Realizing Petabyte Scale Acoustic Modeling

Sree Hari Krishnan Parthasarathi, Nitin Sivakrishnan, Pranav Ladkat, Nikko Strom

***Abstract*—Large scale machine learning (ML) systems such as the Alexa automatic speech recognition (ASR) system continue to improve with increasing amounts of manually transcribed training data. Instead of scaling manual transcription to impractical levels, we utilize semi-supervised learning (SSL) to learn acoustic models (AM) from the vast firehose of untranscribed audio data. Learning an AM from 1 Million hours of audio presents unique ML and system design challenges. We present the design and evaluation of a highly scalable and resource efficient SSL system for AM. Employing the student/teacher learning paradigm, we focus on the student learning subsystem: a scalable and robust data pipeline that generates features and targets from raw audio, and an efficient model pipeline, including the distributed trainer, that builds a student model. Our evaluations show that, even without extensive hyper-parameter tuning, we obtain relative accuracy improvements in the 10 to 20% range, with higher gains in noisier conditions. The end-to-end processing time of this SSL system was 12 days, and several components in this system can trivially scale linearly with more compute resources.**

***Index Terms*—Speech recognition, acoustic models, large scale semi-supervised learning, machine learning.**

I. Introduction

MODERN machine learning (ML) relies on large-scale data; for hard problems more data consistently lead to better models. In the field of automatic speech recognition, there is a well known maxim: *there is no data like more data* [1]. This has naturally led to the use of ever increasing amounts of speech data. From tens of hours of speech (TIDIGITS [2], TIMIT [3], WSJ [4]), to hundreds (Switchboard [5]), and thousands (Fisher corpus [6]). Recently, training data sizes on the order of ten thousand hours of speech are not unusual ([7], [8]), and while building an AM from a hundred thousand hours has still been uncommon, [9] showed that increasing from several thousand hours to a hundred thousand hours of lightly supervised speech data can improve speech recognition accuracy significantly.

This increase in the amount of data in ASR and other ML fields requires ever more efficient data and ML processing pipelines. A rich infrastructure has emerged – commercially and in Open Source communities – to serve both hardware and software requirements of large-scale ML. There has been extensive developments in powerful ML toolkits (Spark MLLib, mlpack, Scikit-Learn), ML cloud services (AzureML, Amazon ML, Google Cloud Machine Learning), distributed storage (S3, HDFS [10], GoogleFileSystem), frameworks for distributed compute (Hadoop [10], Spark [11]) and Deep Learning (PyTorch [12], MxNet [13], TensorFlow [14]). These tools make it easy to train ML models. However, for the largest, most data intensive ML systems it is critical to use generic tools efficiently, and sometimes develop custom solutions, to avoid memory, bandwidth or processing bottlenecks. In this paper we present our end-to-end ML system for building an acoustic model based on 1 Million hours of speech in 12 days. This is one order of magnitude more speech data than has been reported in any previously published work in ASR [9].

All authors are employed by Amazon.com, Inc., {sparta|nitins|ladkat|nikko}@amazon.com

While many of the techniques and tradeoffs are applicable to other fields, we demonstrate the large-scale challenge here by tackling acoustic model (AM) training for ASR. The AM in the ASR system takes a sequence of acoustic feature vectors as input and produces a sequence of phonetic probability densities as output [15]. At each time step, it produces a posterior probability for each phonetic class. Because of the large variability in speaker characteristics, dialects, accents, and acoustic environments, producing a highly accurate AM requires large amounts of speech data, and the model is typically trained as a phonetic classifier with supervision from annotated speech data. Thus, for each training speech utterance, a human annotator listens and assigns the correct spoken words. However, at the scale of 1 Million hours of speech, human annotation becomes impractical. Both the cost and the logistics required to manage such a large undertaking in a reasonable amount of time are prohibiting. Therefore, here we use semi-supervised training, where only a fraction of the speech data is annotated.

To characterize the scale of the task, 1 million hours is equivalent to a constant stream of speech, 24/7, for 114 years – far more than any human will hear in a lifetime. The number of utterances is on the order of one billion, and since we extract hundreds of feature values for every 30 milli second of each utterance, our models process over 1 trillion attributes during training. In terms of raw size for storage, for the commonly used 256 kbps wav formats for audio, 1 million hours of audio uncompressed requires 107 TB of disk space for storage. Building a robust data transformation and training pipeline that handles such data sizes has required close attention to, among other considerations, the failure modes of underlying software, machines, disks and networking channels; limitations of cluster computing frameworks; theoretical algorithmic scaling bottlenecks and monetary cost of resources utilized.

For the SSL system discussed in the paper, we employed the student/teacher learning paradigm, specifically focusing on the student learning subsystem. We factored the student learning subsystem into two pipelines: data preparation and model training. Each has unique scaling challenges. Aside from managing a cluster of compute nodes and distributed data storage, the data preparation step performs shuffling and normalization that is non-trivial in a large scale distributed system. The model training on the other hand must address the challenge of large-sale data-parallel neural network training. This is a fast evolving field that has received significant attention in recent years ([16], [17], [18], [19], [20], [21]).

The remainder of the paper is organized as follows: In the next section we give an overview of the system, followed by section III about the computing infrastructure. In sections: IV, V, and VI we give more details about the most prominent components of the system, and finally section VIII reports our results and IX concludes.

II. Overall System

A. Background on the ASR System

1) Signal processing system: The Alexa family of devices [22] use an array of microphones arranged in different geometries. The signal processing system takes the raw channels and the playback channel as input and returns a single signal for downstream processing by the wake word detector ([23] [24]) and the ASR system ([25], [26]). This system uses beamforming (BF) algorithms to emphasize speech from a desired direction while suppressing audio interference from other directions, and an acoustic echo canceler (AEC) to remove the playback channel from each of the beams.

2) Acoustic and Language Models: The input to the ASR system used in this work is an audio signal, from which we derive a sequence of fixed size acoustic vectors ($X_{1:T} = x_1, ..., x_T$). The ASR system solves the problem of finding the most likely sequence of words ($W_{1:M_W} = w_1, ..., w_{M_W}$) given the sequence of acoustic vectors

$$\hat{W} \propto \arg\max_W \underbrace{p(X|W;\Theta_{\text{AM}})^{\kappa}}_{\text{AM}} \underbrace{P(W;\Theta_{\text{LM}})}_{\text{LM}}, \qquad (1)$$

where Θ_{AM} and Θ_{LM} are the free parameters of the acoustic and language model, and κ balances the impact of the acoustic model against the language model. The AM is based on the standard *HMM/deep learning* hybrid, and we summarize details relevant to this paper in Section II-B. Other aspects of this system have been described elsewhere ([25], [26], [27], [28]).

The LM [29] estimates the *a priori* probability that the speaker will utter a sequence of words. The decoder is an FST based decoder using an optimized dynamic composition approach. In this work, we use a large n-gram model.

