2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

TeamNet: A Collaborative Inference Framework on the Edge

Yihao Fang
Dept. of Computing and Software
McMaster University
Hamilton, Canada
fangy5@mcmaster.ca

Ziyi Jin
Dept. of Computing and Software
McMaster University
Hamilton, Canada
jinz27@mcmaster.ca

Rong Zheng
Dept. of Computing and Software
McMaster University
Hamilton, Canada
rzheng@mcmaster.ca

***Abstract*—With significant increases in wireless link capacity, edge devices are more connected than ever, which makes possible forming artificial neural network (ANN) federations on the connected edge devices. Partition is the key to the success of distributed ANN inference while unsolved because of the unclear knowledge representation in most of the ANN models. We propose a novel partition approach (TeamNet) based on the psychologically-plausible competitive and selective learning schemes while evaluating its performance carefully with thorough comparisons to other existing distributed machine learning approaches. Our experiments demonstrate that TeamNet with sockets and transmission control protocol (TCP) significantly outperforms sophisticated message passing interface (MPI) approaches and the state-of-the-art mixture of experts (MoE) approaches. The response time of ANN inference is shortened by as much as 53% without compromising predictive accuracy. TeamNet is promising for having distributed ANN inference on connected edge devices and forming edge intelligence for future applications.**

***Index Terms*—Distributed Machine Learning, Edge Computing, Wireless Network, Mixture of Experts**

I. Introduction

Recent advances in the Internet of Things and wireless communications witness the arrival of a new computing paradigm, in which computation and data storage is distributed among multiple end-user devices and near-user edge devices. Thousands of distributed smart edge devices such as smart cameras and smart routers are connected seamlessly through protocols such as 5G, LTE, and WiFi, enabling distributed intelligence.

Deep neural networks have been being successfully applied to many domains such as computer vision and natural language processing. In a large-scale image recognition task, deep neural networks normally consume tens of gigabytes in RAM and significant computing power on GPUs and CPUs. Their resource consumption can be accommodated by a cloud environment, but typically cannot be handled by individual edge devices due to their limited processing power and memory. Take the processing power as an example. There are 256 NVIDIA CUDA cores on a Jetson TX2 board. In contrast, the Helios cluster hosted by Compute Canada, a consortium of high-performance computing data centers, has 120 NVIDIA K20 GPUs (with 2496 CUDA cores each) and 96 K80 GPUs (with 4992 CUDA cores each) – roughly 3042 times more cores than a Jetson TX2 board.

Shallow neural networks alone generally have poorer predictive accuracy compared to state-of-the-art (SOTA) deep neural networks, but they allow real-time prediction on the edge devices, which make them more suitable for edge computing. A question arises that *if it is possible to train and coordinate the inference of multiple shallow neural networks (each running on an edge device) to have comparable performance as or even outperform a single SOTA deep neural network in prediction*.

In this work, we propose a novel partition approach called *TeamNet* based on competitive and selective learning. TeamNet is inspired by phenomena in human society, where knowledge is naturally partitioned among people with each individual specializing in only one or a few subject domains. Analogous to the human society, shallow neural networks in TeamNet do not learn the knowledge of the entire dataset, instead, they only master one subset of it. During inference, knowing what they do know and what they know by estimating predictive uncertainties, TeamNet devices running shallow models aggregate their predictions to form final decisions distributively and collaboratively. Unlike existing computation partition approaches [1], [2] that decide where computation should be done for *pre-trained* deep neural network models, TeamNet is a fundamentally different approach to partition by training shallower models using the *similar but downsized architecture* of a given SOTA deep model. Both the number of shallower models and the SOTA deep model can be specified by users.

We have implemented TeamNet using TensorFlow and CUDA on two types of representative edge devices, i.e., Jetson TX2 and Raspberry Pi 3 Model B+. For comparison, we have also parallelized baseline models using the message passing interface (MPI) and implemented the SOTA mixture of experts (MoE) approaches. Experiments using MNIST and CIFAR-10 datasets show that TeamNet can shorten inference latency by as much as 53% compared to baseline neural networks without compromising the predictive accuracy. Therefore, TeamNet is promising for enabling distributed edge intelligence for compute-intensive applications.

The rest of the paper is organized as follows. Section II describes the related works. System architecture is presented

