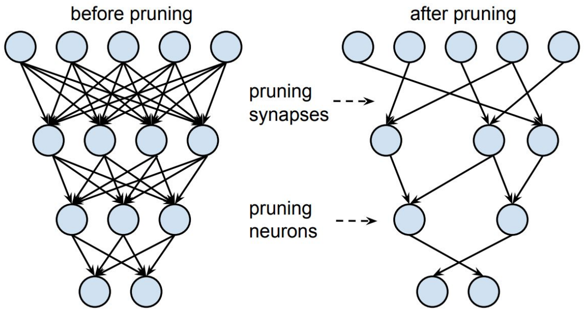
ICLR 2020 Workshop on PML4DC

Motivation

Compressing neural networks can save a large amount of memory and computational cost.



Motivation

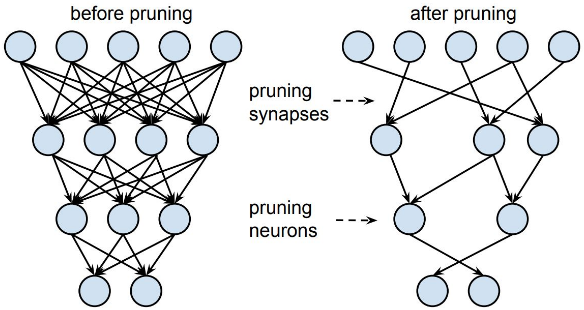


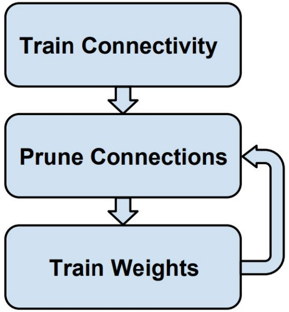
Compressing neural networks can save a large amount of memory and computational cost.

Network pruning is an effective methodology to compress large neural networks.

Han et al. 2015

Motivation





Han et al. 2015

Compressing neural networks can save a large amount of memory and computational cost.

Network pruning is an effective methodology to compress large neural networks, but typically requires training steps (Han *et al.*, 2015, Liu *et al.*, 2019, Frankle *et al.*, 2019).

Motivation

Published as a conference paper at ICLR 2019

SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY

Namhoon Lee, Thalaiyasingam Ajanthan & Philip H. S. Torr
University of Oxford
{namhoon,ajanthan,phst}@robots.ox.ac.uk

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.

Published as a conference paper at ICLR 2020

PICKING WINNING TICKETS BEFORE TRAINING BY PRESERVING GRADIENT FLOW

Chaoqi Wang, Guodong Zhang, Roger Grosse
University of Toronto, Vector Institute
{cqwang, gdzhang, rgrosse}@cs.toronto.edu

ABSTRACT

Overparameterization has been shown to benefit both the optimization and generalization of neural networks, but large networks are resource hungry at both training and test time. Network pruning can reduce test-time resource requirements, but is typically applied to trained networks and therefore cannot avoid the expensive training process. We aim to prune networks at initialization, thereby saving resources at training time as well. Specifically, we argue that efficient training requires preserving the gradient flow through the network. This leads to a simple but effective pruning criterion we term Gradient Signal Preservation (GraSP). We empirically investigate the effectiveness of the proposed method with extensive experiments on CIFAR-10, CIFAR-100, Tiny-ImageNet and ImageNet, using VGGNet and ResNet architectures. Our method can prune 80% of the weights of a VGG-16 network on ImageNet at initialization, with only a 1.6% drop in top-1 accuracy. Moreover, our method achieves significantly better performance than the baseline at extreme sparsity levels. Our code is made public at: https://github.com/alecwangcq/GraSP.

Compressing neural networks can save a large amount of memory and computational cost.

Network pruning is an effective methodology to compress large neural networks, but typically requires training steps (Han *et al.*, 2015, Liu *et al.*, 2019, Frankle *et al.*, 2019).

Pruning can be done at initialization prior to training (Lee *et al.*, 2019, Wang *et al.*, 2020).

