Distributed Learning of Fully Connected Neural Networks using Independent Subnet Training

Binhang Yuan
Rice University
by8@rice.edu

Cameron R. Wolfe
Rice University
crw13@rice.edu

Chen Dun
Rice University
cd46@rice.edu

Yuxin Tang
Rice University
yuxin.tang@rice.edu

Anastasios Kyrillidis
Rice University
anastasios@rice.edu

Chris Jermaine
Rice University
cmj4@rice.edu

ABSTRACT

Distributed machine learning (ML) can bring more computational resources to bear than single-machine learning, thus enabling reductions in training time. Distributed learning partitions models and data over many machines, allowing model and dataset sizes beyond the available compute power and memory of a single machine. In practice though, distributed ML is challenging when distribution is mandatory, rather than chosen by the practitioner. In such scenarios, data could unavoidably be separated among workers due to limited memory capacity per worker or even because of data privacy issues. There, existing distributed methods will utterly fail due to dominant transfer costs across workers, or do not even apply.

We propose a new approach to distributed fully connected neural network learning, called independent subnet training (IST), to handle these cases. In IST, the original network is decomposed into a set of narrow subnetworks with the same depth. These subnetworks are then trained locally before parameters are exchanged to produce new subnets and the training cycle repeats. Such a naturally "model parallel" approach limits memory usage by storing only a portion of network parameters on each device. Additionally, no requirements exist for sharing data between workers (i.e., subnet training is local and independent) and communication volume and frequency are reduced by decomposing the original network into independent subnets. These properties of IST can cope with issues due to distributed data, slow interconnects, or limited device memory, making IST a suitable approach for cases of mandatory distribution. We show experimentally that IST results in training times that are much lower than common distributed learning approaches.

PVLDB Reference Format:
Binhang Yuan, Cameron R. Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. Distributed Learning of Fully Connected Neural Networks using Independent Subnet Training. PVLDB, 15(8): 1581 - 1590, 2022.
doi:10.14778/3529337.3529343

PVLDB Artifact Availability:
https://github.com/BinhangYuan/IST_Release.

Proceedings of the VLDB Endowment, Vol. 15, No. 8 ISSN 2150-8097.
doi:10.14778/3529337.3529343

1 INTRODUCTION

Distributed training of neural networks (NN) over a compute cluster is a common task in modern computing systems [12, 19, 31, 46, 56]. Sometimes, it is the case that distributed training is a choice, and the practitioner is fully in control of the training environment. Namely, practitioners opt for distribution with the goal of using extra hardware to lower the wall-clock time to convergence, or to allow more resources (such as memory or CPU/GPU cycles) to be brought to bear on the problem of training a model. Consider the task of training a model such as GPT-3 [6], which requires on the order of 1000 years of GPU time to train. Thousands of GPUs can be used to lower the time to weeks or months. In such a training scenario, the different sites or compute units are typically connected with a high-speed network, and the hardware is often carefully tailored to the task of distributed training.

However, there are other cases where distribution is *mandatory* and the hardware may be sub-optimal—very far from the idealized environment a company such as OpenAI uses to train GPT-3. For example, consider a case where the training dataset is fragmented across several locations and organizations with privacy mandates preventing the possibility of centralized computing [15, 17, 59, 65, 85]. The training set may be large, and stored across hundreds of machines [11, 18, 32, 45, 53, 55, 74].

Here, the data sits where it happens to sit, and the computing environment is often not under the practitioners' control, forcing NN training to be conducted over a less-than-ideal hardware setup (e.g., too many compute nodes, CPUs, low-end GPUs, low-bandwidth interconnects, etc.). Such scenarios arise often in practice [8, 9, 41]. Even NN training on public compute clouds (e.g., Amazon EC2[1]) suffers from the combination of slow interconnects with high-performance GPUs [50].

We argue that common methods of distributing ML computations cannot be expected to handle such non-ideal environments gracefully, and new methods are needed. In distributed NN training, existing methods are roughly categorized into *model parallel* and *data parallel* methodologies. In practice, data parallel methodologies are most commonly used and supported due to their ease of implementation [1, 52]. In model parallel training [19, 31], portions of the NN are partitioned across different compute nodes, while, in the latter [27, 54, 84], the complete NN is updated with different data on each compute node. Data parallel methods suffer when

[1] 89% of cloud-based deep learning projects are executed on EC2, according to Amazon's marketing materials.

bandwidth is limited because they must transfer an entire model to each site in order to synchronize the computation. For a large model with many parameters, this is not a reasonable requirement. However, in typical mandatory distribution scenarios, model parallel methods are not a reasonable option, either. When data are sharded across sites, model parallel computing implies that different parts of a model can only be updated to reflect the data present at any given node. For these parts to stay synchronized, very fine-grained communication is required. Thus, neither data parallel nor model parallel is fully capable of handling mandatory distribution scenarios.

Independent subnet training. In response to this, we propose independent subnet training (IST), a novel distributed training technique on fully connected NNs that combines techniques from model and data parallel training to maximize communication efficiency. Inspired by dropout [66] and approximate matrix multiplication [25], IST decomposes fully-connected NN layers by distributing the neurons disjointly across different sites, forming a group of *subnets*. Then, each of these subnets is trained independently for one or more local stochastic gradient descent (SGD) iterations before synchronization [47]. After synchronization, parameters are re-distributed based on a new, random neuron sampling, and the local subnet training process repeats.

IST focuses upon the distributed training of NNs with fully-connected layers. Such a focus has also applications in diverse NN architectures (e.g., convolutional NNs [33]): the majority of NN models typically contain large, fully-connected layer, and such layers typically dominate the total number of parameters. As such, IST can be applied to the fully-connected portion of the networks to yield a performance speedup; see Sec. 3.

In cases of mandatory distribution, model parallel training is impractical because it requires that all data is present on the server that passes data into the network's input module.[2] Furthermore, we claim that, under mandatory distribution, IST is more capable than techniques like data parallel training due to its ability to reduce communication volume and memory usage to cope with hardware limitations. Synchronization in IST is simply an exchange of parameters between sites[3] (i.e., no parameters are shared between subnets) and no synchronization is required during local updates, thus reducing per-step communication volume on multiple fronts. Furthermore, IST limits its memory usage by only sending a small portion of its parameters to each device, which prevents model capacity from being limited by the memory of a single device.