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00148

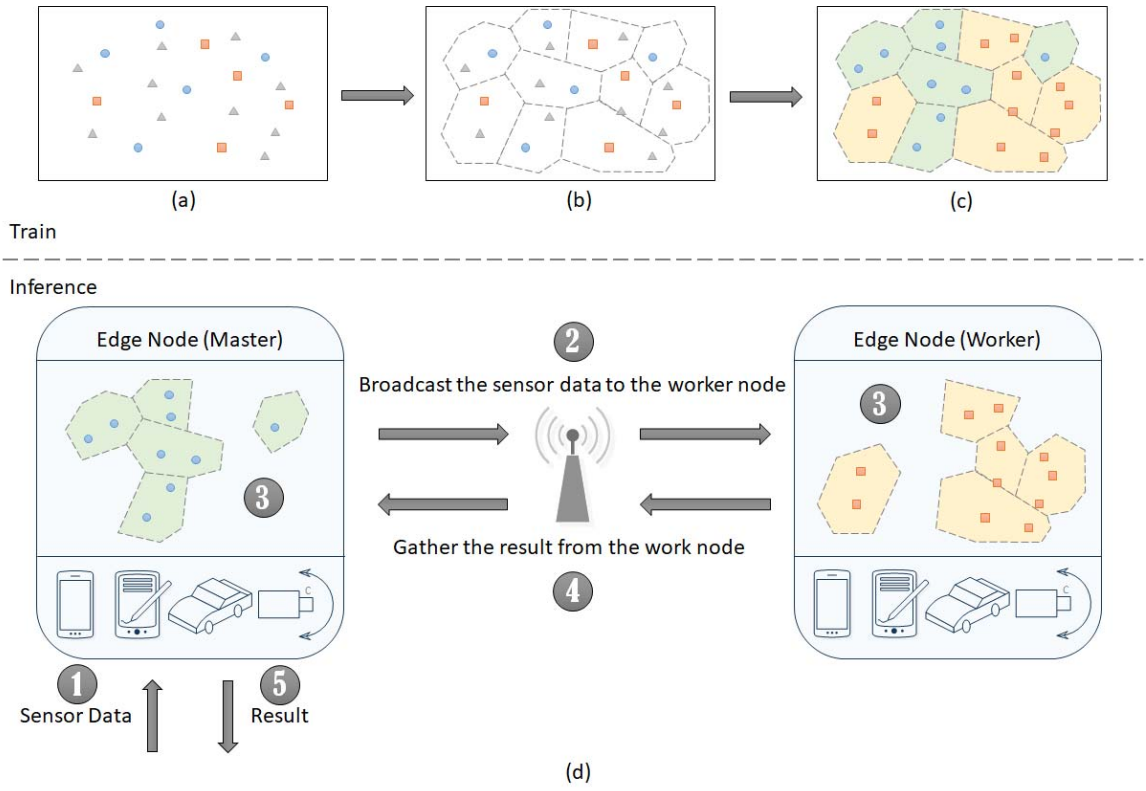


Fig. 1: TeamNet's training and inference are illustrated on two edge nodes: (a) Initially, each expert has very limited knowledge about the dataset but each is randomly more certain of some data than the others. (e.g. One expert is more certain of the square data points, while the other expert is more certain of the circle data points. However, neither of them has knowledge of the triangle data points.) (b) With this preference, it is more likely for each expert to select the more certain data to learn, respectively. (c) Gradually, with the help of gradient descent, each expert has learned its more certain data from the entire dataset. (d) At the inference stage, each expert is deployed to one edge node. When one edge node, called the master, receives the sensor data (Step 1), it broadcasts the data to the other edge node, called the worker (Step 2). Both then perform inference in parallel (Step 3). At the time the inference completes, the master gathers the result (with uncertainty) from the worker (Step 4). Then the master compares its uncertainty with the worker's, and select the one with the least uncertainty as the final result (Step 5). It is necessary to collaborate with the worker since neither of them is trained with all data.

in Section III. Section IV provides more details of TeamNet's training algorithms. Section V described TeamNet's inference. In Section VI, we present implementation details, experimental setups and results. Conclusions and future work are given in Section VII.

II. Related Works

In this section, we provide an overview of related works. There are two groups of work that are relevant, namely, computation partition and the mixture of experts.

Computation Partition of Neural Networks: Computation partition splits a known model represented as a data flow graph into two or more parts and executes them on edge and in the cloud [1]. For deep forward neural networks, the partition is generally performed between layers, which trade-offs extra data transfer time and less total computation time (due to offloading). Recently, two interesting variants of computation partition have been proposed. In [3], Ko et al. considered lossless and lossy compression of the output features of an intermediate layer before transmitting them to the cloud. In [4], Distributed Deep Neural Networks (DDNNs) was designed to perform fast and localized inference using shallow portions of a neural network on edge devices. Using an exit point after device inference, an output is classified locally. When multiple end devices presented, an aggregation method would gather all output from each end device in order to perform classification, where an exit was determined. If the classification could not be made due to low confidence, the task is escalated to a higher exit point (e.g. the edge exit) in the hierarchy until the last exit (the cloud exit). With multiple exit points, DDNNs can significantly reduce communication costs.

Computation partition approaches take a trained DNN model and divide the pipeline among devices with different processing capabilities. Therefore, how partitions can be made

is inherently constrained by the structure of the existing model. In contrast, in TeamNet, we train smaller and specialized expert models.

Mixture of Experts (MoE): Adaptive Mixtures of Local Experts [5] was proposed to combine multiple feed-forward network experts with an adaptive gating network (also feed-forward network). In this architecture, all experts receive the same input and have the same number of outputs. The gating network receives the same input as the expert networks' and outputs a stochastic switch with the probabilities (gate values) that the switch uses to select the outputs from experts. The mixture of experts is partitionable in nature. It is feasible to deploy experts and the gating network respectively to multiple edge devices.

Sparsely-Gated Mixture-of-Experts (SG-MoE) [6] was introduced to increase model capacity through a sparse combination of multiple neural network experts. Different from Jacobs's approach [5], the gating network uses noisy top-K gating that keeps only the top k gate values and set the rest gate values to zero.

Since MoE has been introduced more than two decades ago, it has been the topic of much research. Different MoE architectures have been proposed such as a hierarchical structure [7]–[9], infinite number of experts [10], and sequential increment of experts [11]. However, a majority of the approaches target specific kinds of expert models such as Gaussian, GP, SVM, etc and thus are not architecture independent. They fail to capture recent advances in DNN models. SG-MoE can work with different NN architecture but its training process is not optimized to have experts of comparable capacity. As will be demonstrated through experiments in Section VI, this leads to degraded inference performance.

III. System Architecture

During the training stage, TeamNet takes a neural network architecture, the number of experts K, and training data as input and produce K expert models by semantically partitioning the dataset's knowledge as illustrated in Figure 1. The number of parameters of the expert models is hyperparameters that can be tuned using grid search or other automated machine learning approaches. TeamNet can be thought of as a black box from the developers' point of view. As an example, we input to TeamNet the convolutional neural network (CNN) architecture with 26 hidden layers and then ask TeamNet to generate 4 expert models according to the CIFAR-10 dataset. The outputs of TeamNet are 4 CNN models each with 8 layers, and those models could collaborate with one another on connected edge devices.

At run-time, TeamNet expert models are deployed on edge devices connected through a wireless network. For ease of presentation, we assume each edge device only runs one model and each device has its own sensor input (e.g., visual or audio data). As illustrated in Figure 1, in Step 1, upon a sensing event, an edge device broadcasts the sensor data to all peer edge devices in the network (Step 2). In Step 3, all edge devices will execute their own local models in parallel and each produces an uncertainty measure. Finally, in Step 4 and 5, all the results are being gathered and the output with the least uncertainty will be taken as the final result. This last step can be done distributedly, e.g., using a leader election protocol, or done centrally by sending the results along with the uncertainty measures to a designated device.

IV. Training TeamNet

In this section, we present the details of the algorithm to train TeamNet. The key challenge is the "richer gets richer" phenomenon. Specifically, regardless of how the expert models are initialized, they are predisposed to *biases* – some expert is confident about more input data than the others. Without any correction to the initial bias, the expert that is confident about more input data will be trained with more data leading to skewed partitions of the training data. In the worst case, some experts are subject to little training data and learn nothing.

