:

Documentation of Crypten Deployment on GCN and GP

Gauri Gupta: 2017MT60207

December 25, 2021

1 Deploying Crypten on Graph Convolution Network

In order to design a secure GCN model, we deployed Crypten library on GCN model built on Pytorch. The following is the GCNConv layer on which we implement the Network model.

Listing 1: GCNConv Class in Pytorch

```
class GCNConv(Module):
"""
Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
"""

def __init__(self, in_features, out_features, bias=True):
    super(GCNConv, self).__init__()
    self.in_features = in_features
    self.out_features = out_features
    self.weight = Parameter(torch.FloatTensor(in_features, out_features))
    if bias:
        self.bias = Parameter(torch.FloatTensor(out_features))
    else:
        self.register_parameter('bias', None)
    # initialzes weights with standard deviation = 1/sqrt(weight.size())
    self.reset_parameters()

def reset_parameters(self):
    stdv = 1. / math.sqrt(self.weight.size(1))
    self.weight.data.uniform_(-stdv, stdv)
    if self.bias is not None:
        self.bias.data.uniform_(-stdv, stdv)

def forward(self, input, adj, mask):
    D = 1 / (0.0000001 + adj.sum(axis=1).unsqueeze(-1))
    support = torch.mm(input, self.weight)
    output = D * torch.spmm(adj, support)
    if self.bias is not None:
        return output + self.bias
    else:
        return output

def __repr__(self):
    return self.__class__.__name__ + ' (' \
         + str(self.in_features) + ' -> ' \
         + str(self.out_features) + ')'
```

Now, we use this GCNCov layer to build the model class Net() in Pytorch. Deploying Crypten on GCN model was simple as the Crypten library is particularly designed to crypt the Neural Network models built in Pytorch. It has functions very similar to the ones used in Pytorch that can be directly used to build the crypt model. For instance, I replaced the torch.nn.Linear layer with crypten.nn.Linear and the functions like 'tanh', 'ReLu', 'cat' can be easily implemented in Crypten on the MPCTensor. Listing 2 and 3 shows the model Net() built in Pytorch and Crypten respectively.

Listing 2: GCN Network in Pytorch

```
class Net(torch.nn.Module):
```

```
    def __init__(self):
        super(Net, self).__init__()
        self.conv1Mean = GCNConv(2,8)
        self.conv2Mean = GCNConv(8,4)
        self.lrelu = nn.LeakyReLU(0.1)
        self.lin3 = nn.Linear(4, 1)

    def forward(self,data):
        x, adj = torch.cat((data['aqi'],data['mask'].view(-1,1)),1), data['adj']
        x = self.lrelu(self.conv1Mean(x, adj, data['mask']))
        x = self.lrelu(self.conv2Mean(x, adj, data['mask']))
        x = self.lin3(x)
        return torch.squeeze(x)
```

Listing 3: GCN Network in Crypten

```
class Net(crypten.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1Mean = GCNConv(2,8)
        self.conv2Mean = GCNConv(8,8)
        self.lin3 = crypten.nn.Linear(8, 1)

    def lrelu(self, x, alpha):
        temp1 = ((x < 0) * alpha) + (x > 0)
        return temp1 * x

    def forward(self, data):
        x, adj = crypten.cat([data['aqi'],data['mask'].unsqueeze(-1)],1), data['adj']
        x = self.lrelu(self.conv1Mean(x, adj), 0.1)
        x = self.lrelu(self.conv2Mean(x, adj), 0.1)
        x = self.lin3(x)
        return x.squeeze()
```

2 Deploying Crypten on Gaussian Processes

However, deploying Crypten on Gassuain Processes was not at all easy. We used GPyTorch which is a very large Gaussian process library implemented using PyTorch to implement our GP model. Some of the challenges we are faced are:

- gpytorch converts Pytorch tensor to LazyTensor for kernel and covariance matrix calculations which is not supported on `Crypten` tensor.
- Certain bool comparisons in gpytorch are not compatible with MPC Tensor because `Crypten` doesn't support bool operations without converting it into plaintext.
- GP uses a special Variational ELBO loss function from gpytorch library which is not supported by `Crypten`. `Crypten` currently supports very limited set of loss functions.
- Crypten library does not support in-place operations in autograd. It has limited functionality for the Autograd operations and does not support functions like retain_graph = True

The following is the snippet of the training process of GP model implemented using Crypten. The backward() method in cryten has very limited support where inplace operations are not supported and hence we got an error at the loss.backward() call.

