PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

An algorithm for model fusion for distributed learning

Dinesh Verma, Supriyo Chakraborty, Seraphin Calo, Simon Julier, Stephen Pasteris

Dinesh Verma, Supriyo Chakraborty, Seraphin Calo, Simon Julier, Stephen Pasteris, "An algorithm for model fusion for distributed learning," Proc. SPIE 10635, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, 106350O (4 May 2018); doi: 10.1117/12.2304542

SPIE.

Event: SPIE Defense + Security, 2018, Orlando, Florida, United States

An Algorithm for Model Fusion for Distributed Learning

Dinesh Verma*[a], Supriyo Chakraborty[a], Seraphin Calo[a] , Simon Julier[b], Stephen Pasteris[b]
[a]IBM TJ Watson Research Center, 1110 Kitchawan Road, Yorktown Heights, NY, USA 10598,
[b]Dept. of Computer Science, University College London, London WC1E 6BT, UK

ABSTRACT

In this paper, we discuss the problem of distributed learning for coalition operations. We consider a scenario where different coalition forces are running learning systems independently but want to merge the insights obtained from all the learning systems to share knowledge and use a single model combining all of their individual models. We consider the challenges involved in such fusion of models, and propose an algorithm that can find the right fused model in an efficient manner.

Keywords: distributed learning, coalition operations, federated learning, data efficiency

1. INTRODUCTION

Artificial Intelligence [1] and machine learning in particular is anticipated to play a significant and expanded role in all aspects of future technology, including coalition operations. Most algorithms for machine learning rely on the existence of training data in a centralized location where time-consuming training of machine learning models can be performed. However, in coalition environments, many situations arise when data needs to remain distributed and cannot be moved to a central location. In those cases, the only practical solution is to train several models independently and combine them together. However, the combination of independently trained models is non-trivial. In this paper, we propose an algorithm for distributed learning that can perform the task of fusion of different models in an effective manner.

We begin this paper with an overview of the system environment and assumptions underlying distributed learning. We then discuss the challenges associated with distributed learning in coalition contexts, followed by a review of previous work in the area of distributed learning. We propose two alternative architectures for distributed learning and compare their performance to the base case of centralized learning. Finally, we propose a distributed algorithm for combining different trained models, and discuss its effectiveness.

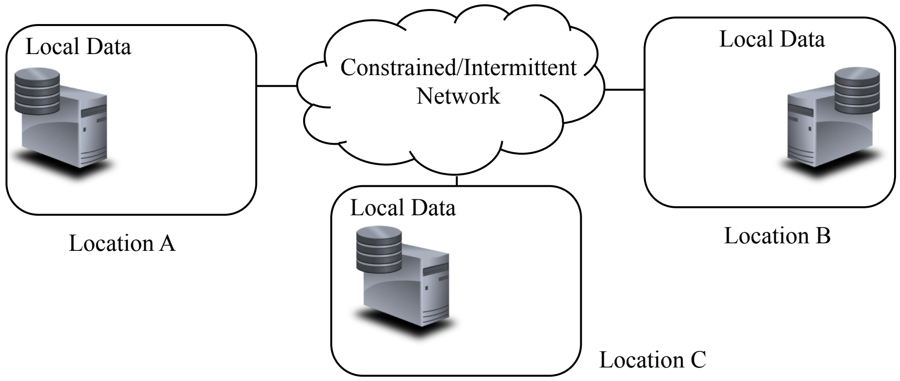

Figure 1. The environment for Distributed Learning

2. DISTRIBUTED LEARNING ENVIRONMENT

In a coalition environment, many countries collaborate together as part of a coalition. However, the trust between different countries is not necessarily absolute. As a result, coalition partners may not be willing to share all the data that they may have collected with one other. However, depending on the level of trust available between partners, they may be willing to share *insights* gleaned from their data. In addition to the level of trust, other impediments may restrict the sharing of

Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX,
edited by Michael A. Kolodny, Dietrich M. Wiegmann, Tien Pham, Proc. of SPIE Vol. 10635,
106350O · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2304542

data among partners. These include the lack of reliable network connectivity, or just that the data size may be too big to transfer over the available bandwidth.

