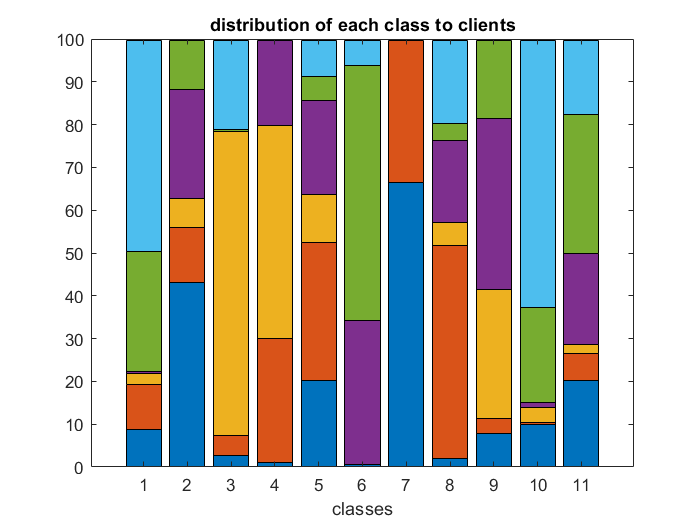
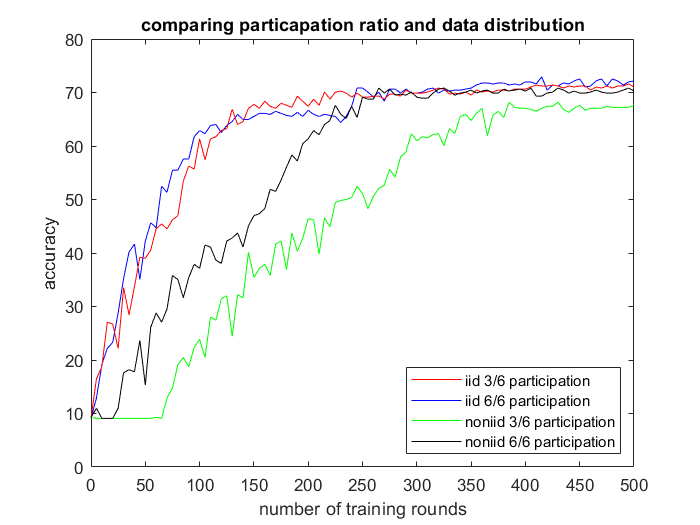
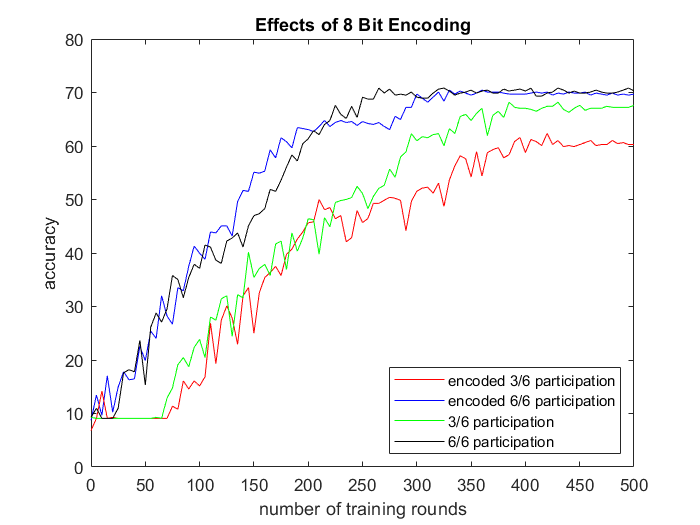
In our tests we have used the raw image data of “Stanford 2D-3D-Semantics Dataset” [Armeni et al.,” Joint 2D-3D-Semantic Data for Indoor Scene Understanding”, Jan 2017]. We have divided our dataset to 6 clients as it was suggested in [yine dataset paper]. How did we handle the actual data distribution? For IID case we have randomly assigned data samples but for non-IID case we have used a dirichlet parameter-based data distribution[Hsu et al.,” Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification”, Sep 2019]. This data distribution can be seen at [alt sol]. The x axis shows the classes/scenes of which we have eleven, and the y axis shows what percentage of this class belongs to a given client. For example, take class 7, it is completely divided between 2 clients and other 4 clients have no samples of it. As we can see our non-IID distribution is highly heterogenous. By using such a heterogeneous data distribution, we want to know if a client can learn a class that it has no samples of. If we turn back to class 7 example, 4 clients have no samples from that class so if we manage to classify this class reliably then we can conclude those 4 clients have learnt to classify this class despite not seeing it at all. This is why we are investigating a federated learning approach to intelligent agent questions.