B. Fully Supervised Acoustic Model

We use an HMM-LSTM hybrid. The HMM models low-frame rate single state triphone units [30]. States are clustered down to 3,183 senones using phonetic decision trees. The acoustic features consist of 64-dimensional log mel-warped energies computed on audio signals every 10 ms with a 25 ms analysis window. These are stacked three at a time and sub-sampled to a 30 ms advance. A causal mean estimate is computed and subtracted, and finally global mean and variance normalization is applied. To compensate for sub-sampling, features are created at three different offsets for each utterance.

The LSTM model is a stack of five layers, each consisting of 768 units resulting in about 24 M parameters. The model has a three-frame look-ahead. The training data is 7,000 hours of labeled US English data. The models are trained first with the cross-entropy criterion (CE), using alignments computed on the labeled data. First, we follow an exponential learning rate decay for ten epochs, with chunked BPTT for greater parallelization efficiency [31]. In this technique, utterances are split into smaller sub-sequence chunks (here, 32 frames) and the sub-sequences are randomized. For each epoch we cycle through a different feature offset. Then the models are fine-tuned using full sequence CE BPTT for two more epochs. Finally, three epochs of the sequence discriminative criterion state-level minimum Bayes risk (sMBR) is applied [32].

C. Semi-Supervised Learning

SSL has a long history in ASR ([33], [34], [35]). Self-training is the most commonly used approach where typically there is a smaller labeled dataset, and a much larger unlabeled dataset. The labeled data is used to train a seed model from a powerful model family, which is used to decode the unlabeled data at the second stage (often large beam sizes are used). The most reliable hypotheses are selected based on confidence measures [36] and the speech data with the selected hypotheses are used for re-training the AM.

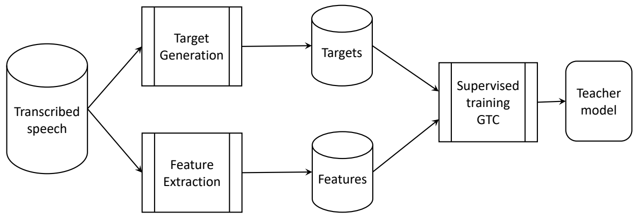


Fig. 1: *SSL Teacher Subsystem: Training the teacher model on transcribed data. We also show the distributed trainer "GTC", of which we will discuss more in Section VI.*

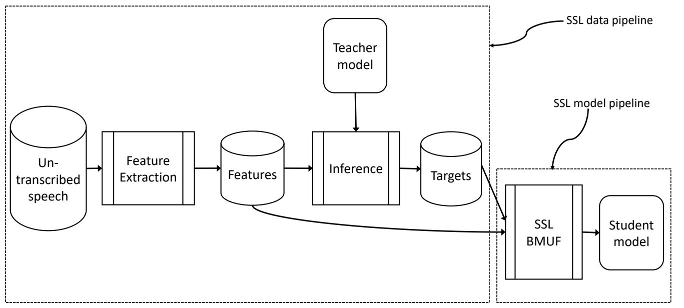


Fig. 2: *SSL Student Subsystem: Training the student model on 1 Mhr of untranscribed data using student/teacher methodology. This figure also illustrates how the data and model pipeline subsystems fit in the overall system. Also shown in the figure is the distributed trainer "BMUF", of which more will be described in Section VI.*

1) Overview of the SSL System: Our approach to SSL was to employ the student/teacher learning paradigm, which avoids explicitly modeling confidence scores, thus taking the ASR decoder out of the SSL recipe. The full teacher and student subsystems are shown in Figures 1 and 2. The teacher and student models output probability distributions over senones; the learning objective optimizes the CE loss between these two distributions. The teacher model is not bound by the same constraints as the runtime system where the student will be deployed. Therefore, we can use a more powerful model for the teacher. In our case we used a bidirectional LSTM model which has access to both the past and future audio. Note that this teacher model is more accurate than a production system where live audio is streamed and we cannot use information from the future. We can also use a larger teacher model without incurring compute cost or latency in the live production system. Apart from the difference in model family, the training of the teacher on the labeled data follows the same recipe as the regular LSTMs, discussed in Section II-B.

We experimented with many different ways to utilize both the annotated and un-annotated data. The final training recipe is periodically inserting the annotated data to the student training interspersed with the un-annotated data. We refer to this method as *scheduled learning*. However, we use only the annotated data in the last steps of the training recipe. Those steps use sequence training which is more sensitive to label errors. A fuller discussion of these modeling challenges and the solutions is presented in [37], and we summarize this in Section V-D.

The focus of this paper is a description of the system level challenges in realizing this large scale system. Over the next few sections, we summarize these challenges, outline the resources at our disposal and elaborate the design tradeoffs.

2) SSL System Challenges: In Section II-C1 we discussed the factorization of the SSL system into teacher and student pipelines. The actual realization of the system is more involved; for simplicity of discussion we factor it into two main components: data pipeline and model pipeline. Each has their own scaling and efficiency challenges.

SSL Data Processing Challenges The major data processing challenges for the SSL pipeline are scalability and robustness. Some computing steps in AM data preparation and training are inherently sequential, and others require cluster inter-worker data transfer; these introduce delays in the processing system. Another challenge is the granularity of a computing step: smaller steps increase the degree of parallelism, but fails to take advantage of joint optimization between steps. In addition for large scale systems, with source data over 100 TB (alternatively, a billion audio files), and with tasks distributed over several thousand computing cores, the system must be resilient against temporary host and process failures, data corruption and network timeout issues. Finally, to make the system viable in practice, it is necessary to do so within tight time and infrastructure budgets.

SSL Modeling Challenges For the modeling subsystem, accuracy and scalability are the most important challenges. From an accuracy standpoint, data selection and filtering for SSL is a question to consider; several sampling strategies have been previously suggested [38]. In addition, SSL with self-training requires good confidence measures, with several previous proposals ([36], [39]). Another challenge with models which have high memorization capability such as LSTM AMs is that label quality becomes even more important [40]. A further challenge is applying sequence discriminative training, where label errors have a larger detrimental effect ([41], [42]). From a scalability standpoint, for the scale of data we consider in this paper, an efficient inference mechanism to generate training targets is an important constraint. Also the model training must address the challenge of large-sale data-parallel neural network training.

III. ML Training Infrastructure

In this section we describe our training infrastructure in terms of compute, storage, and the ML software.

