Advanced settings Client

Adjusting autoencoder architecture/training

to use a different autoencoder, a different architecture can be chosen. Change the encoder model in client.py to your desired encoder model:

```
class Encode(nn.Module):
    def __init__(self):
        super(Encode, self).__init__()
        self.conva = nn.Conv2d(32, 16, 3, padding=1)
        self.convb = nn.Conv2d(16, 8, 3, padding=1)
        self.convc = nn.Conv2d(8, 4, 3, padding=1)

    def forward(self, x):
        x = self.conva(x)
        x = self.convb(x)
        x = self.convc(x)
        return x
```

To pretrain the models, use the pretrainer.py. you can use any dataset/hyperparameter to pretrain the autoencoder in the pretrainer.py.

Attention: client model, encoder model and decoder model in the pretrainer.py have to be the same as in the client.py and server.py. Also, the data-shape at the cut layer hast to correspond to the input/output shape of the autoencoder.

The pretrainer will create files named: convencoder.pth and convdecoder.pth. these files contain the weights for the encoder model/ decoder model for the actual training process. By copying these files into the client.py / srever.py directory, they will be load autonomously by these scripts in their main function before starting the actual training process:

client.py:

```
global encode
encode = Encode()
encode.load_state_dict(torch.load("./convencoder.pth"))
encode.eval()
encode.to(device)
```

Adjusting model architecture:

in order to use a different model architecture, simply adjust the model architecture at the client int the client.py:

```
class Client(nn.Module):
    def __init__(self):
        super(Client, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        self.norm1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        self.relu2 = nn.ReLU()
        self.norm2 = nn.BatchNorm2d(32)
        self.drop1 = nn.Dropout2d(0.2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.norm1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.norm2(x)
        x = self.drop1(x)
        return x
```

Attention: The data-shape at the cut layer hast to correspond to the input/output shape of the autoencoder.

Changing Optimizer/ Loss/ Training device

Optimizer, loss and the device (as well as the models for encoder, decoder, client and server) are globally defined in the main function of the client.py/ server.py and can be changed.

client.py:

```
global device
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

global client
client = Client()
client.to(device)

global encode
encode = Encode()
encode.load_state_dict(torch.load("./convencoderlate.pth"))
encode.eval()
encode.to(device)

global optimizer
optimizer = SGD(client.parameters(), lr=lr, momentum=0.9)

global error
error = nn.CrossEntropyLoss()
```

Measuring Message Size and FLOPs

In order to figure out the size of a message in bytes, the following function can be added to the client.py or server.py:

```
import struct
def get_size_send_msg(msg):
"""
can be called to figure out the meassage size of a message in byte
:param msg: message
:return: string that includes the meassagesize in bytes
"""

    msg = [0, msg]  # add getid
    msg = pickle.dumps(msg)
        # add 4-byte length in network byte order
    msg = struct.pack('>I', len(msg)) + msg
    return ("sendsize: ", sys.getsizeof(msg), " bytes")
```

In order to estimate the FLOPs (Floating Point Operations) of the forward pass of a MODEL, the library thop can be used.

```
import thop
flops_client, params = thop.profile(MODEL, inputs=(torch.rand(batchsize, 3, 32,
32).to(device),))
```

Recreate Environment

In the repository, an environment.yaml file is provided. The Conda environment can be reacreated easily, by running the following command:

`conda env create -f environment.yaml`