Motivation

Published as a conference paper at ICLR 2019

SNIP: Single-shot Network Pruning based on Connection Sensitivity

Namhoon Lee, Thalaiyasingam Ajanthan & Philip H. S. Torr
University of Oxford
{namhoon,ajanthan,phst}@robots.ox.ac.uk

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.

Published as a conference paper at ICLR 2020

Picking Winning Tickets Before Training by Preserving Gradient Flow

Chaoqi Wang, Guodong Zhang, Roger Grosse
University of Toronto, Vector Institute
{cqwang, gdzhang, rgrosse}@cs.toronto.edu

Abstract

Overparameterization has been shown to benefit both the optimization and generalization of neural networks, but large networks are resource hungry at both training and test time. Network pruning can reduce test-time resource requirements, but is typically applied to trained networks and therefore cannot avoid the expensive training process. We aim to prune networks at initialization, thereby saving resources at training time as well. Specifically, we argue that efficient training requires preserving the gradient flow through the network. This leads to a simple but effective pruning criterion we term Gradient Signal Preservation (GraSP). We empirically investigate the effectiveness of the proposed method with extensive experiments on CIFAR-10, CIFAR-100, Tiny-ImageNet and ImageNet, using VGGNet and ResNet architectures. Our method can prune 80% of the weights of a VGG-16 network on ImageNet at initialization, with only a 1.6% drop in top-1 accuracy. Moreover, our method achieves significantly better performance than the baseline at extreme sparsity levels. Our code is made public at: https://github.com/alecwangcq/GraSP.

What about training?

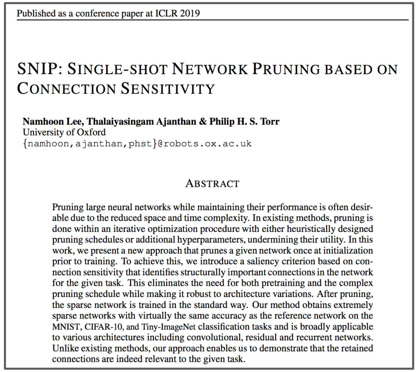
Compressing neural networks can save a large amount of memory and computational cost.

Network pruning is an effective methodology to compress large neural networks, but typically requires training steps (Han *et al.*, 2015, Liu *et al.*, 2019, Frankle *et al.*, 2019).

Pruning can be done at initialization prior to training (Lee *et al.*, 2019, Wang *et al.*, 2020).

Little has been studied about the training aspects of sparse neural networks (Evci *et al.*, 2019, Lee *et al.* 2020).

Motivation



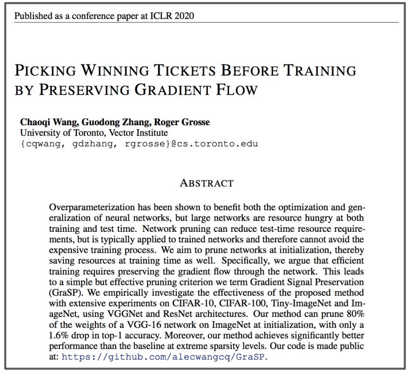
Published as a conference paper at ICLR 2019

SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY

Namhoon Lee, Thalaiyasingam Ajanthan & Philip H. S. Torr
University of Oxford
{namhoon,ajanthan,phst}@robots.ox.ac.uk

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.



Published as a conference paper at ICLR 2020

PICKING WINNING TICKETS BEFORE TRAINING BY PRESERVING GRADIENT FLOW

Chaoqi Wang, Guodong Zhang, Roger Grosse
University of Toronto, Vector Institute
{cqwang, gdzhang, rgrosse}@cs.toronto.edu

ABSTRACT

Overparameterization has been shown to benefit both the optimization and generalization of neural networks, but large networks are resource hungry at both training and test time. Network pruning can reduce test-time resource requirements, but is typically applied to trained networks and therefore cannot avoid the expensive training process. We aim to prune networks at initialization, thereby saving resources at training time as well. Specifically, we argue that efficient training requires preserving the gradient flow through the network. This leads to a simple but effective pruning criterion we term Gradient Signal Preservation (GraSP). We empirically investigate the effectiveness of the proposed method with extensive experiments on CIFAR-10, CIFAR-100, Tiny-ImageNet and ImageNet, using VGGNet and ResNet architectures. Our method can prune 80% of the weights of a VGG-16 network on ImageNet at initialization, with only a 1.6% drop in top-1 accuracy. Moreover, our method achieves significantly better performance than the baseline at extreme sparsity levels. Our code is made public at: https://github.com/alecwangcq/GraSP.

