Master's Thesis of Data Science

Federated Semi-Supervised Learning with Prototypical Network

프로토티피컬 네트워크를 이용한 연합 준 지도 학습

February 2022

Graduate School of Seoul National University
Department of Data Science (Data Science Major)

Woojung Kim

Federated Semi-Supervised Learning with Prototypical Network

Advisor Hyung-Sin Kim

Submitting a master's thesis of
Data Science

December 2021

Graduate School of Seoul National University
Department of Data Science (Data Science Major)

Woojung Kim

Confirming the master's thesis written by
Woojung Kim

January 2022

Chair	Jaejin Lee	(Seal)
Vice Chair	Hyung-Sin Kim	(Seal)
Examiner	Wen-Syan Li	(Seal)
Examiner	Seunggeun Lee	(Seal)

Abstract

Federated Learning (FL) is being actively studied as computing power of edge devices increase. Most of the existing studies assume that there are full labels of data. However, since labeling data on the edge devices requires high cost, this assumption is not suitable in the real world. In general, most of the data each client has is often unlabeled. In this study, we propose a novel federated semi-supervised learning (FSSL) method. It uses *prototype* to utilize other clients' knowledge and pseudo-labeling to compute loss about unlabeled data. It is a communication and computation efficient method than recent FSSL algorithm. In experiments, we showed that our method performed 3.8% better than not using unlabeled data with CIFAR-10 dataset, 4.6% better with SVHN dataset and 3.1% better with STL-10 dataset.

Keyword : federated learning, semi-supervised learning, prototypical network, pseudo-labeling
Student Number : 2020-23120

Table of Contents

Chapter 1. Introduction

Federated Learning (FL)[1] is a machine learning framework that has been actively studied recently. Instead of sending data accumulated on edge devices(clients) to the server, models are local trained using the computing power of the edge devices. Each model is then sent to the server, and the server updates the global model using FedAvg and then broadcasts it to the edge devices.

There are many studies to improve FL algorithms. However, most of the existing studies have been done in a situation of supervised learning [2,3,4] that all clients have full labels of data. This is not the suitable setting for real world because data on the edge devices often do not have labels. For example, if data is collected to the server, some experts can be hired to label them, but data on the edge device is less likely to be labeled by users, and it may be mislabeled because they are not professional. Therefore, it is important to do study Federated Semi-Supervised Learning (FSSL).

Recently, there are some researches about FSSL. In FedMatch [5], they utilized unlabeled data using *inter-client consistency loss*. One client uses other clients' weights to compute loss for unlabeled data. They showed naïve solutions that just applying state of the art semi-supervised learning methods such as FixMatch [7] and UDA [6] to FL are not effective because in federated settings each client has much less data compared to their centralized setting, and they cannot use knowledge of other clients. On the other hand, FixMatch uses knowledge from other clients, so they showed good performance through experiments.

Although their method improved accuracy utilizing unlabeled data, there are some drawbacks. First of all, there is increased communication overhead. In order to use other clients' knowledge, the server has to send other clients' weights of the local model. So, the communication cost increases linearly to the model size as the number of clients for utilizing increases. The other drawback is that increased computation overhead. When the client computes loss for its unlabeled data, it has to run the model as many times as the number of other clients utilized. For both cases, if the size of the model is not so large, it may not be a problem, but vice versa, it will be a lot of overhead. To address these problems, we propose a new method for FSSL. Our method utilizes *prototypes*, not weights, so there is low overhead for communication. Also, there is low computation cost because it does not need to run the model several times. The contributions for our research are as follows:

- We propose new FSSL algorithm that utilizes other clients' knowledge when computing loss for unlabeled data.
- Our method has better accuracy than FedAvg that does not use unlabeled data and FedAvg with a semi-supervised learning method.
- Our method is more efficient in terms of communication and computation cost than FedMatch.

Chapter 2. Backgrounds

2.1. Federated Learning

Federated learning (FL) [1] is distributed learning between a server and clients that the clients do not send their data to the server but send their local models' parameters. The training proceeds by repeating several rounds. When the round starts, the server sends parameters of the global model θ_g to the clients. Then each client trains global parameters θ_g as local parameters θ_c using their own data. After finishing local training, each client sends their local parameters θ_c to the server. Then, the server updates the global model with local parameters using FedAvg.

$$\theta_g = \frac{1}{\sum_{c=1}^{C} n_c} \sum_{c=1}^{C} n_c \cdot \theta_c \tag{1}$$

where C is the total number of clients that participated training in the round and n_c is the number of data that the client c used for training.

