Cost Effective Federated Learning for Heterogeneous Systems

Mentor :

Dr. Deepak Mishra

Members :

- **Mayank Raj (B19CSE053)**
- **Nirbhay Sharma (B19CSE114)**

Introduction

Federated learning approaches are increasingly become popular in today's scenario where data privacy is a key issue among organizations. FedAvg [1] is a simple yet powerful approach to aggregate the individual clients model parameters at the server end thus creating a final model that has the final knowledge from each client. Later on various other approaches are proposed such as fedprox [2] etc. The major issue with these approaches is, they require all the models parameters to be identical in order to aggregate them but they simply ignore the most realistic scenario where none of the architectures can be same. On the same line we design a Federated learning approach which tries to address the architectural differences among the clients and consider the harsh scenario where architectures are completely different that they cannot be aggregated in any manner. Next, we use datafree knowledge distillation approach for transferring knowledge from clients architectures to server architecture.

Background

Knowledge distillation

This is the most widely used technique to transfer knowledge from pretrained teacher model to comparitively lightweight student model. In naive knowledge distillation we train a Deep neural network architecture to achieve a benchmark performance, Next we distill the knowledge from teacher model to student model using KL divergence loss function. Suppose $T(D_x;t)$ denotes teacher output_logits on dataset D_x with parameters t. Similarly $S(D_x;s)$ denotes student output logits on dataset D_x with parameters s, σ denotes softmax activation function. The final loss function can be termed as follows:

$$L_{KL} = KL(\sigma(T(D_x;t))||\sigma(S(D_x;s)))$$

where $KL(y_{student}, y_{teacher})$ denotes KL divergence loss function defined as:

$$KL(y_{student}, y_{teacher}) = y_{teacher} * log(\frac{y_{teacher}}{y_{student}})$$

Data Free Knowledge distillation

In best scenario, the naive knowledge distillation approach involves the original data on which teacher model is trained. Distillation of knowledge from teacher to student using the real data helps student model to match

the performance of teacher model. But this is not the case for a realistic scenario. At industry level many organisations are not interested in sharing the dataset on which the teacher model is trained. In such a harsh scenario, data free knowledge distillation can help to distill the knowledge from teacher to student without involving the real data in the process.

Federated Learning

Federated learning is a technique to train the neural network architecture in a setting where multiple organizations (clients) are involved and data sharing is strictly prohibited. In such a scenario there are various approaches that are proposed, which involves FedAvg [1] , FedProx [2] etc. which aggregates the weight at the server side and train the clients on their respective data at client side. Simple FedAvg strategy is as follows, consider K clients having models weights as w_i at i^{th} client, w is the server network, (x_i, y_i) is the data available at client i^{th} client. For this case each of w_i's and w are identical we can formulate the scenario as

Client side

$$L_{CE} = CE(C(x_i; w_i), y_i)$$

$$w_i = w_i - \eta * \frac{\partial L_{CE}}{\partial w_i}$$

Server side

$$w = \frac{1}{K} \sum_{i=1}^{K} w_i$$

Methods

We extend the idea proposed in [3] to use noise based data free knowledge distillation to transfer the knowledge from client models at server end in federated setting. We modify several approaches to implement the idea of noise based distillation, Our results shows the significance of using noise based distillation in various federated settings.

Modified FedFTG

In original FedFTG [4], each communication round consist of randomly selecting a set of clients and broadcasting global model to them. Each individual client train their models with their local optimizer. The server aggregates the local models. This preliminary model is fine-tuned using knowledge from the extracted models. The server also maintains a conditional generator which generated psuedo data that mimic data distribution of the client. The major disadvantage of this approach is increasing the computation cost at the server end. The approach tries to create a distribution close to client's data distribution which eventually generates data similar to clients thus violating the core motivation behind federated learning. Considering the aforementioned issue we design a computationally efficient yet powerful approach which gives us advantage of getting rid of the generator. We distill the knowledge from clients to server using noise based knowledge distillation. The exact approach is explained below:

Client

The clients receive the server model and updated their local model with the global server model. They then train their local models using the server model by any of the FedAvg, FedProx etc. They then send their

respecitve models to the server side. (D_x, D_y) denotes the dataset that client holds, w_i denotes model parameters at client side, w denotes model parameters at server side, $C(D_x; w_i)$ denotes clients model. L denotes loss function at each client, σ denotes softmax function. Each client tries to update weights w_i as follows:

$$w_i = w$$

$$L = CE(\sigma(C(D_x; w_i), D_y)) + Reg(w, w_i)$$

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

Server

The server on first aggregates and updates server model using the average aggregation of clients models w_i. It then distill the knowledge from each clients models using KL divergence loss function. K denotes total number of clients, w_i denotes i^{th} client model, w denotes global model, Z denotes random noise generated following $N(0, 1)$, $S(Z; w)$ denotes the output of server model, σ denotes softmax function.