What about training?

Compressing neural networks can save a large amount of memory and computational cost.

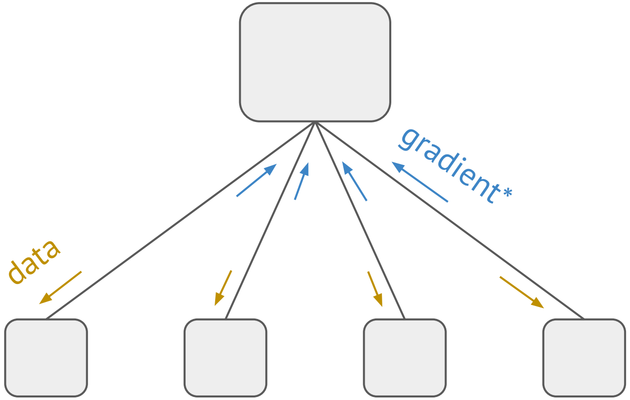
Network pruning is an effective methodology to compress large neural networks, but typically requires training steps (Han *et al.*, 2015, Liu *et al.*, 2019, Frankle *et al.*, 2019).

Pruning can be done at initialization prior to training (Lee *et al.*, 2019, Wang *et al.*, 2020).

Little has been studied about the training aspects of sparse neural networks (Evci *et al.*, 2019, Lee *et al.* 2020).

Our focus ⇒ ***Data Parallelism on Sparse Networks***.

Data parallelism?



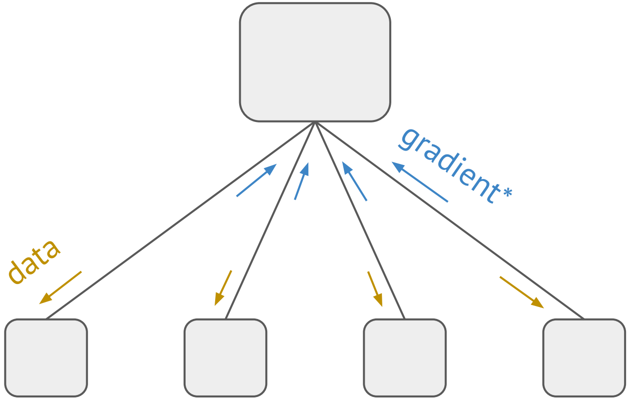
A centralized, synchronous, parallel computing system.

It refers to distributing training data to multiple processors and computing gradient in parallel, so as to accelerate training.

The amount of data parallelism is equivalent to the batch size for optimization on a single node.

*It can be a higher-order derivative.

Data parallelism?



A centralized, synchronous, parallel computing system.

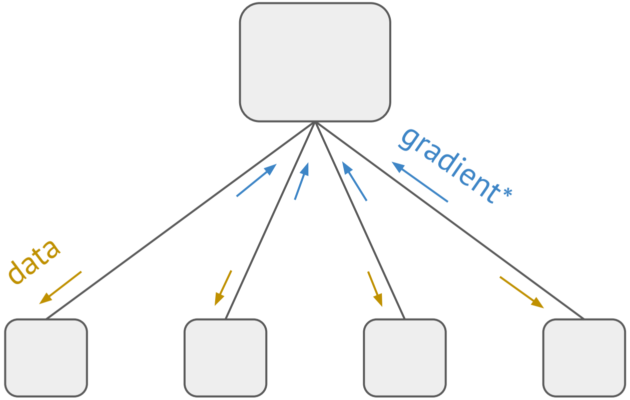
It refers to distributing training data to multiple processors and computing gradient in parallel, so as to accelerate training.

The amount of data parallelism is equivalent to the batch size for optimization on a single node.

Understanding the effect of batch size is crucial and an active research topic (Hoffer *et al.*, 2017, Smith *et al.*, 2018, Shallue *et al.*, 2019).

*It can be a higher-order derivative.

Data parallelism?



A centralized, synchronous, parallel computing system.