Contributions. Our proposal aggressively reduces the communication bottlenecks that plague the scalability of most popular methods of distributed NN training. *As such, IST is most beneficial for training networks with fully-connected layers in cases of mandatory distribution, where training is highly-distributed and hardware is less-than-ideal.* The key contributions of our work can be summarized as follows.

- We propose IST, a distributed training methodology that combines ideas from model and data parallel training by breaking the original NN into a set of disjoint subnetworks that are distributed, locally trained, and re-assembled per iteration.
- We evaluate IST on speech recognition, image classification (CIFAR10 and full ImageNet[4]), and large-scale product recommendation tasks. Using bandwidth-optimal ring all-reduce [72], IST is shown to improve time-to-convergence by as much as 10× in comparison to a state-of-the-art implementation of data parallel training and "vanilla" local SGD [47] (i.e., the only practically viable options under mandatory distribution), as well as surpass the performance of the widely-used ensemble learning method.
- We demonstrate that IST, by enabling models with larger embedding dimensions (i.e., too large for data parallel training) to be trained, is capable of solving an "extreme" product recommendation task with improved generalization.
- Finally, we theoretically show that such IST decomposition still guarantees sublinear convergence to a first-order stationary point on expectation under common assumptions.

2 TRAINING VIA INDEPENDENT SUBNETS

2.1 Methodology

Notation. Assume n sites in a distributed system. Let f_l denote that vector of activations at layer l. f_t denotes the set of activations at the final or "top" layer of the network, and f_0 denotes the feature vector that is input into the network. Assume that the number of neurons at layer l is N_l. Let $\ell(w, \cdot)$ denote the loss function of a NN with parameter w. Given samples $X := \{x_i, y_i\}_{i=1}^q$, we aim to find a $w^\star$ that minimizes the *empirical loss* over a set of labeled examples:

$$w^\star \in \arg\min_w \frac{1}{q} \sum_{i=1}^{q} \ell\left(w, \{x_i, y_i\}\right). \qquad (1)$$

Although (1) can be solved in numerous ways [26, 42, 60, 71, 80], nearly all NN training is accomplished using some variant of SGD. Here, $\eta > 0$ is the learning rate and i_t denotes a subset of training examples from X.

Constructing Subnets. IST is a distributed training regime that randomly partitions hidden neurons via uniform assignment to one of n possible compute nodes. Neurons assigned to the same compute node form a "subnet". Then, the weights of the full NN are partitioned accordingly based on the active neurons for each subnet. Hidden neurons are assigned to exactly one subnet to ensure that i) all neurons are included in training and ii) the same neuron is not simultaneously partitioned to multiple subnets.

Subnet construction is depicted in Figure 1 for a two-hidden-layer NN distributed across $n = 2$ compute nodes. Input and output layers are fully utilized at all sites. Notably, certain parameters are not partitioned to any subnet within multi-layer NNs.[5] However, in contrast to ensemble-style techniques, IST randomly samples new subnets frequently throughout the learning process, ensuring that

[2]If data is fragmented across many machines, model parallel training would struggle greatly to visit the entire dataset during training, as the input module is only stored on a single node and all data used during a particular training round must be present on this node.

[3]Each node in an n-machine cluster will receive a fraction between $\frac{1}{n^2}$ and $\frac{1}{n}$ of total model parameters under IST, while data parallel training requires *all* model weights to be communicated between machines.

[4]We underline the use of the full ImageNet dataset [14, 43] that includes 14,197,122 images, divided into 21,841 classes.

[5]Parameters are only active within a subnet if both input and output neurons associated with that parameter are sampled in the same subnet. For NNs with a single hidden layer, all parameters are included in some subnet because input and output neurons are shared across compute nodes.

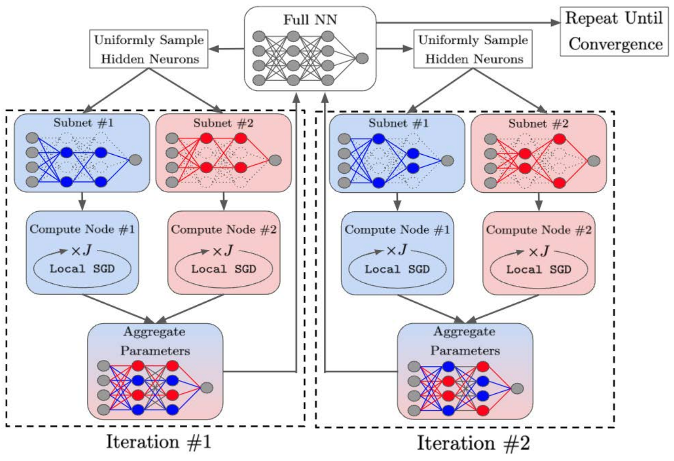

Figure 1: Schematic depiction of a two-hidden-layer NN being trained with IST across two nodes. Each layer's neurons are partitioned randomly to a single node, excluding input and output neurons (i.e., these are shared between sites). The first two iterations of IST are depicted, but the same process of sampling, training, and aggregating subnets repeats until convergence.

all parameters have a high likelihood of being trained sufficiently after several subnet groups have been sampled.

Subnet output is computed by masking (i.e., setting to zero) inactive neurons and scaling remaining activations by a factor of n^2 (i.e., to counteract neuron removal with uniform probability $\frac{1}{n}$). Such a forward pass, which is formalized in Appendix A, provides an unbiased estimate of the full NN forward pass (excluding activation functions). Furthermore, performing uniform sampling of neurons independently at each layer yields sublinear convergence to a first-order stationary point on expectation; see Appendix B. Thus, we adopt this uniform sampling policy in IST due to its unbiased nature and rigorous theoretical guarantees.

Distributing Subnets. The n subnets produced by IST are disjoint, meaning that no model parameters are simultaneously partitioned to multiple subnets. As a result, when distributed to a separate compute node, subnets i) require no cross-site communication during their forward pass, and ii) only require a fraction $\frac{1}{n^2}$ of layer parameters to be sent to each compute node. Thus, *subnets can be distributed to separate compute nodes without significant communication overhead and trained with no dependence upon other subnets—an approach that is adopted directly within IST.*

Training Subnets. For training, IST sends each of the n subnets to a separate compute node and performs J iterations of local SGD [47]. After such independent training iterations, subnet parameters are copied back into the full NN, where no collisions occur because the parameter partition is disjoint. Then, a new group of subnets is constructed through random sampling (i.e., a "re-sampling" of network parameters) and the process repeats.

