WeightRelay: Efficient Heterogeneous Federated Learning on Time Series

Wensi Tang and Guodong Long

Australian Artificial Intelligence Institute
Faculty of Engineering and IT
University of Technology Sydney, Australia
wensi.tang@student.uts.edu.au
guodong.long@uts.edu.au

Abstract. Federated learning for heterogeneous devices aims to obtain models of various structural configurations in order to fit multiple devices according to their hardware configurations and external environments. Existing solutions train those heterogeneous models simultaneously, which requires extra cost (e.g. computation, communication, or data) to transfer knowledge between models. In this paper, we proposed a method, namely, weight relay (WeightRelay), that could get heterogeneous models without any extra training cost. Specifically, we find that, compared with the classic random weight initialization, initializing the weight of a large neural network with the weight of a well-trained small network could reduce the training epoch and still maintain a similar performance. Therefore, we could order models from the smallest and train them one by one. Each model (except the first one) can be initialized with the prior model's trained weight for training cost reduction. In the experiment, we evaluate the weight relay on 128-time series datasets from multiple domains, and the result confirms the effectiveness of WeightRelay.

Keywords: Time series classification · Federated learning · Heterogeneous model.

1 Introduction

With the development of smart devices, an increasing amount of time series data can be collected, such as daily heartbeats, blood oxygen levels, electronic consumption, and motion signals for smart device control [22,9,2,20,7,8,36].

With the advent of sophisticated devices and advanced data analysis technologies, there is potential to bring immense value to society. However, privacy concerns limit the integration of smart devices with state-of-the-art deep learning [60,25]. Specifically, smart devices encounter challenges when training large models due to limited power and computational resources for local training. Concurrently, uploading information from these devices, which are commonly found in homes, can pose significant privacy risks, as they often contain sensitive data [15,24].

Federated learning [15] can be used to train deep learning models with privacy protection. However, there are still many unsolved challenges in applying those solutions to smart devices. Specifically, smart devices even for a similar function, typically have varying hardware configurations. The hardware heterogeneous brings a challenge for federated learning. Specifically, how to transfer knowledge between heterogeneous models [22,15,60]. Specifically, this heterogeneous federated learning aims to obtain models of various structural configurations to fit multiple devices according to their hardware configurations and working environment. Under this setting, it is hard for low-capacity devices to contribute their knowledge to big models, for they might have enough memory, bandwidth or computational power to join the big model training via Federate average [1]. Therefore, solutions that could enable big models to get knowledge from the small models are highly desired [22,15].

Existing solutions tackle the problem by adding one or more resources such as computational, communication, and extra data. For example, distillation-based methods require training cost on multiple models and extra cost for knowledge transfer between those models [59,32,62,31,21]. Pruning-based methods [26,30] require an extra cost to pruning the single model onto multiple smaller models. Weight sharing methods, some of them [52] require the computational cost to match the weight of various models iteratively, and some methods [56,12] need a weight scale module to adjust weight before sharing.

Although existing solutions enable knowledge sharing between heterogeneous models, the extra resource consumption makes it hard to implement them on smart devices. This is because most of those devices do not have strong computational, memory or commutations capacity [22,15]. Therefore, solutions with huge training costs bring an embarrassing burden to low-capacity smart devices. When a device's ability limits it from using big models, it may also limit it from contributing its knowledge to big models or getting knowledge from big models via distillations, pruning or weight matching.

Other than capacity neglection, for smart devices, the appropriate 1D-CNN (one-dimensional-convolutional neural networks) model is also seldom mentioned. Specifically, most solutions were tested with 2D-CNN(two-dimensional-convolutional neural networks) models [59,62,31,21,52,12,56] or language models [32]. However, we should notice that most of the data gathered by smart devices are time-series data [34,54] which can mathematically be described as a series of data points recorded in time order [10,9,14,50]. Such as the heartbeat data collected by smartwatches [40,38], the electricity consumption data gathered by energy management devices [16], building structural vibration data recorded by motion sensors [51,28], etc. According to the University of California, Riverside time series archive (UCR archive) [10], the state-of-the-art solutions for time series classification tasks are all 1D-CNNs [14,50,11].

The characteristics of 1D-CNN allow a novel weight relay solution, which does not need any extra resources. Specifically, suppose we have a small 1D-CNN network and a large 1D-CNN network. For the large network training, we could initialize it with a classic random weight initialization or we could initialize it with the weight from a well-trained small network. We find that these two kinds of initialization will be of similar performance, but the second initialization could reduce the training cost of the large network. This training cost reduction could be used to lower the capacity requirement and allows more low-capacity devices to join the big model training. Via ordering those heterogeneous 1D-CNN models from the smallest to the largest, except for the first smallest model, all the other models' training will be benefited.

In experiments, we show the consistently training cost reduction ability of the weight relay on time series datasets from multiple domains i.e., healthcare, human activity recognition, speech recognition, and material analysis. Despite the dynamic patterns of these datasets, weight relay robustly shows its effectiveness.

2 Related Work

2.1 Deep learning for time series classification

The success of deep learning encourages the exploration of its application on time series data [29,14,13]. Intuitively, the Recurrent Neural Network (RNN), which is designed for temporal sequence, should work on the time series tasks. However, in practice, RNN is rarely applied to TS classification [14]. One widely accepted reason among many is that RNN models suffer from vanishing and exploding gradients when dealing with long seuqnce data [39,14,3]. Nowadays, 1D-CNN is the most popular deep-learning method TSC tasks. [61,53,41,42,23,27]. According to the University of California, Riverside time series archive (UCR archive) [10], the state-of-the-art solutions for time series classification tasks are all 1D-CNNs [23,50,11,53].

2.2 Federated learning on heterogeneous devices

Based on the method of transferring knowledge between heterogeneous models, the solutions to heterogeneous Federated learning could be divided into three columns. These are distillation-based, weight-sharing-based pruning-based methods and prototype-based methods.