It refers to distributing training data to multiple processors and computing gradient in parallel, so as to accelerate training.

The amount of data parallelism is equivalent to the batch size for optimization on a single node.

Understanding the effect of batch size is crucial and an active research topic (Hoffer *et al.*, 2017, Smith *et al.*, 2018, Shallue *et al.*, 2019).

Sparse networks can enjoy a reduced memory and communication cost in distributed settings.

*It can be a higher-order derivative.

Steps-to-result

It refers to the lowest number of training steps required to reach a goal out-of-sample error.

Steps-to-result

It refers to the lowest number of training steps required to reach a goal out-of-sample error.

We measure steps-to-result for **all combinations** of

- workload (data set, model, optimization algorithm)
- batch size (from 1 to 16384)
- sparsity level (from 0% to 90%)

Errors are measured on the entire validation set, at every fixed interval during training.

Our experiments are largely motivated by and closely follow experiments in Shallue *et al.*, 2019.

Steps-to-result

It refers to the lowest number of training steps required to reach a goal out-of-sample error.

We measure steps-to-result for **all combinations** of

- workload (data set, model, optimization algorithm)
- batch size (from 1 to 16384)
- sparsity level (from 0% to 90%)

Errors are measured on the entire validation set, at every fixed interval during training.

Our experiments are largely motivated by and closely follow experiments in Shallue *et al.*, 2019.

Metaparameters

They refer to parameters whose values are set before the learning begins, such as network size for model, or learning rate for optimization.

Steps-to-result

It refers to the lowest number of training steps required to reach a goal out-of-sample error.

We measure steps-to-result for **all combinations** of

- workload (data set, model, optimization algorithm)
- batch size (from 1 to 16384)
- sparsity level (from 0% to 90%)

Errors are measured on the entire validation set, at every fixed interval during training.

Our experiments are largely motivated by and closely follow experiments in Shallue *et al.*, 2019.

Metaparameters

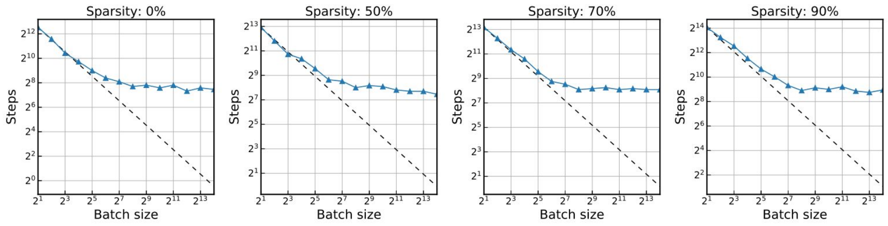
They refer to parameters whose values are set before the learning begins, such as network size for model, or learning rate for optimization.

We tune all optimization metaparameters to avoid any assumptions on the optimal metaparameters as a function of batch size or sparsity level.

The optimal metaparameters are selected based on quasi-random search that yield best performance on a validation set.

We perform the search under a budget of trials, while taking into account a predefined search space for each metaparameter.

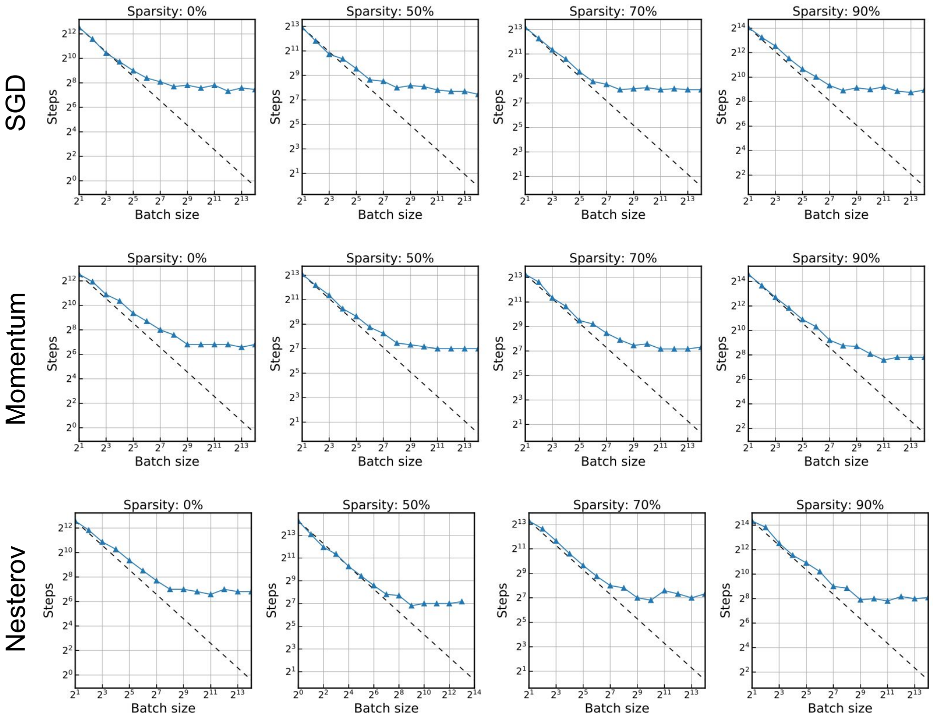
Data parallelism in training sparse neural networks



