Accelerating Deep Neural Network Training for Action Recognition on a Cluster of GPUs

G. Cong, G. Domeniconi, J. Shapiro, F. Zhou, B. Y. Chen

October 29, 2018

Supercomputing 2018
Dallas, TX, United States
November 11, 2018 through November 16, 2018

Accelerating deep neural network training for action recognition on a cluster of GPUs

Guojing Cong, Giacomo Domeniconi, Joshua Shapiro*
IBM TJ Watson Research Center
1101 Kitchawan Road, Yorktown Heights, NY, 10598
{gcong,giacomo.domeniconi1, joshua.shapiro}@us.ibm.com

Fan Zhou
Georgia Tech
Atlanta, GA
fanzhou@gatech.edu

Barry Chen
Lawrence Livermore National Laboratory
Livermore, CA
chen52@llnl.gov

***Abstract*—Due to the additional temporal dimension, large-scale video action recognition is even more challenging than image recognition and typically takes days to train on modern GPUs even for modest-sized datasets. We propose algorithms and techniques to accelerate training of deep neural networks for action recognition on a cluster of GPUs.**

In terms of convergence and scaling, our distributed training algorithm with adaptive batch size is provably superior to popular asynchronous stochastic gradient descent algorithms. The convergence analysis of our algorithm shows it is possible to reduce communication cost and at the same time minimize the number of iterations needed for convergence. We customize the *Adam* optimizer for our distributed algorithm to improve efficiency. In addition, we employ transfer-learning to further reduce training time while improving validation accuracy.

Compared with the base-line single-GPU stochastic gradient descent implementation of the two-stream training approach, our implementation achieves super-linear speedups on 16 GPUs while improving validation accuracy. For the UCF101 and HMDB51 datasets, the validation accuracies achieved are 93.1% and 67.9% respectively. As far as we know, these are the highest accuracies achieved with the two-stream approach that does not involve computationally expensive 3D convolutions or pretraining on much larger datasets.

I. Introduction

Image recognition and object detection have made tremendous progress since the adoption of deep convolutional neural networks (CNN). Current CNN models can classify the ImageNet dataset [20] with over 1 million (M) inputs with very high accuracy (e.g., see [22], [11], [8], [24]), and they can also detect objects in a video frame in real time (e.g., see [19]). Large amounts of data are needed to train these models, and recent studies have tried to accelerate training (e.g., see [27], [7]).

Compared to image classification, video action recognition is even more challenging due to the additional temporal dimension. The CNNs for action recognition are typically more complex than those used in image recognition, which causes their training to take even longer. Currently there are two main approaches of action recognition using CNNs. The first is a two-stream approach (e.g., see [21]) where sample frames of a video are fed to a CNN as a spatial stream (also called RGB stream), and the optical flows between frames are fed to another CNN as a temporal stream (also called the flow stream). The predictions of the two streams are then combined for classification. The second approach employs three-dimensional (3D) convolutions (2D for the spatial dimension with additional 1D for the temporal dimension) (e.g., see [3], [26]). 3D convolutions are computationally very costly, and the validation accuracies achieved are often times not much better than those achieved by the two-stream approaches. In our study we accelerate training for a single-GPU two-stream implementation.

Joshua Shapiro is now at ASAPP: jshapiro@asapp.com

Our primary dataset is UCF101 [23]. UCF101 has over 13,000 video in 101 action categories. There are over 9500 training videos, and over 3700 validation videos. We also show results on the HMDB51 dataset [12].

The base-line single GPU implementation take days to train both streams. We propose a distributed training algorithm to reduce training time. In addition to communication overhead that adversely impacts parallel efficiency, distributed training also faces convergence challenges. Typically if we increase the number of learners in a distributed implementation, more data samples need to be processed to reach convergence. Communication frequency plays a critical role on both communication overhead and convergence speed. We adopt an adaptive-batchsize K-step model averaging algorithm (AAVG) to explicitly manage communication frequency. Our analysis shows that in terms of accuracy and scaling, AAVG is superior to popular asynchronous stochastic gradient descent (ASGD) implementations such as *Downpour* [5] and *EAMSGD* [29]. For our target datasets, we show that communication in AAVG can be very sparse for optimal convergence. From the perspective of distributed processing, AAVG achieves almost linear speedup relative to the single-GPU baseline SGD implementation for the same amount of data samples processed. In addition to communication frequency, AAVG also dynamically adapts batch sizes to balance convergence speed and variance reduction among samples. For training with our target datasets, we use the *Adam* optimizer [10] to avoid costly hyper-parameter tuning (e.g., see [1]). Customizing *Adam* for AAVG brings significant improvement in convergence speed.

AAVG significantly reduces the training time for the RGB stream and the flow stream. Still, it takes much longer to train the flow stream than the RGB stream. We transfer the model learned for the RGB stream to the flow stream, reducing the training time by as much as 30% while increasing validation accuracy.

With our optimizations, AAVG actually processes fewer samples than the baseline single-GPU implementation, and thus achieves super linear speedups. It takes a couple hours for AAVG to train both streams on 16 GPUs for the UCF101 dataset. For the UCF101 and HMDB51 datasets, the validation accuracies achieved are 93.1% and 67.9% respectively. As far as we know, these are the highest accuracies achieved with the two-stream approach that does not involve computationally expensive 3D convolutions or pretraining on much larger datasets.

Our main contributions include a fast and adaptive distributed training algorithm and a transfer learning strategy for speeding up the training of the flow stream.