It must be noted even the raw image part of the original dataset is very large (64 GB) since it uses 1080x1080 images. Using the original dataset as is would be unfeasible so to create our dataset, we have used 64x64 images. As our model we used the model proposed in the original federated learning paper [McMahan et al., “Communication-Efficient Learning of Deep Networks from Decentralized Data”, Feb 2017]. The basis of this model is convolutional layer followed by max pooling layer. We have two such combination. Max pooling is done over a 2by2 windows which means the height and width of channels are halved each time. The first convolutional layer takes 3 channels (RGB) and returns 32 channels whereas the second convolutional layer takes 32 channels and return 64 channels. These layers are followed by a dense layer of size 512 which leads to 11 outputs. As we can see our model is quite simplistic by today’s standards. When it comes to training, we have run our tests for 500 rounds to see them to converge. We are not only interested in the path accuracy plots follow but also the eventual accuracy they settle to.



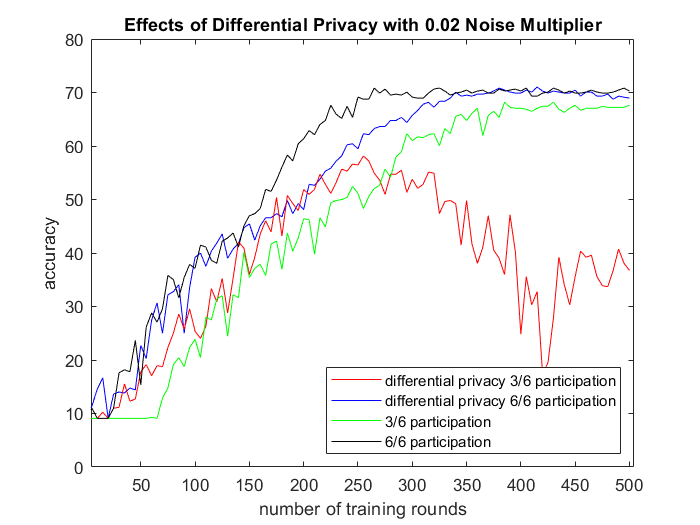
We first wanted to investigate the effects of client selection and non-IID data distribution. There are multiple techniques [Cho et al., “Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies”, Oct 2020 ; Nishi et al., “Learning Advanced Client Selection Strategy for Federated Learning”, July 2019]developed for biased client selection for heterogenous data distributions in which the odds of choosing a specific client for training is either increased or decreased, but these techniques require us to know the data distribution at clients apriori or show preference to some clients on the fly. For that reason, the most common client selection technique is unbiased or random client selection. This unbiased client selection can be used to reduce the total communication cost -note that per client communication cost stays same- or it can be used as a very simplistic model of effects of client availability. To achieve these goals, we have run 4 tests. These tests are combination of IID vs non-IID data and 6/6 client participation vs 3/6 client participation. Now looking at our results we can clearly see when we are using IID data, halving the participation ratio doesn’t negatively affect our accuracy. Both accuracy plots follow the same path and converge at same training rounds. Comparing the IID 6 client participation and non-IID 6 client participation we can clearly see the adverse effects of non-IID data. Despite converging to same accuracy, for a long time there was a significant accuracy drop between the two. Additionally, it took roughly 50 training rounds for non-IID one to reach the same accuracy as IID one. When we use IID data distribution for federated learning for the most part all the clients converge in the same direction in every training round, so the cancellation when averaging is not common. But when we train with non-IID data distribution since all the clients have different local data distributions the cancellations are much more common. These discrepancies lead to server model getting trained more slowly and requiring more training rounds than it would be necessary otherwise. When looking at the IID data distribution we have said using 3 client participation instead of 6 doesn’t negatively affect the training, but it is obvious this observation doesn’t extend to non-IID data distribution. 3-client participation non-IID case not only significantly lagged the 6-client participation, but also it failed to converge to the same accuracy all the others did. This means no matter how long we train, if we are using 3-client participation, we will never reach to 6-client participation accuracy scores. Why did our performance suffer so badly when using 3-client participation was fine with IID data? We believe since our non-IID dataset is highly heterogeneous there are training round cases in which some scenes are not present at all or represented at low numbers that hinder the model training. Take scene 7 for example, all the samples of that scene are only present at 2 clients alone and if those clients are not chosen for training then our model not only doesn’t train on scene 7 but also forgets its past training. Given the fact that almost all of the scenes have at least half of their samples at two clients we can see how prevalent this problem is. We argue this is the main reason why for non-IID case less than full client participation has such detrimental effect.



