Short Paper

DataGossip: A Data Exchange Extension for Distributed Machine Learning Algorithms

Phillip Wenig
phillip.wenig@hpi.de
Hasso Plattner Institute (HPI), University of Potsdam
Potsdam, Germany

Thorsten Papenbrock
papenbrock@informatik.uni-marburg.de
Philipps Universität Marburg
Marburg, Germany

ABSTRACT

Distributed, data parallel machine learning algorithms frequently exchange the gradients of their locally trained models to converge to a consistent global model. This distributed training works reasonably well if the training classes are partitioned evenly. In practice, however, the data partitions are often skewed for various reasons. The local models then strive for different local optima and start to compete rather than collaborate, which spoils the global model. The general solution to this problem is data shuffling, but due to predetermined data placements or physical constraints re-allocating almost all training records in the cluster for some machine learning task is often not possible.

To minimize the negative impact of skewed data distributions, we propose DataGossip, a novel extension for distributed machine learning algorithms. DataGossip runs as a background process and dynamically exchanges relevant training instances between the different cluster nodes. We investigate how many training instances a cluster node needs to exchange, which training instances it should select, and how much foreign training instances need to be emphasized to impact the training positively. Our evaluation shows that DataGossip can improve the accuracy of models that have been trained on skewed data distributions. It also motivates further research on asynchronous and reactive data exchange systems for distributed learning scenarios.

1 INTRODUCTION

The popularity of *distributed machine learning* algorithms increases steadily not only because long training times demand for parallel computing but also because large, pre-partitioned datasets force the training into a distributed setting. Hence, data scientists often need to train a model for a specific task, such as classification, prediction, scoring, or detection, on a training dataset that is scattered across several compute nodes. A widely used solution for such situations is to train multiple models in a data parallel fashion on different, isolated processors that each own a partition of the training data; during the training, the local models frequently exchange their model weights (or gradients) in order to converge to a single global model. The weight exchange can be implemented synchronized via barriers between training steps or asynchronously for better performance and resource utilization; it can further be based on peer-to-peer communication or a dedicated leader node, such as a parameter server.

Regardless of the training subtleties, data parallel training approaches that are based on gradient/weight exchanges usually assume that the classes, events or other training objectives are (at least approximately) evenly partitioned in the cluster. If the data distribution is skewed, i.e., the training classes are represented unevenly over the various data partitions, the local models strive for (very) different local optima that might distort both the global model and other local models when updates are executed.

In practice, skewed data distributions often arise naturally through *recording bias* (e.g. grouping by sensor, department or user), *time bias* (e.g. grouping by experiment, transaction or session), and *semantic bias* (e.g. grouping by sensitivity, size or ownership). The bias is then also reflected in the training classes. For example, chairs and baskets are more common on street cameras than on highway cameras [14], the expression of hate speech and sentiment is very characteristic to certain communities [19, 20] and mammal recordings from wildlife cameras vary depending on the location of the cameras [7]. A commonly applied solution to overcome the class bias is *training data randomization*. Unfortunately, many setups prohibit the shuffling of the training data, because physical, political and/or operational constraints tie the partitions to their specific locations. If the peers then cannot afford to store a second, randomly shuffled partition due to memory constraints (e.g. cloud setups with pay-per-use payment model), full randomization is not possible and the skew prevails.

Biased training data distributions are a well researched issue in Federated Learning (FL) research where the algorithms try to fight the class skew with sophisticated parameter aggregation algorithms. In federated setups, the data partitions are usually strictly private and may, therefore, not be exchanged. For the distributed machine learning extension proposed in this paper, we break with this assumption and assume that training records may temporarily be shared, which is a valid assumption for most of the scenarios described above. While the partitions stay on the machines that they are supposed to be stored on, a background process dynamically selects currently most relevant training records and shares them with all other training nodes to mitigate the class bias. Because the foreign training records are few in number, they are weaved into the local training records in such a way that they receive sufficient emphasize during training.