A. Elastic Computing

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. Our pipelines were deployed on EC2 instances. EC2 provides a wide selection of instance types optimized to serve different types of workloads. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give the flexibility to choose the appropriate mix of resources for our jobs. We used three different instance types across our pipelines depending on the characteristics of the task:

- P3: p3.16xlarge hosts are General purpose GPU instances. The instance type hosts 8 NVIDIA Tesla V100 GPU devices per worker. We used P3s for steps that could take advantage of them, such as model training and inference to generate targets.
- X1: x1.32xlarge machines are dense compute instance types, with 128 cores in their Intel Xeon E7-8880 v3 processors and up to 1,952 GB of DRAM-based memory. We used these hosts for computing steps that have significant data transfer between workers in a cluster. Using X1s reduces the amount of data being exchanged across the network between the workers.
- C4: c4.4xlarge instance types have 16 Intel Xeon Platinum processors and were used for highly parallel and independent steps in the pipeline, where computing speed is the main concern.

B. Distributed Storage

AWS S3 is a highly available, durable, and scalable key-value data store that does not pose practical limitations on object sizes. S3 was used as the distributed store for audio, features and targets produced by the pipeline, and for the teacher and student models during training. We use three APIs/operations to access objects in S3 throughout our pipelines: *PUT Object*, *GET Object*, and *HEAD Object*. Our data layout and cluster configuration were driven by the following considerations:

- S3 has an overhead of around 200 milliseconds on reused connections for the three operations.

- Provided that we are under the network throughput limit of an instance type, and S3 buckets are appropriately partitioned, the average bandwidths for *PUT Object* and *GET Object* operations are 50 MB/s.

C. Cluster Computing

Apache Spark [11] is an open source cluster computing framework that provides a concise programming model for processing large datasets with implicit parallelism and fault tolerance. Central to the Spark programming model is the concept of a Resilient Distributed Dataset (RDD) – a distributed collection of records spread over many partitions. Spark provides abstractions to operate on RDDs and the unit of parallelism is an RDD partition. A common pattern when using Spark is to integrate data loading and staging from distributed file system implementations. However, we used Spark for its fault tolerant model in our iterative algorithms, but relied directly on S3 to persist data for jobs. In our Spark jobs, an RDD starts with a list of objects in S3 that serve as the unit of parallelization. An alternative would have been to use an S3 based implementation of Hadoop's distributed file system interface, which would internally call different S3 operations to load records from S3 files into the RDD. However, to scale effectively, relying on Spark and S3 directly was a early design decision in this work. There are a few fundamental transformations permitted on RDDs such as map, filter, join, and reduce. A key observation here is that operations that perform a shuffle of the elements or involve a repartition of the RDD, involve data movement across workers and cause bottlenecks when working with large datasets.

For scheduling Spark jobs we use the stand-alone Spark default FIFO scheduler: by design our files were of roughly equal size and we have a far greater number of files than CPU cores; so, the default scheduler works well. However, the default scheduler introduced complexity in target generation where we need to schedule tasks on GPUs, instead of on CPUs. For such tasks we implemented a custom scheduler.

D. Machine Learning Software

We made use of the open source ASR toolkit, Kaldi [43] for extracting features from audio, computing and applying normalization techniques, and to serialize data derived from utterances. We use an in-house distributed deep-learning training toolkit [16] for training the acoustic models that has been optimized for the task. We investigated two types of distributed training: Gradient Threshold Compression (GTC) [16] and Blockwise Model Update Filtering (BMUF) [21]. We discuss this in more detail in Section VI.

IV. The Data pipeline

For building acoustic models in the large SSL framework, features computed from the audio and their corresponding machine generated targets need to be generated in a scalable and robust fashion. This involves composing a sequence of several highly scalable steps, which we refer to as the data pipeline. Spark provides resiliency and fault tolerance needed to run some of these steps whose execution time span days. Further, we use S3 as the backing store for some of the intermediate artifacts. We built the pipeline iteratively to aid rapid profiling, fine-tuning, and experimental turnaround. The pipeline is designed to be elastic so that it can scale to even larger datasets.

A. Design Principles

For a system that operated at this scale, the design was a crucial element. The design of the pipeline follows these principles.

1) Optimize input files for parallelism early in the pipeline: Our first step determines how many files the pipeline will consume, and how data will be organized within the files to optimize processing in the future steps. For example, by aggregating and sharding audio based on speaker early in the pipeline, we were able to avoid costly data transfer in later steps, such as in the feature normalization by speaker algorithm that occurs during feature extraction. The trade-off involved in fixing the file partitions early is that we limit the flexibility of reusing intermediate data artifacts for other experiments and pipelines.

2) Avoid distributed algorithms that have high inter-node I/O: Spark group-by, join, and shuffle operations are I/O intensive, requiring heavy data transfer between nodes in the cluster and we avoid them where possible. Consequently, we do not perform utterance level shuffle operations to provide feature randomization as we had run into its scaling limitations even when working with smaller data sets. Instead we shuffle elements hierarchically during feature file generation. This results in features being less uniformly randomized across our all our files. However we did not observe a degradatation in accuracy improvements when increasing the sizes of our datasets in our experiments using the heirarchical shuffle.

3) Aggregate to nearly equal sized files: Working with larger aggregated files reduces the total number of files; this reduces the number of S3 interactions each of which has an unavoidable latency overhead. The *nearly equal file size* property ensures that Spark's default FIFO job scheduler provides a near optimal processing time. We note that files being our unit of parallelization, reducing the total number of files results in the us limiting the ceiling of parallelization, but in practice we are not close to this limit.

4) Perform mini-batch operations locally in a CPU/GPU: Most pipeline steps rely on invoking local, external processes per CPU/GPU on audio data to perform a transformation function. By operating on mini-batches instead of individual data elements, we reduce resource contention and amortize the process start time over the mini-batch. In addition, mini-batches on GPUs take advantage of low level GPU parallel primitives giving a further performance boost. The trade-off with batches is the increased software complexity in implementing batch interfaces and handling failures during processing of individual elements in a batch.

5) Consolidate steps: At this scale, the aggregate S3 operations for a pipeline step can take substantial time. By reducing the total number of steps, we eliminate entire file set transfers, and in turn reduce end-to-end time. However, this involves a cost trade-off in merging steps. If steps that can leverage cheaper compute are merged with a step that requires more expensive compute, the entire set of merged steps may now need to execute on the more expensive compute nodes. Still, we collapsed several steps in feature extraction to a single step with significant savings.

B. Steps in the Data Pipeline

Following the above principles, we implemented a pipeline to produce the data set for the 1 M hour training. It consists of the following steps.

1) Data selection: Prior to any job being run, all potential audio files are stored as individual files in S3. Metadata associated with these individual utterances such as an anonymized speaker id, duration, creation timestamp, locale, and the S3 location of the utterance files are stored as JSON records in aggregated files in S3. The aggregated files are partitioned vertically on time interval and horizontally on speaker id. Since our storage systems could have utterances whose duration in aggregate would be more than a million hours, our first step was to select utterances for training. We followed a simple strategy for this across all data: over the period spanning several months, we uniformly selected 1 million hours of audio. The random selection is performed by first loading the entire utterance metadata set for the duration into an RDD followed by sampling records in the RDD using a mild oversampling strategy based on average utterance length statistics. Once selected, we stored relevant metadata in S3. The output files from this step were stored as nearly equal 200,000 files in S3 (about five hours each). Data from a speaker is grouped into the same file; we retain this partitioning scheme for the rest of the steps in the pipeline. Although this is a step that does need inter-node data transfer for the grouping, the total data being handled is much smaller (200 GB uncompressed); it fits into a single X1 instance and poses no immediate scaling bottleneck.