2.2. FedMatch

In FedMatch [5], the main part of the algorithm is *inter-client consistency loss* that is computed using unlabeled data. Let $p_\theta(y|u)$ be a neural network that is parameterized by weights θ and predicts softmax output y with given unlabeled input u. Then, inter-client consistency loss in client l is defined as

$$loss = \frac{1}{C} \sum_{j=1}^{C} KL[p^*_{\theta^j}(y|u) || p_{\theta^l}(y|u)] \tag{2}$$

where C is the number of other clients that are used in client l, θ^j is weights for C clients (* denotes that the parameters are frozen), θ^l is weights of client l, and KL means Kullback-Leibler divergence loss. In the original paper, loss using pseudo-labeling is also added, but it is omitted for convenience of explanation. As we can see, other clients' weights are needed to compute the loss. So, the server needs to send other clients' weights as well as the global averaged weights. Furthermore, each client needs to run the model as many times as the number of clients. As a result, communication cost and computation cost are increased C times compared to FedAvg.

2.3. Prototypical Network

Prototypical network [8] is originally used for meta-learning algorithm. However, our algorithm uses that it has the training process based on metric learning. The network is trained to learn good embedding vectors for input. Let D is a training set and K is a set of all classes. Then for each class k in K, select *support set* S_k that is a random subset of D_k, a training set of class k. And select *query set* Q_k that is random subset of $D_k \setminus S_k$. Then compute prototype c_k from support set for each class as

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\theta(x_i) \tag{3}$$

where f_θ is a neural network that is parameterized by weights θ. Then, loss is computed as

$$loss = \frac{1}{|K||Q_k|} \sum_{k \in K} \sum_{(x,y) \in Q_k} \left[d(f_\theta(x), c_k) + log \sum_{k' \in K \setminus k} \exp\left(-d(f_\theta(x), c_{k'})\right) \right] \tag{4}$$

where d is Euclidean distance function. The model is trained so that embedding vectors of the same class are located close together, and those of different classes are located far away.

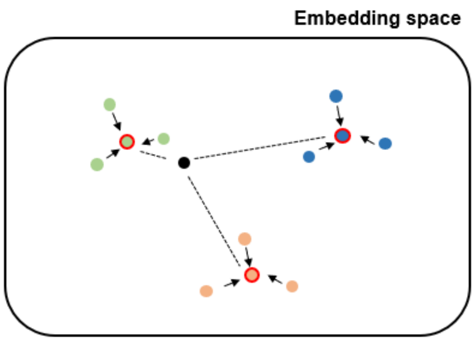

Figure 1. Colored circles without a red border are embedding vectors of support set. Each color means different class. Each prototype of classes is computed by averaging embedding vectors of each class and is expressed with a red border. The black circle means an embedding vector of query set and the distances between prototypes is used for computing loss.

2.4. Pseudo−Labeling

Pseudo−labeling is one of the techniques mainly used in semi−supervised learning. It sets pseudo−label for unlabeled data and then use them to compute loss. There are several methods about pseudo−labeling and we use one of them that used in MixMatch [9].

In MixMatch, it uses several augmented data for one unlabeled data to make pseudo−label. Let p_θ be a neural network parameterized by θ, u_b be one unlabeled data, and $\hat{u}_{b,k}$ be an augmented data of u_b. Then, pseudo−label is computed as

$$q_b = \frac{1}{K}\sum_{k=1}^{K} p_\theta(y|\hat{u}_{b,k}) \tag{5}$$

i.e., average of K augmented data. And then, it uses entropy minimization to lower the entropy of guessed label as follows:

$$\bar{q}_{b,i} = q_{b,i}^{1/T} / \sum_{j=1}^{L} q_{b,j}^{1/T} \tag{6}$$

where L is the number of classes, $q_{b,i}$ is ith class of probability, and T is a hyperparameter. As $T \to 0$, the probability distribution will be a one−hot distribution.