This research project investigates (1) how *relevant* training records for the sharing process can be identified, (2) how *many* records each gossip cycle needs to share, and (3) how much *emphasis* each local trainer must put on foreign records. As a first solution, we propose DataGossip, a non-invasive extension for distributed machine learning algorithms that counteracts class imbalances during training by dynamically sharing training records. DataGossip encapsulates automatic instance selection, asynchronous record exchange and transparent record weaving into a separate process that does not change existing machine learning algorithms. The extension primarily targets the training of non-convex models, such as neural networks, with gradient descent-based learning techniques. Our experiments demonstrate that DataGossip effectively improves the learning accuracy on skewed data distributions.

2 RELATED WORK

DataGossip is an extension for *distributed, data parallel machine learning* algorithms. Such algorithms train the same model architectures on multiple nodes that each hold a unique partition of the entire training dataset. To make progress, the local models are frequently synchronized by sharing model weights/gradients.

Distributed machine learning. Synchronous gradient update techniques for distributed machine learning algorithms introduce blocking barriers after each training step, where a master process collects, aggregates and shuffles training results. Such techniques have been implemented in, for instance, map-reduce [3] and PyTorch [17]. Asynchronous gradient update techniques, on the contrary, decouple the model weight updates from the actual training processes. A popular representative of this class is Hogwild! [18] and its distributed version Hogwild++ [22]. To spread learning results in the cluster, Hogwild++ circulates a weight token that gets updated with local model weights by every training node that it passes. Inspired by this idea, our DataGossip extension also sends a token in a ring of training nodes, but with training instances instead of model weights. Another popular distributed and asynchronous machine learning approach is *DownpourSGD* [5]. This algorithm uses a parameter server that asynchronously (but in some predefined frequency) receives model updates, aggregates the updates into a global model and, then, broadcasts the global model's updates to all local trainees. Our DataGossip extension is a background process that works well in conjunctions with any of the existing distributed learning techniques regardless of whether they are synchronous or asynchronous. In our evaluation, though, we picked DownpourSGD as an example training algorithm to demonstrate DataGossip.

Dynamic data sharing. While most distributed machine learning algorithms exchange only model weights or gradients, a few federated learning [10] algorithms exist that also exchange data. For example, a variant of FL [8] trains a generative model while sharing underrepresented class labels to handle the imbalance. The proposed sharing strategy essentially trains a separate model to learn about underrepresented classes. This technique is basically an alternative instance selection option (also for DataGossip), but it is far more compute intensive than the techniques that we consider and needs to be integrated deeper into the training process. Another FL-related algorithm [23] copes with imbalanced data by sharing small sampled datasets amongst their workers, which generates a small random overlap between partitions. In contrast to DataGossip, this overlap is static and does not adapt to the current state of the training.

Instance selection. A central component of DataGossip is its strategy of selecting local training instances that appear to be relevant for the training and should, therefore, be shared with other trainees. The same task is also solved by *instance selection* algorithms, which identify representative training instances to reduce the amount of training data, and by *curriculum learning* algorithms, which re-order the training instances in a way that improves the learning efficiency. In both areas, *random sampling* is usually the default. Hard Example Mining (HEM) and Easy Example Mining (EEM) [16] are two more advanced techniques that consider an already (partially) trained model to calculate the model's classification *loss* (or error) for all training instances. The techniques then rank the instances by their losses to choose either the top-k *highest* or *lowest* ranked instances for training. Self-Paced Learning (SPL) [12] is a more dynamic technique that judges the *difficulty* of training records throughout the entire training process. The idea is inspired by the observation that a machine learning model learns more effectively from easier data instances in the beginning and ever harder instances towards the end of the training process [1]. Hence, SPL puts a threshold on the difficulty of selected training instances and increases this threshold as the training progresses. Active Bias (AB) [2] is another dynamic technique that selects training records by their *loss variance* during training. Driven by the observation that particularly easy instances (as in EEM and SPL) are ineffective during training and particularly hard instances (as in HEM) are too noisy, AB proposes to select instances by their loss variance, which indicates instances whose classification was impacted the most by the local training. Because DataGossip works with any of the proposed instance selection techniques, we evaluate all of them in this paper.