2) Feature extraction: For each utterance stored across the 200,000 files from the previous step, we fetch audio staged in S3 and group them based on speaker. We then sort this audio by its creation timestamp to provide an audio stream per speaker; we extract features for every frame, one speaker at a time. The algorithm also performs a causal mean normalization over all the audio for a speaker. The total size of the features at this stage is around 135 TB. For training deep learning models, features need to be shuffled across the entire file set. We developed a hierarchical shuffling strategy: that both shuffles the global ordering of the files and the features corresponding to the utterances within each of the 200,000 files. We then normalize features using the global statistics: zeroth, first and second order statistics. Statistics for each shuffled file are computed in parallel, and then aggregated across files. Finally, we subsample the features to a lower frame rate (33Hz), while ensuring no information loss by splicing subsampled frames.

3) Target generation: We use a trained teacher model to generate training targets for the student model. Target generation is more efficient on GPUs with batching. As the stand alone spark scheduler we used in our data pipeline could not efficiently distribute tasks in multi-GPU machines, we implemented a custom local scheduler for this step to distribute the load on all available GPUs in the cluster.

4) Repartition targets and features: The final step of the data pipeline is to repartition the files to nearly 500,000 partitions (each with about 2 hours of audio) to improve the efficiency of distributed model training.

C. Implementation Considerations

The pipeline consists of four steps that starting from audio data in S3 executes sequentially to produce inputs for training. Each step is a cluster job that runs on a fixed pool of identical EC2 instances that takes as input, the output from one or more of the previous steps and a cluster configuration. Inputs and outputs are lists of objects in S3. The steps are modular and can be invoked independently given the inputs required for that step. Apart from advantages of modularity and logical separation of concerns, another reason for this design is the large variance in compute resources for individual steps; thus separating the steps based on the computing needs per step allowed us to be more efficient. Cluster configuration for each step includes:

- EC2 Instance types and the number of instances of that type to be used.
- S3 resource permissions for read and write operations on S3 objects.
- Software dependencies for the job, each of which gets deployed on the entire Spark cluster including the master node and the slave nodes.
- Spark configuration for the job such as the number of executors per node, number of cores to allocate per executor and the Java Virtual Machine(JVM) settings for each executor.

V. The Model pipeline

A few design decisions were critical not just for performance, but also for relatively fast experiment turnaround time as well as to be able to build a model on 1 Mhrs efficiently. In this section we report the key ML design choices in our system.

A. Student-Teacher Learning

We used the student/teacher learning methodology [44], [45], [46] [37] thus simplifying the SSL modeling recipe and eliminating the need for a full ASR decoder. For each feature vector, the teacher and the student networks compute posterior probability distributions over senones. While the teacher parameters are frozen, the student network's parameters are estimated by minimizing the cross-entropy loss between the two posterior distributions. The student network structure is identical to the LSTM AM described in the Section II, but the teacher networks have five bi-directional LSTM layers, each with 768 units (totaling 78 M model parameters) – this is nearly 3 times the size of the student network. The training of the teacher network on labeled data uses the same recipe as the regular LSTMs.

B. Confidence Modeling

It has been reported previously that modeling even with unfiltered data can lead to significant WER improvements in the context of SSL ([34], [47]). Furthermore, as neural network technology has improved, so have the estimated probabilities become better calibrated [48]. Our hypothesis is that the teacher's posteriors are well enough calibrated to act as the confidence measure for student network training. However, in a traditional self-training system, the language model also provides additional information during decoding; in our SSL system the LM is not present. We speculate that this is partially mitigated by using bi-directional teacher LSTM models, which observes more context than the student to make a frame level decision.

C. Target Generation

Since the senone output distribution is large (a 3,183 dimensional vector), generating targets using the teacher model on-the-fly slows down training. As we parallelize training across multiple GPUs, to reduce network bandwidth and to minimize storage, we store only the k highest valued logits. During the student network training, full posteriors are reconstructed; the missing logits are filled with large negative values. While the reconstruction procedure is lossy, our experiments showed that probability mass is dominated by a few top posteriors. The hyper parameter k is selected empirically based on the teacher model family, its model structure, and the amount of data it has been trained on.

D. Scheduled Learning

Although most of the data used for training is unlabeled, we found that using the limited labeled data can be useful in obtaining performance improvements. Our learning algorithm interleaves parameter estimation on unlabeled and labeled data, with slightly higher learning rates on the labeled data. Given the size of unlabeled data, our design was to perform just one epoch through it while visiting the labeled data multiple times. We divided the unlabeled data into a fixed number of *sub-epochs*, with a sub-epoch defined as 55,000 hours. We decayed the learning as we ingested unlabeled data through the sub-epochs, following an exponential learning rate decay. After each sub-epoch through the unlabeled data, we perform CE training on the labeled data, with a rotation through the feature offsets (please refer to Section II). As described in Section II we employ sequence chunked BPTT for training speed; we apply chunked training for the first 15 sub-epochs, and then perform fine-tuning during the last three sub-epochs.

E. Sequence Training for SSL

Sequence discriminative training of a deep learning AM often yields large WER improvements (commonly, around 10% relative [49]). However, discriminative training is a difficult problem for SSL ([50], [41]), since the discriminative loss function can be sensitive to noisy references during training. Our decision was to perform sMBR training of the student model only on labeled data. Previous work [51] indicates that the accuracy gains may be relatively small in such a setup. However, we hypothesized that this result was likely due to a relatively small labeled dataset, and using the full 7,000 hour labeled data in this study could still yield large gains from sequence training.

F. Distributed Training

Identifying a good approach to performing distributed training of the student model was a key element of our design, exploring the tradeoff between scalability and accuracy. We studied Gradient Threshold Compression (GTC) [16] and Blockwise Model Update Filtering (BMUF) [21], of which we discuss more in Section VI.

VI. Distributed Training

For large-scale neural network training, distributing the workload across many GPUs is required to produce a trained model in a reasonable time. Here we will discuss data-parallelism, where different worker nodes process different input data, but share the model that is trained. The widely used stochastic gradient descent (SGD) opimization technique (e.g. [52], [53]) has a serial aspect to it that makes it challenging to scale SGD to large number of workers. Let's assume that the standard technique of using a "mini-batch" is used. A mini-batch is a small batch of data-points that are processed efficiently on a single GPU. Basic data-parallelism then involves computing mini-batches on all workers, aggregating their gradients and updating the shared model. However, by increasing the number of compute nodes, the effective (aggregated) mini-batch size is increased linearly, which has shown to produce lower accuracy on the validation and test datasets, and generally reduces model convergence rate [54], [55]. Several techniques can be employed, such as adjusting learning rate [56], [54], using a warm-up phase (e.g., [16]), etc. which can increase an upper bound on the workable effective mini-batch size, but fundamentally there still remains an obstacle of scaling to large GPU clusters.