In our study all implementations use Pytorch [17]. Communication among learners is done using CUDA-aware openMPI 2.0 through MPI4Py. Our cluster has 4 IBM Minsky nodes. Each node has 2 Power8 GPUs with 10 cores each and 4 NVIDIA Tesla P100 GPUs. The interconnect between the nodes is Infiniband.

The rest of the paper is organized as follows. Section II introduces the two stream action recognition approach and the neural networks used; Section III introduces AAVG and its convergence properties, and demonstrates its performance with UCF101; Section IV customizes *Adam* for AAVG; Section V presents our results of transferring the model from the RGB stream to the flow stream to accelerate the training; Section VI discusses the speedup results achieved by AAVG; and Section VII gives our conclusion and future work.

II. Two-stream action recognition

The baseline two-stream training for action recognition implements the approach described in [21], and is illustrated in Fig. 1. During training for the RGB stream, a random frame from an input video is sampled and fed into a CNN followed by a fully-connected layer for classification. For the flow stream optical flows are computed for two consecutive frames, and 10 consecutive flows are randomly sampled and fed to another CNN. During training validation accuracy for each stream is computed by averaging the prediction of 5 data samples. The two streams are trained separately. For the final validation, 25 samples are used for each stream instead of 5.

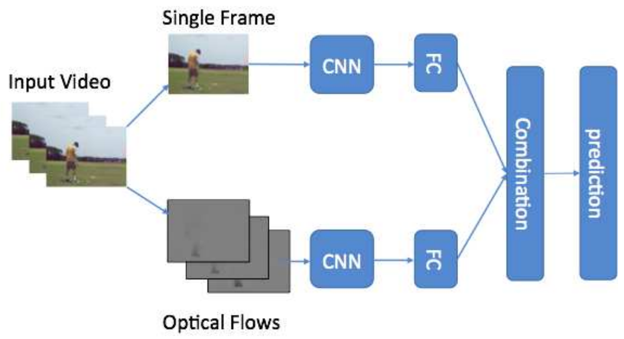

Fig. 1: Two-stream training architecture

Both CNNs in our study are based on ResNet152 [8] with some modifications to the first and last layers. The weights in ResNet152 are pretrained on the ImageNet dataset [20]. For the RGB stream, the last layer of ResNet152 is modified for 101 output classes instead of 1000. For the flow stream, in addition to the modification to the last layer, the input layer is changed to 20 channels instead of the original 3 channels. Simple averaging is used to combine the predictions from the two streams.

The baseline implementation adopts a data augmentation scheme described in [25]. The video frames are first scaled to size 256×340. During training, a multi-scaled crop (random crop whose width and height are randomly chosen from $256, 224, 192, 168$, followed by a re-size of the cropped region to 224×224) and a random horizontal flip are applied to the input. During validation, a center crop of 224×224 without scaling is applied to the input.

The baseline single-GPU training uses an initial learning rate of 0.01, and the learning rate is reduced by a factor of 2 every 50 epochs. The maximum validation accuracies achieved after 500 epochs with the RGB stream and the flow stream are 85.04% and 84.5% respectively. The combined two-stream validation accuracy is 91.3%. On a single GPU the RGB stream takes around 12 hours to train, and the flow stream takes more than two days to train.

Many machine learning practitioners are primarily concerned with achieving the best possible accuracy. Thus it is common to manually tune the hyper-parameters for best accuracy. The hyper-parameters chosen, especially the schedule for reducing the learning rate, can have significant impact on training time. We discuss parallel speedups in detail in Section VI.

III. Adaptive-batchsize model averaging with sparse communication

Parallel and distributed implementation of SGD abound in the literature (e.g., see [18], [5], [15], [28], [4], [13], [14], [29]). We introduce our adaptive-batchsize model averaging algorithm, and analyze its convergence behavior in comparison to existing ASGD implementations. Our analysis helps us set the right communication frequency for convergence.

The AAVG algorithm solves the non-convex optimization problem formulated as $\min_{\mathbf{w} \in \mathcal{X}} F(\mathbf{w})$ where the objective function $F : \mathbb{R}^m \rightarrow \mathbb{R}$ is continuously differentiable but not necessarily convex over $\mathcal{X}$, and $\mathcal{X} \subset \mathbb{R}^m$ is a nonempty open subset. F can be understood as either the expected risk $F(\mathbf{w}) = \mathbb{E} f(\mathbf{w}; \xi)$ or the empirical risk $F(\mathbf{w}) = n^{-1} \sum_{i=1}^{n} f_i(\mathbf{w})$.

A. The AAVG algorithm

The formal description of AAVG is given in Alg. 1. AAVG takes the following inputs: the training dataset $\mathcal{T}$, the validation dataset $\mathcal{V}$, the averaging interval K, the initial batchsize B, the initial learning rate γ, the validation interval m, the number of learners P, the number of steps N, and parameters b_1 and b_2 that are used to adapt batchsize. With AAVG, P learners run stochastic gradient descent concurrently (lines 4 to 11), and average their parameters every K steps (line 12). Every m steps, the algorithm evaluates validation accuracy a on the validation dataset $\mathcal{V}$ and adapts batchsize according to the validation result (lines 13 to 21).

When K=1, AAVG with constant batchsize is equivalent to hard-sync parallelization of SGD [6]. Its convergence behavior is exactly the same as SGD with batchsize PB. $K = 1$ incurs high overhead due to frequent communication. When deploying AAVG, the right K plays a critical role in the convergence speed relative to the number of data samples processed and communication overhead.