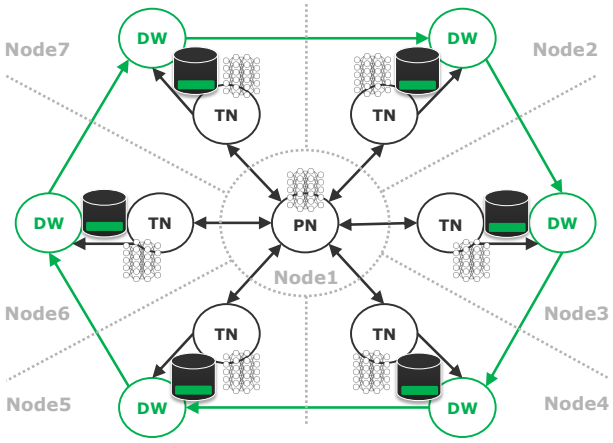

Figure 1: An example setup with a DownpourSGD *Parameter Node* (PN) and six *Training Nodes* (TN) extended with *DataGossip Workers* (DW)

3 DATA GOSSIP

DataGossip is an extension for a variety of distributed, data-parallel machine learning algorithms. It requires access to (1) the training data, (2) the losses of the training records, and (3) some signal as an indicator for when the local training process has finished a training iteration. Any distributed training method that can offer these elements at runtime can use DataGossip. For demonstration purposes, we use DownpourSGD [5] in this paper.

As visualized in Figure 1, a DownpourSGD cluster consists of a *parameter node* that maintains the global model and several *training nodes* that train local models on local partitions of the training data. The parameter node initializes the model and broadcasts its weights to the training nodes. Each training node owns a unique, potentially skewed partition, on which it trains its local model on. After a certain amount of iterations, each DownpourSGD training node sends its local model updates asynchronously to the parameter node. The parameter nodes updates the global model and, then, sends the aggregated global model weights back to the training node.

DataGossip introduces a second process to the learning setup on every training node. This process hosts a DataGossip worker that runs in parallel to the training process. Both the DataGossip worker and DownpourSGD's training node have access to the training data. In the following, we describe the gossip algorithm and how training records are selected, sized and weaved.

DataGossip algorithm. DataGossip expects the training nodes to log the most recent losses for each local training record into a shared memory buffer (a shared PyTorch tensor). After each local training epoch, i.e., every time all local losses got updated, the DataGossip worker receives a signal to attempt a new round of data exchange. The data exchange itself is implemented using the Gloo library [6] and the *ring-allgather* protocol. If a preceding data exchange has not finished when a signal is received, the signal is ignored to not flood the network with gossip attempts; otherwise, all DataGossip workers share a selected set of training records in a ring-shaped pattern. Once a worker receives new foreign training records, it writes them into a predefined shared memory buffer for the training process. If the buffer already contains training records, they get replace by the new ones.

Record selection. Choosing training records that are particularly relevant for the training is important to focus the communication efforts on possibly significant examples. Hence, we tested DataGossip with four different loss-based selection approaches: Random, Hard Example Mining (HEM), Self-Paced Learning (SPL), and Active Bias (AB). The random strategy selects records randomly (without replacement) from the entire local partition. Random sampling is very efficient and also used by SPL and AB until enough losses are recorded. The HEM strategy chooses the most difficult training records from the local partition to be shared with the other training nodes. For this, HEM sorts all records by their most recent loss and, then, selects the top-k hardest instances (highest loss). The SPL strategy defines a threshold to differentiate between sufficiently easy and too hard training records. This threshold increases with every epoch. SPL then selects a random sample of easy records for the gossiping while using the loss as sampling weights. In this way, the sampling focus lies on the hardest records in the set of considered records and follows the moving threshold over time to ever harder training records. The AB strategy samples training records from the entire local partition by using the loss variance of the training records as weights. For this purpose, AB calculates the variance of losses for each training record over the already trained epochs. The loss variance reflects the impact of the local training on a particular record and, hence, the records' potential relevance for the local training.