As we have mentioned in the client selection part it is desirable to reduce the communication costs of federated learning. The main way to do that is encoding [Haddadpour et al. “Federated Learning with Compression: Unified Analysis and Sharp Guarantees” Nov 2020, Konecny et al., “FEDERATED LEARNING: STRATEGIES FOR IMPROVING COMMUNICATION EFFICIENCY”, Oct 2017]. In the standard versions of federated learning it is customery to use 32 bit floating values, but do we really need all these 32 bits? Here we must note we are talking about communication only, so the image itself and the computations done at the clients and server are still using 32 bits. Also note we are not encoding all the gradient values. If a tensor is shorter than a threshold, in our case 10000, then we choose not to encode it since the gains will be small and encoding the final dense layers have detrimental effects on accuracy far overshadowing the gains[Konechy paper]. We wanted to test the effects of encoding and its relation to client participation ratio. Please note all the tests have been conducted using the non-IID data distribution given in [tablo]. When we look at the 6-client participation case we see encoding didn’t effect the accuracy plot much. The encoded case has managed to converge to the same accuracy original one did. But when we look at the 3-client participation case we see the encoded one failed to converge to the nonencoded 3-client’s accuracy- there is a near 7% accuracy drop. Why did 3-client case suffered when 6-client case seems to be doing fine? We believe this can be explianed by treating the encoding as a bounded noise. First let’s discuss why this is the case. When we encode the 32 bits to 8 bits we are basically losing the information that 24 bits carried. The absolute value of this information has the same possiblty to be anywhere from zero to the maximum value we can represent with this 24 bits. Since we might have rounded the leading 8 bits up or down the lost information might be positive or negative. Now obviosuly this lost information is not gaussian distributed but for the sake of argument let’s treat it as AWGN and average it like we are doing in server.

AWGN işlemleri

As we can see averaging signals with AWGN noise reduces the noise power by 1/N. So doubling our client number from 3 to 6 halves the power of AWGN. Of course we aren’t dealing with AWGN here and there is a randomness involved due to very nature of machine learning but we can see why increasing number of clients makes our system more robust to adverse effects of encoding.



The big selling point of federated learning is data privacy. By training each client locally and never transmitting their data to server, federated learning offers a great level of privacy. But we are transmitting the server model to clients and then client gradients after local training to server and if somehow an attacker gains access to both server model and a client’s gradients then the attacker may reverse enginner data distribution of said client. How does that work? When attacker adds the gradients of client to the server model he/she gets the final version of client model at the end of local training. If attacker is aware of which classes are there in this domain then attacker can compare the results of both server model’s and the client model’s accuracies. If client model performed worse at some classes then those classes are likely not present in that client’s local dataset. Or if the client model performed significantly better at some classes then those classes are likely very prevalent in the client’s local dataset. This is a very simplistic example and the privacy leakeage usually happens in multiple communication rounds [advances and open problems] but we can see how this privacy leakeage can lead to problems. One common technique to solve this problem is using encyription. Most common ones are homomorphic encyription [Madi et al., “A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing” May 2021] and secure multiparty computation [Truex et al., “A Hybrid Approach to Privacy-Preserving Federated Learning”, Aug 2019]. This methods make use of encyription methods we can do computations on encyripted values.

F(d(x))

Another method, a more simplistic one, is differential privacy[Abadi et al., "Deep Learning with Differential Privacy", Jul 2016][ McMahan et al., "Learning Differentially Private Recurrent Language Models", Oct 2017]. In differential privacy we basically introduce noise some multiplicant of the clipping of a tensor. We choose this method since this is by far the most prevelant one and it is fairly easy to use. In our tests we have used a noise multiplier of 0.02. For 6-client case, the accuracy plot closely follows the that of original and finally converges at the same ratio. When it comes to 3-client case we see that it diverges around 250 rounds and never recovers. At the end we see near 30% accuracy drop and failure to convergence. Why did 6-client and 3-client cases differ so significantly? We once again use the AWGN modelling and argue the increase in the client number made the model more resilient to noise. But then why did encoding and differencial privacy plots differ so much? We have 2 reasons for that. First, the noise levels are not same. Second, in encoding we do not alter the final fully connected layer whereas in differential privacy we alter those.

We present a federated learning approach to scene recognition task where the dataset is distributed to the agents/clients in a highly heterogenous manner. Our goal is to ensure agents can learn recognizing all the scenes even if they have no data samples from some of the scenes. While doing that we also adhere to a strict privacy constraint of not sharing data samples between agents. We also investigate the fast adaptation to novel places and scenes when a backbone is trained with federated learning.