A specific scaling challenge is communicating gradients between workers which requires high bandwidth and as the size of model or the number of workers are increased, this can lead to a severely communication-bounded algorithm. This also depends on whether workers are on the same host or different, since the cost of communication between different hosts is typically much higher. In case of larger number of workers, the cost of communication can thus dominate the total training time. In particular for cases where the ratio of compute-time to communication is low, having high bandwidth interconnect is then necessary to reduce overall time spent in communication. Empirically we find that with high-end compute nodes such as AWS p3.16xlarge which contains 8 Nvidia V100 GPUs and provides data transfers upto 300 Gbps across peer GPUs on the same host, data parallel SGD can scale almost linearly within a single host. Algorithms such as ring-allreduce [57] and hierarchical ring-allreduce [58] are used which aim to utilize available bandwidth optimally among compute devices. However due to the limited bandwidth of 25 Gbps across hosts on these instances, the scalability of the training beyond single host is severely affected.

Several techniques have been developed to reduce the bandwidth required in gradient communication. The early works of [17] and [16], that introduced the quantization and compression by gradient thresholding, have later been refined and used in many contexts, e.g. [18], [19], [20]. These techniques reduce the amount of data which needs to be communicated between workers to reduce overall communication time. This increases the limit on the number of GPU worker nodes that can be used in parallel without rendering the algorithm communication-bound.

Another well-known general technique to scale distributed training is based on the concept of Model Averaging (MA). In this technique each worker updates a local model based on many mini-batches from the dataset without communicating with other workers. The model is only synchronized across workers after some interval of time or specified number of mini-batches. At that time, all model weights are averaged across workers and synchronized. Because of the infrequent synchronization, this approach can scale training data throughput almost linearly. However in its basic form, it suffers from reduced model training convergence because non-linear divergence between local models that is not well-matched with averaging. A variant, Blockwise model update filtering (BMUF) [21], mitigates this issue significantly.

In this paper we are comparing and contrasting one method from each of the above mentioned data parallel approches, namely gradient averaging and model averaging. We selected synchronous SGD with gradient threshold compression[16] (GTC) and BMUF [21] which both try to address the issue of scaling data parallel training.

A. Gradient Threshold Compression

In GTC, instead of sending the entire gradient tensor for each trainable weight, only gradient elements whose absolute magnitude is greater than a constant, here referred as gradient-threshold (τ) are sent to other workers. This results in a very sparse gradient update – typically reducing the gradient size by several orders of magnitude. Each worker communicates the sparse update to all other workers and conversely receives all sparse updates from other workers. The received sparse gradient updates are aggregated and weights are updated based on the aggregate. The residual gradients which are not sent to other workers are aggregated locally for later iterations. In a naive implementation, a sparse update can be represented by two numbers, an integer element index and floating point number. However this can be compressed further by quantizing gradient and packing quantized gradient and integer index into single 32-bit integer field. In this work, we use 1-bit quantization [17]. Thus, each worker simply sends gradient deltas of $\pm\tau$. This simple coding scheme further compresses the update by 2x. The pseudo-code for this algorithm and a more comprehensive discussion can be found in [16]. The technique can be applied to synchronous as well as asynchronous variants of SGD, however we select the synchronous variant to ensure reproducibility.

B. Blockwise Model Update Filtering

The BMUF algorithm [21] is a variant of model averaging, augmented by considering the model from previous step. First the initial global model (W_g) is broadcasted to all workers. The algorithm then iterates two main steps. In the first step, each worker updates its local model (W) in parallel with its portion of data for a specified number of mini-batches, here referred as block-size. This step is called intra-block parallel optimization and requires no synchronisation between workers. In our implementation, each worker simply updates its local model using mini-batch SGD independently. In the second step, which is referred to as the BMUF step, the global model is updated using the following procedure.

$$\overline{W}(t) = \frac{1}{N}\sum_{i=1}^{N} W(t)^i \tag{2}$$

$$G(t) = \overline{W}(t) - W_g(t-1) \tag{3}$$

$$\Delta(t) = \eta_t \Delta(t-1) + \zeta_t G(t) \tag{4}$$

$$W_g(t) = W_g(t-1) + \Delta(t) + \eta_{t+1}\Delta(t) \tag{5}$$

where hyper-parameters η and ζ are called block-momentum and block-learning-rate respectively. We used following formula

$$\frac{\zeta}{N(1-\eta)} = C \tag{6}$$

to set η and ζ hyper-parameters, where $C \geq 1$ is constant and N is number of workers. We use Nesterov block momentum (NBM) scheme proposed in [21].

The evaluation of these two training methods for frame level accuracy and speedup is described in section VIII-A.

C. Accuracy and Scalability Trade-offs

Both above mentioned algorithms provides flexibility to scale to large number of workers through hyper-parameters, however it may come at the cost of reduced accuracies when number of workers are large. The GTC algorithm can be scaled by controlling gradient-threshold parameter which directly affects sparsity of gradient values. This results in lower update size and reduces overall communication time. The trade-off of different gradient-threshold values on accuracy and speed is described in [16] for asynchronous variant of the algorithm. In our studies we found gradient-threshold of 8 achieved best trade-off between accuracy and scalability. For the BMUF algorithm, the block-size hyper-parameter can be used for controlling how often global model is updated. Setting block-size to large number can enable almost linear scaling, however this results in considerable drop in accuracy for large number of workers. This is further analyzed in [21]. In our studies, we found block-size of 100 achieved best trade-off between accuracy and speed.

VII. Experimental Setup

We discussed system level details in Section II-B. In this section we provide details on our experimental setup, including the training and test data sets. We also discuss various models, and briefly describe the decoding setup.

A. Training Datasets

For our experiments we used three far-field training datasets drawn from the production data of the Alexa family of devices from the US English locale: (a) a 1,000 hour fully labeled dataset for distributed training experiments, (b) a 7,000 hour fully labeled dataset used for training the teacher model, and (c) a 1 Million hour unlabeled dataset for SSL model build. The 1,000 hour dataset is a subset of the 7,000 hour dataset.

B. Test Datasets

We used two test sets in this work. The first is a validation test set (referred to as VAL), which consisted of about 1 hour of data to evaluate the distributed trainers. The accuracy on this test set is evaluated using frame classification accuracy, but more importantly, we use it to measure training speed. The second test set (TST) consists of audio data collected in a real room with about 5,000 utterances roughly equally spread among five device placements. The first device placement (DP1) in the center of the room led to the lowest error rate, while other conditions (DP2 to DP5) were more challenging. On this test set we use word error rate reduction (WERR) to evaluate the model performance. We have also reported on other test sets in [37].