Unlike ensemble methods that independently train each subnet and aggregate parameters into the full NN once at the end of training, IST re-samples subnets frequently and trains them for a shorter number of iterations between re-samplings. Such re-sampling is necessary to avoid the accumulation of random effects, as the expected input to a neuron—despite being unbiased—will shift after backpropagation. Such a shift may be inconsistent across sites, because subnets are trained on data samples from the same distribution; but, re-sampling—which is not present within ensemble methods—guards against such an occurrence.

2.2 Additional Considerations

Correcting Distributional Shift. The analysis of the unbiased subnet forward pass in Appendix A does not consider the NN's non-linear activation function.[6] Within IST, the inputs to each subnet neuron are sub-sampled and scaled by a factor of n^2 to unbias the neuron's activation, which increases the standard deviation of the input to each neuron by a factor of n. As a result, extreme input values are more likely to be observed during training (i.e., when using subnets) than during deployment. To correct this distributional mismatch, we remove the n^2 correction factor and instead compute the mean μ and standard deviation σ of the inputs to each neuron during training and transform subnet output as $x = (x - \mu)/\sigma$ before passing it through the non-linear activation function. After training is complete, we compute μ and σ for each neuron over a small subset of training data using the full network—these values can then be used when the network is deployed.

Although this approach is similar to batch normalization [36], the motivation for its use is much different. Namely, while batch normalization maintains a non-saturated range of neuron input during training to accelerate convergence and improve generalization, IST will not work in the absence of such normalization. The distributional shift encountered when deploying the network must be corrected, making this modification an essential component of IST, rather than an aid to model training and performance.

Other Architectures. IST can be extended to common network architectures (e.g., convolutional NNs) by applying IST only to fully-connected layer(s) within the network (i.e., these exist within most modern convolutional NN architectures). Here, the fully-connected layers would be decomposed as described previously, while the rest

[6] $\mathbb{E}[x] = b$ does not imply that $\mathbb{E}[f(x)] \approx f(b)$ for some random variable x when f is non-linear.

of the network is broadcast to every site during training. Such an approach has significant benefits, as fully-connected layers tend to contain a large portion of network parameters.[7] Thus, improving the efficiency of distributed training over fully-connected layers benefits the distributed training process for the entire network.

2.3 Analysis

IST reduces communication overhead in comparison to data parallel training approaches, which broadcast all parameters across sites during each round of training. Measuring the inflow to each site, the total network traffic of data parallel training per gradient step is (in floating point numbers transferred):

$$\sum_{i=1}^{t} nN_{i-1}N_i.$$

In contrast, in IST, each site receives current parameters every J gradient steps (i.e., assuming J iterations of local SGD are performed between re-sampling rounds). Furthermore, subsampling the NN into multiple subnets further reduces the communication cost of IST because input/ouput layers are partitioned (not broadcast) across nodes and each node receives only a $\frac{1}{n}$ ratio of other network parameters. The total network traffic of IST per gradient step is:

$$\frac{N_0N_1 + N_{t-1}N_t}{J} + \sum_{i=1}^{l} \frac{N_{i-1}N_i}{n \times J}.$$

Similarly, IST reduces computational resource utilization in comparison to data parallel. Given the FLOPs required by matrix multiplications during forward/backward steps, in "classical" data parallel training, the number of FLOPS required per gradient step is:

$$4\sum_{i=1}^{l} BN_{i-1}N_i.$$

In contrast, the number of FLOPS gradient step within IST is:

$$4BN_0N_1 + 4BN_{t-1}N_t + 4B\sum_{i=1}^{l} \frac{N_{i-1}N_i}{n}.$$

Note that this computational reduction indicates that training models with IST reduces memory requirements, which enables the training of larger models as shown in Sec. 3.

Convergence Guarantees. We show that the IST decomposition guarantees convergence to a first-order stationary point on expectation in the distributed setting. Namely, under common assumptions of smoothness, Lipschitz continuity of the objective, and stochastic error boundedness, IST converges sublinearly to a bounded error region around a stationary point; see Appendix B.

3 EMPIRICAL EVALUATION

In this section, we design a set of experiments that showcase the potential benefits of the IST approach under cases of mandatory distribution with limited hardware capabilities. In these cases, we assume slow networks connections with CPUs or GPUs (possibly with limited memory) available on each node. We consider a wide variety of learning tasks and network architectures (i.e., both fully-connected NNs and more complex NNs that contain fully-connected layers).

3.1 Setup and Details

As previously mentioned, model parallel training is not appropriate for mandatory distribution, as all needed data must be stored on the node that houses the network's input module. Furthermore, due to the assumption of limited memory on each device, our experiments typically consider shallow, fully-connected NNs with wide hidden layers.[8] Popular model parallel training packages (e.g., Gpipe [34]) struggle to perform well on wide models with few layers, providing further evidence that model parallel training is not the proper training approach when distribution is mandatory. As such, we adopt local SGD [47], data parallel training, and ensemble learning as our major experimental baselines.

Experimental Settings. We consider the following scenarios:

- *Google Speech Commands* [69]: We learn a 2-layer network of 4096 neurons and a 3-layer network of 8192 neurons to recognize 35 labeled keywords from audio waveforms (in contrast to the 12 keywords in prior reports [69]). We represent each waveform as a 4096-dimensional feature vector [67].
- *Image classification on CIFAR10 and full ImageNet* [33, 63]: We train the Resnet18 model over CIFAR10, and the VGG12 model over full ImageNet (see Section 2.2 for a discussion of IST and non-fully connected architectures). **Note that we include the complete ImageNet dataset with all** $21,841$ **categories and report the top-10 accuracy** [24, 43, 58].
- *Amazon-670k* [5]: We train a 2-layer, fully-connected neural network, which accepts a $135,909$-dimensional input feature, and generates a prediction over $670,091$ output labels.

We train Google speech and Resnet18 on CIFAR10 on three AWS CPU clusters, with 2, 4, and 8 CPU instances (`m5.2xlarge`). We train the VGG model on full ImageNet and Amazon-670k extreme classification network on three AWS GPU clusters, with 2, 4, and 8 GPU machines (`p3.2xlarge`). Our choice of AWS was deliberate, as it is a common platform for distributed training and presents the common challenge faced by many consumers—distributed ML without a super-fast interconnect.

Distributed Implementation Notes. We implement a distributed parameter server for IST in PyTorch. We compare IST to the PyTorch implementation of data parallel learning. We also adapt the PyTorch data parallel learning to realize local SGD [47] and ensemble learning, where learning occurs locally for a number of iterations before synchronizing. For ensemble learning, this synchronization only occurs once at the end of training.