Universal scaling pattern across different sparsity:

- *perfect scaling*
- *diminishing returns*
- *maximal data parallelism*

Data parallelism in training sparse neural networks



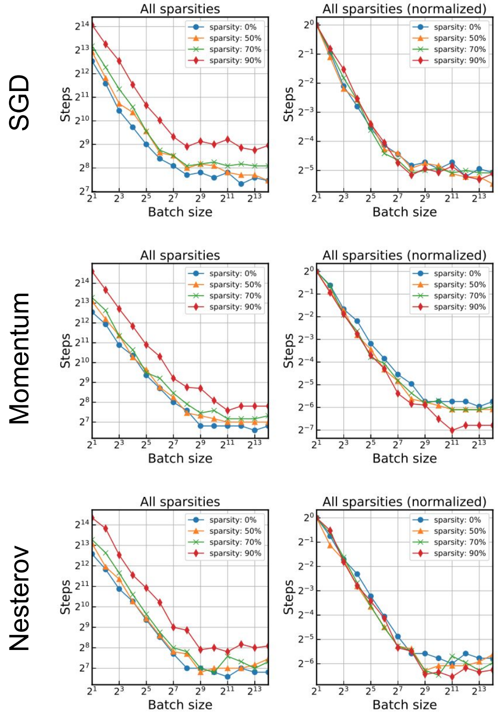
Universal scaling pattern across different sparsity:

- *perfect scaling*
- *diminishing returns*
- *maximal data parallelism*

Same patterns are observed for different optimizers:

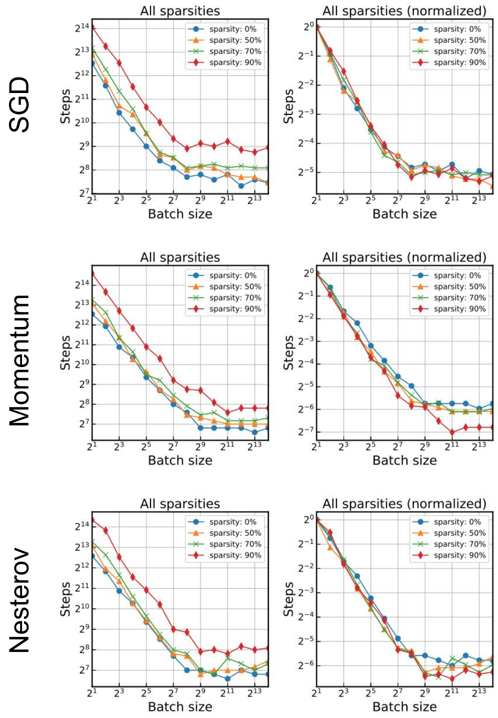
- SGD
- Momentum
- Nesterov

Putting different sparsity together



The higher sparsity, the longer it takes to train.
→ *General difficulty of training sparse networks*.