Record sizing. To keep the network pressure low, not exhaust the memory and stay reactive, the selected set of training records for the gossiping needs to be as small as possible. The optimal amount of gossiped data also depends on the network speed, cluster size and available memory. For this reason, DataGossip offers the size of gossip batches as a parameter that we later investigate in the evaluation.

Record weaving. Because the number of foreign training records on each node needs to be very small in comparison to the local training records, they and their classes are significantly underrepresented, which causes them to have little effect on the training when treated equally to all local records. For this reason, the DataGossip workers weave the foreign records into the local records in a way that foreign records appear more often and, hence, are trained on more frequently. Technically, foreign and local records are still stored in different buffers (to avoid physical duplication), but a weaving protocol defines that a next batch of records needs to be read from the foreign buffer every certain amount of batch readings from the local partition. We analyze the frequency of such readings, i.e., the emphasize that is put on foreign training records, in the evaluation.

4 EXPERIMENTS

In this section, we evaluate the effectiveness of using DataGossip to minimize the impact of skewed data distributions on the learning accuracy. The experiments use w.l.o.g. DownpourSGD [5] as the distributed machine learning algorithm. We refer to DownpourSGD without DataGossip as *baseline* and to DownpourSGD with DataGossip as *datagossip*. Both implementations use PyTorch [17], PyTorch Distributed [13], and Gloo [6]. The source code for algorithms and experiments can be found on GitHub[1].

Hardware. All experiments are run on a server cluster with 14 nodes. Each node has an Intel Xeon E5-2630 v4 CPU with 20 threads and 60 GB RAM. One node of the cluster acts as parameter node and the remaining 13 nodes are training nodes.

Datasets. As training data, we use CIFAR10 [11] with gray-scaled images and Fashion-MNIST [21]. For both datasets, we created a balanced partitioning via random shuffling and an imbalanced partitioning via class skewing, i.e., we sorted the datasets by their class labels and split the sorted datasets horizontally in as many equally sized partitions as there are training nodes.

Parameterization. DownpourSGD's hyper-parameters are the *learning rate* (default 0.003), the *update frequency* (default 5), the weight *batch size* (default 64), and the *optimizer* (default SGD). We opted for SGD, because RMSprop and Adam [9] are more sensitive to staleness [4]; the *batch size* of 64 was chosen, because it is a good compromise between result accuracy and training efficiency considering the results of [15] and our own experiments. All experiments train a Convolutional Neural Network (CNN) with three convolutional and two fully-connected layers using the ReLU activation function. The CNN model is randomly initialized and not pre-trained. The instance selection strategy SPL sets its initial threshold λ_0 to 0.1 and its *growth factor* to 1.2.

Experiments. Because DataGossip increases the number of training records per epoch and adds an overhead for data selection and data sending, it would have an unfair advantage over the baseline if we compare the training progress over epochs; hence, we compare the results over time. We also performed all experiments three times and report the mean and standard deviation, because the asynchronous weight and data exchanges, the random model initialization, and the randomization in the instance selection strategies create fluctuations in the experimental results.

4.1 DataGossip Effectiveness

Figure 2 depicts the learned accuracy of the *baseline* and the *datagossip* setup over time on balanced and unbalanced datasets. If the training classes are evenly partitioned, DataGossip does not improve the training, but with careful configuration ($k = 5$, $f = 4$, AB) it also does not worsen the training. On imbalanced data, however, DataGossip clearly improves the training performance: At the times the baseline stopped, the datagossip setup achieved a 5% higher accuracy on Fashion-MNIST and a 7% higher accuracy on CIFAR10. This is because without data exchange the local optima differ significantly and, therefore, the local models compete rather than support one another. The partitionings used in this experiment are, of course, extreme cases. In practice, we expect class distributions to be in between balanced and fully imbalanced. The effect of DataGossip might therefore be smaller, but it is somewhere in the spectrum of same and significantly higher (7% in our case) accuracy. Hence, we recommend the use of DataGossip if the partition balance is unknown or uneven.