C. Models

In this section we summarize the models relevant to the experimental setup. All the acoustic models (recognition and alignment models) employed in this paper use the hybrid HMM-LSTM approach.

1) Acoustic front-end: The sampling rate of the speech signal in all the datasets used in this work is 16 kHz. The features for the deep learning models come from an acoustic front-end that outputs 64 dimensional log filter bank features at a frame rate of 33 Hz; section Section II-B describes it in greater detail. The phonetic decision tree, however, was built using 40 dimensional features from a different front-end: application of LDA followed by MLLT transforms on 39 dimensional PLP, including delta and delta-delta.

2) Frame level hard targets: The triphone HMM states were clustered down to 3,183 senones using a phonetic decision tree built on the 7,000 hour dataset. Alignments from the alignment model were mapped, and several rounds of realignment followed by parameter reestimation were performed. Using these alignments, an LSTM trained using the CE criterion, discussed in Section II-B is used to generate frame-level targets for training the supervised models in this work. All the alignment models used the GTC trainer in conjunction with 16 V100 GPU cards.

3) Sequence training: Sequence discriminative training in this paper used the sMBR [32] criterion with lattice based methods. sMBR training, for all models including the teacher and the student models, was performed on the fully labeled 7,000 hour dataset. It used GTC trainer with 16 V100 GPU cards. The lattices themselves were shallow, with an average density of around 10, and were stored as compressed files. The space required for storing the lattices for a system was around 6 GB.

4) Teacher model: The teacher is a bidirectional LSTM (BLSTM) model built on the 7,000 hour fully labeled dataset using the features and frame-level targets discussed in this section. This model has 5 bidirectional LSTM layers, each with direction and layer having 768 units. The model parameters were first estimated by minimizing the frame level cross-entropy criterion. The training strategy discussed in Section II-B was followed: 10 epochs of chunked training, followed by 2 epochs of fine-tuning. Finally 3 epochs of sequence training was performed.

5) Features and targets for the student model: Features for the 1 Million hour dataset were generated using the system described in Section IV-B. As in Section VII-C1, this results in 64 dimensional log filter bank features at a frame rate of 33 Hz. Using the trained BLSTM teacher model, frame level soft targets are generated using these features and stored as compressed k-best logits (with k being 20 for the experiments) using the techniques discussed in Sections IV-B3 and V-C.

6) Student model: The student network is identical to the LSTM architecture described in Section II-B. The student model is trained on the features and targets discussed in the previous subsection with scheduled learning, as discussed in Section V-D. We used the BMUF trainer with 8 p3.16xlarge hosts. Lastly 3 epochs of sMBR training restricted to 7,000 hours with GTC trainer was performed.

7) Baseline fully supervised model: The baseline fully supervised system is an LSTM, identical to the network discussed in Section II-B. This network was trained as discussed in Section II-B, on the 7,000 hour dataset, using the same set of features and targets used for training the teacher model. Lastly, 3 epochs of sequence training was performed on the same labeled dataset.

D. Decoding Setup

All decoding on the TST test set use a 4-gram statistical language model (LM). The acoustic model scale factor was tuned on this test set. We compare the SSL model against a strong, fully supervised baseline system [37].

VIII. SYSTEM EVALUATION

We evaluate the system along several dimensions. A key metric is the accuracy of the trained models. For this dataset, accuracy is reported as relative word error rate reduction (WERR) (cf. [37], [26], [27], [25], [28]). Here we show a subset of accuracy results with a more complete picture available in [37]. Other important metrics are the processing time and cost of infrastructure.

A. Distributed Training

We trained models using GTC and BMUF on the 1,000 hour labeled data[1], which were then evaluated on the VAL test set.

The training speedup and relative frame-level classification accuracy improvements of the two trainers are tabulated in Table I. Both metrics are relative to a 1-GPU SGD trainer which

[1]We used AWS EC2 p2.16xlarge instances for experiments in Table I. where each instance consists of 16 NVIDIA K80 GPUs.

TABLE I: *Relative frame-level classification accuracy improvements (in %) and training speedup (as a factor) of GTC and BMUF-NBM compared to 1-GPU SGD trainer. This table illustrates the trade-offs for the two trainers, as a function of the amount of compute (number of GPUs).*

Training Method	Number of Workers	Relative Frame-level Accuracy Improvement (%)	Training Speedup
GTC	8	0.41	7.01
	16	0.41	12.53
	32	0.54	21.75
	64	0.27	16.76
BMUF-NBM block-size: 100	8	-0.27	7.18
	16	-0.06	13.34
	32	-0.10	25.46
	64	-0.13	50.91
	128	-2.46	97.59

does not perform any gradient thresholding or quantization. It can be seen that in terms of accuracy, both trainers are within 1% relative compared to the 1-GPU SGD baseline. BMUF trainer shows higher degradation of about 3% relative when run on 128 GPUs. Synchronous GTC training scales well up to 32 GPU cards, but its throughput tapers off at higher scale. This is due to increased cost of communication needed to synchronize gradients per mini-batch as number of workers are increased. On the other hand, BMUF scales almost linearly with number of workers, at least in terms of throughput, since model synchronization is much more infrequent. However, it comes at a cost of training convergence rate and reduced accuracy at higher number of workers. The Nesterov-like momentum updates at block level recover some of these losses, but empirically we still see some degradation.

B. End-to-End Processing Time

The end-to-end time from data to a fully trained model yields an assessment of our system design. A breakdown of the processing times also gives the constraints that limit our ability to scale the model training to even larger data regimes.

1) Proposed system design: With the design decisions taken in this work we obtain an end-to-end turn around time of the student pipeline in 12 days. Figure 3 breaks down the processing time in different parts of the student training pipeline. As a side-note, the initial training of the teacher model and the storing of utterances corresponding to the 1 Million hours of speech in S3 takes an additional 4 days (the training of the teacher model itself takes 2 days[2]).

The SSL data pipeline for generating features and targets for the 1 Million hours of speech took 7.6 days. This pipeline is relatively straight-forward to parallelize. For most steps, since each data partition is independent, adding more hardware can parallelize the step further. We perform distributed computation on Spark, and all our data is staged in S3. Since both systems

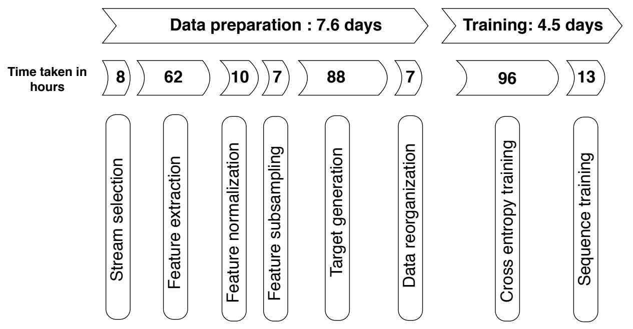


Fig. 3: *This figure presents a breakdown of the end-to-end processing times to train a full SSL student subsystem. The data pipeline scales linearly with more compute and can have even quicker turnarounds; using more compute, the model training can scale further in terms of training times, but it comes at the cost of accuracy.*

are known to scale well, the data pipeline can scale nearly linearly by increasing the cluster size. Further algorithmic improvements are possible with more caching, pre-computations and aggregations, though at the cost of more storage. Model training contributes to a smaller part of the total time (4.5 days). In this project, we increased the number of GPUs to 64, for a very significant speed-up. Adding further compute by using more GPUs can speed up training linearly, but has diminishing returns in terms of accuracy (as presented in Table I).