The knowledge distillation [19,6] allows the knowledge sharing between heterogeneous models. It requires extra data or computational resources to enable knowledge sharing between models [21,31,57], which brings multiple challenges under the federated learning setting. For example, the distillation process requires a large amount of computational resources [62]. What's more, the performance of the distillation is highly related to the similarity of the distributions between the training data and distillation data.

The weight sharing method is based on the assumption that some parts of the weight of various structure models are the shareable, and the shareable part could help to transfer the knowledge between various structure models [52,5,43,37]. In practice, finding which parts of various models should be of the same parameter is hard. Therefore, a large number of computation resources have to be taken to find which parts of models should be matched together [52], or taken to calculate the adjustment module for weight re-scale [12,44].

The pruning-based method aims at training a large neural network and pruning it into various small networks [58] according to the hardware configuration and external environment [26,55,33]. One limitation is that it is hard to control the structure of the pruned network, which challenges fitting those small networks according to the configuration of each edge device. [26,30,48]

The prototype-based federated learning [47,4,17,18] can also be viewed as a solution for heterogeneous devices for it requires a very limited resource for classification calculation. However, as the number of classes increases, prototype learning struggles to scale effectively. Put simply, due to the warp characteristic, comparing the distance between two-time series incurs a computational cost of N^2. Where N is the length of the signal. As the number of classes grows, this cost rises substantially. As a result, the computational cost shifts from training to classification, which is not conducive for small devices. While some methods can map the time series into a feature space [49,46], their classification accuracy cannot match that of larger models.

3 Motivation

To train a 1D-CNN model, we could 1) start from a classic random weight initialization or 2) replace parts of the random initialization weight with a well-trained weight from a small network. The second initialization could reduce the training cost of the large network and won't influence the final performance.

Therefore, to train multiple models on a smaller budget, we don't need to train every model from the stretch. We could initialize some of those models by the trained weight from the others for fast convergence. Figure 3 gives an example of the weight relay on the Crop [45] dataset.

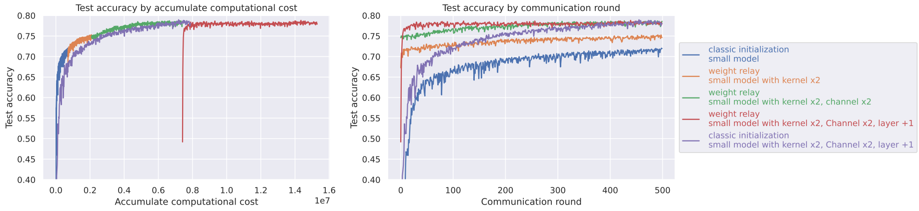

Fig. 1. The left image shows the relationship between the accumulated computational cost and the test accuracy of each model. The accumulated computational cost calculates the computational cost we used to obtain a well-trained model. As the image shows, to obtain a single largest model from classic initialization (purple), we need about $0.8e^7$ computational resources, and we could only get 1 model. However, with weight relay, at the point $0.8e^7$, we have four models(blue, orange, green and red). And the red model, which has the same structure as the purple model, also has the same performance. The right image shows the performance of each model by communication round. We could see that the weight relay model (red) converged much faster than the classic initialization models (purple).

4 Weight relay

In Figure 4, a schematic of weight relay is given. What's more, in this Section, we will explain the weight relay in detail. Specifically, Section 4.1 will introduce heterogeneous models for time series classifications. Section 4.2 will introduce how to align those models. Specifically, when a well-trained weight is passed to a large model, which part of the large network should be replaced. Section 4.3 will discuss the stop criteria of weight relay.

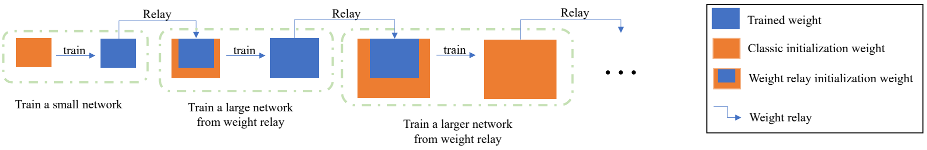
Relay
Relay
Relay
train
train
train
Train a small network
Train a large network from weight relay
Train a larger network from weight relay
. . .
Trained weight
Classic initialization weight
Weight relay initialization weight
Weight relay

Fig. 2. The schematic shows training multiple models with weight relay. Weight relay starts from the training of the smallest network with classic initialization. The trained weight of the smallest network will be used to replace a part of the classic initialization weight of a large network. When the large network is trained, its weight could be used to accelerate the training of a larger network. Since the weight relay only replaces the random initialization with a well-trained one, it requires almost no cost, and the training cost of each model (except the first one) is also smaller than training those from classic initialization.

4.1 Heterogeneous models

According to the result statistics on the UCR archive, all state-of-the-art neural network solutions on time series classification tasks are 1D-CNN models [14,53,50,**?**,35]. Therefore, this paper mainly talked about 1D-CNN models and heterogeneous could happen on all three main structure configurations: the number of layers, kernel sizes and the number of channels.

4.2 Weight alignment

The weight alignment defines which part of the large network should be replaced with the weight of the small network. The alignment will have three steps because the neural network has three hierarchies. Specifically, a neural network is composed of layers. Layers are composed of weight sets. And the weight sets are composed of weight tensors. Therefore, we need to pair weight sets and layers before we align weight tensors.

First step: pair weight sets by layers. The convolutional layer and the batch normalization layer will be indexed from the input to the output. For example, the first convolutional layer is the convolutional layer closest to the input. The fully connected layer will be indexed by **reverse order**. Therefore, the first fully connected layer will be the layer closest to the output. For each of those three types of layers, the $*$-th layer's tensor set of the small network should be paired with the $*$-th layer's tensor set of the large network

Second step: pair tensors by paired weight sets: According to the definitions of the 1D-CNN, tensors in each set will have different functions, such as weight, bias, and running mean. Therefore, for two paired sets, tensors in the two sets will be paired by their function name.