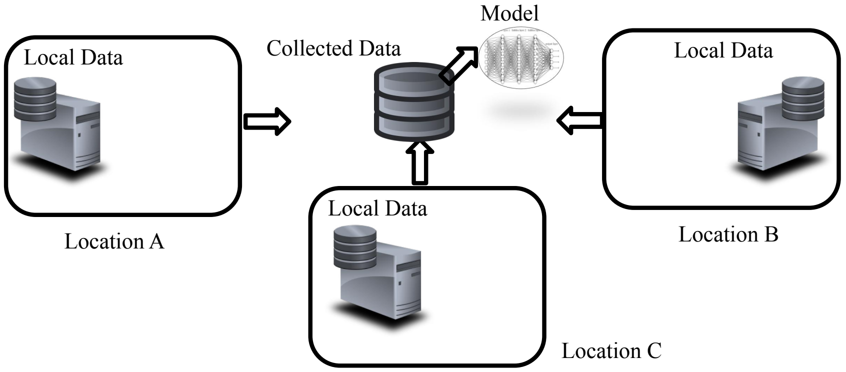

Figure 2. The desired way to learn models with perfect network connectivity

Similar situations can arise in civilian environments as well as military environments. Many companies operate internationally, but data generated in one country may not be allowed to be moved across national boundaries. Industries like telecommunications and finance are subject to many such restrictions. Even if the regulations allowed the movement of data, such movement may be too time-consuming or too expensive. The cost and time taken to transfer the terabytes of data that a bank, insurance company or telecommunications operator has across countries, e.g. from Singapore to United States could be prohibitive. At the same time, banks may want to get insights that combines the knowledge available from all of the data located at many different sites that they may have.

The problem environment is as shown in Figure 1. There are several locations where data is available from which insights can be drawn, i.e. a machine learning model can be built. However, the network between the different locations is less than perfect. The network imperfection depends on the specifics of the environment. In some cases, the network may not have enough capacity to transfer the data across in a reasonable amount of time. In some cases, the network may be unreliable and have intermittent connectivity. This could happen if data is in the tactical battlespace arena, or be located on moving platforms, e.g. ships or planes that only connect at limited points in time. In other cases, the network may be always connected and have good bandwidth, but just may be too expensive. If the data is at a remote area and cellular/satellite connectivity is the only one available, the charges for transferring data on such links may be prohibitive.

If we had a perfect network connecting the different sites, the preferred approach for gleaning insights from the data would be to perform the steps shown in Figure 2. We would first copy all of the data from each of the local sites to a central location, and run a learning algorithm on the data that is collected. This would result in an model which can be used in subsequent applications.

However, as explained above, the approach shown in Figure 2 is not viable in a coalition environment. Instead, the approach that we need to take is to train models at each of the sites on the local data, and then try to combine and merge those models into a larger aggregate. The approach is as shown in Figure 3. With this approach, we only try to transfer the models over the poor network. Since the model is expected to be much smaller than the actual data, it is faster and cheaper to transmit on the network.

