Distributed learning of deep neural network over multiple agents

Otkrist Gupta*, Ramesh Raskar

Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, USA

ARTICLE INFO

ABSTRACT

In domains such as health care and finance, shortage of labeled data and computational resources is a critical issue while developing machine learning algorithms. To address the issue of labeled data scarcity in training and deployment of neural network-based systems, we propose a new technique to train deep neural networks over several data sources. Our method allows for deep neural networks to be trained using data from multiple entities in a distributed fashion. We evaluate our algorithm on existing datasets and show that it obtains performance which is similar to a regular neural network trained on a single machine. We further extend it to incorporate semi-supervised learning when training with few labeled samples, and analyze any security concerns that may arise. Our algorithm paves the way for distributed training of deep neural networks in data sensitive applications when raw data may not be shared directly.

1. Introduction

Deep neural networks have become the new state of the art in classification and prediction of high dimensional data such as images, videos and bio-sensors. Emerging technologies in domains such as biomedicine and health stand to benefit from building deep neural networks for prediction and inference by automating the human involvement and reducing the cost of operation. However, training of deep neural nets can be extremely data intensive requiring preparation of large scale datasets collected from multiple entities (Chervenak et al., 2000; Chuang and Sirbu, 2000). A deep neural network typically contains millions of parameters and requires tremendous computing power for training, making it difficult for individual data repositories to train them.

Sufficiently deep neural architectures needing large supercomputing resources and engineering oversight may be required for optimal accuracy in real world applications. Furthermore, application of deep learning to such domains can sometimes be challenging because of privacy and ethical issues associated with sharing of de-anonymized data. While a lot of such data entities have vested interest in developing new deep learning algorithms, they might also be obligated to keep their user data private, making it even more challenging to use this data while building machine learning pipelines. In this paper, we attempt to solve these problems by proposing methods that enable training of neural networks using multiple data sources and a single supercomputing resource.

2. Related work

Deep neural networks have proven to be an effective tool to classify and segment high dimensional data such as images (Krizhevsky et al., 2012), audio and videos (Karpathy and Fei-Fei, 2015). Deep models can be several hundreds of layers deep (He et al., 2016), and can have millions of parameters requiring large amounts of computational resources, creating the need for research in distributed training methodologies (Dean et al., 2012). Interesting techniques include distributed gradient optimization (Mcdonald et al., 2009; Zinkevich et al., 2010), online learning with delayed updates (Langford et al., 2009) and hashing and simplification of kernels (Shi et al., 2009). Such techniques can be utilized to train very large scale deep neural networks spanning several machines (Agarwal and Duchi, 2011) or to efficiently utilize several GPUs on a single machine (Agarwal et al., 2014). In this paper we propose a technique for distributed computing combining data from several different sources.

Secure computation continues to be a challenging problem in computer science (Sood, 2012). One category of solutions to this problem involve adopting oblivious transfer protocols to perform secure dot product over multiple entities in polynomial time (Avidan and Butman, 2006). While this method is secure, it is somewhat impractical when considering large scale datasets because of resource requirements. A more practical approach proposed in Avidan and Butman (2006) involves sharing only SIFT and HOG features instead of the actual raw data. However, as shown in (Dosovitskiy and Brox), such feature vectors can be inverted very accurately using prior knowledge

---

* Corresponding author.
   E-mail address: otkrist@mit.edu (O. Gupta).

of the methods used to create them. Neural networks have been shown to be extremely robust to addition of noise and their denoising and reconstruction properties make it difficult to compute them securely (Vincent et al., 2010). Neural networks have also been shown to be able to recover an entire image from only a partial input (Pathak et al.), rendering simple obfuscation methods inert.

Widespread application of neural networks in sensitive areas such as finance and health, has created a need to develop methods for both distributed and secure training (Secretan et al., 2007; Chonka et al., 2011; Wu et al., 2007) and classification in neural networks. Under distributed and secure processing paradigms, the owner of the neural network doesn't have access to the actual raw data used to train the neural network (Barni et al., 2006). This also includes secure paradigms in cloud computing (Karam et al., 2012; Subashini and Kavitha, 2011), virtualization (Mackay et al., 2012) and service oriented architectures (Baker et al., 2015). The secure paradigms may also extend to the neural activations and (hyper)parameters. Such algorithms form a subset inside the broader realm of multi-party protocol problems involving secure computation over several parties (Goldreich et al., 1987; Yao, 1986). Some interesting solutions include using Ada-boost to jointly train classifier ensembles (Zhang and Zhong, 2013), using random rotation perturbations for homomorphic pseudo-encryption (Chen and Liu) and applying homomorphic cryptosystem to perform secure computation (Orlandi et al., 2007).

3. Theory

In this paper we propose new techniques that can be used to train deep neural networks over multiple data sources while mitigating the need to share raw labeled data directly. Specifically we address the problem of training a deep neural network over several data entities (Alice(s)) and one supercomputing resource (Bob). We aim at solving this problem while satisfying the following requirements:

1. A single data entity (Alice) doesn't need to share the data with Bob or other data resources.
2. The supercomputing resource (Bob) wants control over the architecture of the Neural Network(s)
3. Bob also keeps a part of network parameters required for inference.

In upcoming sections we will show how to train neural networks between multiple data entities (Alice(s)) and a supercomputing resource (Bob). Techniques will include methods which encode data into a different space and transmit it to train a deep neural network. We will further explore how a third-party can use this neural network to classify and perform inference. Our algorithm can be run using one or multiple data entities, and can be run in peer-to-peer or centralized mode. Please see Fig. 1 for the schematic depiction of algorithm modalities.