Algorithm 1 AAVG ($\mathcal{T}$, $\mathcal{V}$, K, B, m, γ, P, N, b_1, b_2)

```
1: initialize $\widetilde{\mathbf{w}}_1$
2: $a^* \leftarrow 0$, $B_0 \leftarrow B$, $\gamma_0 \leftarrow \gamma$
3: for $n = 1, ..., N$ do
4:     $B_n \leftarrow B_{n-1}$, $\gamma_n \leftarrow \gamma_{n-1}$
5:     for $j = 1, \ldots, P$ in parallel do
6:         set $\mathbf{w}_n^j = \widetilde{\mathbf{w}}_n$
7:         for $k = 1, ..., K$ do
8:             randomly sample a mini-batch of size $B_n$ from $\mathcal{T}$
9:             $\mathbf{w}_{n+k}^j \leftarrow \mathbf{w}_{n+k-1}^j - \frac{\gamma_n}{B_n} \sum_{s=1}^{B_n} \nabla F(\mathbf{w}_{n+k-1}^j; \xi_{k,s}^j)$
10:        end for
11:    end for
12:    $\widetilde{\mathbf{w}}_{n+1} \leftarrow \frac{1}{P} \sum_{j=1}^{P} \mathbf{w}_{n+K}^j$
13:    if $n\%m = 0$ then
14:        $a \leftarrow evaluate(\widetilde{\mathbf{w}}_{n+1}, \mathcal{V})$
15:        if $a < a^* \cdot b_1$ then
16:            $B_n \leftarrow B_n \cdot b_2$
17:        end if
18:        if $a > a^*$ then
19:            $a^* \leftarrow a$
20:        end if
21:    end if
22: end for
```

Let $\|\cdot\|_2$ denote the ℓ_2 norm of a vector in $\mathbb{R}^d$, and $\langle\cdot\rangle$ denote the general inner product in $\mathbb{R}^d$ We analyze the convergence behavior of AAVG with fixed stepsize and batchsize under the following assumptions.

Assumption 1. The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and the gradient function of F is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.

$$\|\nabla F(\mathbf{w}) - \nabla F(\widetilde{\mathbf{w}})\|_2 \leq L\|\mathbf{w} - \widetilde{\mathbf{w}}\|_2$$

for all $\mathbf{w}$, $\widetilde{\mathbf{w}} \in \mathbb{R}^d$.

Assumption 2. The sequence of iterates $\{\mathbf{w}_j\}$ is contained in an open set over which F is bounded below by a scalar F^*.

Assumption 3. For any fixed parameter $\mathbf{w}$, the stochastic gradient $\nabla F(\mathbf{w}; \xi)$ is an unbiased estimator of the true gradient corresponding to the parameter $\mathbf{w}$, namely,

$$\mathbb{E}_\xi \nabla F(\mathbf{w}; \xi) = \nabla F(\mathbf{w}).$$

Assumption 4. There exist scalars $M \geq 0$ and $M_V \geq 0$ such that, for all $k \in \mathbb{N}$,

$$\mathbb{E}_\xi\|\nabla F(\mathbf{w}; \xi)\|_2^2 - \|\mathbb{E}_\xi \nabla F(\mathbf{w}; \xi)\|_2^2 \leq M + M_V\|\nabla F(\mathbf{w})\|_2^2.$$

The average expected squared gradient norm $\mathbb{E}\frac{1}{N}\sum_{n=1}^{N}\|\nabla F(\widetilde{\mathbf{w}}_n)\|^2$ has been used as a metric for measuring convergence. For example, with ASGD, after K updates to the parameter server, the bound on $\mathbb{E}\frac{1}{N}\sum_{n=1}^{N}\|\nabla F(\widetilde{\mathbf{w}}_n)\|^2$ is [14]

$$\mathbb{E}\frac{1}{N}\sum_{n=1}^{N}\|\nabla F(\widetilde{\mathbf{w}}_n)\|^2 \quad \leq \quad \left[\frac{C_0(F(\widetilde{\mathbf{w}}_1) - F^*)}{N\bar{\gamma}} + \frac{C_1 L^2 \bar{\gamma}^2 M^2 P}{2\bar{B}}\right]$$

where C_0 and C_1 are constants independent of P. As P increases, the convergence rate guarantee becomes larger (worse).

The bound on the expected average squared gradient norm as a convergence guarantee for AAVG with fixed stepsize and batchsize is given in Theorem 1.

Theorem 1. *With a fixed stepsize $\gamma_n = \bar{\gamma}$ and a fixed batchsize $B_n = \bar{B}$ for all $n \in \mathbb{N}$ satisfying*

$$\bar{B} \geq L\bar{\gamma}M_G(L\bar{\gamma}K + \frac{1}{P}).$$

the expected average squared gradient norm of F for AAVG satisfy the following bounds for all $N \in \mathbb{N}$:

$$\mathbb{E}\frac{1}{N}\sum_{n=1}^{N}\|\nabla F(\widetilde{\mathbf{w}}_n)\|^2 \quad \leq \left[\frac{2(F(\widetilde{\mathbf{w}}_1) - F^*)}{N(K+1)\bar{\gamma}} + \frac{L\bar{\gamma}M}{\bar{B}}\left(\frac{1}{P} + \frac{LK\bar{\gamma}}{2}\right)\right] \cdot \left(\frac{PL\bar{\gamma}K + 1}{PL\bar{\gamma}(K-1) + 1}\right)$$