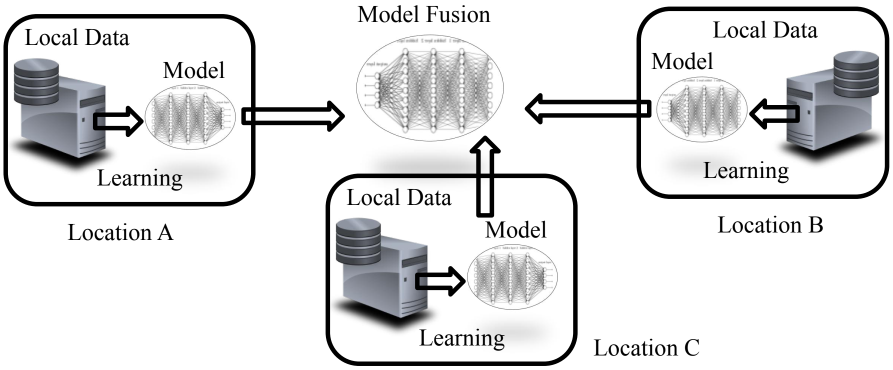

Figure 3. The approach for Distributed Learning

3. RELATED RESEARCH

The challenge of distributed learning in coalition environments was identified in [2] in which several scenarios where distributed learning is required were highlighted. An architecture for coordinating the models that were learnt at each local site was presented, and the needs for algorithms that can fuse neural networks together was highlighted.

When models are represented as decision tables or rules, they can be merged together by finding partitions of the feature space where the different models intersect. In partitions where only a single ruleset of table applies, those are used. In partitions where multiple rulesets/tables intersect, the one with higher confidence are selected. Other merging and selection strategies can also be used [3]. Decision trees can also be merged by converting them to equivalent decision tables and converting them back to decision trees once merging has been completed. However, this approach cannot be used for decision rules which cannot easily expressed as tables. Neural networks, for example, often have decision boundaries which are implicitly encoded in the network but are not explicitly known and would have to be found through exhaustive search.

Another approach to merge models is to view them as functions over the input features space, and to transform the functions into a frequency domain by taking their Fourier Transform [4]. The fusion of models can then be done by adding their transform coefficients, and then inverting them. The challenge with this approach is that it can be applied only to a very limited set of machine learning models, i.e. those that can be easily and naturally expressed as functions. In models such as neural networks, features are latent in intermediate layers of the network, and expressing the model as a function may be non-trivial. Converting these models to a frequency spectrum is difficult.

By limiting neural networks to a very specific domain like optical character recognition, it has been shown that evidential reasoning can be used to combine neural networks in an ensemble [5], provided training data is available. The combination of neural networks is done by viewing them as classifiers, and then creating an ensemble of classifiers using different ensemble methods available in the literature [6] [7]. However, most ensemble methods require access to training data in order to do the proper combination, an assumption which is not viable in the coalition environment.

A parameter sharing solution for distributed learning, assuming each location uses the same model has been proposed [8]. At each location, a model is trained using local data, where a model is simply a parameterized function representing the mapping from the input to the labeled outputs. After a fixed number of training rounds, parameters from the local models are shared with a centralized fusion server, which performs aggregation of these parameters and sends the aggregated parameters back to the different locations. The models at different locations are re-instantiated using the received aggregated parameters, and their training with the local data is re-initiated. The process continues till the aggregated parameters converge. The parameterized representation is typically a lossy compression of the information in the raw data, and the size of the parameter set is smaller than the size of the raw data. This approach requires a network which is always connected (not intermittent) and an ability to coordinate the different nodes to learn the same model. With some modifications, the approach can also protect the privacy of underlying data [9].

Another algorithm [10] to combine multiple models trained on disjoint datasets relies on an auxiliary unlabeled public dataset. The trained models act as teachers, and in combination with an aggregator (which perform majority voting), are used to label samples from the public dataset. A student model is then trained using the labeled public dataset and learns to predict data, similar to the teacher models without being directly exposed to the models. However, this approach requires that we have a training data set available for the fusion operation.

In this paper, we propose an algorithm that can work in environments which are not always connected, and therefore deal with the situation where the models being combined are different.

4. SYSTEM ARCHITECTURES FOR DISTRIBUTED LEARNING

A system architecture refers to the type of services and components that are required in order to perform the task of distributed learning. The choice of the system architecture can have a strong influence in the performance of the learning algorithm. The system architecture can also include the components and standards required for communication and interaction among the different nodes.