3.1. Distributed training over single entity

We will start by describing the algorithm in its simplest form which considers training a neural network using data from a single entity and supercomputing resource. Let us define a deep neural network as a function F, topologically describable using a sequence of layers $\{L_0, L_1, ...L_N\}$. For a given input ($data$), the output of this function is given by $F(data)$ which is computed by sequential application of layers $F(data) \leftarrow L_N(L_{N-1}...(L_0(data)))$.

Let $G_{loss}(output, label)$ denote the customized loss function used for computing gradients for the final layer. Gradients can be backpropagated over each layer to generate gradients of previous layers and to update the current layer. We will use $L_i^T(gradient)$ to denote the process of backpropagation over one layer and $F^T(gradient)$ to denote backpropagation over the entire Neural Network. Similar to forward propagation, backpropagation on the entire neural network is comprised of sequential backward passes F^T

$(gradient) \leftarrow L_1^T(L_2^T...(L_N^T(gradient)))$. Please note that the backward passes will require activations after the forward pass on individual perceptrons.

Finally, $Send(X, Y)$ represents the process of sending data X over the network to entity Y. In the beginning, Alice and Bob initialize their parameters randomly. Alice then iterates over its dataset and transmits encoded representations to Bob. Bob then computes losses and gradients and sends the gradients back to Alice. Algorithm 1 describes how to train a deep neural classifier using a single data source.

3.1.1. Correctness

Here we analyze if training using our distributed algorithm produces the same results as a normal training procedure. Under a normal training procedure we would first compute forward pass $output \leftarrow F(data)$ followed by computation of loss gradients $gradients \leftarrow G(output, label)$. These gradients will be backpropagated to refresh weights $F' \leftarrow F^T(gradients)$.

Algorithm 1
Distributed Neural Network training over 2 agents.

1:	**Initialize:**
	$\phi \leftarrow$ Random Initializer (Xavier/Gaussian)
	$F_a \leftarrow \{L_0, L_1, ...L_n\}$
	$F_b \leftarrow \{L_{n+1}, L_{n+2}, ...L_N\}$
2:	Alice randomly initializes the weights F_a using ϕ
3:	Bob randomly initializes the weights of F_b using ϕ
4:	**while** Alice has new data to train on **do**
5:	Alice uses standard forward propagation on data
	$\triangleright X \leftarrow F_a(data)$
6:	Alice sends n^{th} layer output X and label to Bob
	$\triangleright Send((X, label), Bob)$.
7:	Bob propagates incoming features on its network
	$\triangleright output \leftarrow F_b(X)$
8:	Bob generates gradients for its final layer
	$\triangleright gradient \leftarrow G'(output, label)$
9:	Bob backpropagates the error in F_b until L_{n+1}
	$\triangleright F_b', gradient' \leftarrow F_b^T(gradient)$
10:	Bob sends gradient of L_n to Alice
	$\triangleright Send(gradient', Alice)$
11:	Alice backpropagates gradients received
	$\triangleright F_a', \_ \leftarrow F_a^T(gradient')$
12:	**end while**

Since forward propagation involves sequential application of individual layers we concur that $F(data)$ is same as $F_b(F_a(data))$. Therefore the process of sequential computation and transmission followed by computation of remaining layers is functionally identical to application of all layers at once. Similarly because of the chain rule in differentiation, backpropagating $F^T(gradients)$ is functionally identical to sequential application of $F_a^T(F_b^T(gradients))$. Therefore, we can conclude that our algorithm will produce identical results to a normal training procedure.

Algorithm 2
Distributed Neural Network over N + 1 agents.

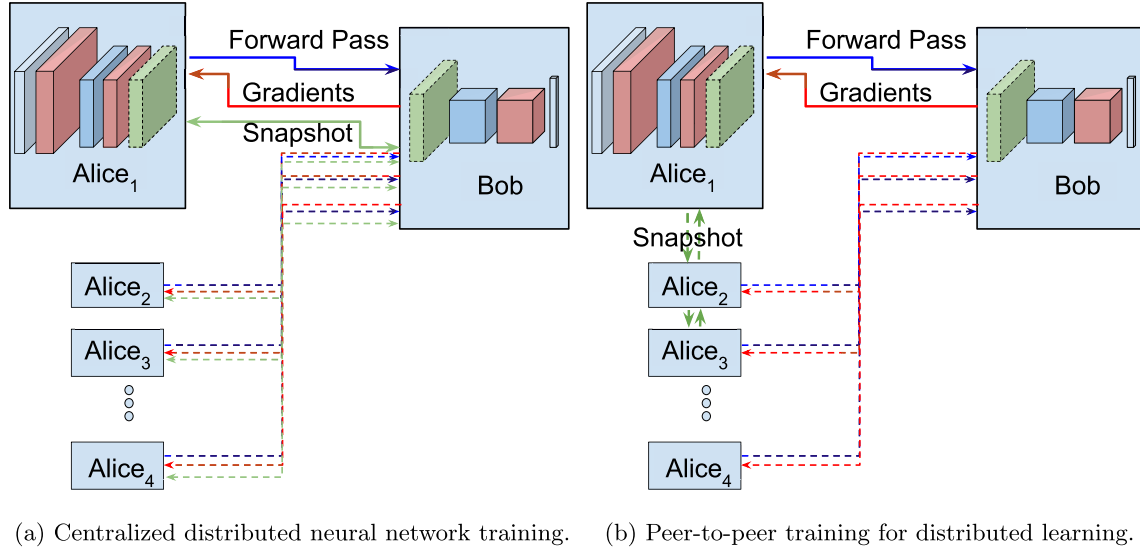|	
1:	**Initialize:**
	$\phi \leftarrow$ Random Initializer (Xavier/Gaussian)
	$F_{a,1} \leftarrow L_0, L_1, ...L_n$
	$F_b \leftarrow L_{n+1}, L_{n+2}, ...L_N$
2:	Alice$_1$ randomly initializes the weights of $F_{a,1}$ using ϕ
3:	Bob randomly initializes the weights of F_b using ϕ
4:	Bob sets Alice$_1$ as last trained

(a) Centralized distributed neural network training. (b) Peer-to-peer training for distributed learning.

