MOONFlower

Vasundhara Agarwal
University of Cambridge
va308@cam.ac.uk

Mina Remeli
University of Cambridge
mr892@cam.ac.uk

Abstract

Federated learning is an increasingly popular method of collaborative training. It is privacy preserving, and it is a solution that enables cross-device learning, with potentially thousands (or millions) of learners! However the more learners we have, the more likely is the issue of non-homogenous data. Finding solutions to this has become one of the central questions of federated learning, and MOON (model contrastive federated learning) (Li et al. [2021]) claims to answer them. In our report, we aim to reproduce and evaluate the central claims of the aforementioned paper, as well as extend it with some of our own insights. We find that the original results (that present MOON as a better alternative to FedAvg) could not be reproduced. Nevertheless, we find a new use case in which MOON could prove to be extremely useful: it could help reduce the divergence between global and local models caused by sparse client sampling.

1 Introduction and Motivation

Federated Learning (Konečnỳ et al. [2016]) allows a large number of participants (clients) to work collaboratively under the coordination of a centralized entity (server) to learn a machine learning model without having to exchange their data with the server or with each other. Since the training data never leaves the client, FL has received widespread adoption in settings where privacy sensitive data is involved, e.g, for training voice assistants (e.g., Apple's Siri) and next-word prediction models for mobile keyboards (e.g., Google's Android keyboard).

Continuing with the examples from above, the training data at each client (here, a mobile device) is characteristic of its user by virtue of being generated locally e.g., texting style in the case of next-word prediction. As a result, a key characteristic of FL settings is that different devices usually possess non-identically distributed datasets such that a single client's data cannot be considered to be representative of the overall population's.

This heterogeneity in the data distribution is not properly accounted for by the training objective used at clients under traditional FL strategies such as `FedAvg` (McMahan et al. [2017]). It has been shown that this can lead to instability (and in extreme cases, non-convergence) of the training process as a result of the local models at the clients diverging from each other towards their differing local optima.

MOON attempts to remedy this by augmenting the client's loss function with a component that does the following. Assuming that round n is currently in progress,

- It *encourages* agreement between the latent representations of a datapoint under the global model the client receives at the beginning of round n and the local model it learns during that round. This tries to limit divergence between local models at different clients.
- It *discourages* agreement between the latent representations of a datapoint under the local model being learnt in round n and the one that was learnt in round $n-1$. This is reminiscent of the traditional objective of training an ML model , i.e., we do want the model to learn something new in every round.

This additional term was inspired by contrastive learning (covered more formally in section 2.2), a paradigm that excels at learning visual representations by nudging embeddings of similar images towards each other, while pushing those of dissimilar ones apart – MOON essentially performs contrastive learning at the level of models. This technique is based around the intuition that a model trained on the whole dataset would learn better feature representations than one trained on a skewed subset of it. In the case of FL, since the global model is an aggregation of multiple local models trained on differently skewed subsets, we would expect the representatios learnt by it to be better than that learnt by a local model on its individual dataset.

Our contributions We reimplement MOON using Flower[1], a federated learning framework, and attempt to reproduce the central results of the original paper. Unfortunately, we find no improvement over FedAvg in case of heterogenous local datasets. We extend the reproduction with our own experiments, and find that MOON could prove useful in improving accuracy in sparse client sampling instead.

2 Theoretical Background

2.1 FedAvg

Federated optimization has some defining characteristics which make it more challenging than traditional distributed optimization. Besides the hetereogeneity of client data distributions, a couple of other properties specific to this scenario are as follows.

High Communication Costs The clients are generally resource-constrained, battery-operated devices connected to the server via the Internet. The regular exchange of updates between rounds incurs significant communication costs in terms of battery utilization and bandwidth usage. Therefore, it might be more efficient to train locally for multiple epochs at each client prior to sending an update to the server. However, this can be problematic in the case of statistical data heterogeneity because the larger the number of local epochs, the larger the divergence between the local models at different clients.

Limited Client Availability For reasons mentioned above, a client might only want to take part in a training round when they are plugged into a charging socket and are on a cheap Internet connection (e.g., public WiFi). As a result, only a fraction of all existing clients might be available to participate at each round. Since the global model is the result of the aggregation of updates from the clients, this translates roughly to only training on a fraction of the cumulative training data (across all the clients) at each round, which can again cause problems in the case of heterogeneity.