Third step: align two paired tensors. For two paired tensors $\mathbf{A}$ and $\mathbf{B}$, the alignment of the two tensors is to **align the output and input dimensions with the left margin** and **align the kernel dimension with the centre**. Specifically, the small network's $*$-th element should be aligned with the middle of the large network's $*$-th element on the input(output) channel. Therefore, the small network's i-th element should be aligned with the middle of the large network's j-th element on the kernel dimension. Then the i and j describe the alignment relationship of $\mathbf{A}$ and $\mathbf{B}$ in the Equation 1

$$j = i + \lfloor (b-1)/2 \rfloor - \lfloor (a-1)/2 \rfloor \tag{1}$$

where the b and a are kernel sizes of large network and small network.

4.3 Stop criteria for each model

The Weight relay is robust to various early stop criteria. Generally, loose stop criteria will make the model stop in a few numbers of epochs than strict stop criteria. However, as we will show in Appendix A, no matter whether the stop criteria are strict or loose, the weight relay keeps working on most datasets in the UCR archive.

5 Analysis of weight relay

This section will have two parts. In Section 5.1, we will show that though the alignment method in Equation 1 is defined in pair, it is reliable for the alignment of multiple models. Secondly, we will give a macro (Section 5.2) and a micro (Section 5.3) explanation of the coverage acceleration of the weight relay.

5.1 Consistency proof for the alignment

This section will show that despite we only define the relationship between two kernels, this operation will keep the **consistency** when we have **multiple** kernels. The consistency of weight alignment can be describe as below: For any three kernel weights $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ and their length relationship are:

$$a < b < c$$

We could use weight alignment to deter the alignment relationship between $\mathbf{C}$ and the other two kernels as:

$$\mathbf{A}_i \xrightarrow{\text{align with}} \mathbf{C}_k \tag{2}$$

$$\mathbf{B}_j \xrightarrow{\text{align with}} \mathbf{C}_k \tag{3}$$

The signal Phase alignment can be called consistency if, with the same operation on $\mathbf{A}$ and $\mathbf{B}$, we should have

$$\mathbf{A}_i \xrightarrow{\text{align with}} \mathbf{B}_j$$

Here, we give an example to illustrate the consistency of multiple kernels. Supposing we have three kernels of length 3, 5, 8. Via Equation. 1, the $\mathbf{A}_1$ should align with $\mathbf{C}_4$, the $\mathbf{B}_3$ should align with $\mathbf{C}_4$. **Consistency** means that the $\mathbf{A}_1$ should align with $\mathbf{B}_3$.

Proof of consistency for Equation.1:

When we combine Equation.1 and Equation.2 we know that the a th element of kernel $\mathbf{A}$ should align with c th element of kernel $\mathbf{C}$ and their index relationship is:

$$k = i + \lfloor (c-1)/2 \rfloor - \lfloor (a-1)/2 \rfloor \tag{4}$$

With Equation.1 and Equation.3 we know the alignment relationship between elements in $\mathbf{B}$ and $\mathbf{C}$ is:

$$k = j + \lfloor (c-1)/2 \rfloor - \lfloor (b-1)/2 \rfloor \tag{5}$$

Using the Equation. 5 to subtract Equation. 4 and we have:

$$k - k = \begin{array}{c} i + \lfloor (c-1)\,/2 \rfloor - \lfloor (a-1)\,/2 \rfloor \\ + \\ -j - \lfloor (c-1)\,/2 \rfloor + \lfloor (b-1)\,/2 \rfloor \end{array} \tag{6}$$

Therefore, we know

$$j \equiv i + \lfloor (b-1)\,/2 \rfloor - \lfloor (a-1)\,/2 \rfloor$$

Which means that

$$\mathbf{A}_i \xrightarrow{\text{should align with}} \mathbf{B}_j$$

as it should be.

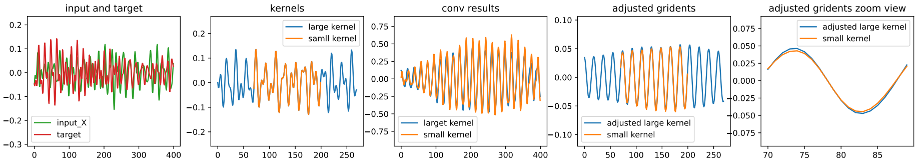

Fig. 3. This example shows that: Given the same input and target, the gradient direction of the small kernel is the same as the gradient direction of the aligned part of the large network. Specifically, images from left to right are: 1) input and target, which are generated by random noise; 2) A large kernel weight which is random noise, and the weight of the small kernel, which is cropped from the large kernel; 3) The convolution results for large kernel and small kernel with the input in image 1); 4) The gradients of kernels which is calculated with the random target in image 1). To demonstrate the overlapping parts of the two kernels are in a similar direction, their magnitude was adjusted; 5). A zoom view of the overlapping gradient parts (indexed 70 to 90).

5.2 Macro Explanation of the training acceleration

One explanation of the training acceleration is that compared with the classic initialization weight, the weight relay initialization is closer to the final weight. An experiment evidence is given in the SectionA.2. We statistic the distance between the initialization weight and trained weight, and we will see that compared with the random initialization, the weight relay initialization are of a smaller distance to the final value.

5.3 Micro explanation of the training acceleration

The micro explanation to the training acceleration is that: the small network's optimization direction is the optimization direction of the large network's sub-network, where the small network should replace the weight. Therefore, the training acceleration is because the training of the small network makes the sub-network close to the final target, which also matches the experiment result in the macro explanation(SectionA.2).

To explain this, we could start from a simple case study in Figure 3.

6 Experiment

6.1 Benchmarks

University of California, Riverside (UCR) 128 archive [10] is selected to evaluate the weight relay under the federated setting. This is an archive of 128 univariate TS datasets from various domains, such as speech reorganizations, health monitoring, and spectrum analysis. What's more, those datasets also have different characteristics. For instance, among those datasets, the class number varies from 2 to 60, and the length of each dataset varies from 24 to 2844. The number of training data varies from 16 to 8,926.