Chapter 3. Algorithm

In this chapter, we would explain about what our algorithm is. The algorithm is explained with four parts: *Server to Clients*, *In Clients*, *Clients to Server*, and *In Server*. These four parts are repeated every round.

```
**Algorithm 1:** Federated Semi-Supervised with Prototypical Network
$c$: prototypes of all classes
**RunServer:**
Initialize global parameter $\theta^0$
**for** each round $r$ = 1, 2, …, R **do**
    $A^r \leftarrow$ (random sample of $N$ clients)
    Send $\theta^{r-1}$, $\{c_t^{r-1}: t \in A^{r-1}\}$ to all clients
    **for** each client t ∈ $A^r$ **in parallel do**
        $\theta_t^r, c_t^r \leftarrow$ RunClient
    **end for**
    $\theta^r \leftarrow \frac{1}{N}\sum_{t=1}^{N} \theta_t^r$
    Store $\{c_t^r: t \in A^r\}$
**end for**

**RunClient:**
$D_s$: labeled dataset, $D_u$: unlabeled dataset
$\bar{y}_u \leftarrow PseudoLabeling(D_u, \{c_t^{r-1}: t \in A^{r-1}\})$
**for** each episode e = 1, 2, …, E **do**
    support set $R \leftarrow RandomSample(D_s)$
    query set $Q_s \leftarrow RandomSample(D_s \setminus R)$
    query set $Q_u \leftarrow RandomSample(D_u)$
    prototype $c \leftarrow MakePrototype(R)$
    $l_s, l_u \leftarrow ComputeLoss(c, Q_s, Q_u, \bar{y}_u)$
    $Loss = l_s + \lambda_u \cdot l_u$
    $\theta \leftarrow \theta - \eta \nabla_\theta Loss$
**end for**
prototype $c \leftarrow MakePrototype(D_s)$
```

3.1. Server to Clients

At the start of the round r, the server sends prototypes c_t^{r-1} of the clients A^{r-1} which participated in the previous round as well as the global weights θ^{r-1} to the clients A^r that participate in the round. In the later section, we will explain how the prototypes of other clients and the global weights are computed.

3.2. In Clients

After receiving global weights and other clients' prototypes, the client first pseudo−labels the unlabeled data using other clients' prototypes. For each unlabeled data $x \in D_u$, compute embedding vector z using local model parameterized by θ.

$$z = f_\theta(x) \tag{7}$$

Then compute Euclidian distance $d_{t,k}$ between each class k of prototypes $c_{t,k}$ for each client t that received from the server.

$$d_{t,k} = ED(z, c_{t,k}) \tag{8}$$

where ED means Euclidean distance. Probability distributions can be computed using softmax of the negative of the distance for each class.

$$p_{t,k} = e^{-d_{t,k}} \Big/ \sum_{j=1}^{L} e^{-d_{t,j}} \tag{9}$$

where L is the total number of classes. After computing probability distributions for every other client' s prototypes, pseudo−labels are computed by averaging probability distributions and entropy minimization.

$$p_k = \frac{1}{N} \sum_{t=1}^{N} p_{t,k} \tag{10}$$

$$\bar{y}_k = p_k^{1/T} / \sum_{k=1}^{L} p_k^{1/T} \tag{11}$$

where T is a hyperparameter for entropy minimization.

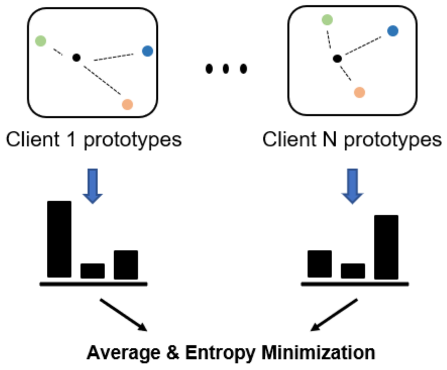

Figure 2. Pseudo−labeling.

Now, we'll explain how to compute loss. First, a support set R is selected using some part of the labeled data D_s, and the prototype c_k for each class is computed by running it through the local model. The query set Q_s is selected from the remaining labeled data and the other query set $Q_u$is also selected from the pseudo−labeled data and create embedding vectors by running the model for each query set. Loss for labeled data is computed using equation (4). Loss for unlabeled data is computed little bit different. A hyperparameter λ_u that regulates the effectiveness of the loss of unlabeled data is multiplied for the loss of the unlabeled data.