Listing 4: GP in Crypten

```
# Now we set both to training mode
model.train()
likelihood.train()

# The use of an optimizer is again very similar to how it is used in neural nets. lr
    is learning rate.
# optimizer = torch.optim.Adam([
#    {'params': model.parameters()},
#    {'params': likelihood.parameters()},
# ], lr=0.05)
crypten_optimizer = crypten.optim.SGD([
   {'params': model.parameters()},
   {'params': likelihood.parameters()},
], lr=0.05)
# Our loss object. We're using the VariationalELBO. This defines the loss function we
    are trying to optimize
# mll = gpytorch.mlls.VariationalELBO(likelihood, model, num_data=train_y.size(0))

num_epochs = 500
for i in range(num_epochs):
   count = 0
   mean_loss = 0
   # Within each iteration, we will go over each minibatch of data
   minibatch_iter = train_loader
   for x_batch, y_batch in minibatch_iter:
      x_batch_enc = crypten.cryptensor(x_batch, src=ALICE)
      y_batch_enc = crypten.cryptensor(y_batch, src=ALICE)
      crypten_optimizer.zero_grad()
      output = model(x_batch_enc)
      print('output', type(output))
      print('y_batch', type(y_batch))
      loss = -mll(output, y_batch_enc)
      print('loss', type(loss))
      loss.backward()
      crypten_optimizer.step()
      batch_loss = loss.get_plain_text()
      count+=1
      crypten.print(f"Loss {batch_loss.item():.4f}")
```

```
    410         if inplace:
    411             if CrypTensor.AUTOGRAD_ENABLED and self.requires_grad:
--> 412                 raise RuntimeError("Autograd is not supported for in-place functions.")
    413
    414             # Note: native in-place support is now deprecated

RuntimeError: Autograd is not supported for in-place functions.
```

Apart from this, some of the major errors we faced are as follows:

Listing 5: Softplus implementation in Pytorch not supported in Crypten

```
softplus = torch.nn.Softplus()
self._transform = softplus
transformed_tensor = self._transform(tensor) if self.enforced else tensor
```

Listing 6: Softplus implementation in Crypten

```
tensor2 = (1 + tensor.exp()).log()
a = (tensor > 20)
tensor = a * tensor + (1-a)*tensor2
```

Crypten does not support Softplus activation function used in GP model class. We tried to implement the function using the functions already supported in Crypten.

Listing 7: Bool operations not supported in crypten

```
x1_eq_x2=False
if x1_eq_x2 and not x1.requires_grad and not x2.requires_grad:
   res.diagonal(dim1=-2, dim2=-1).fill_(0)
```

Since crypten does not support bool operations in MPCTensor, we are not able to implement bool conditions in Crypt tensors. The only way to support bool operations in crypten is to first convert the tensor to plaintext.

Listing 8: Converting to plaintext for performing bool operations

```
x1_eq_x2=False
if x1_eq_x2.get_plain_text() and not x1.requires_grad and not x2.requires_grad:
   res.diagonal(dim1=-2, dim2=-1).fill_(0)
```

Listing 9: Type in crypten

```
def dtype(self):
   return next(self.parameters()).dtype
```

Crypten does not have dtype() unlike the Pytorch tensor. CrypTen defines the ptype (for private-type) attribute of an MPCTensor to denote the kind of secret-sharing protocol used in the CrypTensor. The ptype may have two values: crypten.mpc.arithmetic for ArithmeticSharedTensors and crypten.mpc.binary for BinarySharedTensors