6.2 Evaluation criteria

Following the datasets archive [10], the accuracy score is selected to measure the performance. Following the paper [1], the communication round multiple model size is selected to measure the training cost.

6.3 Experiment setup

Following [14,50,23], for all benchmarks, we follow the standard and unify settings [53] for all 128 datasets in the UCR archive. The training will stop when the training loss is less than 1e-3 or reach 5000 epoch. To mimic the federated learning scenario, the client number is 10. More details can be found in the supplementary material.

6.4 Experiment result

The experiment shows that weight relay has similar performance and fewer computation resources costs than using federated average on all devices to obtain each model. Due to the larger number of datasets, we cannot list the results of all datasets. Therefore, we plot the statistical result of the 128 datasets, and the result is shown in Figure 4.

7 Conclusion

In this paper, we proposed the weight relay method, which could reduce the training cost for heterogeneous model training. We theoretically analyse the mechanism of weight relay and experimentally verify the effectiveness on multiple datasets from multiple domains.

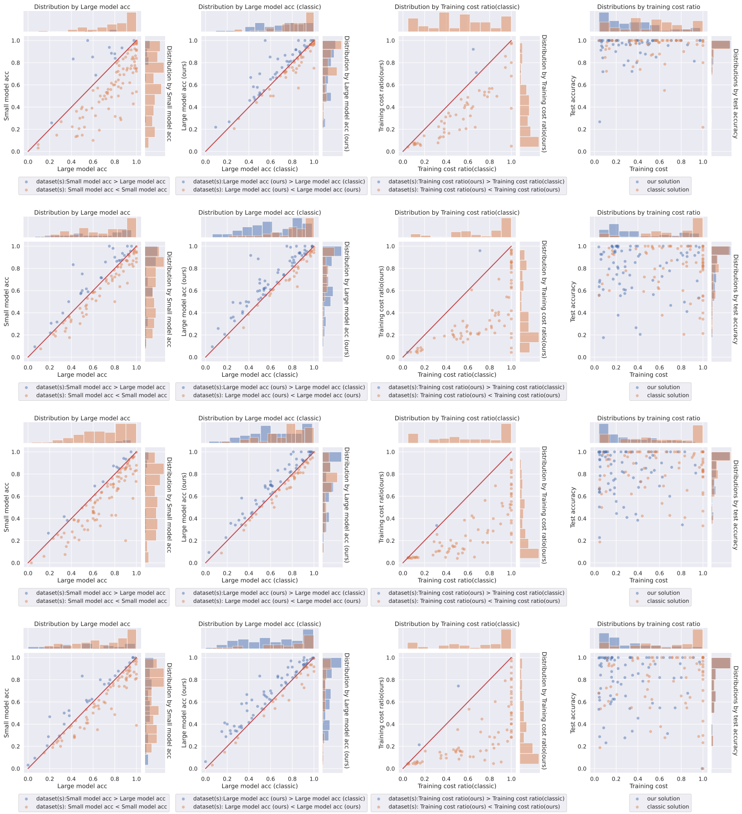

Fig. 4. From top to bottom, each row shows the statistical results when the large model has 2Xkernel size, 2Xchannel number, one extra layer, and all extensions than the smaller model. From the left to the right image, we can see that, for most of the dataset, the large model will perform better than the small model (the first column). The performance of the weight relay is similar to the performance of training from classic (the second column); the weight relay has a lower training cost (the third column); To archive similar performance, the weight relay has a smaller training cost (the fourth column).

References

1.
2. Bagnall, A., Dau, H.A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., Keogh, E.: The uea multivariate time series classification archive, 2018. arXiv preprint arXiv:1811.00075 (2018)
3. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2), 157–166 (1994)
4. Biehl, M., Hammer, B., Villmann, T.: Prototype-based models in machine learning. Wiley Interdisciplinary Reviews: Cognitive Science **7**(2), 92–111 (2016)
5. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791 (2019)

6. Chen, F., Long, G., Wu, Z., Zhou, T., Jiang, J.: Personalized federated learning with graph. arXiv preprint arXiv:2203.00829 (2022)
7. Chen, S., Long, G., Shen, T., Jiang, J.: Prompt federated learning for weather forecasting: Toward foundation models on meteorological data. arXiv preprint arXiv:2301.09152 (2023)
8. Chen, S., Long, G., Shen, T., Zhou, T., Jiang, J.: Spatial-temporal prompt learning for federated weather forecasting. arXiv preprint arXiv:2305.14244 (2023)
9. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The ucr time series classification archive (July 2015), `www.cs.ucr.edu/~eamonn/time_series_data/`
10. Dau, H.A., Bagnall, A., Kamgar, K., et.al.: The ucr time series archive. arXiv:1810.07758 (2018)
11. Dempster, A., Petitjean, F., Webb, G.I.: Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. Data Mining and Knowledge Discovery pp. 1–42 (2020)
12. Diao, E., Ding, J., Tarokh, V.: Heterofl: Computation and communication efficient federated learning for heterogeneous clients. arXiv preprint arXiv:2010.01264 (2020)
13. Dong, X., Kedziora, D., Musial, K., Gabrys, B.: Automated deep learning: Neural architecture search is not the end. arXiv preprint arXiv:2112.09245 (2021)
14. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data Mining and Knowledge Discovery **33**(4), 917–963 (2019)
15. Ferrag, M.A., Friha, O., Maglaras, L., Janicke, H., Shu, L.: Federated deep learning for cyber security in the internet of things: Concepts, applications, and experimental analysis. IEEE Access **9**, 138509–138542 (2021)
16. Gans, W., Alberini, A., Longo, A.: Smart meter devices and the effect of feedback on residential electricity consumption: Evidence from a natural experiment in northern ireland. Energy Economics **36**, 729–743 (2013)
17. Gee, A.H., Garcia-Olano, D., Ghosh, J., Paydarfar, D.: Explaining deep classification of time-series data with learned prototypes. In: CEUR workshop proceedings. vol. 2429, p. 15. NIH Public Access (2019)
18. Ghods, A., Cook, D.J.: Pip: Pictorial interpretable prototype learning for time series classification. IEEE computational intelligence magazine **17**(1), 34–45 (2022)
19. Gou, J., Yu, B., Maybank, S.J., Tao, D.: Knowledge distillation: A survey. International Journal of Computer Vision **129**(6), 1789–1819 (2021)
20. Gu, P., Wu, T., Zou, M., Pan, Y., Guo, J., Xiahou, J., Peng, X., Li, H., Ma, J., Zhang, L.: Multi-head self-attention model for classification of temporal lobe epilepsy subtypes. Frontiers in Physiology **11**, 1478 (2020)
21. He, C., Annavaram, M., Avestimehr, S.: Group knowledge transfer: Federated learning of large cnns at the edge. Advances in Neural Information Processing Systems **33**, 14068–14080 (2020)
22. Imteaj, A., Thakker, U., Wang, S., Li, J., Amini, M.H.: A survey on federated learning for resource-constrained iot devices. IEEE Internet of Things Journal **9**(1), 1–24 (2021)
23. Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., et.al.: InceptionTime: Finding AlexNet for Time Series Classification. arXiv e-prints arXiv:1909.04939 (Sep 2019)
24. Ji, S., Long, G., Pan, S., Zhu, T., Jiang, J., Wang, S.: Detecting suicidal ideation with data protection in online communities. In: International conference on database systems for advanced applications. pp. 225–229. Springer (2019)
25. Jiang, J., Ji, S., Long, G.: Decentralized knowledge acquisition for mobile internet applications. World Wide Web **23**(5), 2653–2669 (2020)

