Gossip Swap SGD

参考にしたソースコード

PDMM SGDのGithubのコードを参考にしました。
“https://github.com/nttcslab/edge-consensus-learning”
大元のソースコードと動かし方はこのページに書いてあります。
環境設定やプログラムの動かし方はこちらも参考にしてください。
このドキュメントでは大元のソースコードの説明と横田が編集を加えた部分について簡単に説明します。

サンプルコードの編集と解説

プログラムはrun_mnist.pyかrun_cifar10.pyがニューラルネットワークに関する記述になります。
どのデータを学習するかによって動かすコードを選んでください。
Mnistのデータセットを学習に利用する場合はrun_mnistを編集と行った形。
この2つのファイルはニューラルネットワークのモデルや、ミニバッチサイズ、エポック数データセットの分布を制御しています。
基本的に編集する部分をリストアップしておきます。

データ分布の編集

cifar10のデータ分布の設定コードになります。

```
class Kings:
    classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
    train_mask_list = {
        "BALTHASAR": [False, False, False, True, True, True, False, False, False, False],
        "CASPER": [True, True, True, False, False, False, False, False, False, False],
        "MELCHIOR": [False, False, True, True, True, False, False, False, False, False],
        "KOUMEI": [False, False, False, False, False, False, True, True, True, False],
        "SHIBAI":[False,False,True,False,False,False,False,True,False,True]
    }
```

デバイス事にどのデータを学習に使用できるかを指定できます。
例えばこのコードだとデバイス”BALTHASAR”はcat deer dogのラベルのデータを訓練用に用いることができます。
新しいデバイスを追加する場合はこのフォーマットに乗っ取ってデバイス名、使用データの順に記載してください。

エポック、バッチサイズの調整

init関数の最初の引数を確認してください。
batch_size=100と記載されています。この値を編集することでミニバッチを調整できます。
他の引数については、デバイスのファイルで指定された引数が優先されるので、ここにあるのは何も指定がなかった場合のデフォルトの数字だと思ってください

```
def __init__(self, name, nodes, algorithm="pdmm", device="cpu",
             batch_size=100, interval=6, offset=0, log_dir="/log/"):
    self.logger = logging.getLogger(name)
    self.model = Net().to(device)
    self.device = device
    self.batch_size = batch_size
```

こちらのtrain関数のmax_epochで実験を行う際のエポック数を指定できます。
200と設定すると200エポック実験が行われることを示します。
test_intervalは何エポック事に精度を測定するかを示します。test_intevalが1だと毎エポック精度を確認できます。エポック数と合わせて設定してください。

```
def train(self, max_epoch=10, test_interval=1):
    self.logger.info('Training start!!')
    criterion = nn.CrossEntropyLoss()
    scheduler = torch.optim.lr_scheduler.StepLR(self.optimizer, step_size=10, gamma=0.95, last_epoch=100)
```

Optimizer部分の編集(スワップ処理、スキップスワップ処理)

サンプルコードである”run_mnist.py”または”run_cifar10.py”の編集の際を行うことで追加できます。

```
#gossip

finish=self.optimizer.update()
```

この更新がGossip SGDによる平均を用いた更新を示しています。
finishは接続先デバイスが見つかれば0、見つからなければ1の値になります。
途中でデバイスの接続が切れた場合、その後の処理で学習が終了します。

Swap処理の追加

finish=self.optimizer.swapupdate()
自分で設定したスワップインターバルに基づいて処理を記述してください。
例えば次ページの冒頭のように記述すると
Gossip SGDの更新とSwapの更新を交互に行う形になり、スワップインターバルが1の状態を再現できます。

self.itcntはこの関数が保持している変数です。

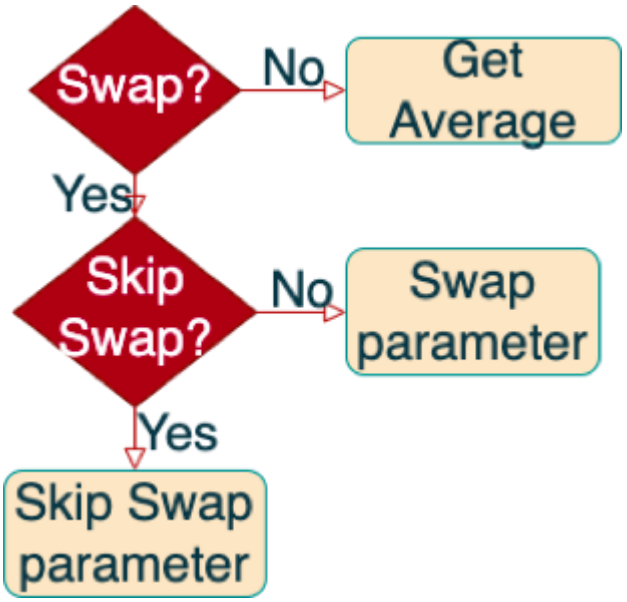
```
if self.itcnt==0:
   finish=self.optimizer.update()
   self.itcnt=1
elif self.itcnt==1:
    finish=self.optimizer.swapupdate()
    self.itcnt=0
```

SkipSwap処理の追加

finish=self.optimizer.skipswap()
Skip Swapはこの記述によって実行されます。

下のように記述すると、Gossip SGDの更新、スワップによる更新Gossip SGDによる更新、スキップスワップによる更新の順に行います。
結果として右図の分岐が全て1:1で行われるようになります。
実行頻度としては平均処理:スワップ:スキップスワップが2:1:1になります。

```
if self.itcnt==0:
   finish=self.optimizer.update()
   self.itcnt=1
elif self.itcnt==1:
   finish=self.optimizer.swapupdate()
   self.itcnt=2
elif self.itcnt==2:
   finish=self.optimizer.update()
   self.itcnt=3
else:
   finish=self.optimizer.skipswap()
   self.itcnt=0
```



