Distributed Asynchronous Domain Adaptation: Towards Making Domain Adaptation More Practical in Real-World Systems

*Shaoduo Gan[1], *Akhil Mathur[2,3], Anton Isopoussu[2], Nadia Berthouze[3], Nicholas D. Lane[4], and Fahim Kawsar[2]

[1]Department of Computer Science, ETH Zurich
[2]Nokia Bell Labs Cambridge
[3]University College London
[4] University of Oxford
sgan@inf.ethz.ch, akhil.mathur@nokia-bell-labs.com,
anton.isopoussu@nokia-bell-labs.com, nadia.berthouze@ucl.ac.uk,
nicholas.lane@cs.ox.ac.uk, fahim.kawsar@nokia-bell-labs.com

(* denotes equal contribution)

Abstract

Despite the rapid advancements in the field of unsupervised domain adaptation (uDA), there remains a wide gap between algorithmic solutions and their manifestation in real-world ML systems. In this paper, we – for the first time – explore the challenges associated with privacy, distributed datasets and communication costs when uDA algorithms are deployed in practice. We propose and evaluate a novel distributed and asynchronous algorithm for domain adaptation which can alleviate several of the aforementioned practical challenges and pave the way for wider adoption of these algorithms in practical applications.

1 Introduction

A fundamental assumption that drives the performance of supervised learning algorithms is that training and testing samples are drawn from the same data distribution. However, in practical scenarios, this assumption is often violated due to variations in data acquisition processes between the training (or source) and testing (or target) domains – caused by for example, different illumination conditions and cameras in the context of visual tasks. This shift in data distributions, known as domain shift, is a core reason which hinders the generalizability of predictive models to new domains. As manual labeling of data in each target domain is prohibitively expensive, *unsupervised domain adaptation* (uDA) has emerged as a promising solution to transfer the knowledge from a labeled source domain to unlabeled target domains. A number of methods have been proposed for uDA [1, 2, 3, 4, 5]. In particular, adversarial training approaches have shown significant promise in learning representations which are domain-invariant and can provide more effective transfer from source to target domain [6, 7, 5, 8].

While existing methods have focused on improving the accuracy of target classifier after adaptation, little attention has been paid to the incorporation of these domain adaptation methods in real-world machine learning systems. These methods assume that both source and target domain data are available on the same machine in order to conduct the joint adaption training, which is not always the case in reality. As a motivating example, consider a medical application scenario wherein a model is trained for the task of fetal head detection from labeled ultrasound images collected in a hospital in

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Finland (source domain S_{fin}). Subsequently, this model has to be deployed in two target hospitals: one in the USA (T_{usa}) and one in China (T_{chi}). Due to variations in sonogram machines and medical training of sonographers, a domain shift is likely to occur in the test samples; hence warranting the need for uDA. Existing uDA methods require the raw source and target data to be available on the same node. Clearly, this raises severe privacy and legal concerns since either the source domain or the target domains will have to send their sonograms to each other in order to perform uDA. In addition, such transfer of data also incurs severe communication costs. In summary, despite the rapid advancements in uDA algorithms, existing methods are not designed to solve practical challenges of distributed data, privacy and communication costs in real-world systems.

Given the gap between the latest domain adaption techniques and distributed learning systems, we consider the distributed setting for unsupervised domain adaption where the labeled source data and unlabeled target data are stored separately within two computational nodes. We explore how to develop distributed adversarial domain adaption methods without sharing the domain data between nodes. The key contributions of this paper are:

- We propose a novel distributed and asynchronous domain adaptation algorithm which divides the model architecture of an adversarial network across different compute nodes.
- We propose a new training strategy for this distributed architecture which does not exchange any raw data or feature representations across nodes. Instead, we propose accumulating and periodic averaging of the discriminator gradients as a way to synchronize the training process across nodes.
- We evaluate our methods on three benchmark uDA tasks and show that it outperforms several distributed training baselines.