26. Jiang, Y., Wang, S., Valls, V., Ko, B.J., Lee, W.H., Leung, K.K., Tassiulas, L.: Model pruning enables efficient federated learning on edge devices. IEEE Transactions on Neural Networks and Learning Systems (2022)
27. Kashiparekh, K., Narwariya, J., Malhotra, P., Vig, L., Shroff, G.: Convtimenet: A pre-trained deep convolutional neural network for time series classification. arXiv:1904.12546 (2019)
28. Kavyashree, B., Patil, S., Rao, V.S.: Review on vibration control in tall buildings: from the perspective of devices and applications. International Journal of Dynamics and Control **9**(3), 1316–1331 (2021)
29. Längkvist, M., Karlsson, L., Loutfi, A.: A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters **42**, 11–24 (2014)
30. Li, A., Sun, J., Li, P., Pu, Y., Li, H., Chen, Y.: Hermes: an efficient federated learning framework for heterogeneous mobile clients. In: Proceedings of the 27th Annual International Conference on Mobile Computing and Networking. pp. 420–437 (2021)
31. Li, D., Wang, J.: Fedmd: Heterogenous federated learning via model distillation. arXiv preprint arXiv:1910.03581 (2019)
32. Liu, R., Wu, F., Wu, C., Wang, Y., Lyu, L., Chen, H., Xie, X.: No one left behind: Inclusive federated learning over heterogeneous devices. arXiv preprint arXiv:2202.08036 (2022)
33. Liu, S., Yu, G., Yin, R., Yuan, J.: Adaptive network pruning for wireless federated learning. IEEE Wireless Communications Letters **10**(7), 1572–1576 (2021)
34. Liu, Y., Garg, S., Nie, J., Zhang, Y., Xiong, Z., Kang, J., Hossain, M.S.: Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach. IEEE Internet of Things Journal **8**(8), 6348–6358 (2020)
35. Long, G., Shen, T., Tan, Y., Gerrard, L., Clarke, A., Jiang, J.: Federated learning for privacy-preserving open innovation future on digital health. In: Humanity Driven AI: Productivity, Well-being, Sustainability and Partnership, pp. 113–133. Springer (2021)
36. Long, G., Tan, Y., Jiang, J., Zhang, C.: Federated learning for open banking. In: Federated Learning: Privacy and Incentive, pp. 240–254. Springer (2020)
37. Long, G., Xie, M., Shen, T., Zhou, T., Wang, X., Jiang, J.: Multi-center federated learning: clients clustering for better personalization. World Wide Web **26**(1), 481–500 (2023)
38. Park, S., Constantinides, M., Aiello, L.M., Quercia, D., Van Gent, P.: Wellbeat: A framework for tracking daily well-being using smartwatches. IEEE Internet Computing **24**(5), 10–17 (2020)
39. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International conference on machine learning. pp. 1310–1318 (2013)
40. Progonov, D., Sokol, O.: Heartbeat-based authentication on smartwatches in various usage contexts. In: International Workshop on Emerging Technologies for Authorization and Authentication. pp. 33–49. Springer (2021)
41. Rajpurkar, P., Hannun, A.Y., Haghpanahi, M., Bourn, C., Ng, A.Y.: Cardiologist-level arrhythmia detection with convolutional neural networks. arXiv:1707.01836 (2017)
42. Serrà, J., Pascual, S., Karatzoglou, A.: Towards a universal neural network encoder for time series. In: CCIA. pp. 120–129 (2018)
43. Singh, A., Vepakomma, P., Gupta, O., Raskar, R.: Detailed comparison of communication efficiency of split learning and federated learning. arXiv preprint arXiv:1909.09145 (2019)
44. Tan, A.Z., Yu, H., Cui, L., Yang, Q.: Towards personalized federated learning. IEEE Transactions on Neural Networks and Learning Systems (2022)
45. Tan, C.W., Webb, G.I., Petitjean, F.: Indexing and classifying gigabytes of time series under time warping. In: Proceedings of the 2017 SIAM international conference on data mining. pp. 282–290. SIAM (2017)
46. Tan, Y., Liu, Y., Long, G., Jiang, J., Lu, Q., Zhang, C.: Federated learning on non-iid graphs via structural knowledge sharing. In: Proceedings of the AAAI conference on artificial intelligence. vol. 37, pp. 9953–9961 (2023)