通信部分の解説

contract.py及びedge.pyに記述されています。

Contract.pyについて

Contract.pyは呼び出されると、init関数において、実行しているターミナルに対応したconfファイルから、接続しているデバイスの情報をEdge関数を用いて作成します。

まず、nodeリストファイルとデバイスの対応を確認することをお勧めします。
nodelistのファイルを元にデバイスにプログラム中で番号が割り当てられます。
self_indexがデバイスの番号になります。
nodes[0]などを行うことで、どの番号がどのデバイスかを確認できます。
基本的にはnodelistのファイルの上から0,1,2のように割り当てられてると思います。

例
'''---self_index 0: CASPER 1:BALTHASAL 2:MELCHIOR 3:KOUMEI----'''

contract.py中のinit.pyの下記の記述によって、接続しているデバイスのEdgeクラスが作成されます。
先述したEdgeクラスの作成箇所がこの部分になります。

self._edges[edge_name]＝Edge[]と書かれている箇所で
Edgeクラスを持った配列が生成されます。
例えば
self._edges[BALTHASAL]
はBALTHASALのaddressや送受信データを保持するEdgeクラスを示します。

```
for edge_name in nodes[self_index]["edges"]:
    keys = list(self._edges.keys())
    if edge_name not in keys:
        con = self.hello(name, edge_info[edge_name]["addr"], model, state_req)
        if con:
            self._edges[edge_name] = Edge(edge_info[edge_name], self_index, name, device,
                                          model.state_dict(), is_state, is_dual, is_avg,
                                          grpc_buf_size, grpc_timeout)
            state_req = False
```

無視する処理

compweightやself.com_weight,comtable,totalweightなどは全て無視してください。
不要な変数になります。(別の手法を検討していた時に追加した処理なので...もし知りたければ後日聞いてください)

Edge.pyについて

def._updateやdef._recv def.send paramsなどがそのデバイスとの通信にどのデータを送るか、どんなデータを受け取ったかなどを示しています。
必要に応じて呼び出します。

難解なので基本的に編集しなくていいと思いますが、2種類以上のパラメータを送受信する場合は,”state”,”dual”,の2種類の変数を送っている箇所を参考にするといいと思います。

このstateとdualは元々のPDMM SGDが主変数、双対変数の2種類のパラメータを送受信する手法だったため残っているものです。
Gossip Skip Swap SGDの改良を行う際に2変数を同時に送信する場合はここを参考にすればよさそうな気がします。(この処理の変更が最も難解だと思います)

Skip Swap、及びSwap自体の編集方法

gossip_sgd.pyで編集を行います。

これが通常のgossip sgdの処理になります。
これを元に関数を作ります。

最後のcontract.swapを行うと、contractクラスがswap処理を動かし,
データの送信を行います。
p.dataは元のデバイスが保持しているパラメータ
d_pはエスケープするためのパラメータ
edge.rcv_stateは受信したパラメータ

```
    @torch.no_grad()
    def update(self):
        #自分と繋がっているedgeが帰ってくる。

        edges = self._contract.edges()  ##([('BALTHASAR', <edgecons.edge.Edge object at 0x7f9c2ecf1e50>)

        '''元のコード '''
        for edge in edges.values(): #それぞれのedgeに対しての処理
            '''if edge.rcv_cnt==1:
                edge.rcv_cnt=0'''
            for group in self.param_groups:
                for i, p in enumerate(group['params']):
                    d_p = p.data
                    p.data = torch.div((d_p + edge.rcv_state()[i]), 2)
                    edge.update(p.data, i)

        finish=self._contract.swap()
        return finish
```

スワップ処理を行う場合は下のように記述しました。
受信していた値をそのまま他デバイスに送るように記述されています。

```
79      @torch.no_grad()
80      def swapupdate(self):
81          #自分と繋がっているedgeが帰ってくる。
82
83          edges = self._contract.edges()  ##([('BALTHASAR', <edgecons.edge.Edge object at 0x7f9c2ecf1e50>)
84
85          '''モデル交換のみを行う '''
86
87          for edge in edges.values():
88              if edge.rcv_cnt==1:
89                  edge.rcv_cnt=0
90              for group in self.param_groups:
91                  for i, p in enumerate(group['params']):
92                      d_p = p.data
93                      p.data =  edge.rcv_state()[i]
94                      edge.update(d_p, i)
95
96
97
98          finish=self._contract.swap()
99          return finish
100
```

skipswapはこの記述です。
2変数を送受信する場合、この記述だとダメなので、1から作り直す必要があります。

```
@torch.no_grad()
def skipswap(self):
    #自分と繋がっているedgeが帰ってくる。

    edges = self._contract.edges()  ##([('BALTHASAR', <edgecons.edge.Edge object at 0x7f9c2ecf1e50>)


    for edge in edges.values():
        if edge.rcv_cnt==1:
            edge.rcv_cnt=0
            if edge._self_name!=self.hold:
                self.hold=edge._self_name
                for group in self.swap_param:
                    for i, p in enumerate(group['params']):
                        d_p = p.data
                        p.data =  edge.rcv_state()[i]
                        edge.update(d_p, i)

            else:
                for group in self.param_groups:
                     for i, p in enumerate(group['params']):
                        d_p = p.data
                        p.data =  edge.rcv_state()[i]
                        edge.update(d_p, i)


    finish=self._contract.swap()
```