A. Overview

Consider K experts, each modeled as a function $f(x;\theta_i)$, parameterized by θ_i, $i = 1, 2, \ldots, K$. For multi-class classification problem with C categories, $f(x;\theta_i)$ can be viewed as the parameters of a multinomial distribution of C classes. Let $p(\hat{y} = c|x, \theta_i)$ be the predictive probability of output $c = 1, 2, \ldots, C$ for input $x \in X$ from Expert i. The predictive entropy of Expert i of input x is defined as,

$$H(\hat{y}|x, \theta_i) := -\sum_c p(\hat{y} = c|x, \theta_i) \log p(\hat{y} = c|x, \theta_i).$$

Predictive entropy is closely related to perplexity. The predictive entropy of a model reflects its "uncertainty" of a data instance drawn from the same distribution of the training data.

The primary objective of TeamNet training is to divide the input data $\mathcal{D}$ into K partitions $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$ such that for any $(x, y) \in \mathcal{D}_i$, Expert i can predict the correct label y and has the least predictive entropy for x among all experts with high likelihood. The secondary objective is to have $|\mathcal{D}_i| \approx \frac{|\mathcal{D}|}{K}, i = 1, 2, \ldots, K$, where $|\mathcal{D}|$ is the cardinality of set $\mathcal{D}$. Thus, TeamNet is both *localized* and *implicit*. *Localized* means that each expert specializes in a subset of the data, and *implicit* means that we do not partition the data explicitly but let the experts *compete* with one another to divide them (roughly equally). The rationale behind equal partitions of training data is to have experts of similar capacity. Otherwise, some experts may be under-fitted while others are over-fitted. One may argue equally divided training data does not imply equally partitioned models. This is indeed true especially in the case of unbalanced training data and will be considered in our future work. In this work, we assume the training data is balanced.

The neural network used to train the experts is given in Figure 2. All expert networks are initialized with random weights. Training is done in multiple epochs. In each epoch, the training data is first reshuffled and then divided into equal-sized batches. Each batch of data is given to all experts

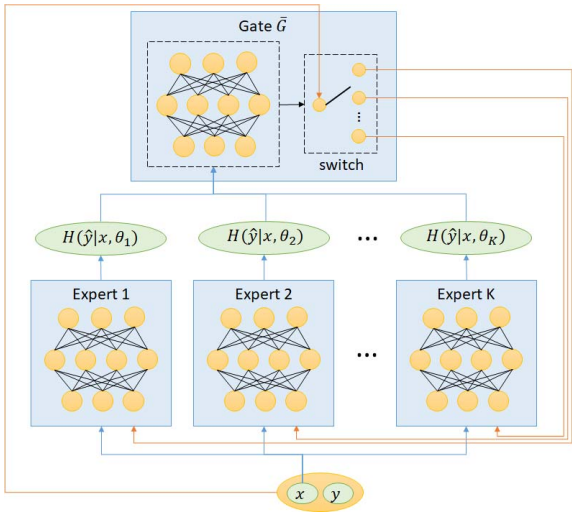


Fig. 2: At the training stage, the gate $\bar{G}$ decides which data example should be assigned to which expert to learn. Such a decision is based on the uncertainty estimation of all experts for the particular data example.

to evaluate their respective predictive entropy values. The calculated predictive entropy values serve as inputs to the gate network $\bar{G}$, which are then trained to assign data samples in the batch to different experts. After the assignment is done, the weights of the expert networks are updated using back-propagation with their own partitions. The procedure is summarized in Algorithm 1.

Algorithm 1 Training TeamNet

▷ Let r be the number of epochs
procedure TRAIN($\theta, \eta, \epsilon, K, r$)
  Shuffle the dataset
  Repeat the dataset for r times
  Get the first batch β
  while $|\beta| > 0$ **do**
    $\mathbf{H} \leftarrow H(\hat{y}|x,\theta_i), \forall i \in \{1...K\}, \forall x \in \beta$
    $\bar{G}^{\beta} \leftarrow$ GATE_TRAIN($\beta, \mathbf{H}, \eta, \epsilon, K$)
    EXPERT_TRAIN ($\beta, \bar{G}^{\beta}, \theta, \eta, K$)
    Get the next batch β
  end while
end procedure

B. Dynamic Gating

The gate function $\bar{G}$ splits the current batch into K partitions for training. When experts are biased, direct application of the $\arg\min$ gate would result in unevenly split data. As the experts become less biased, $\bar{G}$ should approach an $\arg\min$ gate, where the training sample is given to the model with the least uncertainty. Therefore, it is important to update $\bar{G}$ dynamically based on some measure of the biases of the experts.

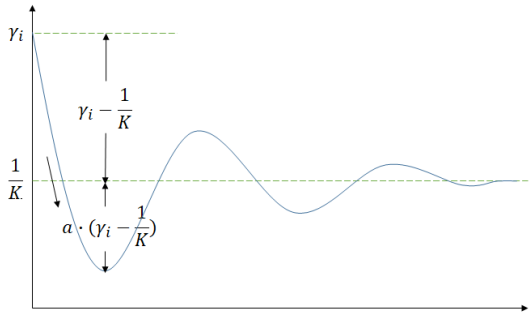


Fig. 3: The term γ_i is defined to reflect the bias from the past. Correcting it induces a momentum (momentum-caused displacement) $a \cdot (\gamma_i - \frac{1}{K})$. The momentum counteracts the bias which has been incurred by the past, and brings γ_i to the mean (set point) $\frac{1}{K}$ eventually.

We define $\bar{G}$ as:

$$\bar{G}(x,\delta) := \arg\min_i \delta_i \cdot H(\hat{y}|x,\theta_i), \tag{1}$$

where $\delta = (\delta_1, \delta_2, \ldots, \delta_K)$ are control variables to be determined. Let $G(x) := \arg\min_i H(\hat{y}|x,\theta_i)$ denote the assignment of the $\arg\min$ gate for input x.

Given the current batch β and δ, let

$$\gamma_i = \frac{\sum_{x\in\beta} \mathbb{1}_{G(x)=i}}{|\beta|}, \tag{2}$$

and

$$\bar{\gamma}_i(\delta) = \frac{\sum_{x\in\beta} \mathbb{1}_{\bar{G}(x,\delta)=i}}{|\beta|}, i = 1, 2, \ldots, K, \tag{3}$$

where $\mathbb{1}$ denotes the Kronecker delta function. In other words, γ_i and $\bar{\gamma}_i(\delta)$ are the portion of data in batch β that are assigned to Expert i according to gate G and $\bar{G}$, respectively. Under the assumption that the data in β is balanced, the bias in Expert i can then be characterized by $\gamma_i - \frac{1}{K}$.