47. Tan, Y., Long, G., Liu, L., Zhou, T., Lu, Q., Jiang, J., Zhang, C.: Fedproto: Federated prototype learning across heterogeneous clients. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 8432–8440 (2022)
48. Tan, Y., Long, G., Ma, J., Liu, L., Zhou, T., Jiang, J.: Federated learning from pre-trained models: A contrastive learning approach. Advances in Neural Information Processing Systems **35**, 19332–19344 (2022)
49. Tang, W., Liu, L., Long, G.: Interpretable time-series classification on few-shot samples. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2020)
50. Tang, W., Long, G., Liu, L., Zhou, T., Blumenstein, M., Jiang, J.: Omni-scale cnns: a simple and effective kernel size configuration for time series classification. In: International Conference on Learning Representations (2021)
51. Vidal, F., Navarro, M., Aranda, C., Enomoto, T.: Changes in dynamic characteristics of lorca rc buildings from pre-and post-earthquake ambient vibration data. Bulletin of Earthquake Engineering **12**(5), 2095–2110 (2014)
52. Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., Khazaeni, Y.: Federated learning with matched averaging. arXiv preprint arXiv:2002.06440 (2020)
53. Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: A strong baseline. In: 2017 international joint conference on neural networks. pp. 1578–1585. IEEE (2017)
54. Xing, L.: Reliability in internet of things: Current status and future perspectives. IEEE Internet of Things Journal **7**(8), 6704–6721 (2020)
55. Xu, W., Fang, W., Ding, Y., Zou, M., Xiong, N.: Accelerating federated learning for iot in big data analytics with pruning, quantization and selective updating. IEEE Access **9**, 38457–38466 (2021)
56. Xu, Z., Yang, Z., Xiong, J., Yang, J., Chen, X.: Elfish: Resource-aware federated learning on heterogeneous edge devices. Ratio **2**(r1), r2 (2019)
57. Yan, P., Long, G.: Personalization disentanglement for federated learning. arXiv preprint arXiv:2306.03570 (2023)
58. Zhang, C., Long, G., Zhou, T., Yan, P., Zhang, Z., Zhang, C., Yang, B.: Dual personalization on federated recommendation. arXiv preprint arXiv:2301.08143 (2023)
59. Zhang, L., Yuan, X.: Fedzkt: Zero-shot knowledge transfer towards heterogeneous on-device models in federated learning. arXiv preprint arXiv:2109.03775 (2021)
60. Zhang, T., Gao, L., He, C., Zhang, M., Krishnamachari, B., Avestimehr, A.S.: Federated learning for the internet of things: Applications, challenges, and opportunities. IEEE Internet of Things Magazine **5**(1), 24–29 (2022)
61. Zheng, Y., Liu, Q., Chen, E., Ge, Y., Zhao, J.L.: Time series classification using multi-channels deep convolutional neural networks. In: International Conference on Web-Age Information Management. pp. 298–310. Springer (2014)
62. Zhu, Z., Hong, J., Zhou, J.: Data-free knowledge distillation for heterogeneous federated learning. In: International Conference on Machine Learning. pp. 12878–12889. PMLR (2021)

A Appendix

Theoretical analysis Mathematically, we would show that: when kernels of various sizes meet the three requirements: 1) they were alignment, 2) they are of the same output, and 3) they are of the same target, the weight of those alignment parts will be optimized with the same direction.

Supposing we have weights of two kernels $\mathbf{A}$ and $\mathbf{B}$ of length a and b. The input signal is $\mathbf{T}$ which length is t.

Each layer of 1D-CNNs requires the output's length equal to the input's length. Thus, we need to pad the input signal. The padding size has a relationship with the length of the kernel. For kernel of length $*$, we denote the padded input as $\mathbf{P}^*$ and the output as $\mathbf{O}^*$.

Based on the definition of convolution operation [1], we know the gradient of the i-th element for kernels $\mathbf{A}_i$

$$\nabla \mathbf{A}_i = \sum_{k=0}^{t-1} \frac{\nabla \mathbf{O}_k^a}{\mathbf{P}_{i+k}^a} \tag{7}$$

and the gradient of the j-th element for kernels $\mathbf{B}_b$ is

$$\nabla \mathbf{B}_j = \sum_{k=0}^{t-1} \frac{\nabla \mathbf{O}_k^b}{\mathbf{P}_{j+k}^b} \tag{8}$$

Based on the assumption 2) and 3) when two kernel are of same output and same target they will be of same gradient. Thus we have

$$\nabla \mathbf{O}_*^a \equiv \nabla \mathbf{O}_*^b \tag{9}$$

and we could simplified to $\mathbf{O}_i$

Based on the assumption 3) Under the weight relay setting, every model share the same target (data on all devices). Thus we have:

$$E\left(\nabla \mathbf{O}_i^s\right) = E\left(\nabla \mathbf{O}_i^l\right) \tag{10}$$

Since $\mathbf{A}$ and $\mathbf{B}$ are aliened, we could combine the Equation 8 with Equation 1, and we get:

$$\nabla \mathbf{B}_{i+\lfloor (b-1)/2 \rfloor - \lfloor (a-1)/2 \rfloor} = \sum_{k=0}^{t-1} \frac{\nabla \mathbf{O}_k}{\mathbf{P}_{i+\lfloor (b-1)/2 \rfloor - \lfloor (a-1)/2 \rfloor + k}^b} \tag{11}$$

Therefore if:

$$\mathbf{P}_{i+k}^a \equiv \mathbf{P}_{i+\lfloor (b-1)/2 \rfloor - \lfloor (a-1)/2 \rfloor + k}^b \tag{12}$$

We could have:

$$\nabla \mathbf{A}_i \equiv \nabla \mathbf{B}_{i+\lfloor (b-1)/2 \rfloor - \lfloor (a-1)/2 \rfloor} \tag{13}$$

[1] https://e2eml.school/convolution_one_d.html

The Equation 13 means the alignment parts will be optimized with the same direction.