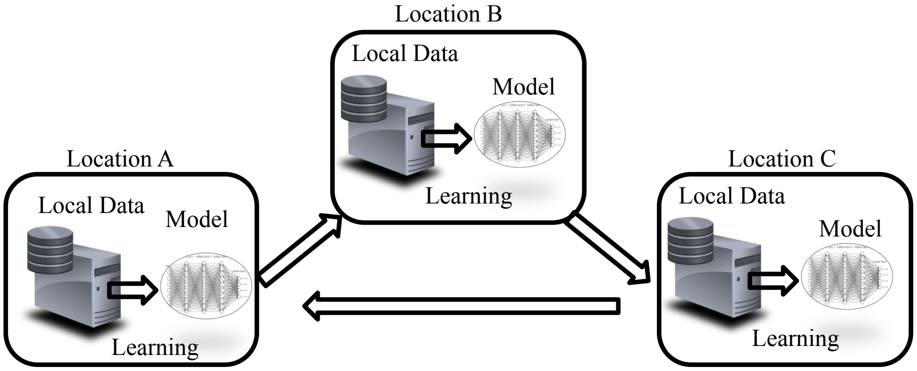

Figure 4. A Peer to Peer Approach for Distributed Learning

For the task of distributed learning, there are two alternative system architectures that can be used. The first system architecture is that of a distributed peer to peer architecture. In this approach, one of the locations will train the model using their local data. It can then pass the trained model to another location, which uses its data to perform additional training and to modify the model. The updated model is passed to the third location and so on, until it has made a pass through all of the locations. At each of the locations, the system can either update the training model or combine that model with a local pre-trained model. In order to be able to pass the model along in a ring linking the locations, we do need to assume that the network connecting each of the locations is always available, or at least available for the time-frame when models need to be exchanged.

Since we need to assume that all the locations are always connected, the training process can be optimized. The training can be done by dividing the local training data into mini-batches. The model is passed to the next locations after each mini-batch, and the models flow into a cycle until each node has completed its batches of data. Nodes that have completed their training would simply pass the model along unchanged in this cycle. When the model makes one complete loop without any updates, it is completely trained.

Passing the model along in this manner has the advantage that it does not require establishing any entity other than the sites which have the local data. Communication among the different locations can be done using web-services, or the sites may even use a distributed block-chain based protocol like Hyperledger [11] to exchange the models. However, this protocol may require several passes through the chain, which may result in degraded performance compared to the situation where we use a centralized system that all the locations trust.

The system architecture which uses a centralized server, which we call the fusion server, to help out the coordination of all locations to learn their data is shown in Figure 5. The fusion server could operate in either a connected or a disconnected mode. When it is operating in a connected mode, each location sends their model to the fusion server after training on a mini-batch of data. The fusion server combines them and sends them back to each of the nodes for training on the next mini-batch. The process continues till each node has completed training on their data. There are approaches which can further optimize how frequently the mini-batches need to communicate with the fusion server to minimize the overhead of communication [12].

When the network is intermittently connected, the fusion server maintains a local copy of the fused model. When new locations connect to it, it gets the fully trained model from the newly connected locations and combines it with the existing fused model. The advantage of this approach is that the locations can connect and disconnect as dictated by their operating environments, and the fusion server is always available to provide them with the models learnt from the locations that it has seen previously. However, this requires the locations to trust the fusion server.