`FedAvg` is a popular FL strategy for tackling the communication costs and low availability problem and this is the baseline we are comparing against.

2.2 Contrastive Learning

Contrastive learning is a technique commonly used in unsupervised learning tasks (recently it has gained great popularity in the self-supervised learning community). It produces embeddings in which similar inputs will be grouped close to one another, while dissimilar inputs will be far apart. It achieves this by applying a *contrastive loss*, in which the representations of *positive pairs* (similar inputs) will be brought closer together, while *negative pairs* (dissimilar inputs) will be forced apart. Where labels are available, we can construct positive pairs by selecting two instances from the same class (or negative pairs by sampling from two different classes). Alternatively, when no class labels are available, we define positive and negative pairs on the level of samples. A common technique used in the image domain is to select an *anchor* (see Figure 1), and then apply various data augmentation techniques (rotation, flip, Gaussian blur, etc.) to obtain a positive pair. A negative pair can be any other image (or its augmented version). We call this learning objective triplet loss [Schroff et al., 2015]:

$$\mathcal{L}_{triplet}(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) = \sum_{\mathbf{x} \in \mathcal{X}} \max\left(0, \|f(\mathbf{x}) - f(\mathbf{x}^+)\|_2^2 - \|f(\mathbf{x}) - f(\mathbf{x}^-)\|_2^2 + \epsilon\right)$$

[1] `https://flower.dev/`

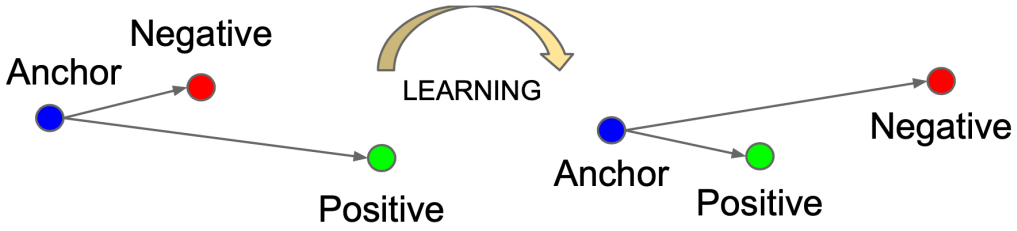


Figure 1: A common contrastive learning objective is to bring representations of similar images closer together (*anchor* and *positive*), while pulling representations of dissimilar images apart (*anchor* and *negative*). Here, anchor and negative are two randomly selected samples, and positive is an augmented version of anchor. Common augmentation techniques in case of images are cropping, flipping, adding Gaussian blur, grayscale conversion, etc. Image from Schroff et al. [2015].

A more generalized version of the triplet loss is the N-pair loss [Sohn, 2016], which allows multiple negative samples:

$$\mathcal{L}_{N-pair}(\mathbf{x}, \mathbf{x}^+, \{\mathbf{x}_i^-\}_{i=1}^{N-1}) = -\log \frac{\exp(f(\mathbf{x})^\top f(\mathbf{x}^+))}{\exp(f(\mathbf{x})^\top f(\mathbf{x}^+)) + \sum_{i=1}^{N-1} \exp(f(\mathbf{x})^\top f(\mathbf{x}_i^-))}$$

A similar loss function was introduced in SimCLR [Chen et al., 2020], a self-supervised contrastive learning technique, called NT-Xent:

$$\mathcal{L}_{NT-Xent}(\mathbf{x}_i, \mathbf{x}_j, \{\mathbf{x}_k\}_{k=1}^{2N} \mathbb{I}_{[k \neq i]}) = -\log \frac{\exp(\text{sim}(x_i, x_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(x_i, x_k)/\tau)}$$

where x_i and x_j are the positive pairs, we have $2N$ augmented negative views and $\text{sim}(.,.)$ is the cosine similarity. The inputs to this loss function are first passed through a *base encoder* $f(.)$ (which learns to extract feature representations), and a *projection head* $g(.)$ (which maps to latent space). Notice how the only difference between this loss function and the previous is how the similarity between samples is calculated (dot-product vs. cosine similarity), an added temperature parameter τ, and the number of negative samples.