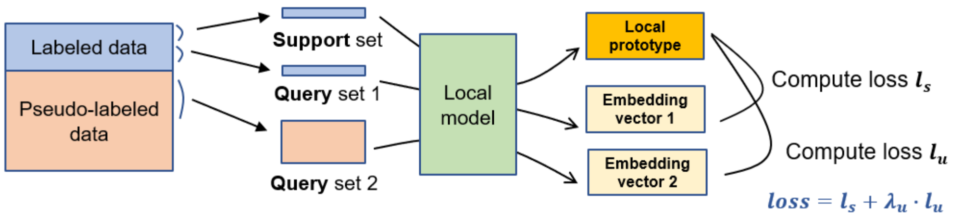

Figure 3. Computing loss

After local training, compute prototypes for sending to the server using all of the labeled data.

3.3. Clients to Server

The client not only sends the updated local weights to the server, but also sends the prototypes for each class.

3.4. In Server

The server updates the global weights using FedAvg with the weights received from each client and keeps the prototypes The server sends the prototypes and the updated global weights to the clients participating training in the next round.

Chapter 4. Experiments

4.1. Experimental Setup

Tasks

We use CIFAR−10, SVHN, and STL−10 datasets. For CIFAR−10, we use 54,000 training set, 3,000 valid set, and 3,000 test set. For SVHN, we use 54,000 training set, 2,000 valid set, and 2,000 test set. For STL−10, we use 108,000 training set, 1500 valid set, and 1500 test set. When split datasets, all splits have the same numbers of data per class. For CIFAR−10 and SVHN, we extract 5,000 labeled data and use the rest of the training set (49,000) as unlabeled data. For STL−10, we use 10,000 labeled data and 98,000 unlabeled data.

There are 100 clients and we randomly select 5 active clients that participate training for each round. Each client has 50 labeled data and 490 unlabeled data for CIFAR−10 and SVHN, and 100 labeled data and 980 unlabeled data for STL−10. For labeled data, there are 5 labeled data per class. For unlabeled data, we assume two settings: IID and non−IID. For IID setting, there are 49 unlabeled data per class and for non−IID setting, each class have the different number of data, (244, 73, 73, 15, 15, 15, 15, 15, 15, 10) with different combination of order that use in [5].

Baseline

Our baselines are: 1) FedAvg−SL and FedProx[10]−SL: each client has full labeled data. They update the global model via FedAvg and FedProx frameworks 2) FedAvg and FedProx: each client trains with 50 labeled data and does not use unlabeled data. 3) FixMatch−FedAvg and FixMatch−FedProx: naive combinations of FixMatch with FedAvg/FedProx. FixMatch is one of the SOTA algorithms for semi−supervised learning. 4) FedMatch: recent federated semi−supervised learning method.

Models

we use ResNet−9 network, same architecture that use in [5]. For our proposal method, it does not need output softmax layer, so we remove the output layer. There is detailed architecture of the network in Table 1. The network has about 6 million parameters.

Layer	Filter Shape	Stride	Output
Input	N/A	N/A	32×32×3
Conv 1	3×3×3×64	1	32×32×64
Conv 2	3×3×64×128	1	32×32×128
Pool 1	2×2	2	16×16×128
Conv 3	3×3×128×128	1	16×16×128
Conv4	3×3×128×128	1	16×16×128
Conv 5	3×3×128×256	1	16×16×256
Pool 2	2×2	2	8×8×256
Conv 6	3×3×256×512	1	8×8×512
Pool 3	2×2	2	4×4×512
Conv 7	3×3×512×512	1	4×4×512
Conv 8	3×3×512×512	1	4×4×512
Pool 4	4×4	4	1×1×512
Softmax	512×10	N/A	1×1×10

Table 1. Network architecture of ResNet−9

Hyperparameters

The number of rounds is 300. We use RMSprop optimizer with 1e−3 initial learning rate. For baseline, each client trains 1 local epochs and 8 batch size. For FedProx, the coefficient of regularization term is 1e−3. For FixMatch, the number of transformations is 2 and the hyperparameter for transformation range is 10. Weight of loss for unlabeled data is 1e−2 and batch size for unlabeled data is 100. For proposed method, the size of support set is 10 (1 per 10 class), the size of labeled query set is 20 (2 per 10 class), and the size of unlabeled query set is 100 (random selection). The number of local episodes is 10. The hyperparameter for entropy minimization is 0.5 and weight of loss for unlabeled data is 3e−1.

4.2. Experimental Results

Table 2. is the details of experimental results. The accuracies of FedAvg/FedProx−SL are upper bounds of accuracies of other methods. When measuring the performance of our proposed method, we use majority votes with the prototypes of the clients that participated the round. We observe that our proposal with FedAvg framework outperforms for all tasks. We can see that naive combinations of FixMatch with FedAvg/FedProx do not perform very well compared to FedAvg/FedProx that do not use unlabeled data. This is because each client has small amount of data and they do not use other clients' knowledge. On the other hands, our method utilizes other clients' knowledge when using unlabeled data. We also observe that our method is robust with non−IID setting.