$$\delta w = \sum_{i=0}^{K} (w_i - w)$$

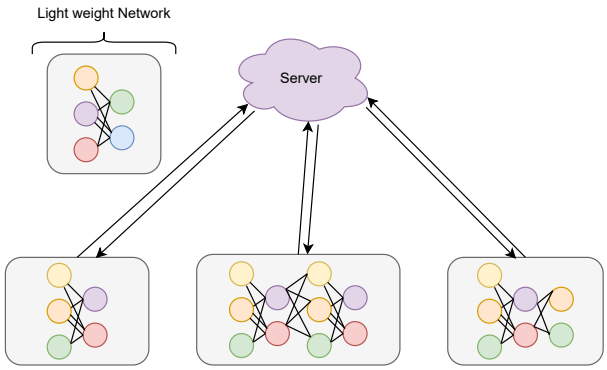
$$w = w + \eta \delta w$$

$$L = \frac{1}{K} \sum_{i=0}^{K} KL(\sigma(C(Z; w_i)), \sigma(S(Z; w)))$$

$$w = w - \eta \frac{\partial L}{\partial w}$$

Light Weight Network (LWN)

In this method, we use a smaller light weight model in the federated setting which will be broadcasted to the clients and it gets mutually train with the client's model. The server then collects the local networks, create their ensemble and distills them into the global network using random noise.



Client

The clients download the smaller lightweight network from the server and then mutually trains it with their local model and update the smaller light weight network and uploads it back to the server. (D_x, D_y) denotes the dataset that client holds, w_i denotes model parameters at client side, w denotes model parameters at server side, $C(D_x; w_i)$ denotes clients model output, $S(D_x; w)$ denotes server's model output. L_c denotes

client loss, L_s denotes server model loss, σ denotes softmax function. Each client tries to update weights w_i as follows

$$L_c = CE(C(D_x; w_i), D_y) + KL(\sigma(S(D_x; w)), \sigma(C(D_x; w_i)))$$

$$L_s = CE(S(D_x; w), D_y) + KL(\sigma(C(D_x; w_i)), \sigma(S(D_x; w)))$$

$$w_i = w_i - \eta \frac{\partial L_c}{\partial w_i}$$

$$w = w - \eta \frac{\partial L_s}{\partial w}$$

Server

The server collects the updated lightweight network from the clients and ensemble them to distill knowledge into the global network. The distillation is done in data-free manner using random noise. K denotes total number of clients, w_i denotes i^{th} client model, w denotes global model, Z denotes random noise generated following $N(0, 1)$, $S(Z; w)$ denotes the output of server model, σ denotes softmax function.

$$L = KL(max\sigma(C(Z; w_i))_{i=0}^{K}, \sigma(S(Z; w)))$$

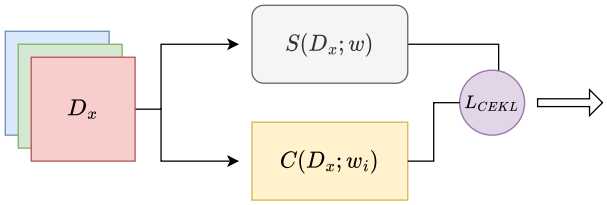
$$w = w - \eta \frac{\partial L}{\partial w}$$

Heterogeneous FedFL

In this setting we train the clients model using the server as a regularizer / teacher. The trained clients models are then send to the server for transferring knowledge from clients models to server models. For transferring knowledge from clients to server we use two strategies. First is to backpropagate the combine distillation loss from server to each client. Second is to first create an ensemble of each client and then distill the knowledge from ensemble to server.

Client

The client use the server model only as teacher to learn the knowledge from different clients as well. (D_x, D_y) denotes the dataset that client holds, w_i denotes model parameters at client side, w denotes model parameters at server side, $C(D_x; w_i)$ denotes clients model output, $S(D_x; w)$ denotes server's model output. L_{CEKL} denotes client loss, σ denotes softmax function. Each client tries to update weights w_i as follows



$$L_{CEKL} = CE(C(D_x; w_i), D_y) + KL(\sigma(S(D_x; w)), \sigma(C(D_x; w_i)))$$

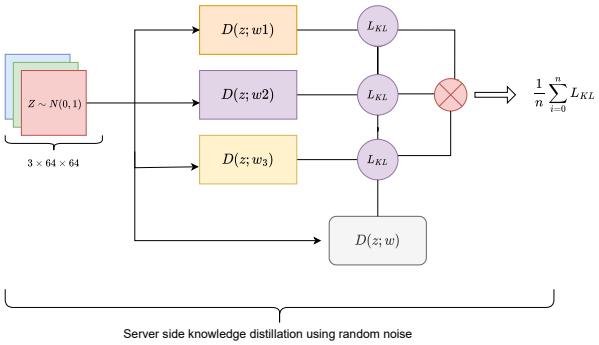
$$w_i = w_i - \eta \frac{\partial L_{CEKL}}{\partial w_i}$$