For MOON, positive and negative pairs will be latent representations of the same sample under different models. The clients' model architectures will be divided into three parts: a base encoder, a projection head, and an output layer. First, we apply the encoder, then the projection head to input x. This input representation of x will be the building block of our contrastive loss: the anchor will be the representation of the global model, the positive and negative samples will be the representation of the current and previous local models respectively. Consequently the contrastive loss pushes consecutive versions of the local models apart, while bringing the representation of the global and the current local model closer together (see Fig. 2).

2.3 Flower

Federated Learning is difficult to implement from scratch, especially in a scalable and reliable manner. Therefore, for our experiments, we have extended pre-existing implementations of the core components of an FL setup made available by Flower.

One of our primary motivations for using Flower was to take advantage of its Virtual Client Engine, which allowed us to run experiments with a very large number of clients on a single GPU in a resource-aware manner. It does away with the need for us to manually partition the limited resources among the clients.

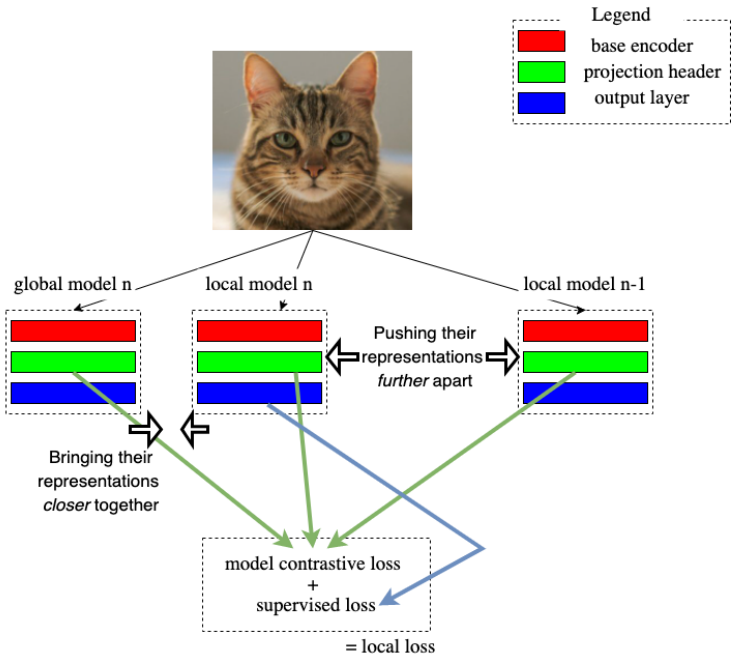


Figure 2: A general overview of architecture and modified loss

3 Implementation

The only major difference between the implementations of vanilla FedAvg and MOON is in terms of their *local* training objectives – the *global* aggregation strategy stays the same.

3.1 Network Architecture

We have only slightly modified the model architecture presented in the official PyTorch tutorial[2] on classification with CIFAR-10 to introduce a projection head right before the final layer. The projection head is a 2-layer MLP with a 256-dimensional output.

Given model weights w, we will use $F_w(.)$ to represent the whole network and $R_w(.)$ to represent the portion of the network upto and including the projection head.

3.2 Modified Training Objective

The modified loss function is a weighted summation of two terms. These are

- the typical **supervised loss** $\mathcal{L}_{sup}$ component used as a proxy for the classification accuracy (implemented here as the cross-entropy loss). This is computed using the probabilities we get from the output layer of the model.
- the **contrastive loss** $\mathcal{L}_{con}$ component (implemented similarly to NT-Xent) computed using the latent representations produced by the projection head.

For any client i, let w^t be the global model it receives at the beginning of round t and w_i^t be the local model it trains in round t. Then, $z^t(x) = R_{w^t}(x)$, $z_i^t(x) = R_{w_i^t}(x)$ and $z_i^{t-1}(x) = R_{w_i^{t-1}}(x)$ are the representations of an input x under the global model, current local model and previous local model respectively[3].

Then, $\mathcal{L}_{con}$ is defined as

[2] `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`

[3] If the fraction of clients sampled in a round is smaller than 1, each client might not participate in each round. In that case, its previous local model will be the local model it had trained when it was last chosen.

$$\mathcal{L}_{con}(x) = -\log \frac{\exp(\text{sim}(z_i^t(x), z^t(x))/\tau)}{\exp(\text{sim}(z_i^t(x), z^t(x))/\tau) + \exp(\text{sim}(z_i^t(x), z_i^{t-1}(x))/\tau)}$$

The loss for an input-output pair (x, y) is then computed as

$$\mathcal{L} = \mathcal{L}_{sup}((x, y); w_i^t) + \mu\mathcal{L}_{con}(x; w^t, w_i^t, w_i^{t-1})$$

Here, μ is a tuneable hyperparameter that controls the relative contributions of the two loss components to the overall loss.