CIFAR−10		
Method	Test Acc.(%)	
	IID	Non IID
FedAvg−SL	81.7	80.0
FedProx−SL	81.6	79.2
FedAvg	62.2	
FedProx	62.0	
	IID	Non IID
FixMatch−FedAvg	62.4	61.8
FixMatch−FedProx	62.9	61.6
FedMatch	57.7	59.1
Proposal−FedAvg	**65.0**	**64.2**
Proposal−FedProx	64.4	63.8

SVHN		
Method	Test Acc.(%)	
	IID	Non IID
FedAvg−SL	93.4	91.7
FedProx−SL	92.9	91.9
FedAvg	93.5	
FedProx	82.8	
	IID	Non IID
FixMatch−FedAvg	86.5	87.0
FixMatch−FedProx	87.6	87.5
Proposal−FedAvg	**88.1**	**87.7**
Proposal−FedProx	87.2	**87.7**

STL−10	
Method	Test Acc.(%)
FedAvg	72.3
FedProx	72.0
	IID
FixMatch−FedAvg	73.7
FixMatch−FedProx	73.6
Proposal−FedAvg	**75.4**
Proposal−FedProx	74.5

Table 2. Test accuracy. We use 100 clients (5 active clients per round) for 300 rounds. We measure the global model accuracy. The ratio of labeled data and unlabeled data is about 1:10

Figure 4. is test accuracy for each communication rounds. We plot IID cases for CIFAR−10 and SVHN dataset.

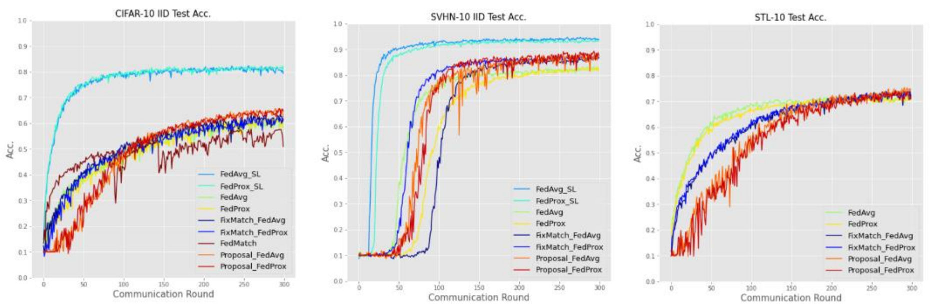

Figure 4. Test accuracy for communication rounds

Figure 5. is performances with different ratio of labeled and unlabeled data. It shows that as the number of unlabeled data decreases and the number of labeled data increases, the performance between FedAvg that does not use unlabeled data and our method becomes similar. It means that our method can have better performance when it can utilize more unlabeled data.

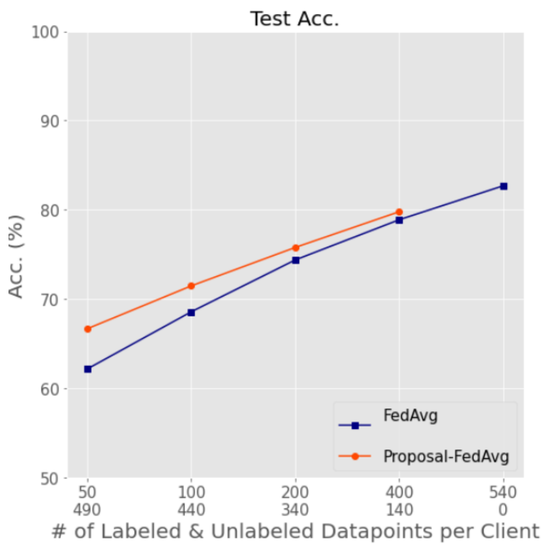

Figure 5. Performances with different ratio of labeled and unlabeled data.

Figure 5. is performances with different ratio of labeled and unlabeled data. It shows that as the number of unlabeled data decreases and the number of labeled data increases, the performance between FedAvg that does not use unlabeled data and our method becomes similar. It means that our method can have better performance when it can utilize more unlabeled data.