Server

The server distills the knowledge from the clients in two ways as follows:

Combine distillation loss

In this strategy the server takes the distillation loss with each clients and combine them to backpropagate the final loss. w_i denotes i^{th} client model, w denotes global model, Z denotes random noise generated following $N(0,1)$, $S(Z;w)$ denotes the output of server model, σ denotes softmax function.



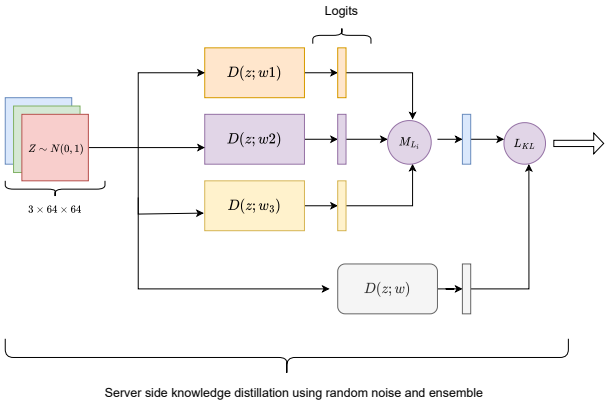
$$L_{KL} = KL(\sigma(C(Z;w_i)), \sigma(S(Z;w)))$$

$$L = \frac{1}{K}\sum_{i=0}^{K} L_{KL}$$

$$w = w - \eta\frac{\partial L}{\partial w}$$

Ensemble distillation loss

In this strategy the server first takes the ensemble among the clients. Next it distills the knowledge from ensemble to the server model. w_i denotes i^{th} client model, w denotes global model, Z denotes random noise generated following $N(0,1)$, $S(Z;w)$ denotes the output of server model, σ denotes softmax function.



$$M_l = max\{\sigma(C(Z;w_i))\}_{i=0}^{K}$$

$$L_{KL} = KL(M_l, \sigma(S(Z;w)))$$

$$w = w - \eta\frac{\partial L_{KL}}{\partial w}$$

The heterogeneous approach can be deployed in the setting where each client holds different models and there is no way we can combine them as in case of FedAvg and FedProx. Our method would be efficient as it

only holds one loss function at the client side and also it utilizes noise at the server side by using only loss function between client and server model reducing the overall time at the server side as well.

Experiments

We perform rigourous experiments to show the significance of federated learning using noise-based knowledge distillation.We implement naive FedAvg and naive Fedprox to create a baseline for comparison. We implement and compare the modified FedFTG. We also implement the knowledge network and heterogeneousFL and compare all the three aproaches. We use Cifar10 dataset for all the experiments.

Results & Analysis

We evaluate the model performance using test set of cifar10 dataset.

Approach	Accuracy (%)
FedAvg original	69.70
FedProx original	69.60
Modified FedFTG + FedAvg + noise	64.80
Modified FedFTG + FedProx + noise	63.70
Light weight network + noise	73.40
HeterogeneousFL + noise	72.00
HetereogeneousFL + ensemble + noise	66.40

Analysis

The significance of using noise based knowledge distillation in federated settings is clearly visible from the above Table. We observe that Hetereogeneous federated fl technique and LWN network when used with noise performs significantly better than Modified FedFTG approach. The major observation here is the knowledge is distilled primarily using random noise. The above results shows the significance and importance of using noise in federated setting where the data is not available and knowledge combination from different architectures is of utmost importance.

Conclusion & Future works

We extend the idea of noise based knowledge distillation in federated setting to create an efficient pipeline that also preserves accuracy. We have tried the method with various approaches such as FedFTG, Light weight network and so on, we finally propose the method which is efficient and does not compromise much with the accuracy. The future work includes experimentation with various types of noise such as dead leaves noise etc. To test it for various non-iid distribution as well. The method can be very useful in the setting where different architectures cannot be aggreated in any manner thus providing federated learning benefits to any setting.

References

[1] https://arxiv.org/pdf/1602.05629.pdf

[2] https://arxiv.org/pdf/1812.06127.pdf

[3] https://openreview.net/pdf?id=K8JngctQ2Tu

[4] https://arxiv.org/pdf/2203.09249.pdf