2 Related Work

In our paper, the training data from two domains is distributed in two separate nodes, which relates our problem to distributed learning algorithms. Parameter Server [9] is a common strategy for distributed learning which coordinates the gradients of all nodes to update the model in a centralized way. Lian [10] proposed a decentralized learning system, in which each node conducts model averaging with only connected neighbors. Besides, in order to save communication costs, a number of asynchronous algorithms have been proposed [11, 12]. Compared with previous works, this paper explores the distributed domain adaption scenario, which, to the best of our knowledge, has not been studied before. It is a special case in the sense that training data split across the two nodes is biased, i.e., data in each node belongs to one domain. Moreover, it is unclear from past works how to distribute the adversarial training model across nodes.

3 Technique

In this section, we present our proposed approach of distributed asynchronous domain adaptation.

3.1 Preliminary

We tackle the problem space of unsupervised domain adaption (uDA). In uDA, the training data consists of a labeled source dataset $S_s = \{(x_s, y_s)\}$ drawn from a source domain $\mathcal{X}_s$ and an unlabeled target dataset $S_t = \{(x_t)\}$ drawn from a target domain $\mathcal{X}_t$, with no labeled observations. Because direct supervised learning is not possible on the target dataset, uDA methods aim to adapt a source model for use in the target domain.

Our solution is focused on Representation-level Adversarial Domain Adaption (RADA), a class of uDA methods that have achieved many exciting results recently [7, 5, 6, 13]. The core intuition in RADA is to use adversarial learning to align the feature representations of the source and target domains, thereby allowing a source classifier to be used in the target domain. Broadly, there are three components in RADA and its variants: a *feature extractor* E, a *task classifier* C and a *domain discriminator* D. E takes training instances as input and generates feature representations for them, which sequentially are passed as input to C and D. The classifier C conducts the task classification while D aims to differentiate the two domains based on their feature representations. The goal of domain adaptation is to learn a E and a C that result in low target error $\mathcal{E}_t$:

$$\mathcal{E}_t(E, C) = \Pr_{(x,y)\sim\mathcal{X}_t}[C(E(x)) \neq y] \tag{1}$$

Two losses are optimized during the training process: classification loss $\mathcal{L}_{cla}$ and adversarial loss $\mathcal{L}_{adv}$. The former one refers to the supervised task classification loss over the source dataset. The latter one represents the domain discrimination loss over source and target data – this is computed by assigning 0 and 1 as domain labels for source and target domains respectively. The two losses can be formalized as follows: (note L is the loss function)

$$\mathcal{L}_{cla} = \sum_{(x_s,y_s)\sim S_s} L(C(E(x_s)), y_s) \tag{2}$$

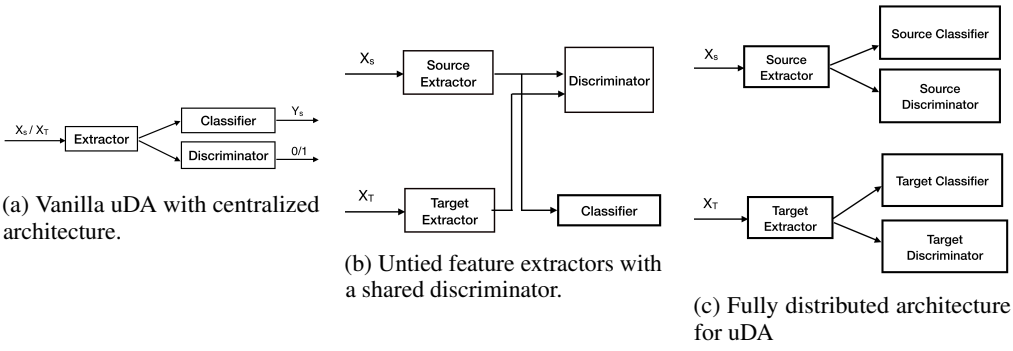
$$\mathcal{L}_{adv} = \sum_{x_s\sim S_s} L(D(E(x_s)), \mathbf{0}) + \sum_{x_t\sim S_t} L(D(E(x_t)), \mathbf{1}) \tag{3}$$