For the CPU experiments, we use PyTorch's `gloo` backend. For the GPU experiments, data parallel learning and local SGD use PyTorch's `nccl` backend, which leverages the most advanced Nvidia collective communication library (the set of high-performance multi-GPU and multi-node collective communication primitives optimized for NVIDIA GPUs). `Nccl` implements ring-based all-reduce

[7]Consider the full ImageNet dataset [21, 44] for a deep model like ResNet50: the convolutional layers have 17,614,016 parameters (67.2MB, 28.2%), whereas the fully-connected layer has 44,730,368 parameters (170.6MB, 71.8%) that utilized in IST.

[8]The memory usage of IST scales linearly with increasing network depth, but small portions of each hidden layer can be partitioned to each subnet in order to limit increased memory usage due to larger hidden layers.

Table 1: The time (in seconds) to reach various levels of accuracy.

Google Speech 2 Layer									
	Data Parallel			Local SGD			IST		
Accuracy	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node
0.63	118	269	450	68	130	235	35	28	**24**
0.75	759	1708	2417	444	742	1110	231	**167**	192

Google Speech 3 Layer									
	Data Parallel			Local SGD			IST		
Accuracy	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node
0.63	376	1228	1922	182	586	1115	**76**	141	300
0.75	4534	9340	14886	2032	4107	6539	812	**664**	1161

CIFAR10 Resnet18									
	Data Parallel			Local SGD			IST		
Accuracy	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node
0.85	21775	13689	6890	18769	12744	7020	15093	7852	**5241**
0.90	54002	38430	17853	36891	22198	**12157**	33345	16798	13425

Full ImageNet VGG12									
	Data Parallel			Local SGD			IST		
Accuracy	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node	2 Node	4 Node	8 Node
0.20	108040	278542	504805	6900	14698	30441	**3629**	4379	5954
0.26	225911	393279	637188	15053	22055	39439	**6189**	7711	10622

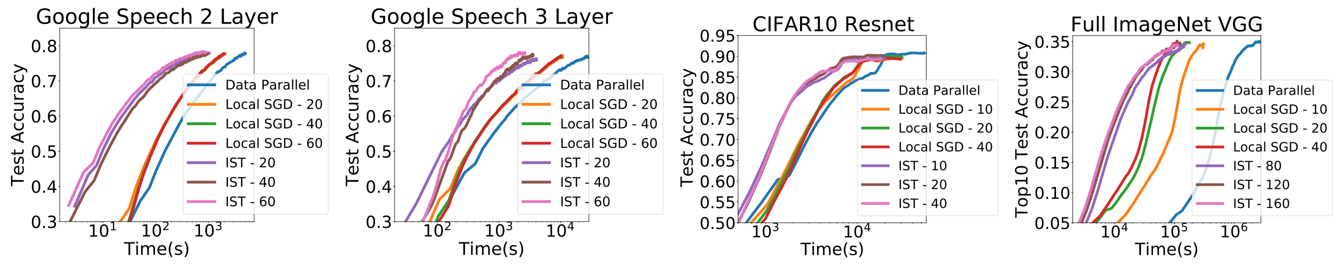

Figure 2: Test accuracy versus time. 2-/3- layer Google speech models are trained using an 8-CPU cluster; Resnet18 on CIFAR10 is trained using 4-CPU cluster; VGG12 on full ImageNet is trained using a 8-GPU cluster. The number after local SGD or IST legend represents the local update iterations.

[72], which is used in well-known distributed learning systems such as Horovod [62].

IST cannot use the `nccl` backend because it does not support the `scatter` operator required to implement IST. As a result, IST must use the `gloo` backend (meant for CPU-based learning), which is a serious handicap but does not reflect any intrinsic flaw of the method—high-performance GPU libraries simply lack support for required operations.

3.2 Results and Analysis

Convergence speed. While IST can process data quickly (i.e., due to previously-described improvements in communication efficiency), there are questions regarding its statistical efficiency and generalization performance in comparison to baseline methods. Figure 2 plots the hold-out test accuracy for selected benchmarks as a function of time, while Table 1 shows the training time required for IST and relevant baselines to reach specified levels of hold-out test accuracy.

Our results generally indicate that IST achieves high accuracy on the test set much faster than other frameworks. For example, in reaching an accuracy of 77% with a 2-layer, fully-connected network, IST exhibits a 4.2× speedup compared to local SGD and a 10.6× speedup compared to data parallel. Similarly, IST exhibits a 6.1× speedup compared to local SGD and a 16.6× speedup compared to data parallel in reaching the same accuracy with a 3-layer model. Note that the above improvements were observed even though IST was handicapped by its use of the `gloo` backend for its GPU implementation.

Table 2: Final accuracy on each benchmark.

	Data Parallel	Local SGD	IST
Speech 2 layer	0.7938	0.7998	**0.8153**
Speech 3 layer	0.7950	0.7992	**0.8327**
CIFAR10	**0.9128**	0.9087	0.9102
Full Imagenet	0.3688	0.3685	**0.3802**

Table 3: Test accuracy on Google Speech Commands.

Compute Nodes	2 Layer		3 Layer	
	IST	Ensemble	IST	Ensemble
2 Node	0.82	0.82	0.84	0.74
4 Node	0.79	0.80	0.82	0.71
8 Node	0.76	0.77	0.78	0.70

Because CPUs were used for training on CIFAR10, the network was less of a bottleneck and all methods were able to scale, thus slightly negating the advantages of IST. Despite reaching 90% accuracy slower on an 8-CPU cluster, however, IST was still the fastest to reach 85% accuracy. Furthermore, for the full ImageNet data set, the communication bottleneck using AWS is so severe that the smaller clusters were always faster. At each cluster size, IST was still the fastest option.

Trained model accuracy. In Table 2 we give the final accuracy of each method, trained on a 2-node cluster in various experimental settings. As can be seen, despite partitioning the full networks into several independently-trained subnets, IST achieves better final accuracy in comparison to data parallel and local SGD training on all datasets except for CIFAR10. On the CIFAR10 dataset, IST achieves test accuracy 0.26% lower than data parallel training, but outpeforms local SGD. Furthermore, IST continues to achieve high final accuracy as the number of compute nodes is increased, as shown in Table 3. Thus, IST achieves highly-comparable or improved final accuracy in comparison to local SGD, data parallel, and ensemble-based training in all settings, *revealing that the partitioning strategy of IST does not deteriorate the network's ability to match or exceed the accuracy achieved by baseline methodologies.*

As previously mentioned, IST also enables models to be trained that are too large to be handled by a single device. In cases of mandatory distribution, such a property is useful for training sufficiently large models despite limited memory on individual compute nodes. To demonstrate the utility of this property of IST, we study the relationship between embedding dimension and test accuracy for fully-connected models trained on the Amazon-670K recommendation task in an 8-GPU cluster. As shown in Table **??**, IST is able to scale to larger model sizes without exceeding the memory capacity of individual nodes. Such scalability enables a $>$ 15% test accuracy improvement in comparison to data parallel training, thus demonstrating that IST allows models with sufficient capacity to be trained despite the restricted memory of each device.