Fig. 1. Two modalities of our algorithm: centralized mode (1a) and peer-to-peer mode (1b).

Algorithm 2 (*continued*)

5:	**while** Bob waits for next $Alice_j$ to send data **do**
6:	$Alice_j$ requests Bob for last $Alice_o$ that trained
7:	$Alice_j$ updates its weights
	$\triangleright F_{a,j} \leftarrow F_{a,o}$
8:	$Alice_j$ uses standard forward propagation on data
	$\triangleright X \leftarrow F_{a,j}(data)$
9:	$Alice_j$ sends nth layer output and label to Bob
	$\triangleright Send((X, label), Bob)$.
10:	Bob propagates incoming features on its network
	$\triangleright output \leftarrow F_b(X)$
11:	Bob generates gradients for its final layer
	$\triangleright gradient \leftarrow G'(output, label)$
12:	Bob backpropagates the error in F_b until L_{n+1}
	$\triangleright F'_b, gradient' \leftarrow F_b^T(gradient)$
13:	Bob sends gradient of L_n to $Alice_j$
	$\triangleright Send(gradient', Alice_j)$
14:	$Alice_j$ backpropagates the gradients it received
	$\triangleright F'_{a,j}, \_ \leftarrow F_{a,j}^T(gradient')$
15:	Bob sets $Alice_j$ as last trained
16:	**end while**

3.2. Distributed training over multiple entities

Here we demonstrate how to extend the algorithm described in 3.1 to train using multiple data entities. We will use the same mathematical notations as used in 3.1 when defining neural network forward and backward propagation. In Algorithm 2 we demonstrate how to extend our algorithm when there are N data entities, each of them is denoted by $Alice_i$.

In Algorithm 2 at the first initialization step, Bob sends $Alice_1$ topological description of first N layers. Alice and Bob use standard system level libraries for random initialization of their parameters. Bob then sets $Alice_1$ as the last agent used for training and begins training using data from $Alice_1$. We modify 1 and add a step which uses data from multiple entities in a round robin fashion, allowing for a distributed learning framework. However, for consistency, $Alice_j$ may be required to update weights before they begin their training. We solve this by providing two separate methodologies involving peer-to-peer

and centralized configurations. In the *centralized* mode, Alice uploads an encrypted weights file to either Bob or a third-party server. When a new Alice wishes to train, it downloads and decrypts these weights. In *peer-to-peer* mode, Bob sends the last trained Alice's address to the current training party and Alice uses this to connect and download the encrypted weights. The implementation details for both methods can be seen in Supplementary Material. Once the weights are updated, $Alice_j$ continues its training. Since the same weights are initialized in both centralized and peer-to-peer mode, the final result of training is identical in both modalities.

3.2.1. Correctness

We analyze if training using our algorithm produces results which are identical when training with all the data combined on a single machine (under the assumption that the data arriving at multiple entities preserves the order and random weights use same initialization). The algorithm correctness stems from the fact that Bob and at least one of $Alice_o$ have identical neural network parameters to regular training at $iteration_k$. We use inductive techniques to prove that this is indeed the case.

Lemma 1. *The neural network being trained at $iteration_k$ is identical to the neural network if it was trained by just one entity.*

Base Case: One of $Alice_{1...N}$ has the correct weights at beginning of first iteration.

Proof. $Alice_1$ randomly initialized weights and Bob used these weights during first iteration. We assume that this initialization is consistent when training with single entity. In case another $Alice_j$ attempts to train, it will refresh the weights to correct value.

Recursive Case: Assertion: If $Alice_j$ has correct weights at beginning of $iteration_i$ it will have correct weights at beginning of iteration $i + 1$.

Proof. $Alice_j$ performs backpropagation as the final step in iteration i. Since this backpropagation is functionally equivalent to backpropagation applied over the entire neural network at once, $Alice_j$ continues to have correct parameters at the end of one training iteration. ($F^T(gradient)$ is functionally identical to sequential application of $F_{a,j}^T(F_b^T(data))$, as discussed in 3.1.1).

3.3. Semi-supervised application

In this section we describe how to modify the distributed neural network algorithm to incorporate semi-supervised learning and generative losses when training with fewer data points. In situations with fewer labeled data-samples, a reasonable approach includes learning hierarchical representations using unsupervised learning (Shin et al., 2013). Compressed representations generated using unsupervised learning and *autoencoders* can be used directly for classification (Coates et al., 2012). Additionally, we can combine the losses of generative and predictive segments to perform semi supervised learning, adding a regularization component while training on fewer samples (Weston et al., 2012).

Over here we demonstrate how we can train autoencoders and semi-supervised learners using a modified version of Algorithm 1. Such unsupervised learning methods can be extremely helpful when training with small amounts of labeled data. We assume that out of n layers for Alice, the first m layers are encoder and the remaining $n - m$ layers belong to its decoder. $F_{e,i}$ denotes the forward propagation over encoder (computed by sequential application $L_m(L_{m-1}...(L_0(data))))$. $F_{d,i}$ denotes application of decoder layers. During forward propagation Alice propagates data through all n layers and sends output from mth layer to Bob. Bob propagates the output tensor from Alice through $L_{n...N}$ and computes the classifier loss (logistic regression).