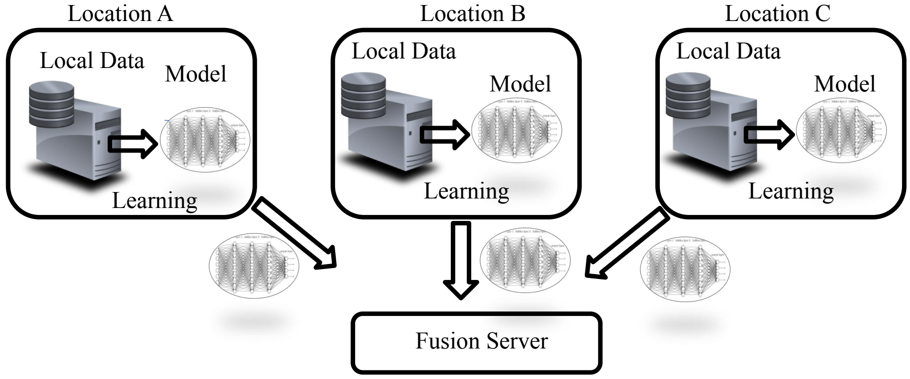

Figure 5. A Centralized Fusion Server Approach for Distributed Learning

The relative time for training the data in the two different system architectures in the fully connected mode can be computed and would depend on the latency of network communication, and the time taken to train the model on each mini-batch. If there are *N* locations, and each location has equal amounts of data which is divided into *K* mini-batches each containing *B* Mbytes, network connectivity is *C* Mbytes per second, the model is *M* Mbytes, and the training for each mini-batch takes *T* seconds, we can obtain the expected time for training the model using each of the two system architectures for distributed learning, as well as for the centralized approach shown in Figure 2.

For the centralized architecture shown in Figure 2, the total time would be $K.B/C + N.K.T$.

For the peer to peer architecture shown in Figure 4, the total time would be $K.N.M/C + N.K.T$.

For the fusion server architecture shown in Figure 5, the total time would be $K.M/C + K.T$.

This assume that the fusion server or the central server in the location can handle concurrent communications with all of the learning nodes at the same time. In most real-life scenarios, $B >> M$, the two approaches for distributed learning should take smaller time for overall training than trying to move the data to a central location. Furthermore, the central fusion architecture is *N* times faster than the peer to peer architecture, where *N* is the number of different locations.

In addition to the performance, an important question is whether the fusion server architecture shown in the diagram above can be as accurate as the centralized architecture. In the next section, we look at the algorithm for distributed learning and fusion of models.

5. FUSION ALGORITHM

The fusion algorithm that we propose assumes that the local systems have trained the model completely and that the completed system is available at the fusion server for combination. This assumption is necessary in order to support disconnected operations. If we could assume a fully connected network then an incremental learning approach, which gets the agents to agree to the same type of model e.g. a neural network with same number of layers and nodes in each layer, [2], and use parameter aggregation [8] with or without optimization [9], where training is done in mini-batches. However, when we have a disconnected network, incremental training cannot be used, and we have to combine fully trained models, that potentially may have different set of parameters.

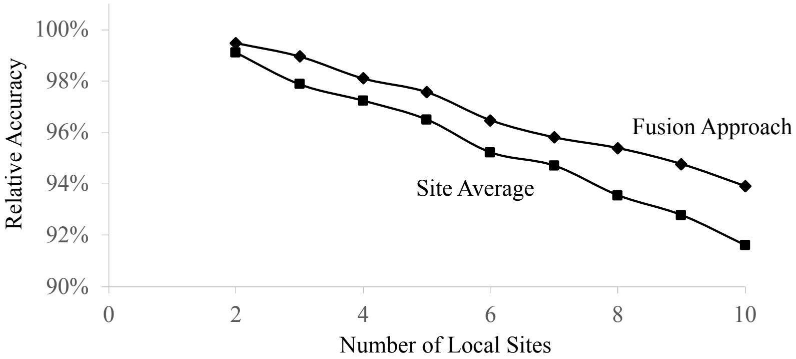

Figure 6. Performance of the Fusion Algorithm with Ensembles

We propose that the fusion server create an ensemble of all the fully trained models. However, the challenge with creating the ensemble is that most of the common techniques for ensemble building, e.g. boosting, Bayesian parameter estimation, Bayesian parameters, stacking, require access to the original training data. When we do not have access to the training data, only bagging approaches which take a majority vote for combining outputs are the ones that can be used. However, these also happen to be the ensemble approaches that perform worse than the ones where access to training data is available. Therefore, the main challenge at the fusion server would be to determine what weight to assign to the results of each individual model when combining them, without any access to either training data or to a test data.