Let us go over the steps in each local update to clarify how the global and local models change in each round. First, client i receives the global model w^t, and sets their current local model to be the same: $w_i^t \leftarrow w^t$ (let us assume that they have w_i^{t-1} stored in memory). Then, they start going over their data in batches, and for each batch they calculate the supervised loss $\mathcal{L}_{sup}$ and the contrastive loss $\mathcal{L}_{con}$. Client i updates w_i^t with the gradient calculated on the overall loss $\mathcal{L}$: $w_i^t \leftarrow w_i^t - \mu\nabla\mathcal{L}$. In the first batch, $\mathcal{L}_{con}$ will be 0 (due to the global and local model being the same), however, as local training progresses, w_i^t will change, and the contrastive loss term will be nonzero.

3.3 Maintaining local model state across rounds

We have used functionality defined in `flwr.simulation` to delegate the. management of resources amongst the virtualized clients to Ray. Since this module was designed to support extremely large-scale experiments on a limited number of CPU and GPU instances, it does not guarantee that it will keep all existing clients in memory at a time. Instead, this module requires us to implement a `client_fn()` method that creates clients on demand whenever it needs a particular client to call `fit()` on. This method takes a parameter `cid`, which uniquely identifies the client we want an instance of (this is necessary to allow different clients to have different datasets).

As stated in the official Flower documentation, the client instances created are ephemeral and are often destroyed after a single use. Therefore, we cannot expect these instances to carry over state from one training round to the next. Hence, each client saves its model state at the end of each round to a file named `net.pt` under a folder corresponding to its `cid`. It loads in this updated model state at the beginning of each round in order to generate the latent representations required for computing the contrastive loss.

3.4 Heterogeneous data partitioning

We partition the data in each class among N clients in proportions determined by a sample from an N-dimensional symmetric Dirichlet distribution, whose concentration parameter β we can control. Since we have 10 classes, we need to take 10 independent samples from this distribution.

β essentially controls the degree of statistical heterogeneity of the data distribution among the clients. Put simply, the larger the value of β, the more identically distributed the local datasets at the clients will be.

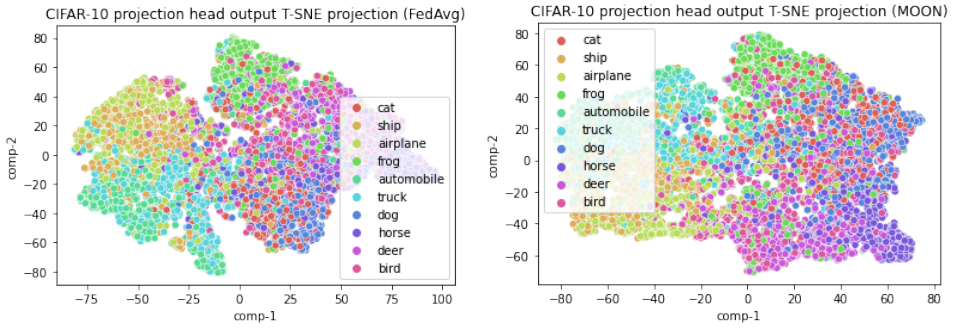
4 Experiments

In this section we will document the experiments that we have conducted. In Section 4.1, we compare MOON to FedAvg. Then, motivated by the plots in the original paper, we compare and contrast the visualization of input representations of models trained with MOON versus FedAvg in Section 4.2. We then go onto exploring performance on different levels of data heterogeneity in Section 4.3. We conclude with Section 4.4, where we examine the effect of changing the sampling fraction. Experiments in Section 4.1 and 4.3 are reproductions, while Section 4.2 and 4.4 are extensions of the original paper.