Based on the definition of same padding, we know:

$$\mathbf{P}^{*}_{m} = \begin{cases} \mathbf{T}_{m-\lfloor(*-1)/2\rfloor}, & \text{if } 0 \leq m - \lfloor(*-1)/2\rfloor \leq t-1 \\ \text{padding value}, & \text{otherwise} \end{cases} \tag{14}$$

We could see values in $\mathbf{P}^{a}_{i+k}$ are:

$$\mathbf{P}^{a}_{i+k} = \begin{cases} \mathbf{T}_{i+k-\lfloor(a-1)/2\rfloor}, & \text{if } 0 \leq i+k - \lfloor(a-1)/2\rfloor \leq t-1 \\ \text{padding value}, & \text{otherwise} \end{cases} \tag{15}$$

and values in $\mathbf{P}^{b}_{i+\lfloor(b-1)/2\rfloor-\lfloor(a-1)/2\rfloor+k}$ are also:

$$\mathbf{P}^{b}_{i+\lfloor(b-1)/2\rfloor-\lfloor(a-1)/2\rfloor+k} = \begin{cases} \mathbf{T}_{i+k-\lfloor(a-1)/2\rfloor}, & \text{if } 0 \leq i+k - \lfloor(a-1)/2\rfloor \leq t-1 \\ \text{padding value}, & \text{otherwise} \end{cases} \tag{16}$$

Based on Equation 15 and Equation 16, we know the Equation 12 is always correct. Thus, weight of those alignment parts will be optimized with the same direction as it described in Equation 13

In real cases, we cannot expect the target of small and large networks are the same. However, it is reasonable to assume that those two networks' targets are similar. In Figure 5, we random 5000 sets of kernels and two targets and plot the relationship between gradient and targets. We could see that the small kernel's gradient direction is similar to the gradient direction of the aligned part when the target is similar. Therefore, as long as their targets are similar, the weight relay could accelerate the training of the large network.

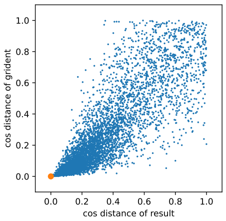

Fig. 5. The blue points are results from 5000 tests with random input, targets, and weight. The orange point is the origin point. We could see that when the value of the x-axis approaches 0 (target similar), the value of the y-axis comes to 0 (gradient direction similar).

A.1 Robust to various early stop criteria

This section shows that the weight relay is robust to various early stop criteria. Specifically, the experiment in Figure 4 is run with the learning rate of 0.0001; the train will stop when the loss is less than 1e-3 or 5000 epochs. In figure 6, the experiment will stop if the value of loss changing is less than 2% within 200 epochs. In Figure 7, we change the learning rate to 0.01 while keeping other settings the same as Figure 6. We could see that the weight relay consistently shows effectiveness with these various hyper-parameter settings.

A.2 Case study of the convergence acceleration

In this section, the top three datasets in the UCR archive of most training samples are selected. As we explained in Section 5.2, the visualization in Figure A.2 confirms that the weight relay could accelerate convergence because the weight from a well-trained small network is closer to the target weight than random initialization.

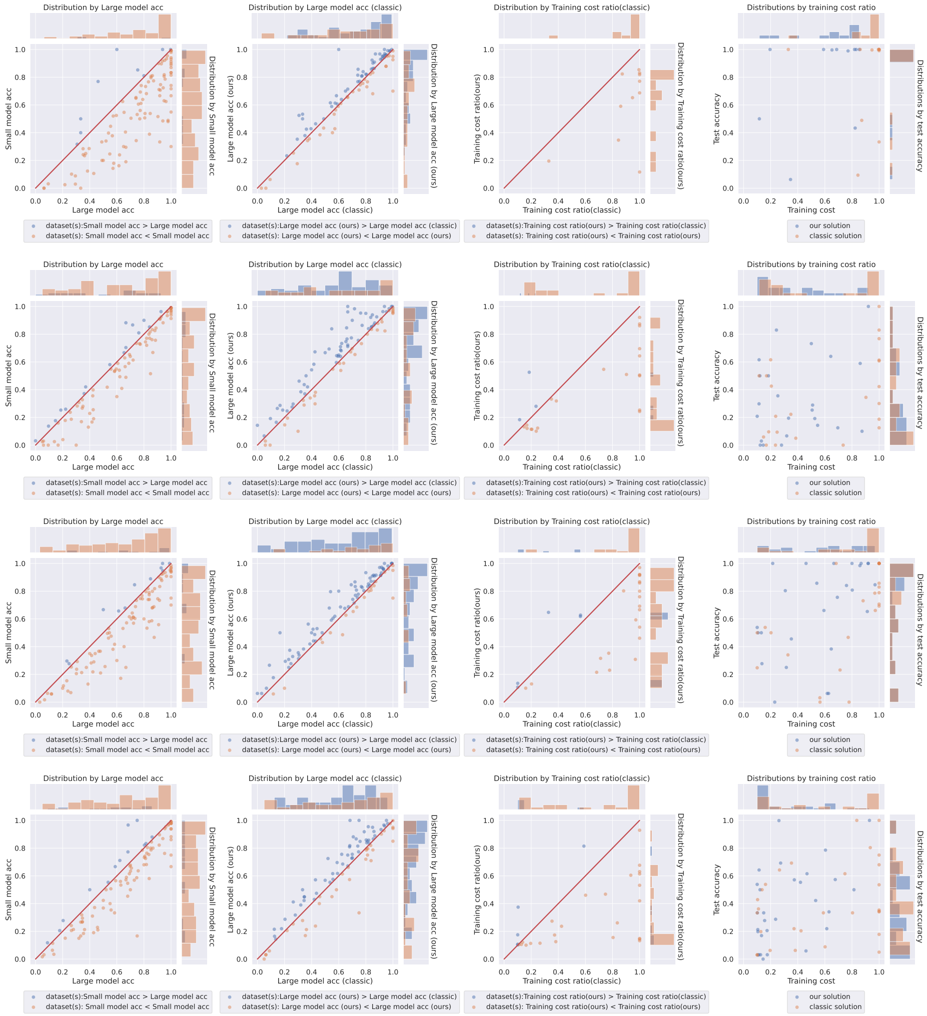

Fig. 6. The experiment results under the stop criteria: the value of loss changing is less than 2% within 200 epochs.