Given $\mathcal{L}_{cla}$ and $\mathcal{L}_{adv}$, the adversarial domain adaption training is a minimax game. Feature extractor E is trained to minimize $\mathcal{L}_{cla}$ and maximize $\mathcal{L}_{adv}$ at the same time, because it is supposed to learn the representation that is informative enough for classification but domain-invariant for D. Whereas, domain discriminator D tries to minimize $\mathcal{L}_{adv}$ such that it can still differentiate the representations of the two domains. Task classifier C is updated to minimize $\mathcal{L}_{cla}$. The optimization problems can be stated as:

$$\min_{E,C} \mathcal{L}_{cla}, \min_{D} \mathcal{L}_{adv}, \max_{E} \mathcal{L}_{adv}. \tag{4}$$

3.2 Distributed Domain Adaption

As discussed above, the training process of RADA methods follows the "one model, two data flows" paradigm, that is it requires both source and target datasets to be on the same node in order for the adaptation to work. Recall that our goal is to come up with a system design that allows domain adaptation to work without exchanging source and target datasets or their features representations, in order to make it practical from a privacy and data communication perspective. As source and target datasets reside at separate nodes in our setting, a logical option would be to split the model across the nodes. In vanilla RADA as shown in Figure 1(a), the three model components E, C and D are shared by both domains and reside on the same node. Prior works such as ADDA [7] shown in Figure 1(b) proposed to use different feature extractors for the source and target domains, however they still use a shared domain discriminator, which makes them impractical for our problem.



(a) Vanilla uDA with centralized architecture.

(b) Untied feature extractors with a shared discriminator.

(c) Fully distributed architecture for uDA

Figure 1: Network architectures for Adversarial Domain Adaptation.

We propose a fully distributed architecture for adversarial domain adaption. As shown in Figure 1(c), E, C and D are split into source components (E_s, C_s, D_s) and target components (E_t, C_t, D_t) respectively. Source data and target data can therefore be fed separately into their own model components, thus preventing any exchange of raw data or feature representations across nodes. Under this architecture, the most straight-forward training strategy is to aggregate the gradients of two discriminators for every step. From an adaptation perspective, this is equivalent to the centralized

Algorithm 1 DADA (Source Node)

Input: E_s and C_s, which are pre-trained on source dataset S_s; randomly initialized D_s; sync-up step p; number of total steps N
for $n = 1, 2, ..., N$ **do**
  Randomly sample a batch $\xi_s^{(n)}$ from source dataset S_s
  Feed $\xi_s^{(n)}$ to E_s and D_s, get adversarial loss $\mathcal{L}_{adv}^s = L(D_s(E_s(\xi_s^{(n)})), \mathbf{0})$.
  Calculate gradients over D_s: $g^{(n)}(D_s; \mathcal{L}_{adv}^s)$
  Add $g^{(n)}(D_s; \mathcal{L}_{adv}^s)$ to the accumulated gradients buffer G_{acc}^s
  if $n\%p == 0$ **then**
    Exchange G_{acc}^s with target node, average the accumulated gradients , then update D_s and clear buffer G_{acc}^s.
  end if
end for

Algorithm 2 DADA (Target Node)

Input: E_t, C_t and D_t, which are all initialized by source node; sync-up step p; number of total steps N
for n = 1,2,...,N **do**
  Randomly sample a batch $\xi_t^{(n)}$ from target dataset S_t
  Feed $\xi_t^{(n)}$ to E_t and D_t, get adversarial loss $\mathcal{L}_{adv}^t = L(D_t(E_t(\xi_t^{(n)})), \mathbf{1})$.
  Calculate gradients over E_t and D_t: $g^{(n)}(E_t; \mathcal{L}_{adv}^t)$ and $g^{(n)}(D_t; \mathcal{L}_{adv}^t)$
  Update E_t by $g^{(n)}(E_t; \mathcal{L}_{adv}^t)$
  Add $g^{(n)}(D_t; \mathcal{L}_{adv}^t)$ to the accumulated gradients buffer G_{acc}^t
  if n%p == 0 **then**
    Exchange G_{acc}^t with source node, average the accumulated gradients, then update D_t and clear buffer G_{acc}^t.
  end if
end for