Our approach therefore is a multi-step process in which the fusion server maintains the definition of a common model (e.g. a neural network with a given number of layers and a set of nodes per layer) which combines the concepts of weighted averaging and parameter aggregation:

- **Step 1**: All the local agents that have current connectivity with the fusion server provide it with their trained network model
- **Step 2**: The fusion server creates an ensemble model consisting of the models provided by the currently connected agents, and the fused model that exists at the fusion server. In the ensemble model, each input sent to the system for inference is passed through each of the models, and their results combined using a weight assigned to each of the ensembles. The weights at this point assigned to the newly provided models are undetermined.
- **Step 3**: Each of the agents use their local data to compute the best weights for the ensemble available on their training data. One approach to compute the weights is for each agent to run each of the models from the other agents on its local data, find the accuracy of the other model when used on its own local data, and propose that as the ensemble weight to the fusion server.
- **Step 4**: The fusion server averages the weights from all of the ensembles to create a new fused ensemble network which is provided to each local agent. When averaging the weights, the number of agents that have contributed to the fused model is used to determine the relative importance of existing models and newly provided models.

We tested the resulting accuracy from the ensemble approach accuracy that can be achieved in this manner by implementing and running the algorithm on the MNIST data set [14]. The relative accuracy is the accuracy of each of the approaches compared to training the same model on all of the data in a central location. We assumed that the training data was divided equally in a random manner among all the different local sites, and each site was training a simple convolutional network with two convolutional layers using rectified linear unit activation followed by two fully connected layers, the first one using rectified linear unit activation and the second one using softmax. The fusion server had no data, but simply acted as a coordination point. Each site used the accuracy of other models on its data as a measure of the weight to be assigned to the other model, and the fusion server provided the aggregation capability for the same. As the number of sites decreases, with each site having access to only a fraction of the data, their relative accuracy decreases. The fusion process can increase the relative accuracy by a few percentage points.

The result of this algorithm, when run between 2 and 10 sites with distributed data is shown in Figure 6. While the ensemble approach improves the accuracy of the resulting fusion, especially when the data is distributed across a larger number of

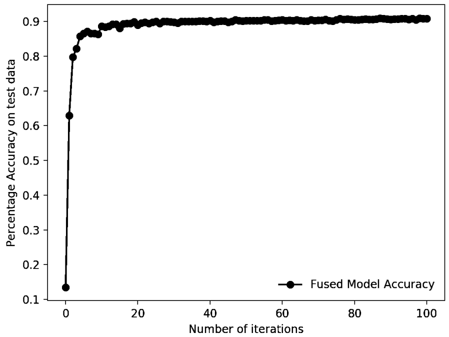

Figure 7. Performance of the Fusion Algorithm

sites, the increase in accuracy is relatively modest and the simple ensemble based averaging approach is relatively inaccurate compared to the approach of collecting all the data into a single location for training.

In order to address this limitation, we propose a different algorithm for fusing the models. In this approach, we assume that one of the local sites can act as the fusion server, and it has a data set available for validation. We further assume that the fusion server can ask each of the local agent to train a specific type of model, which have the same parameters across all of the sites. The new algorithm therefore becomes as follows:

- **Step 1**: All the local agents that have current connectivity with the fusion server provide it with their trained network model, which they have trained with a mini-batch size parameter specified by the fusion server
- **Step 2**: The fusion server computes the predictions of each model against its validation data. It then minimizes the squared error between the weighted linear combination of the predicted outputs (from each model) and the actual labels of the validation dataset. The weights obtained from the optimization are used to combine the parameters of the models for that iteration.
- **Step 3**: Each of the agents is sent back the model with the fused parameters, and they use the next mini-batch to train a new set of model weights to be shared with the fusion server.
- **Step 4**: Each such iteration continues till all the mini-batches are used for training by each of the local agents.