[1] https://github.com/HPI-Information-Systems/DataGossip

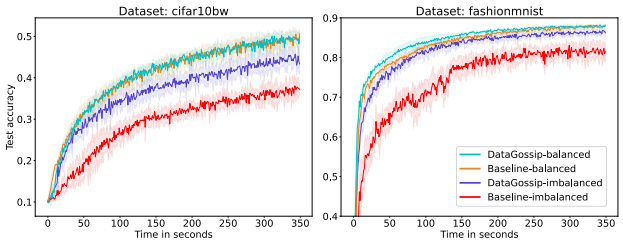

Figure 2: Model accuracies on balanced/imbalanced partitionings with/without DataGossip ($k = 5$, $f = 4$, AB).

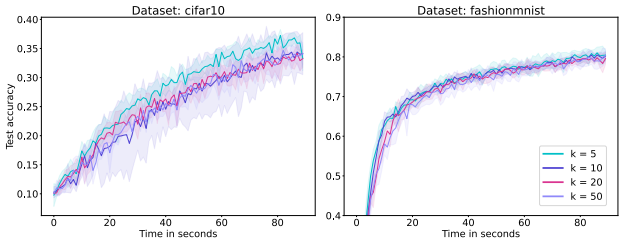

Figure 3: Model accuracies for different batch sizes k on skewed partitionings with DataGossip ($f = 1$, AB).

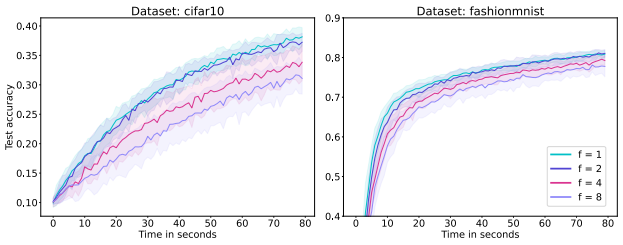

Figure 4: Model accuracies for different frequencies f on skewed partitionings with DataGossip ($k = 5$, AB).

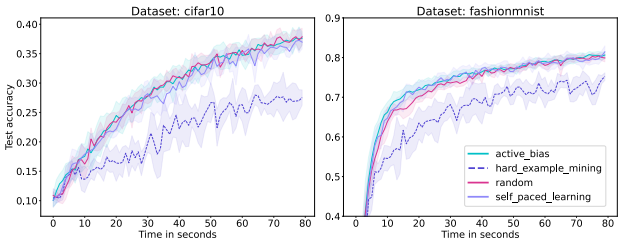

Figure 5: Model accuracies for different instance selectors on skewed partitionings with DataGossip ($k = 5$, $f = 1$).

4.2 DataGossip Instance Selection

If activated, DataGossip is configured to happen at most once per local epoch to align the gossip frequency with the learning progress and, hence, not flood the network with possibly redundant training data, as an epoch is needed to re-evaluate all training losses. With the gossip frequency fixed, we now investigate three DataGossip parameters: the *batch size* k of gossiped training records, the *training frequency* f for foreign records, and the *instance selector*, which is either random, HEM, SPL, or AB.

How many training records should be sent? The parameter k is the number of local records that each training node shares with all other training nodes in every round of gossip. Hence, with 13 training nodes, a round of gossip sends $k \times (13 - 1)$ records to every training node. The measurements in Figure 3 show that the number of exchanged records impacts the performance of the training only slightly. This is because the record weaving always ensures the same emphasize on foreign records (and their classes) regardless of k and finer class variances are learned well enough by the regular model weight updates. Smaller values of k, however, deliver slightly better results than larger ones, because they are more efficient and consume less resources so that the overall training converges a bit faster. In our experiments, we found $k = 5$ to be an overall effective and robust setting.