Proof. Omitted for brevity. □

Theorem 1 provides some guidelines for deploying AAVG. The $\frac{2(F(\widetilde{\mathbf{w}}_1) - F^*)}{N(K+1)\bar{\gamma}}$ term in the bound is primarily impacted by N and goes to zero as N increases. Increasing K and γ decreases this term. This in part explains why model averaging may be effective with infrequent synchronizations (large K) in our experiments with AAVG. The $\frac{L\bar{\gamma}M}{\bar{B}}\left(\frac{1}{P} + \frac{LK\bar{\gamma}}{2}\right)$ term is independent of N, and is impacted by the choice of P, B, and K. It favors large B, large P, and small γ. If the total amount of data samples collectively processed by all learners is kept constant, this term forces some intricate relationships on the parameters. For example, when p increases, the constant data samples constraint forces N to decrease and thus increases the $\frac{2(F(\widetilde{\mathbf{w}}_1) - F^*)}{N(K+1)\bar{\gamma}}$ term. Increasing the batchsize B seems beneficial without side effects. Yet N decreases when B increases. Increasing B reduces $\frac{L\bar{\gamma}M}{\bar{B}}\left(\frac{1}{P} + \frac{LK\bar{\gamma}}{2}\right)$ but also decreases N.

The next theorem establishes that frequent synchronization (e.g., $K = 1$) is not necessarily beneficial to convergence.

Theorem 2. *Let $S = N * K$ be a constant. Suppose AAVG is run with a fixed stepsize $\gamma_n = \bar{\gamma}$ and a fixed batchsize $B_n = \bar{B}$ for all $n \in \mathbb{N}$ satisfying*

$$\frac{LK\bar{\gamma}}{2} \geq \frac{1}{P}, \text{ and } \bar{B} \geq \frac{3L^2K\bar{\gamma}^2M_G}{2}.$$

If $1 \geq PL\bar{\gamma}$, then the optimal K to choose is $K_{opt} = 1$. If $1 < PL\bar{\gamma}$ then there is a unique optimal K value satisfying $K_{opt} > 1$.

Proof. Under the assumptions $S = N{*}K$ and $LK\bar{\gamma}/2 \geq 1/P$, we can rewrite the bound on the convergence guarantee as

$$\begin{aligned}&\mathbb{E}\frac{1}{N}\sum_{n=1}^{N}\|\nabla F(\widetilde{\mathbf{w}}_n)\|_2^2 \\ &\leq \Big[\frac{2(F(\widetilde{\mathbf{w}}_1) - F^*)K}{S\bar{\gamma}(K+1)} + \frac{L^2\bar{\gamma}^2MK}{\bar{B}}\Big]\Big(\frac{PL\bar{\gamma}K+1}{PL\bar{\gamma}(K-1)+1}\Big).\end{aligned} \quad (1)$$

To move on, we set

$$B(K) := \Big(\frac{\alpha K}{K+1} + \beta K\Big)\Big(\frac{aK+1}{a(K-1)+1}\Big)$$

where

$$\alpha = \frac{2(F(\widetilde{\mathbf{w}}_1) - F^*)}{S\bar{\gamma}}, \ \beta = \frac{L^2\bar{\gamma}^2M}{\bar{B}}, \text{ and } a = PL\bar{\gamma}.$$

To figure out the monotonicity of $B(K)$, one needs to solve $B'(K) = 0$ which is equivalent to solving a quartic equation. Instead of solving it explicitly, we investigate the solution analytically. The bound on the convergence guarantee can be decomposed into two parts:

$$f(K) = \frac{\alpha K}{K+1} * \Big(\frac{aK+1}{a(K-1)+1}\Big), g(K) = \beta K * \Big(\frac{aK+1}{a(K-1)+1}\Big). \quad (2)$$

One can easily get that

$$f'(K) = \frac{\alpha(2aK+1)(1-a)}{(K+1)^2(a(K-1)+1)^2}$$

which indicates that if $1 > PL\bar{\gamma}$, then it is increasing with respect to K. Otherwise, it is decreasing. On the other hand, we have

$$g'(K) = \frac{\beta(a^2K^2 + 2a(1-a)K + 1 - a)}{(a(K-1)+1)^2}.$$

By analyzing the quadratic equation with respect to K, we conclude: If $1 \geq a$, then $g(K)$ is increasing with respect to K; if $a > 1$, then $g(K)$ obtains its minimum when

$$\widehat{K}_{opt} = \frac{a - 1 + \sqrt{a(a-1)}}{a}.$$

Using the condition $a > 1$ and $K \geq 1$, one can easily check that $\widehat{K}_{opt} \in (0, 2)$.

If $1 \geq PL\bar{\gamma}$, the bound is always an increasing function with respect to K. Together with the constraint $LK\bar{\gamma}/2 \geq 1/P$, we have $K_{opt} = \frac{2}{LP\bar{\gamma}} \geq 2$. Thus $K = 1$ is an optimal choice in this case. However, when $1 < PL\bar{\gamma}$, the situation becomes complicated. As we discussed above, $f(K)$ is a decreasing function of K with a sublinear decreasing rate. $g(K)$ is decreasing at first and then grows almost linearly as K is increasing. If $(F(\widetilde{\mathbf{w}}_1) - F^*)/(S\bar{\gamma})$ is large enough such that $f(K)$ is dominant, the bound (1) decreases if K increases. Meanwhile, a large enough K can make $g(K)$ dominant, which eventually makes the bound (1) increase as K increases. As a consequence, when $1 < PL\bar{\gamma}$, the function value $B(K)$ can be decreasing at the very beginning and increasing almost linearly later on. Thus the optimal K value of $B(K)$ in this case satisfies $K_{opt} \geq 1$. □

Theorem 2 establishes a foundation for explicitly managing convergence and communication cost that are both impacted by K. If the optimal K for convergence is not 1, as a lucky coincidence, $K > 1$ also reduces communication overhead on a cluster.