Clearly, $\gamma_i - \frac{1}{K} \equiv 0$ for all experts implies the batch are divided equally by the expert models trained thus far. Otherwise, by adjusting δ_i's, we can "correct" the biases by assigning more data to the expert who would have received less training data according to gate G. Thus, we formulate the following optimization problem,

$$\min_{\delta} \sum_{i=1}^{K} \left| \bar{\gamma}_i(\delta) - \left(\frac{1}{K} - a \cdot (\gamma_i - \frac{1}{K}) \right) \right|, \tag{4}$$

where $0 < a < 1$ is a hyperparameter analogous to the gain of a proportional controller (Figure 3).

The optimization in (4) cannot be solved directly since there is no closed-form expression for $\bar{\gamma}_i$'s for arbitrary K. Instead, we represent the possible values of δ parametrically.

Let $E(x) = \frac{1}{K}\sum_{i=1}^{K} H(\hat{y}|x,\theta_i)$ be the mean entropy and $D(x) = \frac{1}{K}\sum_{i=1}^{K} |H(\hat{y}|x,\theta_i) - E(x)|$ be the absolute deviation

Algorithm 2 Finding Gate $\bar{G}$

▷ Let β be a mini-batch with size n
▷ Let $\mathbf{H}$ be the matrix of $H(\hat{y}|x,\theta_i)$ for all i in $1...K$ for all x in β
▷ Let Δ be the relative mean absolute derivation of $\mathbf{H}$
▷ Let $\bar{G}^\beta$ be the vector of $\bar{G}(x,\delta)$ for all x in β

procedure GATE_TRAIN($\beta, \mathbf{H}, \eta, \epsilon, K$)
  Calculate Δ for β
  Create $z \sim \mathcal{U}(-1,1)$ ▷ z is a latent variable
  Calculate γ_i for all i in $1...K$
  while $J > \epsilon$ **do**
    $\Phi \leftarrow W(z,\Theta)$ ▷ W transforms z into Φ
    $\delta \leftarrow 1 + \Phi \cdot \Delta$
    $\bar{G}^\beta \leftarrow \arg\min \delta \odot \mathbf{H}$
    Calculate $\bar{\gamma}_i(\delta)$ for all i in $1...K$
    $J \leftarrow \frac{1}{K}\sum_{i=1}^{K} \left|\bar{\gamma}_i(\delta) - \left(\frac{1}{K} - a \cdot (\gamma_i - \frac{1}{K})\right)\right|$
    $\Theta \leftarrow \Theta - \eta \nabla_\Theta J$ ▷ Θ descends w.r.t. gradients
  end while
  return $\bar{G}^\beta$
end procedure

of the entropy of instance x from K experts. We introduce Δ, the average normalized absolute deviation of batch β as

$$\Delta = \frac{1}{|\beta|}\sum_{x\in\beta}\frac{D(x)}{E(x)}.$$

In other words, Δ measures how "diverse" the uncertainty of different expert models is.

Next, we represent Δ by

$$\delta = 1 + \Delta \cdot W(z,\Theta),$$

where z is a vector of length N randomly drawn from a uniform distribution $\mathcal{U}(-1,1)$ and W is a multilayer perceptron (MLP) function parametrized by Θ. Essentially, we transform the problem of solving for δ into estimating parameters Θ. Doing so allows us to evaluate the gradients of Θ with respect to the objective function in (4) by approximating $\arg\min$ in $\bar{G}$ with a continuous function. The gradients are then used to update Θ until convergence.

To this end, we summarize the procedure to find $\bar{G}$ for batch β in Algorithm 2. The proof sketch of its convergence is given in Appendix A.

Soft Argument of the Minimum: The $\arg\min$ function in (1) is not differentiable. Gradients cannot be propagated back to Θ. In order to calculate the partial derivatives with respect to Θ, the $\arg\min$ function must be softened [12], defined by:

$$\text{soft}\arg\min(x) = \sum_i \frac{e^{-bx_i}}{\sum_j e^{-bx_j}} i \tag{5}$$

Meta-Estimator: An over-large b in (5) leads to an over-steep slope in gradient descent, while an over-small b leads to an over-gentle one. Neither benefits gradient propagation. To solve this problem, the scalar b is found by optimizing

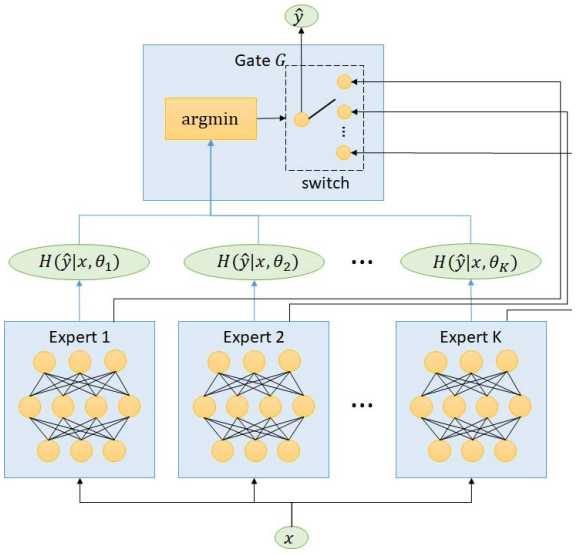


Fig. 4: At the inference stage, it is sufficient to have the $\arg\min$ function to determine which expert is least uncertain of the given data input. TeamNet's inference time is generally faster than other mixture of experts (MoE) approaches since its gate is much simpler.

a meta-estimator (neural network) with the objective that minimizes the distance between a small ϵ and the expectation of $\bar{G}(x,\delta)$ to its closest integer. The introduction of ϵ avoids from happening of an over-steep slope and smoothens the loss surface. Formally, the objective function is written as follows:

$$\min_b \left| \frac{\sum_{x\in\beta} \min_{i=1,2,...,K} \left|\bar{G}(x,\delta) - i\right|}{|\beta|} - \epsilon \right| \tag{6}$$

Kronecker Delta's Approximation: The Kronecker delta function $\mathbb{1}$ in (3) is not differentiable. Thus, instead, $\mathbb{1}$ is replaced by a differentiable approximation, denoted by:

$$\mathbb{1}_{\bar{G}(x,\delta)=i} \approx \tanh\left(c \cdot \text{ReLU}(0.5 - |\bar{G}(x,\delta) - i|)\right), \quad i = 1,2,\ldots,K, \tag{7}$$

where the rectified linear unit (ReLU) [13] is given by $\text{ReLU}(x) = \max(x,0)$, and the hyperbolic tangent function is defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

The differentiable approximation in (7) is the combination of shifting $(\bar{G}(x,\delta) - i)$, ramping (ReLU), and discretization (tanh). In our experiments in Section VI, the constant c is set to 10 to satisfy the needs of discretization while letting gradients propagate through. The constant 0.5 is added to approximate rounding.