Let *loss* define the logistic regression loss in the predictive segment of the neural network (last $N - n$ layers owned by Bob), and let $loss_{enc}$ define the contrastive loss in autoencoder (completely owned by Alice (s)). Bob can compute *loss* using its softmax layer and can back-propagate gradients computed using this loss to layer L_{n+1} giving gradients from classifier network [$gradient' \leftarrow F_b^T(gradient)$]. Alice$_i$ can compute the autoencoder gradients and can backpropagate it through its *decoder* network [$F_{d,i}^T(gradient_{enc})$]. We can facilitate semi-supervised learning by combining a weighted sum of two losses. The weight α is an added hyperparameter which can be tuned during training.

$$\eta \leftarrow F_b^T(gradient) + \alpha * F_{d,i}^T(gradient_{enc}) \tag{1}$$

Algorithm 3
Distributed Neural Network with an Autoencoder over N + 1 agents.

1:	**Initialize:**
	$\phi \leftarrow$ Random Initializer (Xavier/Gaussian)
	$F_{e,1} \leftarrow L_0, L_1, ...L_m$
	$F_{d,1} \leftarrow L_m, L_{m+1}, ...L_n$
	$F_b \leftarrow L_{n+1}, L_{n+2}, ...$
2:	Alice$_1$ randomly initializes the weights of $F_{a,1}$ using ϕ
3:	Bob randomly initializes the weights of F_b using ϕ
4:	Alice$_1$ transmits weights of $F_{a,1}$ to Alice$_{2...N}$
5:	**while** Bob waits for next feature vector from Alice$_j$ **do**
6:	Alice$_j$ requests Bob for last Alice$_o$ that trained
7:	Alice$_j$ updates its weights
	$\triangleright F_{a,j} \leftarrow F_{a,o}$
8:	Alice$_j$ uses standard forward propagation on data
	$\triangleright X_m \leftarrow F_{e,j}(data)$
	$\triangleright X \leftarrow F_{d,j}(X_m)$
9:	Alice$_j$ sends m^{th} layer output and label to Bob
	$\triangleright Send((X_m, label), Bob).$
10:	Bob propagates incoming features on its network F_b
	$\triangleright output \leftarrow F_b(X_m).$
11:	Bob generates gradient for its final layer
	$\triangleright gradient \leftarrow G'(output, label)$
12:	

Algorithm 3 (*continued*)

	Bob backpropagates the error in F_b until L_{n+1}
	$\triangleright F_b', gradient' \leftarrow F_b^T(gradient)$
13:	Bob sends gradient for L_n to Alice$_j$
	$\triangleright Send(gradient', Alice_j)$
14:	Alice$_j$ generates autoencoder gradient for its decoder
	$\triangleright F_{d,j}', gradient_{enc}' = F_{d,j}^T(X)$
15:	Alice$_j$ backpropagates combined gradients
	$\triangleright F_a, \_ \leftarrow F_a^T(\eta(gradient', gradient_{enc}'))$
16:	Bob sets Alice$_j$ as last trained
17:	**end while**

After the initialization steps, Alice propagates its data through its network and sends output from the encoder part to Bob. Bob does a complete forward and backward to send gradients to Alice. Alice then combines losses from its decoder network with gradients received from Bob and uses them to perform backpropagation (please see Algorithm 3 for detailed description).

3.4. Online learning

An additional advantage of using our algorithm is that the training can be performed in an online fashion by providing Bob output of forward propagation whenever there is new annotated data. In the beginning instead of transmitting the entire neural net, Alice$_i$ can initialize the weights randomly using a seed and just send the seed to Alice$_{1...N}$ preventing further network overhead. When Alice is requested for weights in peer-to-peer mode, it can simply share the *weight updates*, which it adds to its parameters during the course of training. The combined value of weight updates can be computed by subtracting weights at beginning of training from current weights. For security, Alice can also upload the encrypted weight updates to a centralized weight server, making it harder to reverse engineer actual weights when using man-in-middle attack. Weights can be refreshed by Alice by combining its initial weights with subsequent weight updates downloaded from the centralized weight server (or Alice(s) depending on mode). To facilitate centralized modality, we can modify step 6 of Algorithm 2, replacing it with a request to download encrypted weights from weight server. Once training is over Alice$_j$ can upload the new encrypted weights to the weight server (please refer to step 15 in Algorithm 2).

3.5. Analyzing security concerns

While a rigorous information theoretical analysis of security is beyond the scope of this paper, over here we sketch out a simple explanation of why reconstructing the data sent by Alice is extremely challenging. The algorithm security lies in whether Bob can invert parameters (F_a) used by Alice during the forward propagation. Bob can indeed build a decoder for compressed representations transmitted by Alice, but it requires Alice revealing the current parameters of its section of neural network (Dosovitskiy and Brox).

In this section we make an argument that Bob cannot discover the parameters used by Alice as long as its layers (denoted by F_a) contain at least one fully connected layer. We will use the word "*configuration*" to denote an isomorphic change in network topology which leads to functionally identical neural network.

Lemma 2. *Let layer M be a fully connected layer containing N outputs then layer M has at least N! functionally equivalent "configurations".*

Proof. We construct a layer M and transpose N output neurons. The output of neurons is reordered without changing weights or affecting learning in any way. Since there are N! possible orderings of these
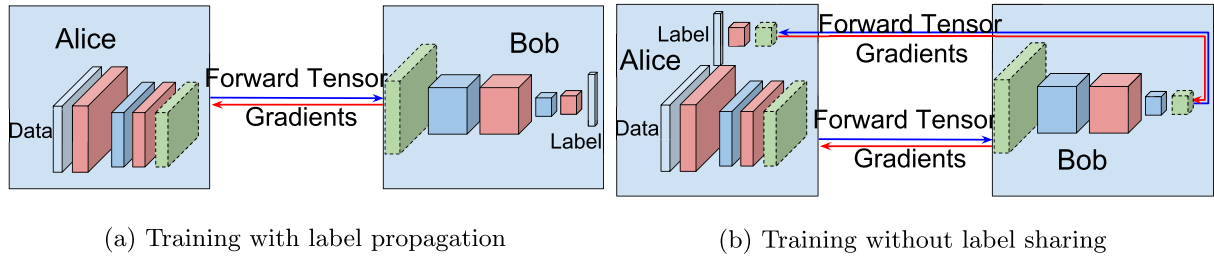
(a) Training with label propagation (b) Training without label sharing

Fig. 2. Figure (2a) shows the normal training procedure while figure (2b) demonstrates how to train without transmitting labels, by wrapping the network around at its last layers.

neurons at least $N!$ unique *configurations* are possible depending on how the weights were initialized.

