The Impact of Federated Learning on Distributed Remote Sensing Archives

The repository consists of the code for Federated Learning Experiments for Remote Sensing image data using convolution neural networks. It contains the implementation of three Federated Learning models:

- FedAVG
- FedProx
- BSP

The implementation is specifically made for the multi-label *UCMerced Landuse* dataset. To apply other datasets it requires some modification.

Table of Contents

Setup Details

Option 1 : using Anaconda Distribution

Python version 3.8.5
It is recommended to use conda environments when using deep learning frameworks especially GPU supported libraries as there can be clash of versions if not. A `environment.yml` file is provided in the main directory which can be used to install the environment directly using conda. On linux follow these steps:

```
wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64.sh
chmod +x Anaconda3-2020.11-Linux-x86_64.sh
bash Anaconda3-2020.11-Linux-x86_64.sh
```

This will install Anaconda on the machine. Next create a virtual environment using the `environment.yml` file provided by

```
conda env create -f environment.yml
```

This creates the virtual environment `cv4rs` on the machine. Depending on the shell used the environment can be activated using

```
conda activate cv4rs
```

This environment should be able to run all the scripts and notebooks provided with the project.

Option 2 : Use pip

The packages used are provided in `requirements.txt` file which can be used to setup all the dependencies on a machine from scratch. This can be done using pip by

```
pip install -r requirements.txt
```

Description of the files & Usage

- `main.py` : The file to run to start training based on the required experiment setup. Required Arguments are data path and the CNN model to use. For more details on how to use the arguments use `python main.py -h`. E.g. `python main.py -c resnet34 -d ./UCMerced_LandUse/Images` trains FedAVG using resnet34 with the images in the provided path. The following parameters can be chosen for training:
  - CNN Model
  - Number of clients
  - Federated Learning Model / Centralized
  - Percentage of label skewness
  - Number of Epochs
  - Number of local Epochs (FedAVG and FedProx)
  - Learning Rate
  - Small Skew
  - C Fraction (For FedAvg and FedProx)
  - Batch Size
  - Validation Split
  - Data directory and multilabel excelfile path
    The details are given below
- `visualize.py` : This file is used to plot the results from training. When training the FL models using `main.py` a `csv` is generated containing *loss*, *accuracy* and *F1-Score*. For more details on how to use the arguments use `python visualize.py -h`.
- `legacy_notebooks` : Code before combining all notebooks to a single project. Initial work was done almost individually.

- `multilabels` : Contains the multilabel excel files for the UCMerced_LandUse dataset.
- `cnn_nets.py` : Contains the CNN architectures that can be used: `ResNet34` , `LeNet` and `AlexNet`
- `custom_dataloader.py` : Includes the functions to split data across multiple clients in both IID and non-IID distributions. Furthermore it checks which classes are least correlated.
- `custom_loss_fns.py` : Custom loss functions can be found here. One loss function is specifically for the FedProx Federated Learning algorithm. The other one is a wrapper to the Pytorch loss function. This was included to have a generic train function that supports custom loss functions.
- `CustomDataSet.py` : Inherits the abstract class `torch.utils.data.Dataset` and overrides `__len__` and `__getitem__` method. This custom dataset class supports multilabel for each image.
- `federated_train_algorithms.py` : Includes the implemented Federated Learning models. Supported FL algorithms:
    - FedAvg (Federated Averaging) (Paper)
    - FedProx (Paper)
    - Bulk Synchronous Processing (BSP) (Paper)
- `FL_with_pytorch_only.ipynb` : The final notebook version from which the modular code was written from.
- `train.py` : The file has the training loop. This training loop is used by all federated algorithms.
- `requirements.txt` : Can be used directly with pip/conda to setup the required packages.

Data Preparation and Splitting

// tbd

Implementation of the Federated Learning Models

All clients, given to the the FL algorithms as parameter, are `torch.utils.data.Dataloader` containing the respective data partition. The communication between *server* and *client* is simulated for all three algorithms.

FedAVG (Federated Averaging)

1. For each round, `FedAVG` first chooses a random fraction of clients `client_subset` to be trained. The number of chosen clients is bounded by the given parameter `c_fraction` .

2. In the next step, we iterate through `client_subset` to train each chosen client. For each client, the current model (still untrained at first round) is sent out for training using the respective data partition. To avoid issues we need to initialize a new optimizer and scheduler. Moreover, we have to create a new model and (deep-) copy the parameteras from the original model to it. The training is carried out through the `train` function from the `train.py` file which returns the trained model. Each trained model is stored in the array `model_client_list` which is later used in step **3** for averaging.
3. Once we trained a model for each client from `client_subset` we proceed with the averaging: for each client we iterate through all layers and average the weights. Each client (not only from `client_subset`) ist weighted depending on the size of its data partition. E.g. let *n=1000* be the size of the whole dataset and *m=100* is the size of a client's data partition. Its model will be weighted *m/n=1/10*. Finally, we receive an averaged model.
4. Start again fromo step **1** for the next round using the averaged model.

FedProx

The algorithm for `FedProx` is with exception of the used loss function exactly like for `FedAVG`. `FedProx` adds a proximal term to the standard local loss function that penalizes models that deviate too much from the global model. The loss function is defined in `custom_loss_fns.py` under the class `FedProxLoss`. This class takes in basic loss function `base_criterion` which can again be same as the one used for FedProx and a hyper parameter `mu` which impact how much the proximal term affects the overall loss. The proximal term is a penalty on the local client weights for deviating too much from the central weights.

BSP

The implementation of `BSP` is similar to `FedAVG` with a few exceptions. First, it skips step **1** of choosing a fraction of clients. Furthermore, `BSP` doesn't average the models of all clients but rather passes them from one client to the next. A round is finished once the model has been passed to all clients (and has been trained by them).