C. Expert Trainer

The training process of experts is guided by gate $\bar{G}$. The gate decides which expert to learn a particular data example in each batch. A mini-batch β is then partitioned into K subsets $\beta_1, \beta_2, ..., \beta_K$. Gradients of Expert i (i in $\{1,...,K\}$)

(a) Accuracy (b) Inference Time (c) Memory (d) CPU Usage

Fig. 5: TeamNet's performance is evaluated on Raspberry Pi 3 Model B+ for handwritten digit recognition. With more experts in TeamNet, inference becomes faster, and memory and CPU consumption become smaller on the edge node. The accuracy is generally not compromised.

	Baseline	Double Nodes				Quadro Nodes			
		TeamNet	MPI-Matrix	SG-MoE-G	SG-MoE-M	TeamNet	MPI-Matrix	SG-MoE-G	SG-MoE-M
Accuracy (%)	98.8	**98.7**	**98.7**	98.6	98.6	**98.7**	**98.7**	98.5	98.5
Inference Time (ms)	3.4	**3.2**	108.2	5.9	6.9	**3.3**	189.0	4.1	10.3
Memory Usage (%)	8.2	6.0	8.55	7.8	9.1	4.4	6.78	6.1	6.6
CPU Usage (%)	55.3	30.7	31.9	21.9	54.3	21.2	34.6	11.5	48.2

(a) Jetson TX2 CPU only

	Baseline	Double Nodes				Quadro Nodes			
		TeamNet	MPI-Matrix	SG-MoE-G	SG-MoE-M	TeamNet	MPI-Matrix	SG-MoE-G	SG-MoE-M
Accuracy (%)	98.8	**98.8**	**98.8**	98.7	98.6	98.7	**98.8**	98.5	98.5
Inference Time (ms)	0.3	**1.5**	104.8	5.8	3.2	**2.6**	187.7	4.5	6.9
Memory Usage (%)	10.0	9.9	15.1	15.3	15.1	8.3	14.5	13.9	14.0
CPU Usage (%)	37.0	21.7	30.8	15.1	49.5	15.9	34.7	9.2	42.5
GPU Usage (%)	5.0	3.8	1.9	3.4	3.0	2.8	6.8	1.6	1.6

(b) Jetson TX2 GPU and CPU

TABLE I: TeamNet is compared with the baseline, a MPI approach, and SG-MoE with gRPC and MPI for handwritten digit recognition. (a) On Jetson CPUs, TeamNet is faster than other approaches while the accuracy is not compromised. MPI requires frequent communication among Jetson devices, thus it is slower than other approaches. (b) On Jetson GPUs, TeamNet is still faster than MPI and the SG-MoE approaches. However, TeamNet is not better than the baseline. The root cause is that there is a fixed cost over the WiFi communication. The performance gain from a smaller model is overwhelmed by the communication cost, especially when the device's computing power is significantly larger than the computation needed by the model.

are calculated with respect to the loss from the data examples in batch β_i. The parameters of Expert i (noted by θ_i) are then updated by subtracting the product of the learning rate η and the normalized gradients of the batch (batch normalization) [14]. Details of the process are given in Algorithm 3.

Algorithm 3 Training Experts

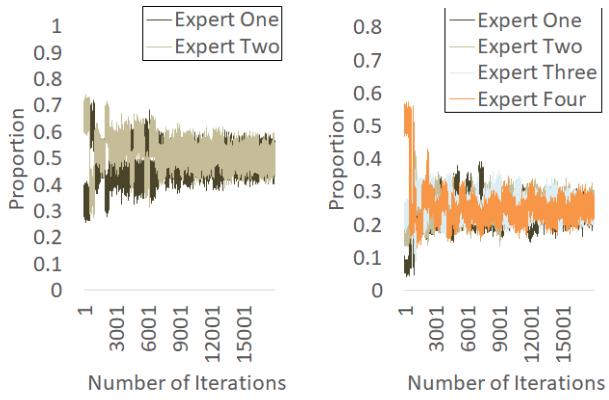
▷ Let β be a mini-batch with size n
▷ Let $\bar{G}^{\beta}$ be the vector of $\bar{G}(x, \delta)$ for all x in β
1: **procedure** EXPERT_TRAIN($\beta, \bar{G}^{\beta}, \theta, \eta, K$)
2: $\beta_1, \beta_2, ..., \beta_K \leftarrow$ group β by $\bar{G}^{\beta}$
3: **for** i **in** $1...K$ **do** ▷ in parallel
4: $\theta_i \leftarrow \theta_i - \eta \frac{1}{|\beta_i|} \sum_{(x,y)\in\beta_i} \nabla_{\theta_i} \sum_c y \log f(x; \theta_i)$
5: **end for**
6: **end procedure**

At Line 4, the formula $\sum_c y \log f(x; \theta_i)$ gives the cross entropy loss (objective) for optimization. Only batch β_i is used to update the parameters of Expert i. No expert learns from all data examples in β.

V. TeamNet Inference

Once the experts are trained, in the inference phase, given a new instance x, each expert makes its own prediction and computes the respective predictive entropy. A gate function is then applied to the predictive entropy and selects the prediction of the expert with the least uncertainty as the final output. This procedure is illustrated in Figure 4.

The $\arg\min$ gate in Figure 4 is not the only way to combine outputs of multiple expert models. For example, one can take the (weighted) majority vote from all experts as in ensemble learning [15]. However, since the experts are trained to highly



(a) Double Experts (b) Quadro Experts

Fig. 6: TeamNet's convergence is evaluated for handwritten digit recognition by monitoring at each iteration the proportion of data assigned to each expert. (a) With two experts, the proportion deviates from the set point (0.5) initially and converges to it at about the 12000^{th} iteration. (b) With four experts, the proportion fluctuates at the beginning, but finally reaches the set point (0.25) at about the 15000^{th} iteration.

specialize on a subset of the data, considering the prediction of "non-expert" can be detrimental.

VI. Performance Evaluation

In this section, we evaluate the proposed algorithms using two image datasets, namely, the MNIST handwritten digit and the CIFAR-10 datasets. In addition to testing TeamNet's predictive accuracy, we also evaluate its inference time, memory usage and CPU/GPU usage for image classification tasks. Its performance is compared with three different message passing interface (MPI) implementations to parallelize baseline models and the state-of-the-art Sparsely-Gated Mixture-of-Experts (SG-MoE).