Comparison to Ensemble Learning. IST intermittently aggregates subnet parameters and re-samples a new group of subnets for independent training. Although ensemble learning trivially improves communication efficiency and wall-clock training time (i.e., due to utilizing fewer synchronizations), re-sampling is necessary for achieving high network performance. To show this, we perform tests with ensemble learning—i.e., training a group of subnet-sized models independently and aggregating their parameters once at the end of training—and IST on the Google Speech Commands dataset; see Table 3.

Ensemble learning and IST perform similarly for two-layer networks 2.2. Such comparable performance is expected because *i*) two-layer networks are separable [13] (i.e., each hidden neuron contributes independently to network output without inter-neuron interaction) and *ii*) all parameters within the one-hidden-layer NN are partitioned to some subnet. In such a simplified case, ensemble learning is able to performing well by independently learning meaningful neuron representations that can be aggregated into the global network.

For deeper networks, no re-sampling during training leads numerous network parameters to be excluded from the learning process and allows random effects to accumulate throughout training, thus drastically deteriorating ensemble learning performance. As such, IST significantly outperforms ensemble learning with three-layer networks (e.g., 10% absolute improvement with $n = 2$ compute nodes), revealing that IST has a significant performance advantage relative to ensemble learning for complex, multi-layer network architectures. Thus, although ensemble learning is faster to complete a fixed number of training epochs, it cannot yield comparable performance to networks trained with IST.

Discussion. There are a few takeaways from the experimental results. First, as expected, IST is able to process far more data in a short amount of time than the other distributed training algorithm. Interestingly, we find that the IST speedups in CPU clusters are more significant than that in GPU clusters. There are two reasons for this. First, for GPU clusters, IST suffers from its use of PyTorch's `gloo` backend, compared to the `all-reduce` operator provided by `nccl`. Second, since the GPU provides a very high level of computation, there is less benefit to be realized from the reduction in FLOPS per gradient step using IST (as the GPU does not appear to be compute bound).

It is interesting that *some of the frameworks actually do worse with additional machines, especially with a fast GPU.* This illustrates a significant problem with distributed learning. Unless a super-fast interconnect is used (and such interconnects are not available from typical cloud providers), it can actually be *detrimental* to add additional machines, as the added cost of transferring data can actually result in *slower* running times. We see this clearly in Table 1, where the state-of-the-art PyTorch data parallel implementation (and the local SGD variant) does significantly *worse* with more machines. IST shows the best potential to utilize additional machines without actually becoming much slower or slower to reach high accuracy.

Finally, we note that various compression schemes can be used to increase the bandwidth of the interconnect (e.g., gradient sparsification [2], quantization [3], sketching [37], and low-rank compression [68]). However, these methods could be used with *any* framework—including IST. We conjecture that while compression may allow effective scaling to larger clusters, it would not affect the efficacy of IST.

Table 4: Precision @1, @3, @5 on Amazon 670k.

	Data Parallel			IST		
Dim.	@1	@3	@5	@1	@3	@5
512	0.386	0.345	0.316	0.396	0.360	0.331
1024		Fail to handle		0.409	0.369	0.339
1536		Fail to handle		0.432	0.391	0.361
2048		Fail to handle		0.437	0.394	0.364
2560		Fail to handle		**0.438**	**0.394**	**0.366**

4 RELATED WORK

Data parallelism often suffers from the high bandwidth costs to communicate gradient updates between workers. Quantized SGD [3, 16, 22, 30, 35, 61, 70] and sparsified SGD [2] both address this. Quantized SGD uses lossy compression to quantize the gradients. Sparsified SGD reduces the exchange overhead by transmitting the gradients with maximal magnitude. Such methods are orthogonal to IST, and could be used in combination with it.

Recently, there has been a series of papers on using parallelism to *"Solve the* YY *learning problem in* XX *minutes"*, for ever-decreasing values of XX [14, 29, 64, 73, 76–78]. Often these methods employ large batches. It is generally accepted—though still debated [23]—that large batch training converges to "sharp minima", hurting generalization [20, 39, 75]. Further, achieving such results seems to require teams of PhDs utilizing special-purpose hardware: there is no approach that generalizes well without extensive trial-and-error.

Distributed local SGD [49, 81, 82, 86] updates the parameters, through averaging, only after several local steps are performed per compute node. This reduces synchronization and thus allows for higher hardware efficiency [82]. IST uses a similar approach but makes the local SGD and each synchronization round less expensive. Recent approaches [47] propose less frequent synchronization towards the end of the training, but they cannot avoid it at the beginning.

Finally, asynchrony avoids SGD synchronization cost [19, 51, 57, 83]. It has been used in distributed-memory systems, such as DistBelief [19] and the Project Adam [42]. While such systems, asymptotically, show nice convergence rate guarantees, there seems to be growing agreement that *unconstrained* asynchrony does not always work well [10], and it seems to be losing favor in practice.

Overall, the goal of such distributed training methods is to lower the wall-clock time-to-convergence with the addition of extra hardware. As such, empirical analysis of these methods is often conducted using state-of-the-art computing hardware with high-bandwidth interconnects. Even with access to such an ideal environment, however, data parallel approaches struggle to scale. In particular, per-node compute requirements are reduced while synchronization costs remain constant or increase, leading to cases where the addition of more nodes makes training *slower* as communication costs begin to dominate the training procedure. This issue could theoretically be mitigated with the use of larger batches, but such an approach often degrades statistical efficiency and leads to poor generalization [14, 28, 29, 48, 64, 73, 76–78].

5 CONCLUSION

In this work, we propose *independent subnet training* for distributed training of neural networks. By stochastically partitioning the model into non-overlapping subnets, IST reduces the communication overhead for model synchronization, and the computation workload of forward-backward propagation for a thinner model on each worker. This results in two advances: *i*) IST significantly accelerates the training process comparing with standard data parallel approaches for distributed learning, and *ii*) IST scales to large models that cannot be learned using standard data parallel approaches.