uDA methods such as ADDA [7] where E_s and E_t are trained sequentially by source and target data, and only D is trained jointly by two domains. Even though we distribute the discriminator training across nodes, by averaging the discriminator gradients after each step, we ensure that the two discriminators are always synchronized, thereby this method obtains similar levels of adaptation performance as centralized training. Please refer to § 4 for the empirical findings that justify this approach. Besides, it is worth mentioning that most existing distributed training methods [9, 10, 14] are not applicable in this setting because the domain label spaces across the nodes are completely disjoint (i.e. source node has only samples from class 0 and target node has samples only from class 1).

While this approach of synchronously updating the discriminators can guarantee the statistical accuracy for the proposed distributed domain adaption, it does require gradients exchange between nodes after every single training step, which in turn can lead to significant communication costs over long training processes. In the next section, we relax the synchronization constraint, i.e., we propose that source and target nodes exchange discriminator gradients after every n steps, thereby significantly reducing the communication costs at the expense of minimal accuracy loss.

3.3 Distributed Asynchronous Domain Adaption

The communication content between source and target nodes are gradients of D_s and D_t. Note that the target extractor E_t is always updated for every step regardless of the communication pattern of discriminators. In the synchronous case, D_s and D_t are always identical because they are initialized with the same weights and updated with same averaged gradients after every step. For asynchronous gradient exchange, we explore three strategies that are summarized in Table 1.

Table 1: Summary of the three asynchronous strategies studied in this paper.

	In Local Step	**In Sync-up Step**	**Expected Divergence between D_s and D_t**
Naive	Apply local gradients to D_s and D_t	Average weights of D_s and D_t	High
Moving Average (MA)	Apply local gradients to D_s and D_t, then do average with D_{sync}	Average weights of D_s and D_t, store the averaged model D_{sync}	Low
Gradient Accumulating (GA)	Accumulate the gradients of D_s and D_t	Average accumulated gradients and apply it to D_s and D_t	None

We refer to the training step at which the gradient synchronization takes place as the *Sync-up Step* while the other steps during which both nodes are computing the gradients locally are called *Local Steps*. In the *Naïve* method – during the local steps, both D_s and D_t are updated based on the gradients computed over their local datasets. In the sync-up step, the weights of the two discriminators are exchanged and averaged as a way to synchronize them. As we mentioned before, since the label spaces of the two discriminators are completely disjoint, it is likely that averaging models trained on such biased data will degrade the discriminator performance.

Structure	Sync.	Sync Step	Method	0 → 60	0 → 120	0 → 180	0 → 240	0 → 300	Avg
Centralized	-	-	-	69.86	29.89	59.95	49.58	38.2	49.496
Distributed	Sync	-	-	69.84	29.42	59.12	49.45	38.40	49.246
Distributed	Async	2	**Naive**	45.98	24.95	45.43	38.15	31.67	37.236
			MA	65.29	**29.86**	56.58	38.13	**38.01**	45.574
			GA	**68.73**	29.39	**59.82**	**48.12**	37.25	**48.652**
Distributed	Async	3	**Naive**	50.86	23.86	41.09	26.39	29.48	34.336
			MA	49.77	27.0	47.20	27.81	38.10	37.976
			GA	**68.43**	**29.82**	**59.70**	**45.34**	**38.56**	**48.37**
Distributed	Async	4	**Naive**	33.17	22.71	40.02	20.67	22.96	27.906
			MA	46.21	21.27	44.95	24.61	33.30	34.068
			GA	**68.29**	**29.31**	**59.40**	**41.81**	**37.95**	**47.352**

Table 2: Target Domain Accuracy (%) for various distributed and asynchronous approaches for Rotated MNIST