We experiment with different K values and observe their impact on the validation accuracy. The results are shown in Figures 2 and 3. In the figures the evolution of validation accuracy is plotted.

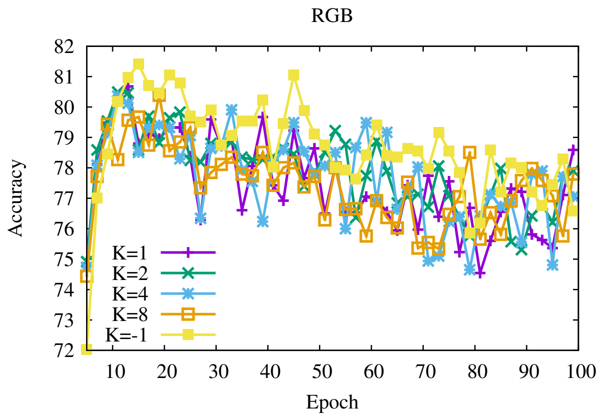

Fig. 2: AAVG performance relative to K. For clarity, validation accuracies are shown for every other epoch. $K = -1$ is for averaging the models once after an entire epoch

Fig. 2 shows the impact of K on accuracy for the RGB stream. We experiment with K=1, 2, 4, 8. We also experiment with model averaging after an entire epoch is done, represented by the K=-1 line in the figure. The number of batches each learner processes in an epoch depends on B and P. With $B = 32$ and $P = 16$, the number of batches processed by a learner in an epoch is bout 18. The optimizer we use is *Adam*. For K=1, 2, 4, and 8 there are no clear winners in terms of the validation accuracy achieved after each epoch. K=-1 achieves the highest validation accuracy after 100 epochs among all runs.

Fig. 3 shows the validation accuracies for the flow stream for K=1, 2, 4, 8, and -1. AAVG is run for 200 epochs. We use batchsize 64, and the optimizer is *Adam*. The plots for $K = 1$ and $K = 2$ almost completely overlap with each other. Again $K = -1$ yields the best validation accuracy among all runs.

For both streams, our experiments show that K_{opt} can be fairly large. Very sparse communication among the learners actually helps convergence. We use $K = -1$ for all our AAVG implementations. With $K = -1$, computation dominates

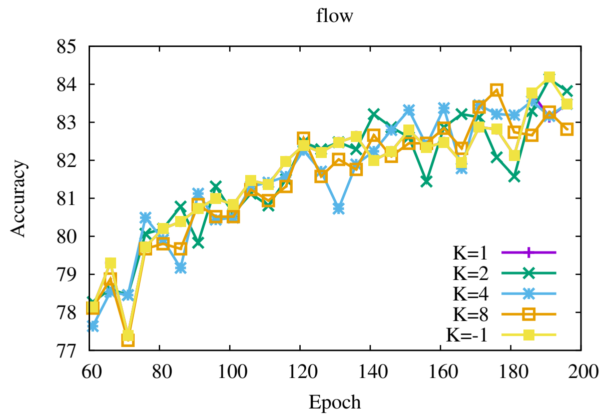

Fig. 3: AAVG performance relative to K. For clarity, validation accuracies are shown for every 5th epoch

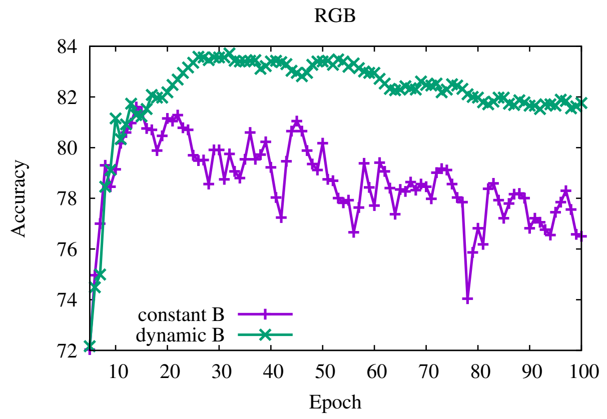

Fig. 4: Validation accuracy with adaptive batchsize

communication. Ignoring I/O overhead (determined by the storage type and file system), AAVG with $K = -1$ achieves near linear speedup with the UCF101 dataset.

AAVG with constant batchsize achieves 81.6% validation accuracy for the RGB stream , and 84.1% validation accuracy for the flow stream.

B. Dynamically adapting batchsizes

The bound on the convergence guarantee in Theorem 1 decreases as the learning rate decreases and/or the batch size increases. The Adam optimizer adaptively scales the learning rate for each individual gradient component. We consider dynamically adapting batchsize for faster convergence.

When the objective function $F(w)$ is strongly convex, new developments for efficient SGD computation focus on methods that construct increasingly accurate estimates of the gradient using larger sample batches B_n as iterations progress, i.e. $B_n \to \infty$ as $n \to \infty$. The intuition behind this ([16]) is that while gradient estimates from small batches are sufficient at the beginning of a procedure and make rapid progress, SGD with increasing batchsize should eventually start to resemble deterministic algorithms for strongly convex problems via larger B_n batch sampling (When F is the empirical loss function, a natural upper bound on B_n is the full dataset). The batchsize increase schedule can be carefully tuned such that the overall method enjoys convergence that is linear in the total computational effort (function evaluations at samples) expended (see [2] section 5.1). If $M_n = \sum_n B_n$ represents total effort up to iteration n, then the optimality gap $(F(w_n) - F^*) = O(M_n^{-1})$, and so one needs to expend $O(\epsilon^{-1})$ effort to reduce the optimality gap to ϵ. No such guarantee exists for general non-convex F formulations.