All experiments are performed on the CIFAR-10 dataset, and on the network architecture specified in Section 3.1. Unless otherwise specified, the default values for the experiments are the following: 10 clients, 100 training rounds, 10 local epochs per round, 1.0 sampling sampling fraction (all clients are selected in each round), 0.01 learning rate, β is 0.5 and μ is 5. Reported results are the average of

	fedAvg	moon
mean	**0.5312**	0.4751
std	0.0289	0.0129

Table 1: Comparison of MOON to FedAvg, listed values are average and standard deviation of the test accuracy.



(a) Global model, FedAvg

(b) Global model, MOON

Figure 3: T-SNE representations of CIFAR-10 inputs.

three different runs. Top-1 accuracy is calculated on the centralised test set (10k samples). Our code for running the experiments is publicly available on Github[4].

4.1 Comparison to FedAvg

In this section we reproduce the first set of experiments from the original paper, where they compare MOON to various other approaches. We only attempt to reproduce the results comparing MOON to FedAvg. Unfortunately, we did not achieve desirable results with the hyperparameters proposed in the paper (neither of the models learned at all, accuracy stayed at 10%, which is equivalent to a random guess). Therefore we performed a hyperparameter search for the learning rate, which we found improved the loss value (cross-entropy) /accuracy the most. We evaluated the following learning rate values on FedAvg: [0.01, 0.005, 0.001, 0.0005, 0.0001], and found that a 0.001 learning rate worked the best.

With the new learning rate, we ran the experiment and found that MOON did not outperform FedAvg. Our results can be found in Table 1. The original paper also achieved greater accuracies in both cases. Their average accuracy for FedAvg was 66.3%, while for us it was 53.1%. We are unsure whether this arises from a difference in optimal hyperparameter selection, or an implementation error. One feasible hypothesis would be that the contrastive loss "holds back" the local model from diverging from the global model, thus slowing down learning. We also checked whether it would pay off to train a bit longer, and plotted the average cross-entropy loss curves (see Fig. 2). We found that the loss for both FedAvg and MOON begin to flatten after 100 rounds, therefore we saw no justification to train the models for longer periods. We also attempted to tune μ - however we discovered that increasing μ only decreased the resulting model's accuracy (Fig. 5). This was unfortunately in line with our reported results in Table 1, meaning the greater the "deviation" from FedAvg with our contrastive loss component, the greater the damage to our accuracy. We experimented with the following values for μ (besides the original 5.0 value): [0.1, 1.0, 3.0, 7.0]. Our last shot at producing results resembling those in the paper was to increase the number of local epochs to 20 (and decrease the number of training rounds to 50, to result in 1000 gradient updates just like before). However, that did not yield better results either.

	mean		std	
strategy **beta**	fedAvg	moon	fedAvg	moon
0.1	**0.4135**	0.3868	0.0081	0.0239
0.5	**0.5312**	0.4751	0.0289	0.0129
5.0	**0.5672**	0.5323	0.0038	0.0078

Table 2: Heterogeneity experiment results. Listed values are average and standard deviation of the test accuracy.

	mean		std	
strategy **frac**	fedAvg	moon	fedAvg	moon
0.05	0.5027	**0.5112**	0.0338	0.0373
0.1	**0.5599**	0.5398	0.0163	0.0248
0.2	**0.5650**	0.5633	0.0167	0.0114

Table 3: Sparse sampling experiment results. Listed values are average and standard deviation of the test accuracy.

4.2 Input representations

Inspired by Figure 2 from the original report, we plot the T-SNE representations of the global projection head outputs of trained models. More specifically, the input to the T-SNE algorithm is $\mathcal{R}_w(x)$, where x is the input image. We plot the two main components, and color code the samples belonging to the same class. We find no significant difference between projections of the two models (Figure 3). It is also interesting that contrary to the author's results, the projection of the classes of MOON does not seem any more separable than FedAvg's. However, we can nicely see that classes that represent vehicles and animals form two separate big clusters.

4.3 Heterogeneity

We reproduce the heterogeneity experiments from the paper, by varying the value of β. With smaller values of β, we generate more diverse local datasets (that are highly heterogenous) while for larger values of β we generate more homogenous datasets. The default parameter for β was 0.5, and we conduct further experiments with 0.1 and 5.0 (see results in Table 2). As we can see, the FedAvg test accuracy outperforms MOON in all three cases, contrary to the results that the authors presented (where MOON outperformed all other approaches).