Structure	Sync.	Sync Step	Method	A → W	W → D	D → W	Avg	M → U	U → M	S → M	Avg
Centralized	-	-	-	77.61	98.18	92.45	89.39	90.95	95.51	69.61	85.35
Distributed	Sync	-	-	77.23	98.0	92.01	89.08	90.73	95.59	69.55	85.29
Distributed	Async	2	**Naive**	**76.60**	97.38	89.05	87.67	65.4	71.32	55.65	64.12
			MA	76.47	96.58	89.18	87.41	69.0	70.5	57.1	65.53
			GA	76.35	**97.8**	**90.94**	**88.36**	**91.22**	**95.31**	**67.87**	**84.8**
Distributed	Async	3	**Naive**	75.84	96.38	87.5	86.57	62.1	70.44	54.46	62.33
			MA	76.72	96.78	88.67	87.39	67.1	70.32	57.1	64.84
			GA	**77.10**	**97.18**	**90.56**	**88.28**	**90.81**	**79.62**	**67.1**	**79.17**
Distributed	Async	4	**Naive**	75.47	94.77	87.16	85.8	62.0	70.34	54.63	62.32
			MA	76.35	96.98	88.55	87.29	66.23	69.3	55.25	63.59
			GA	**76.85**	**97.03**	**89.05**	**87.64**	**90.31**	**75.35**	**65.39**	**77.01**

Table 3: Target Domain Accuracy (%) for distributed and asynchronous approaches for Office-31 and Digits adaptation tasks.

To mitigate the effects of biased gradients, we use the *Moving Average* method which aims to minimize the divergence between the two discriminators in the local steps. The key idea is to store a copy of the last synchronized discriminator (called D_{sync}) from the previous Sync-up step and after each local update of the discriminators, average the weights of the local discriminators with weights of D_{sync}. This technique prevents the weights of the two discriminators from diverging significantly from the last synchronized model.

Finally, we propose *Gradient Accumulation* as a simple but effective method to ensure there is no divergence between the discriminators, and at the same time we are able to benefit from asynchronous training. As the consistency of D_s and D_t are crucial for the adversarial training, we do not update discriminators with local biased gradients. Instead, we only accumulate the gradients during the local steps. In the sync-up step, both nodes exchange the accumulated gradients, average them and apply them to update the discriminators. The target extractor E_t is always updated for every step. Our experiment results in § 4 show that the Gradient Accumulation method provides the best adaptation performance as the asynchronicity of the system increases.

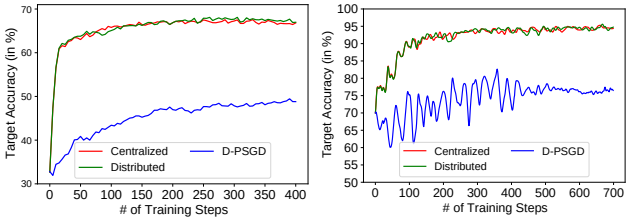
Combining the distributed training with the Gradient Accumulation technique, we propose a distributed asynchronous domain adaption (DADA) algorithm, which is summarized in Algorithm 1 and 2. In these algorithms, $p = 1$ corresponds to the synchronous distributed training discussed in §3.1.

4 Evaluation

This section reports on the experimental validation of our approach on multiple image datasets.

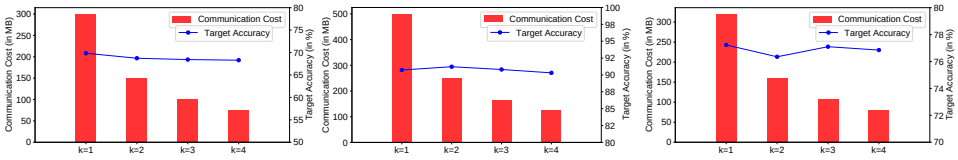
Datasets. We conduct experiments on three widely used uDA datasets: Rotated MNIST, Digits, and Office-31. *Rotated MNIST* is a variant of the MNIST [15] dataset wherein the numbers are rotated from 0°to 300°at increments of 60°. Each rotation is considered as a separate domain and hence we have 6 domains in total (0°, 60°, 120°, . . . , 300°). The *Digits* adaptation task has three domains: MNIST [15], USPS, and SVHN [16] and each domain consists of 10 digit classes ranging from 0-9. Finally, the *Office-31* dataset contains images of everyday office objects from 31 classes, but collected from different sources: Amazon (2817 images), DSLR camera (498 images) and a Web camera (795 images). We refer to these domains as A, D and W respectively.