A. Implementation

In the experiment, communication among the edge devices is done through TCP sockets over WiFi. Each edge device runs a listening socket to accept incoming data. Two types of edge devices (Jetson TX2 and Raspberry Pi 3 Model B+) are employed with TensorFlow, CUDA and cuDNN pre-installed. Trained experts (neural networks) are executed on the TensorFlow framework at the inference stage.

Message Passing Inference (MPI): Two types of neural networks are evaluated in the experiments: MLP and convolutional neural network (CNN) with the Shake-Shake regularization. In the first case, matrix (weights) multiplication can be split among multiple edge nodes using the MPI protocol (MPI-Matrix). In the second case, there are two main branches in the Shake-Shake CNN, which can be split into two edge nodes and coordinated through the MPI protocol (MPI-Branch). Therefore, MPI-Branch is only evaluated in experiments employing two edge devices. Alternatively, we can distribute convolutional kernels and their associated computation onto multiple edge devices. This approach is called (MPI-Kernel). It can be tested with multiple edge devices.

All MPI approaches are executed on the TensorFlow framework at the inference stage.

Sparsely-Gated Mixture-of-Experts (SG-MoE): SG-MoE requires both experts and a gate to be trained together as a joint architecture. Similar to TeamNet, SG-MoE is trained with the TensorFlow framework on NVIDIA 1080TI GPUs. At the inference stage, each expert is executed on one edge node, and the gate is placed on one of the edge nodes.

Two protocols are evaluated for communication among SG-MoE experts, namely, gRPC, an open source remote procedure call (RPC) system, and MPI. The respective SG-MoE implementations are called SG-MoE-G and SG-MoE-M.

B. Experimental Setup

In the experiments, two datasets are evaluated: MNIST and CIFAR-10. MNIST is a dataset of handwritten digits on grey color images. There are 10 classes in total from digit zero to digit nine. It consists of a training set of 60000 images and a test set of 10000 images.

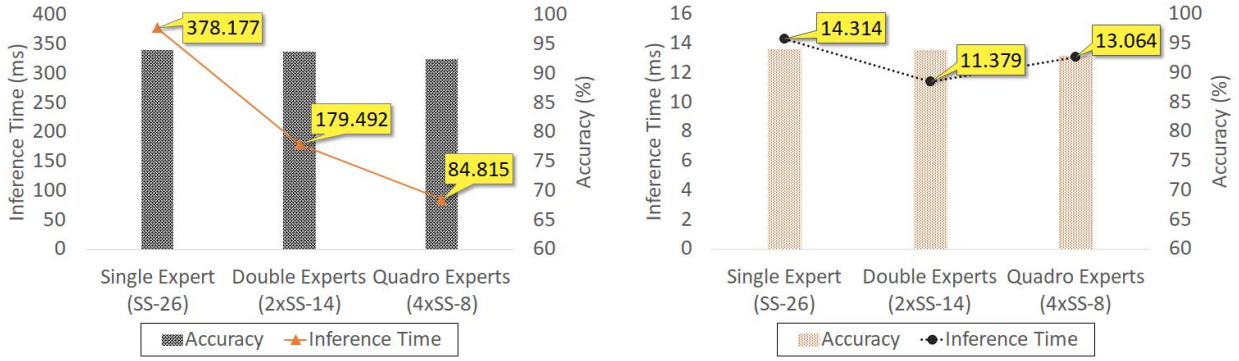
CIFAR-10 [16] is a benchmark dataset in image classification. There are 10 classes in the dataset such as the airplane, automobile, bird, and dog. It consists of 60000 32-by-32 color images, with 50000 images for training and 10000 images set aside for evaluation. CNN allows fast inference and is commonly used in image classification such as the CIFAR-10 image classification task.

C. Handwritten Digit Recognition

Multi-layer perceptron (MLP) classifiers are trained with the MNIST dataset to recognize handwritten digits on images. With TeamNet, four 2-layer (4xMLP-2) and two 4-layer multi-layer perceptrons (2xMLP-4) are trained respectively. They collaborate with each other in prediction using $\arg\min$ gate in Figure 4. For comparison, an 8-layer MLP (MLP-8) is trained on the same dataset with the same number of epochs as the baseline model.

The first experiment is conducted on the edge device: Raspberry Pi 3 Model B+. Accuracy, inference time, memory and CPU usages are measured for all three scenarios. From Figure 5, we observe that TeamNet with quadro experts (4xMLP-2) has the shortest inference time, lowest memory and CPU consumption due to a smaller model size on each edge device. At the same time, the predictive accuracy is not compromised of both 4xMLP-2 (quadro experts) and 2xMLP-4 (double experts).

The second experiment compares TeamNet with MPI in the context of distributed edge computing (Table I). Jetson devices participate in the experiment, with WiFi connecting with each other. Accuracy, inference time, memory, CPU and GPU usages are measured during the experiment. The results show that TeamNet far excels MPI in inference time. Inference



(a) Jetson TX2 CPU only

(b) Jetson TX2 GPU and CPU

Fig. 7: TeamNet's performance is evaluated on both Jetson CPUs and GPUs for the image classification task. (a) On Jetson CPUs, inference becomes faster with more experts in TeamNet, while accuracy is generally not compromised. (b) On Jetson GPUs, the fastest inference is acheived when there are two experts in TeamNet. Because there is a fixed communication cost over the WiFi network, a shorter time cannot be achieved when the communication latency is too close to the actual latency caused by the computation unless the model size is significantly larger and more computation is needed.