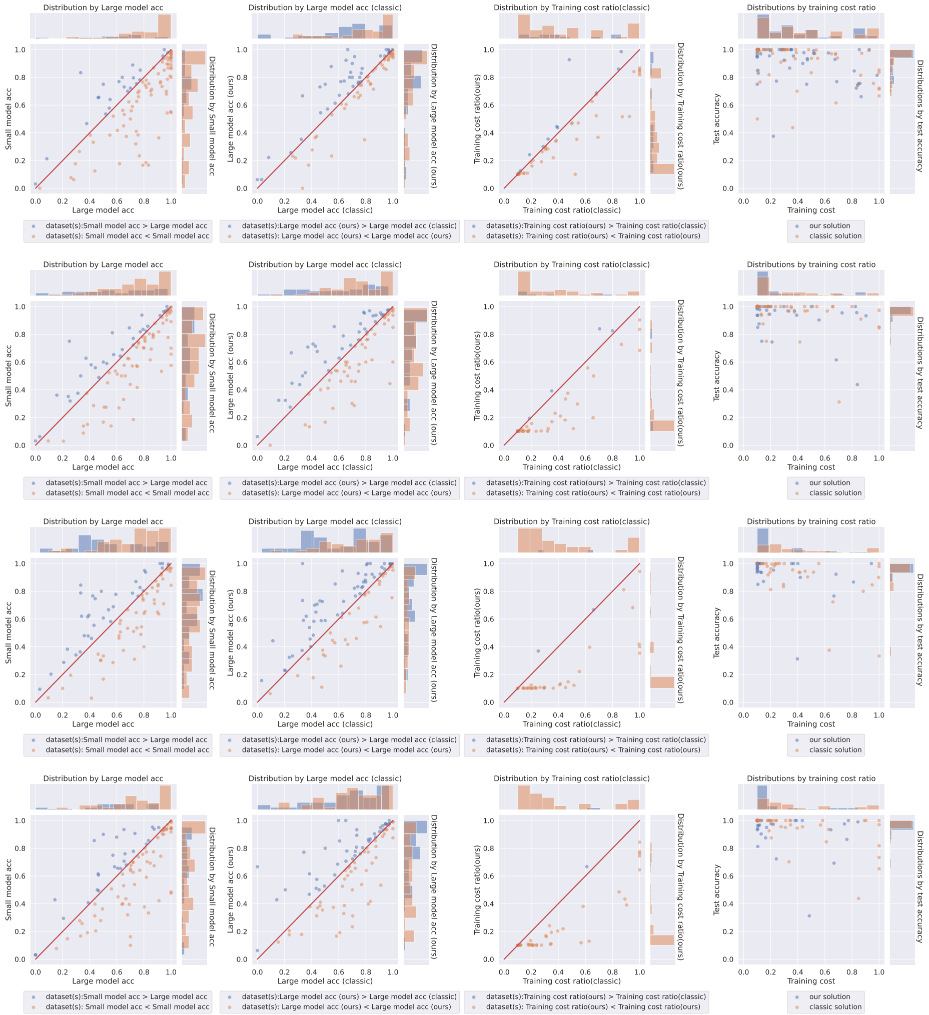

Fig. 7. The experiment results under the learning rate 0.01.

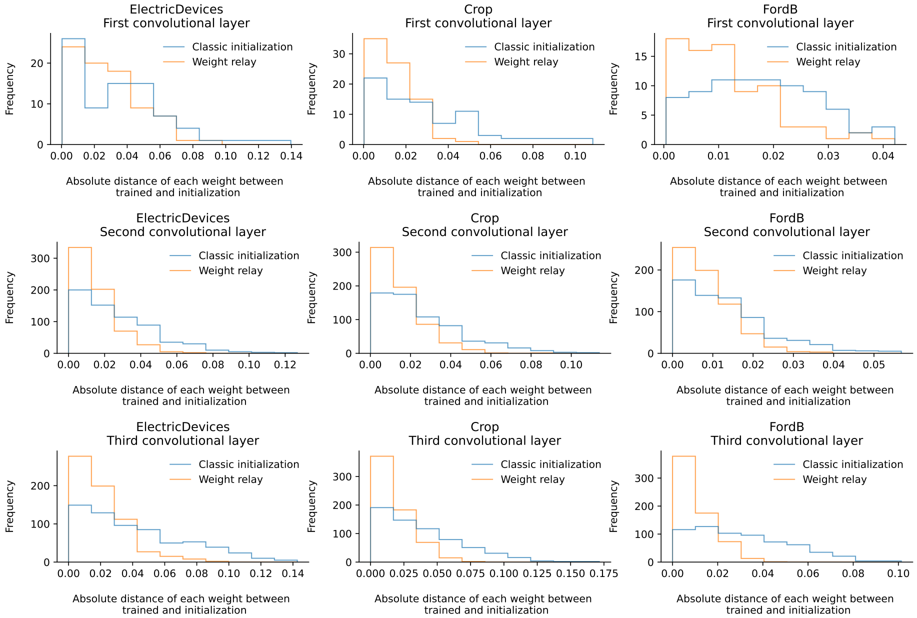

Fig. 8. This histogram statistic the distance between the initialization weight and trained weight. As we can see, compared with the random initialization, the weight relay initialization are of a smaller distance to the final value. Therefore, it will be easier to converge than a random initialization one.