Bob will have to go through at least $N!$ possible *configurations* to invert the transformation applied by Alice. Since $N! > (N/2)^N > e^N$ this will require an exponential amount of time in a layer of size N. For example if the fully connected layer has 4096 neurons and each configuration could be tested in a second, it would take Bob more than the current age of the universe to figure out parameters used by Alice.

3.6. Training without label propagation

While the algorithm we just described doesn't require sharing raw data, it still does involve sharing labels. We can mitigate this problem by presenting a simple adjustment to the training framework. In this topological modification, we wrap the network around at its end layers and send those back to Alice (see Fig. 2). While Bob still retains majority of its layers, it lets Alice generate the gradients from the end layers and uses them for backpropagation over its own network. We can use a similar argument as one used in Lemma 1 to prove that this method will still work after the layers have been wrapped around. Please see Fig. 2 for a schematic description of our training methodology without label sharing.

4. Datasets and implementation

We use standard *json* communication libraries for asynchronous RPC for implementation. On top of those, we implement a custom protocol for training once a secure connection is established using SSL. Our protocol defines several network primitives (implemented as remote functions) which we broadly divide in 3 parts (1) Training request, (2) Tensor transmission and (3) Weight update. Please refer to appendix for a complete list of network primitives. We describe these three network primitives categories in our Supplementary Material.

4.1. Mixed NIST

Mixed NIST (MNIST) database (LeCun et al., 1989) contains handwritten digits sampled from postal codes and is a subset of a much larger dataset available from the National Institute Science and Technology. MNIST comprises of a total of 70,000 samples divided into 60,000 training samples and 10,000 testing samples. Original binary images were reformatted and spatially normalized to fit in a 20×20 bounding box. Anti-aliasing techniques were used to convert black and white (bilevel) images to grey scale images. Finally the digits were placed in a 28×28 grid, by computing the center of mass of the pixels and shifting and superimposing images in the center of a 28×28 image.

4.2. Canadian Institute For Advanced Research

The Canadian Institute For Advanced Research (CIFAR-10) dataset is a labeled subset of tiny images dataset (containing 80 million

images). It is composed of 60,000, 32×32 color images distributed over 10 different class labels. The dataset consists of 50,000 training samples and 10,000 testing images. Images are uniformly distributed over 10 classes with training batches containing exactly 6000 images for each class. The classes are mutually exclusive and there are no semantic overlaps between the images coming from different labels. We normalized the images using GCA whitening and applied global mean subtraction before training. The same dataset also includes a 100 class variation referred to as CIFAR-100.

4.3. ILSVRC (ImageNet) 2012

This dataset includes approximately 1.2 million images labeled with the presence or absence of 1000 object categories. It also includes 150,000 images for validation and testing purposes. The 1000 object categories are a subset of a larger dataset (ImageNet), which includes 10 million images spanning 10,000 object categories. The object categories may be internal or leaf nodes but do not overlap. The dataset comprises images with varying sizes which are resized to 256×256 and mean subtracted before training.

5. Experiments and applications

We implement our algorithm and protocol using python bindings for *caffe* (Jia et al.). We test our implementation on datasets of various sizes (50 K–1 M) and classes (10, 100 or 1000 classes). We demonstrate that our method works across a range of different topologies and experimentally verify identical results when training over multiple agents. All datasets were trained for an equal number of epochs for fair evaluation.

In 3.2.1 we show why our algorithm should give results identical to a normal training procedure. We experimentally verify our method's correctness by implementing it and training it on a wide array of datasets and topologies including MNIST, ILSVRC 12 and CIFAR 10. Table 1 lists datasets and topologies combined with their test accuracies. Test accuracies are computed by comparing the number of correctly labeled samples to the total number of test data points. As shown in Table 1, the network converges to similar accuracies when training over several agents in a distributed fashion.

Table 1

Accuracies when training using multi-agent algorithm vs when training on a single machine.

Dataset	Topology	Accuracy (Single Agent)	Accuracy using our method	Epochs
MNIST	LeNet (LeCun et al., 1989)	99.18%	99.20%	50
CIFAR 10	VGG (Simonyan and Zisserman)	92.45%	92.43%	200
CIFAR 100	VGG (Simonyan and Zisserman)	66.47%	66.59%	200
ILSVRC 12	AlexNet (Krizhevsky et al., 2012)	57.1%	57.1%	100
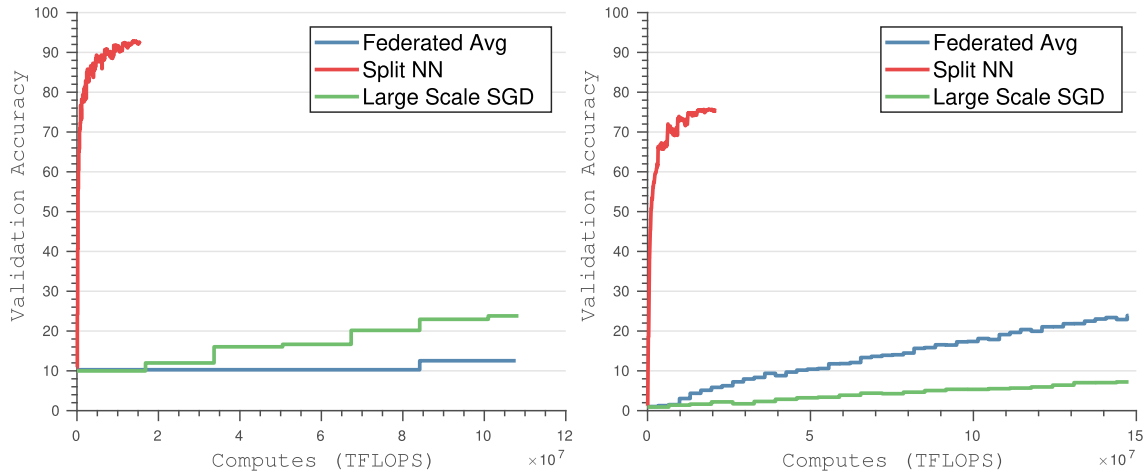
(a) Validation accuracy with client side flops when training 100 clients (VGG and CIFAR 10).