	Base	Double Nodes					Quadro Nodes			
		TeamNet	MPI-Kernel	MPI-Branch	SG-MoE-G	SG-MoE-M	TeamNet	MPI-Kernel	SG-MoE-G	SG-MoE-M
Accuracy (%)	94.0	93.7	**93.9**	**93.9**	89.7	90.1	92.4	**93.6**	87.1	87.8
Inference Time (ms)	378.2	179.5	2684.3	1227.8	**157.3**	192.4	84.8	6722.7	**67.8**	71.6
Memory Usage (%)	27.3	20.9	18.6	11.8	15.0	15.6	18.5	14.9	10.7	14.1
CPU Usage (%)	95.4	91.9	40.3	47.6	45.7	75.7	82.0	34.5	21.4	57.7

(a) Jetson TX2 CPU only

	Base	Double Nodes					Quadro Nodes			
		TeamNet	MPI-Kernel	MPI-Branch	SG-MoE-G	SG-MoE-M	TeamNet	MPI-Kernel	SG-MoE-G	SG-MoE-M
Accuracy (%)	93.9	93.8	93.9	**94.0**	89.4	89.0	92.8	**93.5**	87.3	87.3
Inference Time (ms)	14.3	**11.4**	2611.7	1002.7	31.7	29.4	**13.1**	7062.9	30.6	29.5
Memory Usage (%)	43.8	36.8	27.5	26.3	26.8	26.1	33.5	22.6	22.6	22.7
CPU Usage (%)	24.3	20.3	33.4	35.4	16.9	43.7	16.2	34.2	9.4	46.3
GPU Usage (%)	34.6	26.7	2.3	2.7	15.2	14.2	18.1	1.3	5.3	5.3

(b) Jetson TX2 GPU and CPU

TABLE II: TeamNet is compared with the baseline model, two MPI approaches, and SG-MoE with gRPC and MPI for the image classification task. TeamNet is faster and it consumes fewer resources on each edge device than the baseline model. (a) On Jetson CPUs, SG-MoE with gPRC is slightly faster than TeamNet, but generally less accurate. MPI approaches are slightly more accurate than TeamNet but much slower. (b) On Jetson GPUs, TeamNet is the fastest approach among the others, though MPI approaches are slightly more accurate.

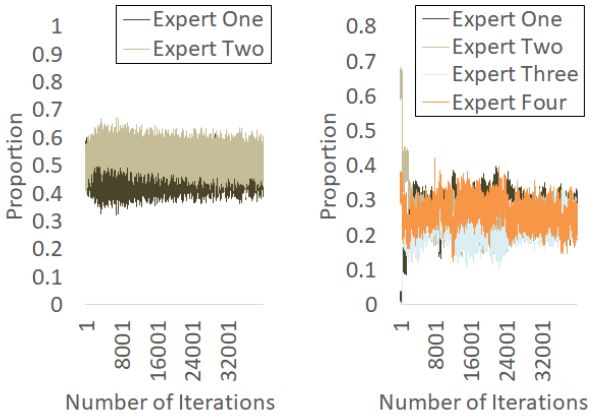
time with MPI is even slower than the one with the baseline model. The reason is that MPI requires frequent communication among Jetson devices per each matrix multiplication, while TeamNet requires only communication at first and at last. WiFi is much slower than the infinite band (mostly used in the data center). It turns out to be overkill when the WiFi communication between edge devices is over frequent in parallel computing. Hence, TeamNet is a better architecture for distributed edge computing.

In the third experiment, TeamNet is compared with two SG-MoE approaches: SG-MoE with gRPC and SG-MoE with MPI. Accuracy, inference time, memory, CPU and GPU usages are measured during the experiment. TeamNet is more accurate than SG-MoE most of the time since SG-MoE adopts random data assignment and does not promote specialization on the experts (Table I).

The fourth experiment is to test whether the proportion of data assigned to each expert eventually converges to the set point. Figure 6a shows that the proportion deviates from the set point (0.5) initially, and as the training goes, the proportion gets closer and closer to the set point (0.5) and eventually converges to it. Figure 6b demonstrates that with four experts in TeamNet, the proportion fluctuates at the beginning but eventually reaches the set point (0.25).

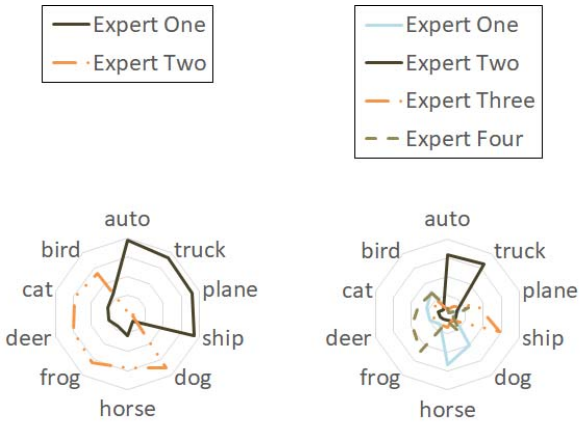
D. Image Classification

In the first experiment, we evaluate the performance of our algorithm on CNNs with the Shake-Shake regularization. With TeamNet, four 8-layer (4xSS-8) and two 14-layer Shake-Shake CNNs (2xSS-14) are trained respectively. Predictive accuracy,



(a) Double Experts (b) Quadro Experts

Fig. 8: TeamNet's convergence is evaluated for the image classification task by monitoring at each iteration the proportion of data assigned to each expert. (a) The proportion is initially close to the set point (0.5) by luck, but it deviates from the set point (0.5) very fast since both experts had limited knowledge of the dataset at this stage and cannot have a clear judgment on uncertainty. As the training goes, both experts become certain of an equal amount of data, and the proportion becomes closer and closer (and converges) to the set point (0.5). (b) With four experts in TeamNet, the proportion deviates from the set point (0.25) at the beginning but converges to it finally at about the 32000^{th} iteration.



(a) Double Experts (b) Quadro Experts

Fig. 9: Specialization is emphasized in TeamNet. (a) With two experts in TeamNet, Expert One is more certain of machines such as airplanes, automobiles and trucks, while Expert Two is more certain of animals such as cats and dogs. (b) With four experts in TeamNet, Expert One and Expert Four are more certain of animals, while Expert Two and Expert Three are more certain of machines.

inference time, memory, CPU and GPU usages on Jetson TX2 are compared to those of the 26-layer Shake-Shake CNN (SS-26) model. The experiments demonstrate that TeamNet nearly halves the inference time on Jetsons with CPU only, while does not sacrifice accuracy (Figure 7a). On Jetson GPUs, 2xSS-14 achieves the fastest inference. The performance gain from smaller model size is overwhelmed by the communication cost when 8-layer and 14-layer Shake-Shake models are running on Jetson GPUs (Figure 7b).

In the second experiment, TeamNet is compared with MPI and Sparsely-Gated Mixture-of-Experts (SG-MoE) [6] on the Jetson TX2 edge devices. The experiment shows that TeamNet has much shorter inference time than that of MPI, and more accurate than SG-MoE. Faster inference time is benefited by infrequent communication among edge devices, and better accuracy is owing to the specialization of each expert. In SG-MoE, data examples are randomly assigned to experts for learning, thus specialization is not being emphasized. On the other hand, TeamNet assigns the most certain data examples to the experts for learning, thus experts will only learn the ones which they are most familiar with. That is how the specialization being emphasized in TeamNet and why TeamNet outperforms SG-MoE (Table II).