AAVG increases the batchsize by a factor of b_2 whenever the validation accuracy does not improve by a margin of b_1. In our implementation, we use $b_1 = 1$ and $b_2 = 2$. We keep the maximum batchsize to 576.

Fig. 4 shows the impact of dynamically adapting batchsize on the validation accuracy for the RGB stream. Adaptive B (starting at 32) brings significant improvement of accuracy over constant B=32. The best validation accuracy achieved with constant B is at 81.6 %, while the best validation accuracy achieved with adaptive B is at 83.7%. In the figure batchsize adaption first occurs after 5 epochs. After 20 epochs, the validation accuracy with batchsize adaptation is much higher than without.

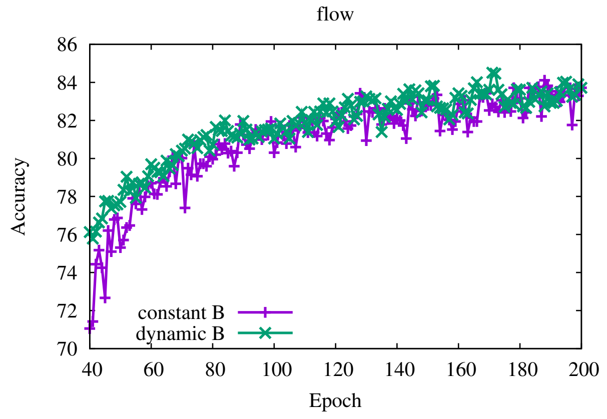

Fig. 5: Validation accuracy with adaptive batchsize

The impact of dynamically adapting batchsize on the validation accuracy for the flow stream can be found in Fig. 5. Again we notice adaptive B brings improvement of accuracy over constant B, although the improvement is not as dramatic as that for the RGB stream. The best validation accuracy achieved with constant B is at 84.1 %, while the best validation accuracy achieved with adaptive B is at 84.5%.

IV. Customizing the Optimizer for AAVG

AAVG averages the weights from all learners after each K steps and adapts the batchsize. To avoid manually tuning the learning rates, we use the *Adam* optimizer to manage them n our experiments. *Adam* maintains two quantities, m, the weighted average of historical gradients, and v, the weighted average of the historical squared gradients. The learning rate of each component in the gradient is scaled by $\frac{m}{\sqrt{v}}$. From the perspective of each Adam optimizer, model averaging disrupts its internal state, that is, m and v, as the model weights have changed. We adjust m and v in the Adam optimizer for AAVG.

The ResNet model used in our two-stream training employs batch normalization [9]. Batch normalization has been shown to significantly boost neural network performance. During validation batch normalization maintains a running mean and variance of the input for the specific layer and rescaling these inputs according to the mean and variance. After averaging in AAVG, the current mean and variance may no longer be a good fit for the next batch, and need to be adjusted.

Alg. 2 implements the customization of *Adam* and batch normalization for AAVG. Alg. 2 takes 4 inputs, $\{m_j\}$, $\{v_j\}$, $\{m_j^b\}$, and $\{v_j^b\}$, $1 \le j \le P$. Among them, $\{m_j\}$ and $\{v_j\}$ are the internal states for the Adam optimizer. They are first summed in reductions (line 2) and each optimizer updates its local m_j and v_j as the average (line 5).

After model averaging, the running mean and variance for batch normalization at each learner are re-computed by combining the mean and variance from other learners (line 6).

Algorithm 2 Adjust-optimizer($\{m_j\}$, $\{v_j\}$, $\{m_j^b\}$, $\{v_j^b\}$, P)

```
1: $z_j \leftarrow v_j^b + (m_j^b)^2$
2: set $m = \sum_j^P m_j$, $v = \sum_j^P v_j$
3: set $m^b = \sum_j^P m_j^b$, set $z = \sum_j^P z_j$
4: for $j = 1, \ldots, P$ in parallel do
5:     $m_j \leftarrow m/P$, $v_j \leftarrow v/P$
6:     $m_j^b \leftarrow m^b/P$, $v_j^b \leftarrow z/P - (m_j^b)^2$
7: end for
```

For the RGB stream the impact of customizing the optimizer for AAVG on the validation accuracy is illustrated in Fig. 6. In the figure, AAVG with customized *Adam* consistently outperforms AAVG after 20 epochs. The highest accuracy achieved with optimizer customization is 85.06 %, while the highest accuracy achieved with AAVG is 83.7%. It takes AAVG with customized *Adam* 61 minutes to train the RGB stream with 16 GPUs, while it takes the base-line single-GPU SGD implementation 2067 minutes to train to achieve similar validation accuracy.

Fig. 7 shows for the flow stream the impact of customizing the optimizer for AAVG on the validation accuracy. In the figure, the highest accuracy achieved by AAVG with customized *Adam* is 84.7 %, while the highest accuracy achieved with AAVG is 84.5%. It takes AAVG with customized *Adam* 439 minutes to train the flow stream on 16 GPUs. It takes a few days for the single-GPU implementation to train the flow stream.

The accuracy after combining results from the two streams is 91.4%.