(b) Validation accuracy with client side flops when training 500 clients (Resnet-50 and CIFAR 100).

Fig. 3. Comparison of client side computational cost of our method against existing state of the art methods.

5.1. Comparison with existing methods

We compare our method against the modern state-of-the-art methods including large-batch global SGD (Chen et al.) and federated averaging approaches (McMahan et al.). We perform several different comparisons using the best hyperparameter selections for federated averaging and federated SGD. We compare client side computational costs when using deep models and demonstrate significantly lower computational burden on clients when training using our algorithm (see Fig. 3). We also analyze the transmission cost of state-of-the-art deep networks including ResNet and VGG on CIFAR-10 and CIFAR-100. We demonstrate higher validation accuracy and faster convergence when considering a large number of clients.

We demonstrate significant reductions in computation and communication bandwidth when comparing against federated SGD and federated averaging (McMahan et al.). Reduced computational requirements can be explained by the fact that while federated averaging requires forward pass and gradient computation for the entire neural network on the client, our method requires these computations for only the first few layers, significantly reducing the computational requirements (as shown in Fig. 3). Even though federated averaging requires a lot fewer iterations than large-scale SGD, it is still outperformed by our method requiring only a fraction of computations on the client.

Reduction in communication bandwidth can be attributed to the fact that federated averaging involves transmitting the gradient updates for the entire neural network from all clients to a central server, accompanied by transmission of updated weights to every single client (please refer to Fig. 4). While the federated averaging algorithm is able to converge in fewer transmission cycles, each transmission cycle requires huge amounts of data download and upload to the client and server. The split neural network algorithm reduces data transmitted by restricting the size of the client neural network to only the first few layers, thereby greatly reducing the total amount of data transmitted during training. Additionally, federated averaging fails to achieve optimal accuracy for higher numbers of clients since general non-convex optimization averaging models in parameter space could produce an arbitrarily bad model (phenomenon described in (Goodfellow et al.)).

5.2. Impact of amount of data on final accuracy

An important benefit of our method lies in its ability to combine multiple data-sources. When using deep neural networks, larger datasets have been shown to perform significantly better than smaller datasets. We experimentally demonstrate the benefits of pooling several agents by uniformly dividing dataset over 10 agents and training topologies using 1, 5 or 10 agents. We observe that adding more agents causes accuracy to improve significantly. Please see Table 2 for analysis on how accuracy will improve as we add more data sources in real world scenarios.

6. Conclusions and future work

In this paper we present new methods to train deep neural networks over several data repositories. We also present algorithms on how to train neural networks without revealing actual raw data while reducing computational requirements on individual data sources. We describe how to modify this algorithm to work in semi-supervised modalities, greatly reducing number of labeled samples required for training. We provide mathematical guarantees for correctness of our algorithm.

We devise a new protocol for easy implementation of our distributed training algorithm. We use popular computer vision datasets such as CIFAR-10 and ILSVRC12 for performance validation and show that our algorithm produces identical results to standard training procedures. We also show how this algorithm can be beneficial in low data scenarios by combining data from several resources. Such a method can be beneficial in training using proprietary data sources when data sharing is not possible. It can also be of value in areas such as biomedical imaging, when training deep neural network without revealing personal details of patients and minimizing the computation resources required on devices.

In this paper we describe a method to train a single network topology over several data repositories and a computational resource. A reasonable extension to this approach can be to train an ensemble of classifiers by transmitting forward and backward tensors for all classifiers every iteration. A deep neural network classifier ensemble can comprise several individual deep neural network topologies which perform classification. The network topologies are trained individually by computing forward and backward functions for each neural network, and during the testing phase the results are combined using majority vote to produce classification. We can train such an ensemble by generating separate forward and backward propagation tensors for each neural network and transmitting them during each training iteration. This is equivalent to training individual networks one by one, but it saves time by combining iterations of various networks together. Ensemble classifiers have also been shown to be more secure against network copy attacks and have also been shown to perform better in
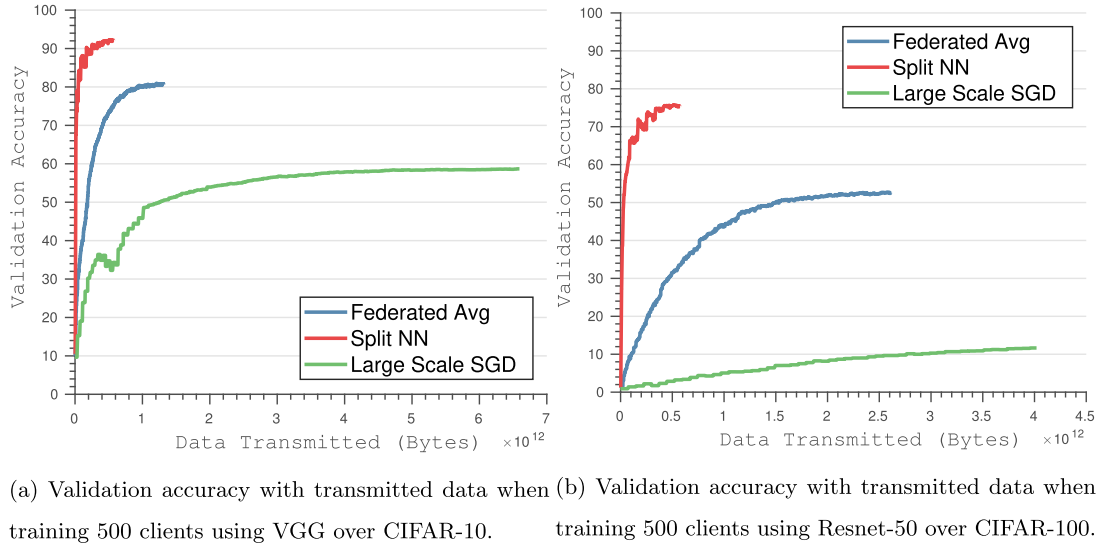
(a) Validation accuracy with transmitted data when training 500 clients using VGG over CIFAR-10.