How often should be trained on foreign records? The parameter f is the frequency for the training on local records w.r.t. foreign records. It denotes that the setup should train on f batches of local records after every batch of foreign records. Figure 4 plots the model accuracies with different frequencies f. The measurements show that with a small f, i.e., a high proportion of foreign records, the training on skewed partitions works best. On balanced partitions, however, we observed that small f can slightly reduce the training accuracy (by about 2-3% in our tests) due to the training bias towards gossiped records. We therefore recommend a small value $f = 1$, only if the data is known to be skewed; if the skew is unknown, a slighly higher value, such as $f = 4$, is a safe choice.

Which training records should be sent? Figure 5 shows the impact of the chosen instance selection strategy on the training accuracy. The measurements show that random, SPL and AB all perform similarly well. This is because all three introduce randomness in the selection, which seems to be the most important selection feature, as small differences in the quality of the selected records are compensated for by the weight exchange of DownpourSGD. Because AB performs slightly best on our datasets w.r.t. accuracy and variance over the training time, we recommend this instance selector as default. The HEM strategy, on the contrary, performs worst, because the chosen hard examples are too deterministic, noisy and confusing for the local training processes.

5 CONCLUSION

DataGossip is, to the best of our knowledge, the first approach that extends arbitrary data parallel distributed machine learning algorithms with the capability to dynamically exchange records at training time. Our implementation of DataGossip demonstrates that sharing data at runtime can effectively reduce the impact of imbalanced training data distributions. The extension focuses on learning setups with large, potentially skewed corpora of training data; the data is tied to specific physical locations but parts of the data may temporarily be shared with other nodes. In our experiments, training setups with DataGossip achieved up to 7% more accurate training results on imbalanced class distributions; with careful parametrization, the extension does no harm when used on balanced training datasets.

Although we already tested the idea of data gossip on different datasets (two are shown is this paper) and machine learning algorithms (one is shown in this paper), we still need to experiment with different skew patterns, data sizes, and data types. Our experiments also indicate that a more intelligent parametrization (e.g. dynamic adjustment of f and k at runtime) and an asynchronous data exchange protocol (e.g. a reactive gossiping strategy) can further improve the accuracy with DataGossip.

REFERENCES

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. ACM, 41–48.

[2] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. 2017. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*. 1002–1012.

[3] Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. 2007. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*. 281–288.

[4] Wei Dai, Yi Zhou, Nanqing Dong, Hao Zhang, and Eric P Xing. 2019. Toward understanding the impact of staleness in distributed machine learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[6] facebookincubator. 2020. Gloo: Collective communications library with various primitives for multi-machine training. https://github.com/facebookincubator/gloo

[7] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. 2020. The Non-IID Data Quagmire of Decentralized Machine Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. 5819–5830.

[8] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. 2018. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479* (2018).

[9] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

[11] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[12] M Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*. 1189–1197.

[13] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

[14] Jiahuan Luo, Xueyang Wu, Yun Luo, Yunfeng Huang, Yang Liu, Aubu Huang, and Qiang Yang. 2019. Real-world image datasets for federated learning. *arXiv preprint arXiv:1910.11089* (2019).

[15] Dominic Masters and Carlo Luschi. 2018. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612* (2018).

[16] J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Francisco Martínez-Trinidad, and Josef Kittler. 2010. A review of instance selection methods. *Artificial Intelligence Review* 34, 2 (2010), 133–143.

[17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NeurIPS Autodiff Workshop*.

[18] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[19] Julian Risch and Ralf Krestel. 2018. Aggression Identification Using Deep Learning and Data Augmentation. In *Proceedings of the Workshop on Trolling, Aggression and Cyberbullying (co-located with COLING)*. 150–158.

[20] Julian Risch, Victor Künstler, and Ralf Krestel. 2020. HyCoNN: Hybrid Cooperative Neural Networks for Personalized News Discussion Recommendation. In *Proceedings of the International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT)*.

[21] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[22] Huan Zhang, Cho-Jui Hsieh, and Venkatesh Akella. 2016. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In *Proceedings of the International Conference on Data Mining (ICDM)*. 629–638.

[23] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).