Figure 7 shows the accuracy of this result when 6 agents were connected to the fusion server, and the training data was divided randomly among the different agents, so that each agent had access to one sixth of the training data. This approach provides for a much higher accuracy in the distributed fusion process, and a small number of iterations (less than 10), the distributed process approaches the overall accuracy as if it was trained on a centralized data.

The ensemble-based approach, which has worse performance as shown in Figure 6, allows for each location to use an independent form of model, as long as each node has the ability to execute the models provided by the other nodes. The parameter fusion approach has much better accuracy but requires that the locations coordinate the model being learnt and trained. Both of these approaches can be applicable depending on the operational environment in a coalition.

6. SUMMARY

In this paper, we have discussed the need for having distributed fusion algorithms, specially in the context of coalition operations. We have compared two different architectures for distributed fusion, and shown that the use of a fusion server can significantly improve overall system performance compared to a completely distributed peer to peer approach. We have proposed algorithms for creating a fusion of models trained independently, and shown that one of the algorithms can approach the accuracy of a centralized learning model very rapidly.

The algorithms described in this paper are just the early stage of exploration for distributed learning algorithms that can apply to the many challenging requirements of a coalition network. In future work, we intend to develop algorithms which can further reduce the assumptions inherent in distributed fusion, and study the performance of the algorithms under different conditions, including situations where training data for some classes is only available at selected sites.

.

7. ACKNOWLEDGEMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] Rasch, R., Kott, A., & Forbus, K. D., "AI on the battlefield: An experimental exploration," Proc. AAAI Conference on Innovative Applications of Artificial Intelligence, 906-912 (2002).

[2] Verma, D., & Julier, S., "Dynamic network based learning systems for sensor information fusion," In International Society for Optics and Photonics Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VIII, 10190- (2017)

[3] Strecht, P., "A Survey of Merging Decision Trees Data Mining Approaches," Proc. 10th Doctoral Symposium in Informatics Engineering, 36-47 (2015).

[4] H. Kargupta and B. Park, “A fourier spectrum-based approach to represent decision trees for mining data streams in mobile environments,” IEEE Transactions on Knowledge and Data Engineering, 16(), 216–229 (2004).

[5] Rogova, G., "Combining the results of several neural network classifiers," Neural networks 7(5), 777-781, (1994).

[6] Opitz, D., & Maclin, R., "Popular ensemble methods: An empirical study," Journal of Artificial Intelligence Research, vol 11, pp. 169–198, (1999).

[7] Rokach, L., "Ensemble-based classifiers". Artificial Intelligence Review. 33 (1-2): 1–39, (2010).

[8] McMahan, H. B., and Moore, E., and Ramage, D., and Hampson, S., and Arcas, B. A., “Communication-Efficient Learning of Deep Networks from Decentralized Data”, https://arxiv.org/pdf/1602.05629.pdf

[9] Bonawitz, K., et. al., “Practical Secure Aggregation for Privacy-Preserving Machine Learning,” Proc. Conference on Computer and Communications Security, 1175-1191, URL https://eprint.iacr.org/2017/281.pdf (2017).

[10] Papernot, N., Abadi, M., Erlingsson, U., Goodfellow, I., and Talwar, K., “Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data,” Proc. International Conference on Machine Learning (2017).

[11] Cachin C., "Architecture of the Hyperledger blockchain fabric,” Proc. Workshop on Distributed Cryptocurrencies and Consensus Ledgers, (2016).

[12] Wang, S., Tuor, T., Salonidis, T., Leung, K., Makaya, C., He, T. and Chan, K., “When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning,” Proc. IEEE International Conference on Computer Communications – INFOCOM, (2018).

[13] Hastie, T., Rosset, S., Zhu, J., and Zou, H., “Multi-class adaboost,” Statistics and its Interface, *2*(3), 349-360 (2009)

[14] Yann, L., Corinna, C. and Christopher, J., “The MNIST database of handwritten digits”, URL http://yann.lecun.com/exdb/mnist/, (1998)