Functions, and Variables in Client.py, FedBase.py, and FedAvg.py

Client

Client.py is the node class, it has all the client node attributes. The inheritance of these three classes goes as such Client.py → FedBase.py → FedAvg.py, in descending order.

Client Variables

Model: neural network structure. (ie, layers, loss function, ...)
Id: a primary key to identify an individual node.
Group:
train_data: an array of the data the node will be training on (local data)
eval_data: an array of training data the node will be validating on
num_samples: total length of the training data.
test_samples: total length of the testing data.

Client Functions

set_params(self, model_parameters): this function sets the model the clients are working with, it is set using a parameter in the set_params function. Reference model variable for what the model is.

get_params(self): this function returns the model, model a structure within Client that is set during the constructor or set_params function.

get_grads(self, model_len): This function returns the gradients of the model. Using the local training data and the model's length. The gradient is a vector that consists of partial derivates of a function. The function we look at to calculate the gradient is the loss function.

solve_grads(self): bytes_w is the size of the model; the size of the model is the number of weights the model contains. grads = the gradients are going to be a list of derivatives, one for each variable. comp = flops * number of samples, flops are floating point operations per second, which should be relatively constant per sample. Therefore, by multiplying these two we can get number of flops per training cycle. bytes_r is equivalent to bytes_w. The function then returns all this information.

solve_inner(self, num_epochs=1, batch_size=10): & solve_iters
Solves local optimizations:
Num_samples: number of samples used in training.
Soln: local optimization solutions.
Bytes read: number of bytes received.
Comp: number of FLOPs executed in training process
Bytes write number of bytes transmitted.

This function used the inners and iters functions that are defined in the model class in tensorflow.

test(self):
this function uses the test function of the model, as defined within tensorflow, this test function Is ran against the local evaluation data and returns the total correct as well as the loss value of the model.

train_error_and_loss(self):
this returns the total correct in the training data, as well as the loss, and the number of local samples. This does not train the data, but it does use the training data.

FedBase.py

Class BaseFedarated (object)

Variables
Inner_opt: learning rate of the model is set, this is a globally used variable as Federated Base is the base to the machine learning network.

Params, these are all the gradients of the model.

Learner, apart of the model, still discovering specifics.

Dataset, this is the chosen data set that the user has decided to work on, the data is passed as a parameter to BaseFederated

Functions

__init__ : constructor, sets the parameters explicitly

Setup_clients(dataset, model):
this sets up all the clients with the correct dataset and model. The zip() function allows for parallel iteration (ie: giving all the clients all the iterable at the same time.)
This function returns all_clients

train_error_and_loss():

this function calls the client method of the same name, but in the BaseFederated class it will apply this too all the clients. It then adds all the client ids and their groups into arrays then returns them, along with the compilation of the client data.

show_grads():

the BaseFederated retrieves the gradients from the clients using get_grads, (see get_grads the client function above) they then find the length of the client sample data and use this to weight each client gradient – giving a new global gradient, this is then added onto the intermediate_grads numpy array which is then returned.

test():

runs on the clients through the client test function (see client test() function) and then has the sum of the systems total samples tested, and total correct, this function also returns the client IDs and their groups.

save():
This function is not implemented yet, as per the pass.

select_clients(round, num_clients = 20):

this function returns the list of all the self.clients along with a list that tells you what those specifically used client's indexes are. The parameter equaling twenty is saying that the greatest number of clients we will use is 20. This is arbitrary and should be changed to an actual variable such that we can manipulate the number of maximum clients.

aggregate(wsolns):

This function handles all the client gradients. Solns is passed from FedAvg as csolns, csolns comes from the client.py function solve_inners. Solve_inners passes the trainable_varaiables, a TensorFlow member. This function returns the gradients for each data sample tested.
W is the number of local samples,
I is the client,
V is the gradient value.
The Array wsolns has the dimension: [i][w][v]
Averaged solution is an array of weighted gradients that is the sum of (gradient/number of samples) for each parameter, therefore aggregate averages the gradients of every client for every parameter while weighing them for how much data they trained with.

FedAvg.py

Server Class

Server class is the parent of the Federated Base class. The Federated Base class is made such that we can either make a normal server class, or a FedProx class.

Variables
Params, learner, dataset, all found in the parent class. No new class variables.

Functions
train() starts the training process of the clients. Stats variable is equal to test() (see FedBase test function) this chooses the clients we use. Then stats_train is initialized with train_error_and_loss (see FedBase functions) this selects the correct training data. Next the function displays the current training statistics, then selects clients to pull data from for an averaged gradient. The end of train is where the communication occurs – the distribution of the gradients occurs as this is where we implemented a top-k algorithm.