2) Comparison to fully supervised system: We close the discussion on processing times by making a note on the fully supervised AM described in Section II-B, and the model was built as presented in Section VII-C7. This involved 2 steps:

1) *Feature extraction and target generation*: This was the bottleneck: this system was impractical for feature extraction and target generation at the scale of 1 Million hours. For the 7,000 hour data, the feature and target generation took nearly 2 weeks.
2) *Model building including the CE and sMBR training stages*: Model building was done with GTC trainer using 16 GPU cards for both the CE and sMBR stages. This took about 21 hours.

C. System Accuracy

The final results including sequence discriminative training, with sMBR loss function, are reported in Table II on the TST test set. We compare against the baseline fully supervised model trained on the 7,000 hour data (please refer to Section VII-C7 for more details on this model). The results are reported as relative WER improvements.

Except for device position one (DP1), relative WER reductions are all greater than 10%, and indicating that the improvement is greater for harder conditions. We take this as validation that large scale SSL can not only significantly

[2]It might be surprising that the training of the teacher model on 7,000 hours takes 2 days, while the training of the student model takes "only" 4.5 days; but the teacher is a BLSTM, using GTC trainer employing 16 GPUs. Also we perform 12 epochs of training on 7,000 for the teacher. The student model is an LSTM (trains 2x faster), using BMUF trainer employing 64 GPUs (about 4x faster), doing 1 epoch through the data.

TABLE II: *On TST test set (in DP1 to DP5), relative WER reduction (%) of the final 1 Million hour model against a baseline LSTM AM that is sMBR trained on the fully labeled 7,000 hour training data.*

Test Conditions	WERR (%)
DP1	9.8
DP2	22.2
DP3	21.8
DP4	16.5
DP5	18.9

improve accuracy overall, but also yield significant improvement in more challenging conditions.

IX. Conclusion

In this paper we presented an in depth discussion of the design of an efficient end-to-end SSL system starting from 1 Million hours of raw audio and its metadata. Following the student/teacher paradigm for SSL, we focused on the student subsystem, factoring it into two main pipelines: data preparation and model training. To address the challenges of scalability and robustness, our discussion on data pipeline laid out the key bottlenecks and proposed corresponding design principles. These principles were then used in decomposing the pipeline into smaller steps to efficiently address the challenges.

The model pipeline, including the distributed trainer, addressed the twin challenges in ML design for this problem: accuracy improvement and scalability. Scaling posterior generation with k best selection, using scheduled learning to leverage transcribed data, restricting sequence training to transcribed data were among the methods we presented. Our system evaluations showed that even without extensive hyper-parameter tuning, we can obtain relative WER improvements in the 10 to 20% range, with much higher gains in more difficult conditions. The end-to-end processing time of this SSL system was 12 days, and several components in this system can trivially scale linearly with more compute resources for further speed-up.

Acknowledgment

We would like to thank Xing Fan for providing the setup for baseline models; Harish Mallidi for help with the decoding infrastructure; Oleg Rybakov and Tianjun Ye for help with early debugging with our deep learning toolkit.

References

[1] F. Jelinek, "Some of my best friends are linguists," in *LREC*, 2004.
[2] R. G. Leonard and G. Doddington, "TIDIGITS speech corpus," *Texas Instruments, Inc*, 1993.
[3] J. S. Garofolo, L. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continuous speech corpus," *NASA STI/Recon technical report*, vol. 93, 1993.
[4] D. B. Paul and J. M. Baker, "The design for the Wall Street Journal-based CSR corpus," in *Proc. of Workshop on Speech and Natural Language*, 1992.
[5] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "SWITCHBOARD: Telephone speech corpus for research and development," in *Proc. of ICASSP*.
[6] C. Cieri, D. Miller, and K. Walker, "The Fisher Corpus: a resource for the next generations of speech-to-text." in *LREC*, vol. 4, 2004, pp. 69–71.
[7] D. Amodei, S. Ananthanarayanan *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *Proc. of ICML*, 2016.
[8] Y. Huang, Y. Wang, and Y. Gong, "Semi-supervised training in deep learning acoustic model." in *Proc. of Interspeech*, 2016.
[9] H. Soltau, H. Liao, and H. Sak, "Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition," *arXiv preprint arXiv:1610.09975*, 2016.
[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. Ieee, 2010, pp. 1–10.
[11] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache Spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.
[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.
[13] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015. [Online]. Available: http://arxiv.org/abs/1512.01274
[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/
[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.
[16] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. of Interspeech*, 2015.
[17] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs," in *Proc. of Interspeech*, 2014.
[18] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: randomized quantization for communication-optimal stochastic gradient descent," *CoRR*, vol. abs/1610.02132, 2016. [Online]. Available: http://arxiv.org/abs/1610.02132
[19] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *CoRR*, vol. abs/1712.01887, 2017. [Online]. Available: http://arxiv.org/abs/1712.01887
[20] C. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp : Adaptive residual gradient compression for data-parallel distributed training," *CoRR*, vol. abs/1712.02679, 2017. [Online]. Available: http://arxiv.org/abs/1712.02679
[21] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *Proc. of ICASSP*, 2016.
[22] R. Prasad, "Spoken Language Understanding for Amazon Echo," 2015, keynote in Speech and Audio in the Northeast (SANE).
[23] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, "Multi-Task Learning and Weighted Cross-Entropy for DNN-Based Keyword Spotting," in *Proc. of Interspeech*, 2016, pp. 760–764.
[24] R. Maas, S. H. K. Parthasarathi, B. King, R. Huang, and B. Hoffmeister, "Anchored speech detection." in *Proc. of Interspeech*, 2016.
[25] S. H. K. Parthasarathi, B. Hoffmeister, S. Matsoukas, A. Mandal, N. Strom, and S. Garimella, "fMLLR based feature-space speaker adaptation of DNN acoustic models," in *Proc. of Interspeech*, 2015.
[26] S. Garimella, A. Mandal, N. Strom, B. Hoffmeister, S. Matsoukas, and S. H. K. Parthasarathi, "Robust i-vector based adaptation of DNN acoustic model for speech recognition," in *Proc. of Interspeech*, 2015.
[27] B. King, I.-F. Chen, Y. Vaizman, Y. Liu, R. Maas, S. H. K. Parthasarathi, and B. Hoffmeister, "Robust speech recognition via anchor word representations," in *Proc. of Interspeech*, 2017.
[28] L. Mosner, M. Wu, S. H. K. Parthasarathi, R. Maas, A. Raju, K. Kumatani, S. Sundaram, and B. Hoffmeister, "Improve noise robustness of automatic speech recognition via teacher-student learning," in *Proc. of ICASSP*, 2019.