(b) Validation accuracy with transmitted data when training 500 clients using Resnet-50 over CIFAR-100.

Fig. 4. Comparison of data transmission cost of our method against existing state of the art methods.

Table 2
Comparison on how accuracy improves as more data is added when training.

Dataset	Accuracy using 1 agent (10%)	Accuracy using 5 agents (50% of data)	Accuracy using all agents
MNIST	97.54	98.93	99.20
CIFAR 10	72.53	89.05	92.45
CIFAR 100	36.03	59.51	66.59
ILSVRC 12	27.1	56.3*	57.1

real world applications (Granitto et al., 2005).

In future work, a learned neural network could be shared using student-teacher methods for transferring information learned by neural network (Papernot et al.). After the training phases are over, Alice and Bob can use any publicly available dataset to train secondary (student) neural network using outputs from the primary (teacher) neural network. Alice can propagate the same training sample from the public dataset through the layers from the previously trained network and Bob can propagate them through its network. Bob can use the output of its layers to train the student network by doing forward-backward for the same data sample. This way, knowledge from the distributed trained network can be transferred to another network which can be shared for public use. Such algorithms can help in introducing deep learning in several areas such as health, products and finance where user data is an expensive commodity and needs to remain anonymized.

Tor like layer-by-layer computation could allow for training this network over multiple nodes with each node carrying only a few layers. Such a method could help protect not just the data but the identity of the person sharing the data and performing classification. In Tor like setup, additional entities $Eve_{0...M}$ are added which do not have access to data or complete network topology. Each Eve is provided with a few network layers $F_k^{eve} \leftarrow L_q, L_{q+1}...L_r$. During forward propagation Alice computes F_a and passes it to Eve_0, which then passes it to Eve_1 and so on until it reaches Eve_M. Eve_M is analogous to the exit node in Tor network and it passes the tensor to Bob. Similarly, when back-propagating, Bob computes *loss* and sends it to Eve_M, which sends it to Eve_{M-1} and so on until it reaches Eve_0 and then Alice. The onion like organization of network layers can be used to keep the identity of Alice confidential.

We can also apply our algorithm on not just classification tasks but also on regression and segmentation tasks. We can also use this over LSTMs and Recurrent Neural Networks. Such neural networks can be easily tackled by using a different loss function (euclidean) on Bob's side when generating gradients.

Appendix A. Supplementary data

Supplementary data related to this article can be found at http://dx. doi.org/10.1016/j.jnca.2018.05.003.

References

Agarwal, A., Duchi, J.C., 2011. Distributed delayed stochastic optimization. In: Advances in Neural Information Processing Systems, pp. 873–881.

Agarwal, A., Chapelle, O., Dudík, M., Langford, J., 2014. A reliable effective terascale linear learning system. J. Mach. Learn. Res. 15 (1), 1111–1133.

Avidan, S., Butman, M., 2006. Blind vision. In: European Conference on Computer Vision, pp. 1–13.

Baker, T., Mackay, M., Shaheed, A., Aldawsari, B., 2015. Security-oriented cloud platform for soa-based scada. In: Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on. IEEE, pp. 961–970.

Barni, M., Orlandi, C., Piva, A., 2006. A privacy-preserving protocol for neural-network-based computation. In: Proceedings of the 8th Workshop on Multimedia and Security, pp. 146–151.

Chen, K., Liu, L., A Random Rotation Perturbation Approach to Privacy Preserving Data Classification.

Chen, J., Monga, R., Bengio, S., Jozefowicz, R., Revisiting Distributed Synchronous Sgd, arXiv preprint arXiv:1604.00981.

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S., 2000. The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. J. Netw. Comput. Appl. 23 (3), 187–200.

Chonka, A., Xiang, Y., Zhou, W., Bonti, A., 2011. Cloud security defence to protect cloud computing against http-dos and xml-dos attacks. J. Netw. Comput. Appl. 34 (4), 1097–1107.

Chuang, J.C.-I., Sirbu, M.A., 2000. Distributed network storage service with quality-of-service guarantees. J. Netw. Comput. Appl. 23 (3), 163–185.

Coates, A., Karpathy, A., Ng, A.Y., 2012. Emergence of object-selective features in unsupervised feature learning. In: Advances in Neural Information Processing Systems, pp. 2681–2689.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al., 2012. Large scale distributed deep networks. In: Advances in Neural Information Processing Systems, pp. 1223–1231.

Dosovitskiy, A., Brox, T., Inverting Visual Representations with Convolutional Networks, arXiv preprint arXiv:1506.02753.