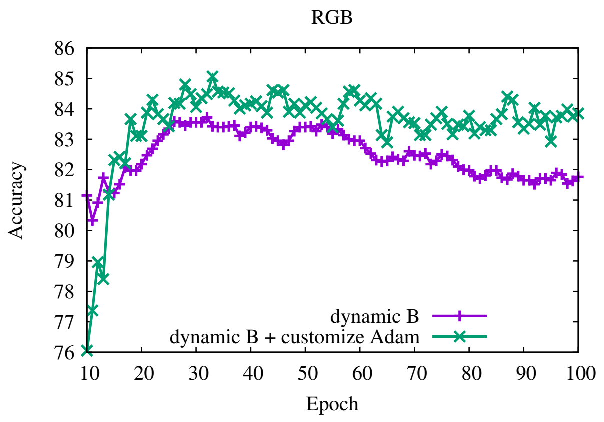

Fig. 6: Improvement with adjusting the optimizer for AAVG

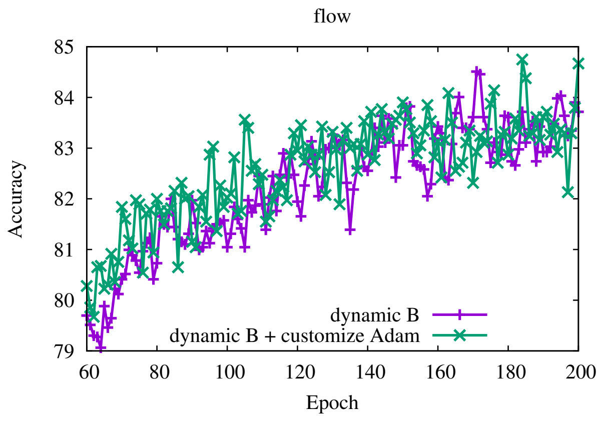

Fig. 7: Improvement with adjusting the optimizer for AAVG

V. Transfer learning for training the flow stream

Two factors contribute to the slower training of the flow stream compared to the RGB stream. The first is that the pretrained ResNet was trained with RGB images not flow inputs. The second is that the input layer of the modified ResNet used in the flow stream has significantly more channels and thus more weights to train.

To further accelerate training for the flow stream, one option is to use more GPUs. This is not always attractive as it demands more computing resources. In addition, our analysis of the convergence guarantee in Theorem 1 indicates convergence challenges with large P.

Instead of using ResNet with weights trained on ImageNet, we use the network trained for the RGB stream. For the input layer, we populate the filter weights for each channel with the mean of the weights of the 3 RGB channels in the ResNet for the RGB stream.

Figure 8 compares the convergence behavior of AAVG (with customized *Adam*) and AAVG with transfer learning in addition to customized *Adam* for the flow stream. In the figure, the validation accuracy with transfer learning is significantly higher than AAVG without transfer learning for the first 100 epochs. Eventually the highest validation accuracies achieved by the two approaches are similar, but with transfer learning the highest accuracy is achieved at around epoch 150, while without transfer learning the highest accuracy is achieved at around epoch 185.

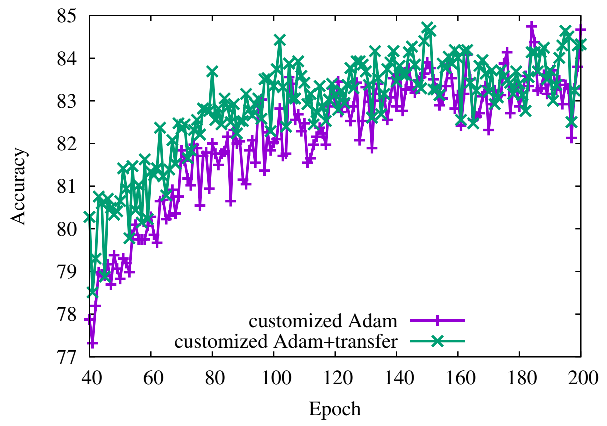

Fig. 8: Impact of transfer learning on the flow stream

In addition to accelerating the training of the flow stream, the accuracy from combining the two streams is also improved with transfer learning. Tab. I shows the accuracies achieved by the four AAVG implementations with the RGB stream, the flow stream, and the combined two streams for the UCF101 and the HMDB51 datasets. In the table, $AAVG_1$ implements $AAVG$ with constant batchsize; $AAVG_2$ implements $AAVG$ with adaptive batchsize; $AAVG_3$ implements AAVG with both adaptive batchsize and customized optimizer; and $AAVG_4$ is $AAVG_3$ with weights transferring from the RGB stream to the flow stream. For the HMDB51 dataset, we show only results for $AAVG_3$ and $AAVG_4$ due to limited machine time available on the cluster to run experiments. For the UCF101 dataset, from $AAVG_1$ to $AAVG_4$, the combined two-stream accuracy consistently increases. Note that the RGB stream of $AAVG_4$ is exactly the same as the RGB stream of $AAVG_3$. Although transfer learning does not improve the validation accuracy of the flow stream, it does significantly increases the accuracy of the two streams combined.

	UCF101			HMDB51		
	RGB	Flow	2stream	RGB	Flow	2stream
$AAVG_1$	81.6	84.1	88.3	-	-	-
$AAVG_2$	83.7	84.5	89.7	-	-	-
$AAVG_3$	85.6	84.7	91.4	61.4	55.9	67.0
$AAVG_4$	85.6	84.7	93.04	61.4	56.0	67.9

TABLE I: Comparison of different AAVG implementations

VI. Discussion on speedups

In training deep neural networks, the additional accuracy dimension makes comparing speedup results less straightforward. It is extremely rare that training with different implementations results in the same model with the same training/validation accuracy. Moreover, a slight increase in accuracy may require many epochs of additional training.