[29] A. Gandhe, A. R. Rastrow, and B. Hoffmeister, "Scalable language model adaptation for spoken dialogue systems." in *Accepted for Proc. of Spoken Language Technologies*, 2018.

[30] G. Pundak and T. N. Sainath, "Lower frame rate neural network acoustic models." in *Proc. of Interspeech*, 2016.

[31] P. Doetsch, M. Kozielski, and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," in *Proc. of ICFHR*, 2014.

[32] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2009, pp. 3761–3764.

[33] T. Kemp and A. Waibel, "Unsupervised training of a speech recognizer: recent experiments." in *Proc. of Eurospeech*, 1999.

[34] L. Lamel, J.-L. Gauvain, and G. Adda, "Lightly supervised and unsupervised acoustic model training," *Computer Speech & Language*, vol. 16, no. 1, pp. 115–129, 2002.

[35] J. Ma, S. Matsoukas, O. Kimball, and R. Schwartz, "Unsupervised training on large amounts of broadcast news data," in *Proc. of ICASSP*, 2006.

[36] M.-h. Siu, H. Gish, and F. Richardson, "Improved estimation, evaluation and applications of confidence measures for speech recognition," in *Proc. of Fifth European Conference on Speech Communication and Technology*, 1997.

[37] S. H. K. Parthasarathi and N. Strom, "Lessons from building acoustic models with a million hours of speech," in *Proc. of ICASSP*, 2019.

[38] J. Pylkkönen, T. Drugman, and M. Bisani, "Optimizing speech recognition evaluation using stratified sampling." in *Proc. of Interspeech*, 2016, pp. 3106–3110.

[39] Y. Huang, D. Yu, Y. Gong, and C. Liu, "Semi-supervised GMM and DNN acoustic model training with multi-system combination and confidence re-calibration." in *Proc. of Interspeech*, 2013.

[40] Y. Huang, Y. Wang, and Y. Gong, "Semi-supervised training in deep learning acoustic model." in *Proc. of Interspeech*, 2016.

[41] V. Manohar, D. Povey, and S. Khudanpur, "Semi-supervised maximum mutual information training of deep neural network acoustic models." in *Proc. of Interspeech*, 2015.

[42] J.-T. Huang and M. Hasegawa-Johnson, "Maximum mutual information estimation with unlabeled data for phonetic classification." in *Proc. of Interspeech*, 2008.

[43] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi Speech Recognition Toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[44] J. Li, R. Zhao, and J.-T. a. Huang, "Learning Small-Size DNN with Output-Distribution-Based Criteria," in *Proc. of Interspeech*, 2014.

[45] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proc. of Advances in NIPS*, 2014.

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

[47] L. Lamel, J.-L. Gauvain, and G. Adda, "Unsupervised acoustic model training," in *Proc. of ICASSP*, 2002.

[48] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *arXiv preprint arXiv:1706.04599*, 2017.

[49] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks." in *Interspeech*, 2013, pp. 2345–2349.

[50] J.-T. Huang and M. Hasegawa-Johnson, "Semi-supervised training of gaussian mixture models by conditional entropy minimization," *Optimization*, vol. 4, p. 5, 2010.

[51] N. Kanda, S. Harada, X. Lu, and H. Kawai, "Investigation of semi-supervised acoustic model training based on the committee of heterogeneous neural networks." in *Proc. of Interspeech*, 2016.

[52] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[53] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "The Microsoft 2016 Conversational Speech Recognition System," *CoRR*, vol. abs/1609.03528, 2016. [Online]. Available: http://arxiv.org/abs/1609.03528

[54] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[55] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *CoRR*, vol. abs/1802.09941, 2018. [Online]. Available: http://arxiv.org/abs/1802.09941

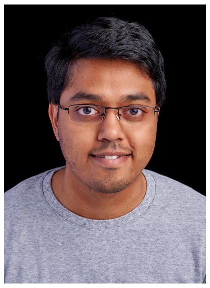
[56] Y. You, I. Gitman, and B. Ginsburg, "Scaling sgd batch size to 32k for imagenet training," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-156, Sep 2017. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-156.html

[57] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, pp. 117–124, 02 2009.

[58] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *CoRR*, vol. abs/1807.11205, 2018. [Online]. Available: http://arxiv.org/abs/1807.11205

Sree Hari Krishnan Parthasarathi is a senior machine learning scientist with the Alexa Speech team. He joined Amazon in 2013 as a member of the team that built Speech Recognition for Alexa. At Amazon he develops algorithms and technologies for large-scale automatic speech recognition and keyword spotting problems. Since 2005 he has academic and industrial research experience working on problems in speech recognition, speaker diarization, speech activity detection, and keyword spotting. Before joining Amazon, he was a Postdoc at ICSI, working on acoustic modeling. He graduated with a PhD from EPFL in 2011. His interests lie in machine learning methods for speech and language processing tasks.

Nitin Sivakrishnan is a Principal Engineer with the Alexa Machine Learning Platform Organization where he guides the architectural strategy for efficient consumption of large-scale data for business activities across Alexa, especially modeling. He has been with Alexa since its launch on the first generation Echo device in 2014. Prior to Alexa, he worked on some of the foundational technologies in Amazon's ecommerce platform. He has a Masters degree in Computer Science from The Ohio State University. His interests lie at the intersection of Machine Learning, Distributed Systems and Big Data Processing.



Pranav Ladkat is a software development engineer with the Alexa Machine Learning Platform Organization since 2016. He contributed to development of deep learning toolkit used by Alexa teams. He received his Masters degree in 2016 from University at Buffalo, State University of New York in the field of Mechanical Engineering with research area in Computational Physics and High Performance Computing (HPC). His interests lies in HPC, distributed systems and Deep Learning at scale.

Nikko Strom was born on July 21st, 1966 in Stockholm, Sweden. He received his Ph.D. from the Speech Communications Laboratory at the Royal Institute of Technology (KTH) in Stockholm in 1997 in the field of Automatic Speech Recognition. He joined Amazon in 2011 as a founding team member of the Amazon Echo and Alexa organization. Before joining Amazon, he was a Principal Engineer at Microsoft, and before that Tellme Networks, and earlier he held a post-doc position at the Speech Communications Lab at MIT. His current research interests are in Speech Technologies, in particular Automatic Speech Recognition, Dialogue Management, and Deep Learning at scale.