4.4 Sparse sampling

So far all experiments were conducted on 10 clients, where all of them were sampled in each round. Now, we turn to experiments with more clients - more specifically to cases where only a small subset of clients are selected in each round to participate. This is a fairly common scenario, since clients might only be available if they are plugged in or are using WiFi. Since the contrastive component of the loss incorporates the current and the previous local model, we wanted to see the effect sparse sampling would have on the final accuracy. Since models are rarely selected (a few rounds could go by before being selected), the received global model at the beginning of a round could be very much different from the previous local model. In this case, it would make a lot of sense to incorporate a loss that brings the current local model and the global closer together, while pushing the two consecutive local models apart. Our results for training on 100 clients for 250 rounds, and sampling fractions of [0.05, 0.1, 0.2] can be found in Table 3. We selected the learning rate to be 0.01 again. As suspected, for sparse sampling fraction (i.e. 0.05), the MOON strategy yields slightly better results.

[4] `https://github.com/minaremeli/MOON`

5 Discussion

To conclude, unfortunately we were unable to reproduce most of the results that the authors presented. We tried a number of ideas to try to improve our results. In most cases (and for most hyperparameters) MOON proved to be the inferior algorithm. This could be either because of (i) inconsistency in the hyperparameter settings mentioned in the original paper vs what the authors actually implemented (ii) some implementation error on our end or (iii) the contrastive loss "holding back" the local model, by not letting it diverge from the global model.

However, in the last section (Section 4.4) we were able to present some promising results on experiments with sparse client sampling. Since clients are selected less frequently, their previous local models can differ greatly from the global model when they finally get selected. We have shown that it could prove beneficial to use MOON to make sure that the new local models are closer to the global model than the previous local model. For future work we intend to explore preventing model divergence with model contrastive learning in more depth (another central question of federated learning). For example, we could try to alleviate model divergence caused by a large number of local epochs.

References

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. 2020. URL http://arxiv.org/abs/2002.05709.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

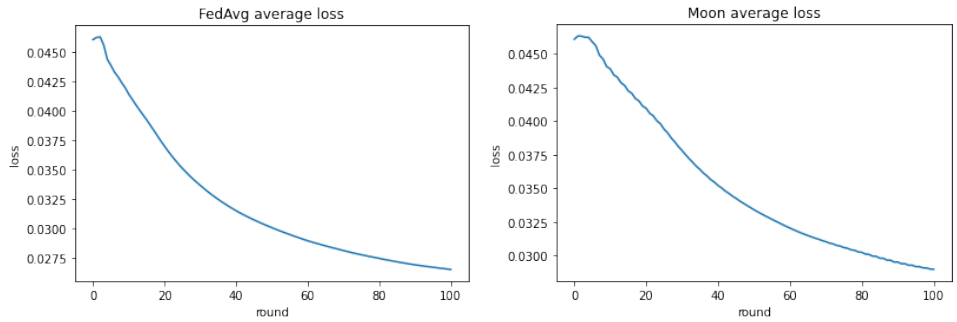
Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10713–10722, 2021.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. pages 815–823, 2015. doi: 10.1109/CVPR.2015.7298682. URL http://arxiv.org/abs/1503.03832.

Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://papers.nips.cc/paper/2016/hash/6b180037abbebea991d8b1232f8a8ca9-Abstract.html.

A Additional figures



(a) FedAvg (b) MOON

Figure 4: Average loss curve of the 'Comparison to FedAvg' experiments. We calculated the cross-entropy loss in both cases. We wanted to see whether it would pay off to train MOON for longer periods. Since both of the curves begin to flatten after 100 rounds, we saw no reason to train longer than that.

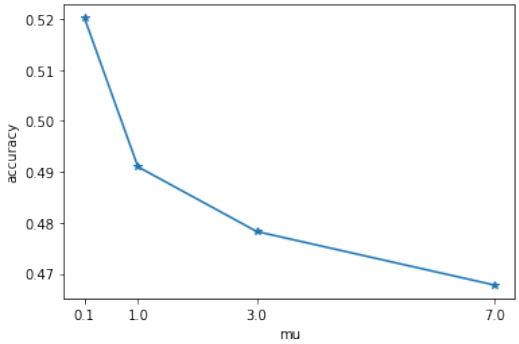


Figure 5: Hyperparameter search for mu. We wanted to see whether 0.5 mu caused the poor results, however, there is a clear negative effect on training accuracy as we increase mu.