AAVG achieves superlinear speedup over the base-line single-GPU implementation for similar accuracies achieved due in part to the fact that the single-GPU SGD implementation runs for more epochs. Note that AAVG uses Adam as the optimizer. It is entirely possible that extensive hyper-parameter search may yield a better learning rate adaptation schedule with which SGD may converge with fewer epochs. Yet hyper-parameter search itself is extremely time consuming. As AAVG uses Adam, it does not rely on hyper-parameter search to find the right learning rate adaptation schedule. In our analysis of AAVG, we establish that sparse communication actually results in better convergence behavior. Thus ignoring I/O overhead, we expect close to linear speedup if we compare the time spent on one epoch by AAVG and the single-GPU SGD implementation. The performance of the algorithm in terms of both accuracy achieved and time spent on training depends almost entirely on its convergence property.

It is therefore interesting to compare the convergence behavior of AAVG with single-GPU *Adam*. We run AAVG with 16 GPUs and *Adam* with a single GPU. The evolution of validation accuracies are shown in Fig. 9.

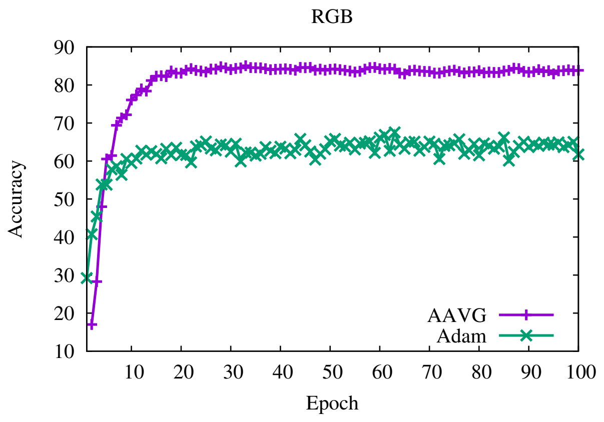

Fig. 9: AAVG vs. Adam

In Fig. 9, in the first a few epochs, *Adam* reaches higher validation accuracy than AAVG. After 5 epochs, AAVG catches up, and maintains higher validation accuracy than *Adam*. The best accuracy achieved with *Adam* is 67.5%.

VII. Conclusion and future work

Training deep neural networks for action recognition is time consuming. AAVG is an efficient distributed training algorithm with adaptive batchsize that explicitly manages the impact of model averaging frequency on both convergence and communication overhead. According to our analysis and experimental evaluation, AAVG with very sparse synchronization (i.e. once per epoch), shows very good convergence behavior. As a happy conincidence, the communication overhead is very low. Customizing the *Adam* optimizer and its batch normalization for AAVG improves training time and accuracy. For the two-stream action recognition approach, we employ transfer-learning from the spatial stream to the temporal stream that further reduces training time for the flow stream with improved validation accuracy.

In its best convergence results, AAVG shows super-linear speedups on 16 GPUs over the base-line single-GPU SGD implementation while improving accuracy. For the UCF101 dataset, our validation accuracy is 93.1%. As far as we know, this is the highest accuracy achieved with the two-stream approach that does not involve 3D convolution or pretraining on much larger datasets. For the HMDB51 dataset, AAVG achieves a validation accuracy of 67.9%.

In our future work, we plan to evaluate our algorithm on even larger datasets.

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(1):281–305, February 2012.

[2] L Bottou, F. E. Curtis, and J Nocedal. Optimization methods for large-scale machine learning. 2017.

[3] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[4] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.

[5] J. Dean, G. Corrado, R. Monga, and et al. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

[6] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[7] P. Goyal, P. Dollár, R.B. Girshick, and et al. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

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

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[10] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[11] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[12] H. Kuehne, H. Jhuang, R. Stiefelhagen, and T. Serre. HMDB51: A large video database for human motion recognition. In W.E. Nagel, D.H. Kröner, and M.M. Resch, editors, *High Performance Computing in Science and Engineering '12*, pages 571–582, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[13] M. Li, D. G. Andersen, J. W. Park, and et al. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[14] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[15] T. Paine, H. Jin, J. Yang, and et al. Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*, 2013.

[16] R Pasupathy, P Glynn, S Ghosh, and F Hashemi. On sampling rates in simulation-based recursions. *SIAM Journal of Optimization*, 2017. in revisions.

[17] A. Paszke, S. Gross, S. Chintala, and et al. Automatic differentiation in pytorch. 2017.

[18] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[19] J. Redmon, S.K. Divvala, R.B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[20] O. Russakovsky, J. Deng, H. Su, and et al. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[21] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[23] K. Soomro, A.R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[24] C. Szegedy, W. Liu, Y. Jia, and et al. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[25] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao. Towards good practices for very deep two-stream convnets. *CoRR*, abs/1507.02159, 2015.

[26] S. Xie, C. Sun, J. Huang, and et al. Rethinking spatiotemporal feature learning for video understanding. *CoRR*, abs/1712.04851, 2017.

[27] Y. You, I. Gitman, and B. Ginsburg. Scaling SGD batch size to 32k for imagenet training. *CoRR*, abs/1708.03888, 2017.

[28] R. Zhang and J.T. Kwok. Asynchronous distributed admm for consensus optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pages 1701–1709, 2014.

[29] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 685–693, 2015.