The third experiment evaluates whether the proportion of data assigned to each expert converges eventually to the set point. Figure 8a demonstrates that the proportion is close to the set point (0.5) at the beginning while very fast deviates from it. The reason is that both experts have limited knowledge of the dataset at this stage and cannot have a clear judgment of uncertainty. As the training goes, both experts know more about the dataset, and the proportion is closer and closer to the set point. Figure 8b shows that when there are four experts in TeamNet, the proportion converges to the set point (0.25) eventually at about 32000^{th} iteration.

In the fourth experiment, we further investigate the effects of specialization which is forced by the training algorithm. The experiment is performed on the CIFAR-10 dataset. There are 10 classes in the dataset such as airplanes, birds, and cats. When there are two experts in TeamNet, we observe that Expert One is more certain of machines such as airplanes, automobiles, ships, and trucks; on the other hand, Expert Two knows more about animals such as birds, cats, and horses (Figure 9a). When there are four experts in TeamNet, we observe that Expert One and Expert Four are more certain of animals, and each masters half of this category; on the other hand, Expert Two and Expert Three are more certain of machines: Expert Two has more knowledge about trucks and automobiles, while Expert Three knows more about ships and airplanes (Figure 9b).

VII. Conclusions

In this paper, we propose TeamNet, a novel framework for collaborative inference among storage-and-compute-limited edge devices. Unlike existing computation-partition solutions that take a pre-trained model and determine the best partitions

between edge devices and the cloud, TeamNet is a *model partition* approach by training multiple small *specialized* models. It is proven to work well with the commonly used neural networks (CNNs and MLPs) on different numbers and types of edge devices. Extensive experiments have demonstrated that TeamNet can be executed on low-end edge devices such as Raspberry Pi and Jetson TX2 devices with superior performance than the baseline solution and the state-of-the-art MoE approach in inference time with marginal degradation in inference accuracy.

Currently, TeamNet is trained with the objective of equally partitioning training samples among experts. As part of future work, we will explore other objective functions especially those can adapt to the imbalances among different classes in training data.

REFERENCES

[1] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 615–629.

[2] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE, 2016, pp. 2464–2469.

[3] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," *arXiv preprint arXiv:1802.03835*, 2018.

[4] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 328–339.

[5] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[6] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[7] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[8] C. M. Bishop and M. Svenskn, "Bayesian hierarchical mixtures of experts," in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 57–64.

[9] B. Yao, D. Walther, D. Beck, and L. Fei-Fei, "Hierarchical mixture of classification experts uncovers interactions between brain regions," in *Advances in Neural Information Processing Systems*, 2009, pp. 2178–2186.

[10] C. E. Rasmussen and Z. Ghahramani, "Infinite mixtures of gaussian process experts," in *Advances in neural information processing systems*, 2002, pp. 881–888.

[11] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts." in *CVPR*, 2017, pp. 7120–7129.

[12] O. Chapelle and M. Wu, "Gradient descent optimization of smoothed information retrieval metrics," *Information retrieval*, vol. 13, no. 3, pp. 216–235, 2010.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[15] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[16] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

APPENDIX A
PROOF SKETCH OF CONVERGENCE OF ALGORITHM 2

In this section, we analyze the convergence of Algorithm 2 based on some simplifying assumptions. Experimental results in Section VI have shown empirically the procedure converges.

Assumption 1. *The expert model trained with $x\%$ of training data has the least uncertainty of $x\%$ of test instances among all expert models.*

Assumption 2. *Each batch in training is sufficiently random.*

Assumption 3. *Solution to* (4) *gives*

$$\sum_{i=1}^{K}\left|\bar{\gamma}_i(\delta)-\left(\frac{1}{K}-a\cdot(\gamma_i-\frac{1}{K})\right)\right|=0, \tag{8}$$

Proof. From Assumption 3, we have

$$\bar{\gamma}_i(\delta)\approx\frac{1}{K}-a\cdot(\gamma_i-\frac{1}{K}), \tag{9}$$

for all $i=1,...,K$. Let $|\beta|$ be the number of examples in a batch, and $n_{i,l}$ be the number of examples in the l^{th} batch used to train Expert i. Consider the L^{th} batch.

From Assumption 1,

$$\gamma_{i,L}=\frac{\sum_{l=1}^{L-1}n_{i,l}}{(L-1)\cdot|\beta|} \tag{10}$$

From (9), we have

$$\bar{\gamma}_{i,L}(\delta)=\frac{1}{K}-a\cdot(\gamma_{i,L}-\frac{1}{K}), \tag{11}$$

where $\bar{\gamma}_{i,L}(\delta)$ is the percentage of samples in the Lth batch ith expert is least uncertain of according to $\bar{G}$. Since

$$n_{i,L}=\bar{\gamma}_{i,L}\cdot|\beta|, \tag{12}$$

we have

$$\begin{aligned}\gamma_{i,L+1}&=\frac{\sum_{l=1}^{L-1}n_{i,l}+n_{i,L}}{L\cdot|\beta|}\\&=\frac{\gamma_{i,L}\cdot(L-1)+\frac{1}{K}-a\cdot(\gamma_{i,L}-\frac{1}{K})}{L}.\end{aligned}$$

Subtracting $\frac{1}{K}$ from both sides and with further simplification, we have

$$\begin{aligned}\gamma_{i,L+1}-\frac{1}{K}&\approx\frac{\gamma_{i,L}\cdot(L-1)+\frac{1}{K}-a\cdot(\gamma_{i,L}-\frac{1}{K})-\frac{L}{K}}{L}\\&=\frac{(\gamma_{i,L}-\frac{1}{K})(L-1)-a\cdot(\gamma_{i,L}-\frac{1}{K})}{L}\\&=\frac{L-1}{L}(1-\frac{a}{L-1})(\gamma_{i,L}-\frac{1}{K})\end{aligned}$$

Therefore,

$$\begin{aligned}\left|\gamma_{i,L+1}-\frac{1}{K}\right|&=\frac{L-1}{L}(1-\frac{a}{L-1})\left|\gamma_{i,L}-\frac{1}{K}\right|\\&=\frac{1}{L}\prod_{l}\left(1-\frac{a}{l-1}\right)\times\left|\gamma_{i,1}-\frac{1}{K}\right|.\end{aligned}$$

With $a>0$, clearly, as $L\to\infty$, $\gamma_{i,L+1}\to\frac{1}{K}$ In other words, the training data will be *equally* partitioned among experts. □