Experiment Setup. We follow the same evaluation procedure as earlier uDA works [5, 7] wherein all training instances are used for adaptation and the adapted model is evaluated on the target test data. We choose three adaptation tasks each for the Digits and Office-31 datasets as shown in Table 3.



(a) Rotated MNIST (0°→60°) (b) Digits (USPS→MNIST)

Figure 2: Comparison of target domain accuracy across centralized and distributed training methods.



(a) Rotated MNIST (0°→60°) (b) Digits (MNIST→USPS) (c) Office-31 (A→W)

Figure 3: Communication Cost across different methods.

For the rotated MNIST dataset, without loss of generality, we assume 0°as the labeled source domain and remaining 5 rotations as target domains, leading to a total of 5 adaptation tasks.

Results. In Figure 2, we plot the target domain accuracy with the number of training steps, and compare our approach (without adding asynchronization) against a centralized baseline (ADDA [7]) and a state-of-the-art decentralized training algorithm D-PSGD [10]. D-PSGD is proposed to conduct decentralized training among multiple nodes by averaging the model only between the connected neighbors. Here there are only two nodes, therefore we apply the model averaging strategy of D-PSGD between source and target discriminators. The results clearly demonstrate that our distributed training approach (without asynchronization) can achieve similar target accuracy as the centralized baseline, and moreover it outperforms D-PSGD on all datasets.

Next, we evaluate our proposed approach of gradient-accumulation based asynchronization against a number of baseline techniques. Tables 2 and 3 show the target domain accuracy post-adaptation, comparing the centralized ADDA method against our distributed technique as we vary the number of sync-up steps from 1 to 4. Sync up step of 1 corresponds to the fully synchronized training. For Rotated MNIST, we observe that the Gradient Accumulation approach provides comparable accuracy to fully synchronous training, and also outperforms the other async baselines. Similar results are obtained for the Digits and Office-31 datasets.

Finally, we evaluate the total communication cost in various training strategies. As discussed before, the primary communication overhead in our proposed solution is the exchange of discriminator gradients at the sync-up step. We calculate the total data transfer over the entire training process and report it in Figure 3. It is observed that as the sync-up step (k) increases (on the x-axis), the communication cost goes down with minimal drop in target domain accuracy. It is also worth highlighting that in our experiments, we did not have very large-scale datasets that are expected in real-world. Existing domain adaptation algorithms require data exchange, hence their communication cost will scale with dataset size, however our communication costs are dependent on the size of the discriminator and hence largely independent of the dataset size.

5 Conclusion

In this paper, we presented the challenges of deploying uDA algorithms in practice. We proposed a distributed and asynchronized algorithm which allows for privacy-preserving uDA on distributed datasets. As this is an early work, we did not perform extensive hyper-parameter search, which might have improved the performance. We hope to get feedback on this idea from fellow workshop attendees, which will help us in making the paper stronger.

Finally, our method provides user privacy by only shares the gradients of the domain discriminators between nodes. The raw data from both domains as well as its corresponding gradients from the feature extractor are completely never shared between nodes. While no prior work has shown that discriminator gradients can leak raw data, we do not discount the possibility that privacy attacks could be developed in the future. We leave a detailed privacy study of our method as a future work.

References

[1] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

[2] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2208–2217. JMLR. org, 2017.

[3] Pedro O Pinheiro. Unsupervised domain adaptation with similarity learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8004–8013, 2018.

[4] Swami Sankaranarayanan, Yogesh Balaji, Carlos D Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2018.

[5] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International Conference on Machine Learning*, pages 1994–2003, 2018.

[6] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[7] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176, 2017.

[8] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

[10] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5330–5340, 2017.

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

[12] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[13] Zhangjie Cao, Lijia Ma, Mingsheng Long, and Jianmin Wang. Partial adversarial domain adaptation. In *European Conference on Computer Vision*. Springer, 2018.

[14] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[16] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.