	Comm. Cost per Round	Comp. Cost per Round
FedMatch	14.46 MB	1229.9 GFLOP
Proposal	13.16 MB	789.7 GFLOP

Table 3. Communication cost and Computation cost for FedMatch and proposal.

Table 3. is communication cost and computation cost for FedMatch and the proposal. For communication cost, we use prototypes, not the model parameters, so communication cost is 9% less than FedMatch. For computation cost, FedMatch has to run other clients' models to compute loss for unlabeled data, so it has more overhead. By the way, our method only need to compute distance between prototypes and embedding vectors, so computation cost is about 35% less than FedMatch.

Chapter 5. Conclusion

We propose the new framework for FSSL. With experiments, we show that our method can utilize well other clients' knowledge with unlabeled data. Furthermore, there is small additional costs for communication and computation. The dimension of the prototype is much lower than the number of parameters of weights. Computing distance between embedding vectors and prototypes is also much lower overhead compared to running the model. In conclusion, our method is effective and efficient framework for FSSL.

FSSL is a field that has not been studied much yet. Therefore, it is necessary to conduct more experiments in various setting, such as additional datasets, different number of clients, different distribution of labeled and unlabeled dataset, and so on.

Bibliography

[1] McMahan, Brendan, et al. ”Communication−efficient learning of deep networks from decentralized data.” Artificial intelligence and statistics. PMLR, 2017.
[2] He, Chaoyang, Murali Annavaram, and Salman Avestimehr. ”Group knowledge transfer: Federated learning of large cnns at the edge.” arXiv preprint arXiv:2007.14513 (2020).
[3] Zhu, Zhuangdi, Junyuan Hong, and Jiayu Zhou. ”Data−Free Knowledge Distillation for Heterogeneous Federated Learning.” arXiv preprint arXiv:2105.10056 (2021).
[4] Li, Qinbin, Bingsheng He, and Dawn Song. ”Model−Contrastive Federated Learning.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.
[5] Jeong, Wonyong, et al. ”Federated Semi−Supervised Learning with InterClient Consistency & Disjoint Learning.” arXiv preprint arXiv:2006.12097 (2020).
[6] Xie, Qizhe, et al. ”Unsupervised data augmentation for consistency training.” arXiv preprint arXiv:1904.12848 (2019).
[7] Sohn, Kihyuk, et al. ”Fixmatch: Simplifying semi−supervised learning with consistency and confidence.” arXiv preprint arXiv:2001.07685 (2020).
[8] Snell, Jake, Kevin Swersky, and Richard S. Zemel. ”Prototypical networks for few−shot learning.” arXiv preprint arXiv:1703.05175 (2017).
[9] Berthelot, David, et al. "Mixmatch: A holistic approach to semi−supervised learning." arXiv preprint arXiv:1905.02249 (2019).
[10] Li, Tian, et al. "Federated optimization in heterogeneous networks." Proceedings of Machine Learning and Systems 2 (2020): 429−450.

초　록

연합 학습은 엣지 디바이스의 계산 능력이 증가하면서 활발하게 연구되고 있는 분야이다. 대부분의 기존 연구는 클라이언트가 가지고 있는 데이터에 레이블이 모두 있다고 가정한다. 하지만, 엣지 디바이스의 데이터를 레이블링 하는 작업은 비용이 많이 들기 때문에, 이러한 가정은 실생활에서 적합하지 않다. 일반적으로, 클라이언트가 가지고 있는 데이터의 대부분은 레이블이 없는 경우가 많다. 본 연구에서, 우리는 새로운 연합 준 지도 학습 방법을 제안한다. 이것은 다른 클라이언트의 지식을 이용하기 위해 프로토타입이라는 것을 사용하고, 레이블이 없는 데이터를 학습시킬 때 수도 레이블링이라는 작업을 한다. 제안 방법은 최신 기법보다 높은 성능을 보이고, 또한 통신 비용과 계산 비용 측면에서 더 효율적인 방법론이다. 실험을 통해 우리의 알고리즘이 레이블이 없는 데이터를 사용하지 않은 경우에 비해 CIFAR-10 데이터셋에서는 3.8%, SVHN 데이터셋에서는 4.6%, 그리고 STL-10 데이터셋에서는 3.1% 성능이 더 좋다는 결과를 얻었다.

Keywords : **연합 학습, 준 지도 학습, 프로토타피컬 네트워크, 수도 레이블링**
Student Number : 2020-23120