A IST IS AN UNBIASED ESTIMATOR

We formalize subnet construction in IST with a set of neuron membership indicators $m_{l,i}^{(s)} \in \{0,1\}$ at each layer l where s ranges over the n compute nodes and i ranges over all the neurons in layer l. $m^{(s)}$ contains a binary mask for subnet s across all layers and neurons. For each entry $m_{l,i}^{(s)}$, a value of $\{0,1\}$ is assigned with marginal probability $\mathbb{P}[m_{l,i}^{(s)}=1]=\frac{1}{n}$ to exactly one of the n subnets, implying that $\sum_s m_{l,i}^{(s)}=1$ (i.e., neurons are partitioned to exactly one subnet) and $\mathbb{E}[m_{l,i'}^{(s)} m_{l-1,i}^{(s)}]=\frac{1}{n^2}$ (i.e., sampling is independent at each layer).

Using these constructions, we can define the forward pass of subnet s at layer l as

$$\hat{f}_l^{(s)} = n^2\left(m_l^{(s)} \odot \left(W_l\left(m_{l-1}^{(s)} \odot \bar{f}_{l-1}\right)\right)\right) \tag{2}$$

where $\odot$ denotes the Hadamard product, W_l is the weight matrix between layers $l-1$ and l, and $\bar{f}_l^{(s)}=\mathcal{S}(\hat{f}_l^{(s)})$ (i.e., $\hat{\cdot}$ and $\bar{\cdot}$ denote representations before and after the non-linear activation function $\mathcal{S}$). To gather the activations produced by each subnet into a single vector, we sum over subnet activations as $\hat{f}_l=\sum_s \hat{f}_l^{(s)}$. The Hadamard products in (2) mask out neuron activations—both in $\hat{f}_l^{(s)}$ and $\bar{f}_{l-1}$—that are not relevant to subnet s.[9]

Interestingly, if $\bar{f}_{l-1}$ is an unbiased estimator of the full network output $f_{l-1}^{\star}$, then $\hat{f}_l$ is an unbiased estimator of $W_l f_{l-1}^{\star}$. To show this, we consider the jth entry of

$$n^2\sum_s m_l^{(s)} \odot \left(W_l\left(m_{l-1}^{(s)} \odot \bar{f}_{l-1}\right)\right),$$

for which the expectation can be written as

$$\begin{aligned}\mathbb{E}\left[n^2\sum_s\sum_i\sum_{i'} W_{l,j,i} m_{l,i'}^{(s)} m_{l-1,i}^{(s)} \bar{f}_{l-1,i}\right] &= n^2\sum_s\sum_i \frac{1}{n^2} W_{l,j,i}\mathbb{E}\left[\bar{f}_{l-1,i}\right]\\ &= \sum_i W_{l,j,i}^{l} f_{l-1,i}^{\star}\end{aligned}$$

which is precisely the jth entry in $W_l f_{l-1}^{\star}$.

B CONVERGENCE GUARANTEES FOR IST

We will discuss the convergence behavior of IST in this section. Given space constraints, we do not give a proof of the central Theorem and corollary; those are left to the full version of the paper [79].

Consider minimizing $\ell(w)=\frac{1}{n}\sum_{i=1}^{n}\ell_i(w)$ as in Equation 1. Our analysis adopts six assumptions, labeled Assumption 1 through Assumption 6.

[9]In practice, such masking is not actually performed. Rather, we partition the weight matrix such that inactive neurons are never computed.

Assumption 1. *(L_i-smoothness) Given component ℓ_i of ℓ function, there exists constant $L_i > 0$ such that for every $w_1, w_2 \in \mathbb{R}^p$ we have that:*

$$\|\nabla \ell_i(w_1) - \nabla \ell_i(w_2)\|_2 \leq L_i \cdot \|w_1 - w_2\|_2$$

or, equivalently,

$$\ell_i(w_2) \leq \ell_i(w_1) + \langle \nabla \ell_i(w_1), w_2 - w_1 \rangle + \tfrac{L_i}{2}\|w_1 - w_2\|_2^2.$$

Further, define $L_{\max} := \max_i L_i$.

Assumption 2. *(Q-Lipschitz continuity) Given ℓ function, there exists constant $Q > 0$ such that for every $w_1, w_2 \in \mathbb{R}^p$ we have that:*

$$|\ell(w_1) - \ell(w_2)| \leq Q \cdot \|w_1 - w_2\|_2$$

or, equivalently, $\|\nabla \ell(w)\|_2 \leq Q, \quad \forall w \in \mathbb{R}^p$.

Assumption 3. *(Error Bound) Let $w^\star$ denote the global optimum of ℓ. Then, under the Error Bound assumption, there exists constant $\mu > 0$ such that for every $w \in \mathbb{R}^p$ we have that:*

$$\|\nabla \ell(w)\|_2 \geq \mu \|w^\star - w\|_2$$

Per [38], Error Bound ≡ Polyak-Łojasiewicz inequality.

Assumption 4. *(Stochastic gradient variance) such that*

$$\mathbb{E}_{i_t}\left[\|\nabla \ell_{i_t}(w)\|_2^2\right] \leq M + M_f \|\nabla \ell(w)\|_2^2,$$

where ℓ_{i_t} is a randomly selected component from the sum $\frac{1}{n}\sum_{i=1}^n \ell_i(w)$.

Note that we make the distinction between the general indexing term i and the randomly selected index per SGD round, i_t. We follow the problem formulation in [40] on compressed iterates, where IST performs the following motions:

- Given model w_t at iteration t, we generate a mask $\mathcal{M} : \mathbb{R}^p \to \mathbb{R}^p$ such that:

$$(\mathcal{M}(w_t))_i = \begin{cases} \frac{w_{t,i}}{\xi}, & \text{with probability } \xi, \\ 0, & \text{with probability } 1 - \xi. \end{cases}$$

  Input and output neurons are always selected in this mask.
- Given mask $\mathcal{M}(\cdot)$, we generate the subnetwork as in:

$$z_t \equiv \mathcal{M}(w_t) \in \mathbb{R}^p,$$

  where z_t—a *compressed* version of w_t—has zeros at positions for deactivated subnetwork weights at iteration t and non-zeros for active weights.
- We perform gradient descent on the compressed z_t:

$$w_{t+1} = z_t - \eta \nabla \ell_{i_t}(z_t),$$

  with η the learning rate and i_t randomly selected from $[n]$.