Goldreich, O., Micali, S., Wigderson, A., 1987. How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. 218–229.

Goodfellow, I.J., Vinyals, O., Saxe, A.M., Qualitatively Characterizing Neural Network Optimization Problems, arXiv preprint arXiv:1412.6544.

Granitto, P.M., Verdes, P.F., Ceccatto, H.A., 2005. Neural network ensembles: evaluation of aggregation algorithms. Artif. Intell. 163 (2), 139–162.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T., Caffe: Convolutional Architecture for Fast Feature Embedding, arXiv preprint arXiv:1408.5093.

Karam, Y., Baker, T., Taleb-Bendiab, A., 2012. Security support for intention driven elastic cloud computing. In: Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on. IEEE, pp. 67–73.

Karpathy, A., Fei-Fei, L., 2015. Deep visual-semantic alignments for generating image descriptions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3128–3137.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105.

Langford, J., Smola, A.J., Zinkevich, M., 2009. Slow learners are fast. Adv. Neural Inf. Process. Syst. 22, 2331–2339.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. Neural Comput. 1 (4), 541–551.

Mackay, M., Baker, T., Al-Yasiri, A., 2012. Security-oriented cloud computing platform for critical infrastructures. Comput. Law Secur. Rep. 28 (6), 679–686.

Mcdonald, R., Mohri, M., Silberman, N., Walker, D., Mann, G.S., 2009. Efficient large-scale distributed training of conditional maximum entropy models. In: Advances in Neural Information Processing Systems, pp. 1231–1239.

McMahan, H.B., Moore, E., Ramage, D., Hampson, S., et al., Communication-efficient Learning of Deep Networks from Decentralized Data, arXiv preprint arXiv:1602.05629.

Orlandi, C., Piva, A., Barni, M., 2007. Oblivious neural network computing via homomorphic encryption. EURASIP J. Inf. Secur. 2007, 18.

Papernot, N., Abadi, M., Erlingsson, Ú., Goodfellow, I., Talwar, K., Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data, arXiv preprint arXiv:1610.05755.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A., Context Encoders: Feature Learning by Inpainting, arXiv preprint arXiv:1604.07379.

Secretan, J., Georgiopoulos, M., Castro, J., 2007. A privacy preserving probabilistic neural network for horizontally partitioned databases. In: 2007 International Joint Conference on Neural Networks, pp. 1554–1559.

Shi, Q., Petterson, J., Dror, G., Langford, J., Strehl, A.L., Smola, A.J., Vishwanathan, S., 2009. Hash kernels. In: International Conference on Artificial Intelligence and Statistics, pp. 496–503.

Shin, H.-C., Orton, M.R., Collins, D.J., Doran, S.J., Leach, M.O., 2013. Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4d patient data. IEEE Trans. Pattern Anal. Mach. Intell. 35 (8), 1930–1943.

Simonyan, K., Zisserman, A., Very Deep Convolutional Networks for Large-scale Image Recognition, arXiv preprint arXiv:1409.1556.

Sood, S.K., 2012. A combined approach to ensure data security in cloud computing. J. Netw. Comput. Appl. 35 (6), 1831–1838.

Subashini, S., Kavitha, V., 2011. A survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. 34 (1), 1–11.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., 2010. Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. 11, 3371–3408 (Dec).

Weston, J., Ratle, F., Mobahi, H., Collobert, R., 2012. Deep learning via semi-supervised embedding. In: Neural Networks: Tricks of the Trade. Springer, pp. 639–655.

Wu, B., Wu, J., Fernandez, E.B., Ilyas, M., Magliveras, S., 2007. Secure and efficient key management in mobile ad hoc networks. J. Netw. Comput. Appl. 30 (3), 937–954.

Yao, A.C.-C., 1986. How to generate and exchange secrets. In: Foundations of Computer Science, 1986., 27th Annual Symposium on, pp. 162–167.

Zhang, Y., Zhong, S., 2013. A privacy-preserving algorithm for distributed training of neural network ensembles. Neural Comput. Appl. 22 (1), 269–282.

Zinkevich, M., Weimer, M., Li, L., Smola, A.J., 2010. Parallelized stochastic gradient descent. In: Advances in Neural Information Processing Systems, pp. 2595–2603.
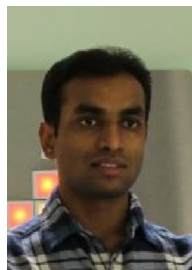
Otkrist Gupta is a Ph.D. candidate at MIT Media Lab. He works at camera culture, his research is focused on inventing new algorithms for deep learning for health screening and diagnosis, hidden geometry detection, exploiting techniques from optimization, linear algebra and compressive sensing. He also works on designing algorithms for futuristic 3D projective displays. Before joining MIT Media Lab Otkrist worked in Google Now team where he built voice actions such as take a picture and what's on my Chromecast and worked on voice response quality from Google Now. He also worked in LinkedIn where he developed services such as Smart ToDo, Ultra fast auto-complete, Notifications and CheckIn platform. He completed his bachelors from Indian Institute of Technology Delhi (IITD) in Computer Science with emphasis on algorithms and linear algebra. After graduating from IITD, he worked for one year in field of High Frequency Trading at Tower Research Capital.

Ramesh Raskar joined the Media Lab from Mitsubishi Electric Research Laboratories in 2008 as head of the Labs Camera Culture research group. His research interests span the fields of computational photography, inverse problems in imaging and humancomputer interaction. Recent projects and inventions include transient imaging to look around a corner, a next generation CAT-Scan machine, imperceptible markers for motion capture (Prakash), long distance barcodes (Bokode), touch+hover 3D interaction displays (BiDi screen), low-cost eye care devices (Netra,Catra), new theoretical models to augment light fields (ALF) to represent wave phenomena and algebraic rank constraints for 3D displays(HR3D).