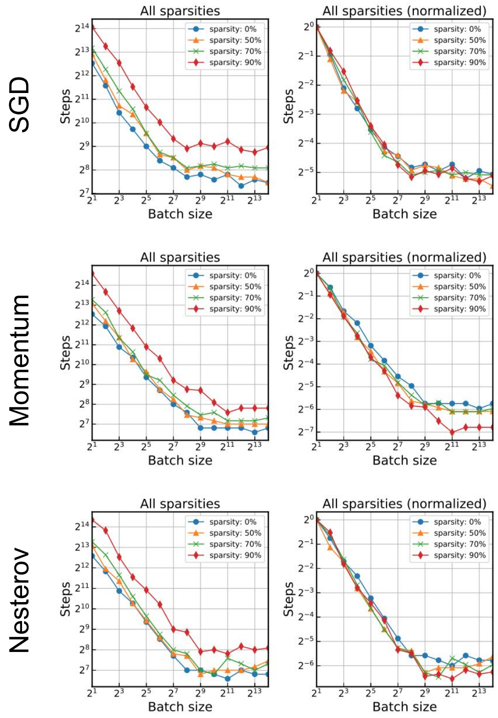
Putting different sparsity together



The higher sparsity, the longer it takes to train.
→ *General difficulty of training sparse networks*.

The regions of diminishing returns and maximal data parallelism appear at a similar point.
→ *The effects of data parallelism on sparse network is comparable to the dense case*.

Putting different sparsity together

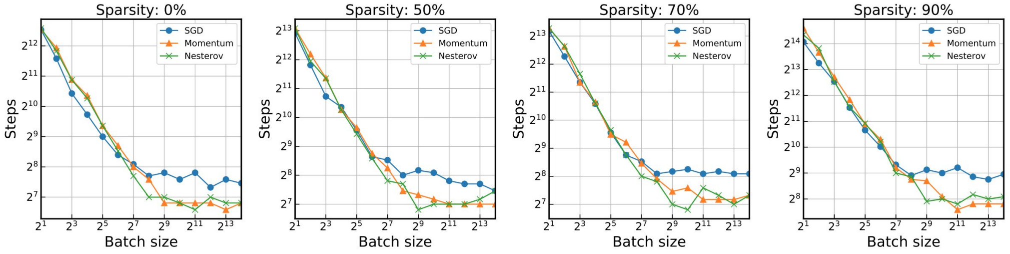


The higher sparsity, the longer it takes to train.
→ *General difficulty of training sparse networks*.

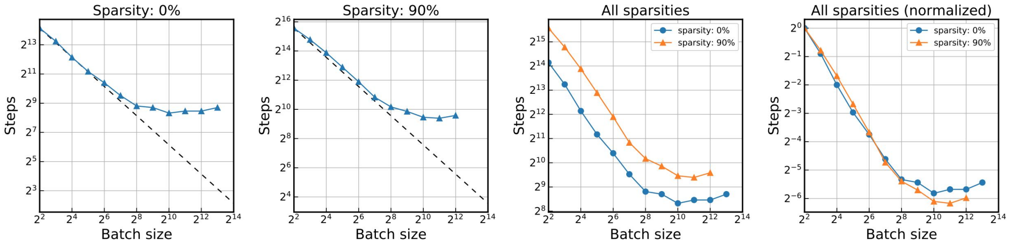
The regions of diminishing returns and maximal data parallelism appear at a similar point.
→ *The effects of data parallelism on sparse network is comparable to the dense case*.

A bigger critical batch size is achieved with highly sparse networks when using a momentum based SGD.
→ *Resources can be used more effectively*.

Continuing results



Comparing SGD, Momentum, and Nesterov optimizers.



CIFAR-10, ResNet-8, Nesterov with a linear learning rate decay.

Momentum based optimizers are better at exploiting large batch for all sparsity levels.

The data parallelism on sparse networks hold across different workloads.

Our results on sparse networks were unknown and is difficulty to estimate a priori.

More results can be found in the paper.

Summary

- *A universal scaling pattern for training sparse neural networks is observed across different workloads.*
- *Despite the general difficulty of training sparse neural networks, data parallelism on them remains no worse than that on dense networks.*
- *When training using a momentum based SGD, the critical batch size is often bigger for highly sparse networks than for dense networks.*
- *Our results render a positive impact on the community, by potentially helping practitioners to utilize resources more effectively.*