This setting resembles *gradient descent with compressed iterates* (GDCI) [40], but our analysis considers a different function class. Our final two assumptions are on $\mathcal{M}(\cdot)$ with respect to the gradient of ℓ.

Assumption 5. *(Additive gradient error assumption with bounded energy) Let w_t be the current model and let $z_t = \mathcal{M}(w_t)$ be the compressed model. Consider the stochastic gradient term $\nabla \ell_{i_t}(z_t)$; we assume that, on expectation, the following holds:*

$$\mathbb{E}_{\mathcal{M}, i_t}\left[\nabla \ell_{i_t}(z_t) \mid w_t\right] = \nabla \ell(w_t) + \varepsilon_t,$$

for $\varepsilon_t \in \mathbb{R}^p$ such that $\|\varepsilon_t\|_2 \leq B$ for $B > 0$.

Assumption 6. *(Norm condition) $\exists\, \theta \in [0, 1)$ such that:*

$$\left\|\mathbb{E}_{\mathcal{M}, i_t}\left[\nabla \ell_{i_t}(z_t) \mid w_t\right] - \nabla \ell(w_t)\right\|_2 = \|\varepsilon_t\|_2 \leq \theta \|\nabla \ell(w_t)\|_2,$$

where w_t and z_t are current and compressed models, respectively.

The above assumption is commonly used in derivative free optimization [4, 7]. We are now able to derive the following theorem and corollary, which imply the convergence of IST.

Theorem 1. *Let $\ell(w) := \frac{1}{n}\sum_{i=1}^n \ell_i(w)$ have L_i-smooth components ℓ_i for $L_{\max} := \max_i L_i$, and consider the following recursion:*

$$w_{t+1} = z_t - \eta \nabla \ell_{i_t}(z_t), \quad \text{where } z_t = \mathcal{M}(w_t).$$

Suppose $\mathcal{M}(w_t)$ and ℓ satisfy Assumption 5. After T iterations for step size $\eta = \frac{1}{2L_{\max}}$, we obtain:

$$\min_{t \in \{0,\dots,T\}} \mathbb{E}_{\mathcal{M}, i_t}\left[\|\nabla \ell(w_t)\|_2^2\right]$$
$$\leq \frac{\ell(x_0) - \ell(w^\star)}{\alpha(T+1)} + \frac{1}{\alpha} \cdot \left(\frac{BQ}{2L_{\max}} + \frac{5L_{\max}\omega}{2} \cdot \|w^\star\|_2^2 + \frac{M}{4L_{\max}}\right)$$

where the expectation is over the random selection on the compression operator $\mathcal{M}(\cdot)$ and the stochastic selection i_t, $\alpha = \frac{1}{2L_{\max}}\left(1 - \frac{M_f}{2}\right) - \frac{5\omega L_{\max}}{2\mu^2}$, and $\omega = \frac{1-\xi}{\xi} < \frac{\mu^2}{10L_{\max}^2}$.

If we exchange the bounded assumption $\|\varepsilon_t\|_2 \leq B$ and Q-Lipschitzness, with the norm condition in Assumption 6, we obtain the following corollary.

Corollary 1. *Let ℓ be L-smooth, and consider the recursion over compressed iterates:*

$$w_{t+1} = z_t - \eta \nabla \ell_{i_t}(z_t), \quad \text{where } z_t = \mathcal{M}(w_t).$$

We further assume that the operator mask, along with ℓ, satisfy the norm condition Assumption 6 with parameter $\theta \in [0, 1)$. Then, after running the above recursion for T iterations for step size $\eta = \frac{1}{2L_{\max}}$, we obtain:

$$\min_{t \in \{0,\dots,T\}} \mathbb{E}_{\mathcal{M}, i_t}\left[\|\nabla \ell(w_t)\|_2^2\right]$$
$$\leq \frac{\ell(w_0) - \ell(w^\star)}{\alpha(T+1)} + \frac{1}{\alpha} \cdot \left(\frac{5L_{\max}\omega}{2} \cdot \|w^\star\|_2^2 + \frac{M}{4L_{\max}}\right)$$

where the expectation is over the random selection on the compression operator $\mathcal{M}(\cdot)$ and α and ω are expressed as:

$$\alpha = \frac{1}{2L_{\max}}\left(\frac{1}{2} - \theta - \frac{M_f}{2}\right) - \frac{5L_{\max}}{2} \cdot \frac{\omega}{\mu^2}$$
$$\omega = \frac{1-\xi}{\xi} < \frac{\mu^2}{5L_{\max}^2\left(\frac{1}{2} - \theta - \frac{M_f}{2}\right)}$$

ACKNOWLEDGMENTS

This work is supported by NSF FET:Small no. 1907936, NSF MLWiNS CNS no. 2003137 (in collaboration with Intel), NSF CMMI no. 2037545, NSF CAREER award no. 2145629, and Rice InterDisciplinary Excellence Award (IDEA).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021* (2017).

[3] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*. 1709–1720.

[4] A. Berahas, L. Cao, K. Choromanski, and K. Scheinberg. 2019. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *arXiv preprint arXiv:1905.01332* (2019).

[5] Kush Bhatia, Kunal Dahiya, Himanshu Jain, Yashoteja Prabhu, and Manik Varma. [n.d.]. *The Extreme Classification Repository: Multi-label Datasets and Code*. http://manikvarma.org/downloads/XC/XMLRepository.html.

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

[7] R. Carter. 1991. On the global convergence of trust region algorithms using inexact gradient information. *SIAM J. Numer. Anal.* 28, 1 (1991), 251–265.

[8] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. 125–136.

[9] Zheng Chai, Hannan Fayyaz, Zeshan Fayyaz, Ali Anwar, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig, and Yue Cheng. 2019. Towards taming the resource and data heterogeneity in federated learning. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*. 19–21.

[10] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).

[11] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635* (2019).

[12] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System.. In *OSDI*, Vol. 14. 571–582.

[13] Lenaic Chizat and Francis Bach. 2018. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in neural information processing systems* 31 (2018).

[14] Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. 2017. Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291* (2017).

[15] E Cordis. 2019. Machine learning ledger orchestration for drug discovery.

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.

[17] Pierre Courtiol, Charles Maussion, Matahi Moarii, Elodie Pronier, Samuel Pilcer, Meriem Sefta, Pierre Manceron, Sylvain Toldo, Mikhail Zaslavskiy, Nolwenn Le Stang, et al. 2019. Deep learning-based classification of mesothelioma improves prediction of patient outcome. *Nature medicine* 25, 10 (2019), 1519–1525.

[18] Walter de Brouwer. 2019. The federated future is ready for shipping.

[19] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[20] Aaron Defazio and Léon Bottou. 2018. On the Ineffectiveness of Variance Reduced Optimization for Deep Learning. *arXiv preprint arXiv:1812.04529* (2018).

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[22] Tim Dettmers. 2015. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561* (2015).

[23] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. 2017. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1019–1028.

[24] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[25] Petros Drineas, Ravi Kannan, and Michael W Mahoney. 2006. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM J. Comput.* 36, 1 (2006), 132–157.

[26] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[27] Philipp Farber and Krste Asanovic. 1997. Parallel neural network training on multi-spert. In *Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on*. IEEE, 659–666.

[28] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. 2018. On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent. *arXiv preprint arXiv:1811.12941* (2018).

[29] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[30] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[31] Stefan Hadjis, Ce Zhang, Ioannis Mitliagkas, Dan Iter, and Christopher Ré. 2016. Omnivore: An optimizer for multi-device deep learning on CPUs and GPUs. *arXiv preprint arXiv:1606.04487* (2016).

[32] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

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

[34] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. arXiv:1811.06965 [cs.CV]

[35] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *Journal of Machine Learning Research* 18, 187 (2017), 1–30.

[36] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 448–456.

[37] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Ion Stoica, Raman Arora, et al. 2019. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*. 13144–13154.

[38] H. Karimi, J. Nutini, and M. Schmidt. 2016. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 795–811.

[39] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[40] A. Khaled and P. Richtárik. 2019. Gradient descent with compressed iterates. *arXiv preprint arXiv:1909.04716* (2019).

[41] Latif U Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. 2021. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Communications Surveys & Tutorials* (2021).

[42] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[43] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. 2020. Big transfer (bit): General visual representation learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*. Springer, 491–507.

[44] A. Krizhevsky, I. Sutskever, and G. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[45] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. 2019. Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 6341–6345.

[46] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*, Vol. 14. 583–598.

[47] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. 2018. Don't use large mini-batches, use local SGD. *arXiv preprint arXiv:1808.07217* (2018).

[48] Linjian Ma, Gabe Montague, Jiayu Ye, Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. 2019. Inefficiency of K-FAC for Large Batch Size Training. *arXiv preprint arXiv:1903.06237* (2019).

[49] Ryan Mcdonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S Mann. 2009. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*. 1231–1239.

[50] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. 2012. Performance evaluation

of Amazon EC2 for NASA HPC applications. In *Proceedings of the 3rd workshop on Scientific Cloud Computing*. 41–50.

[51] Thomas Paine, Hailin Jin, Jianchao Yang, Zhe Lin, and Thomas Huang. 2013. GPU asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186* (2013).

[52] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[53] Sundar Pichai. 2019. Privacy should not be a luxury good. *The New York Times* (2019).

[54] Rajat Raina, Anand Madhavan, and Andrew Y Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 873–880.

[55] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329* (2019).

[56] Alexander Ratner, Dan Alistarh, Gustavo Alonso, Peter Bailis, Sarah Bird, Nicholas Carlini, Bryan Catanzaro, Eric Chung, Bill Dally, Jeff Dean, et al. 2019. SysML: The New Frontier of Machine Learning Systems. *arXiv preprint arXiv:1904.03257* (2019).

[57] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[58] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. 2021. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972* (2021).

[59] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. 2020. The future of digital health with federated learning. *NPJ digital medicine* 3, 1 (2020), 1–7.

[60] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[61] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[62] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[63] K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[64] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. 2017. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489* (2017).

[65] Chetan L Srinidhi, Ozan Ciga, and Anne L Martel. 2020. Deep neural network models for computational histopathology: A survey. *Medical Image Analysis* (2020), 101813.

[66] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[67] Stanley Smith Stevens, John Volkmann, and Edwin B Newman. 1937. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America* 8, 3 (1937), 185–190.

[68] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. 2019. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems*. 14236–14245.

[69] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[70] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*. 1509–1519.

[71] Stephen Wright and Jorge Nocedal. 1999. Numerical optimization. *Springer Science* 35, 67-68 (1999), 7.

[72] Alfred Xu. 2018. NCCL BASED MULTI-GPU TRAINING. http://on-demand.gputechconf.com/gtc-cn/2018/pdf/CH8209.pdf. (2018). Accessed: 2020-02-06.

[73] Omry Yadan, Keith Adams, Yaniv Taigman, and Marc'Aurelio Ranzato. 2013. Multi-GPU training of convnets. *arXiv preprint arXiv:1312.5853* (2013).

[74] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).

[75] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. 2018. Hessian-based analysis of large batch training and robustness to adversaries. In *Advances in Neural Information Processing Systems*. 4949–4959.

[76] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Scaling SGD batch size to 32K for ImageNet training. *arXiv preprint arXiv:1708.03888* (2017).

[77] Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large-Batch Training for LSTM and Beyond. *arXiv preprint arXiv:1901.08256* (2019).

[78] Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. 2019. Reducing BERT Pre-Training Time from 3 Days to 76 Minutes. *arXiv preprint arXiv:1904.00962* (2019).

[79] Binhang Yuan, Anastasios Kyrillidis, and Christopher M Jermaine. 2019. Distributed learning of deep neural networks using independent subnet training. *arXiv preprint arXiv:1910.02120* (2019).

[80] Matthew D Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[81] Ce Zhang and Christopher Ré. 2014. Dimmwitted: A study of main-memory statistical analytics. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1283–1294.

[82] Jian Zhang, Christopher De Sa, Ioannis Mitliagkas, and Christopher Ré. 2016. Parallel SGD: When does averaging help? *arXiv preprint arXiv:1606.07365* (2016).

[83] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu. 2013. Asynchronous stochastic gradient descent for DNN training. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 6660–6663.

[84] Xiru Zhang, Michael Mckenna, Jill P Mesirov, and David L Waltz. 1990. An efficient implementation of the back-propagation algorithm on the connection machine CM-2. In *Advances in neural information processing systems*. 801–809.

[85] Wan Zhu, Longxiang Xie, Jianye Han, and Xiangqian Guo. 2020. The application of deep learning in cancer prognosis prediction. *Cancers* 12, 3 (2020), 603.

[86] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. 2010. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*. 2595–2603.