情報工学演習 II 岡田先生の課題

1 XOR

Listing 1 に作成したソースコードを示す．また図 1，図 2，図 3，図 4，図 5，図 6 に学習率を 0.001, 0.01, 0.1, 1, 10, 100 にしたときの loss の変化及び勾配の最大値を示す．

Listing 1: XOR

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn

class MLP(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super(MLP, self).__init__()

    self.l1 = nn.Linear(input_size, hidden_size)
    self.l2 = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    h = torch.sigmoid(self.l1(x))
    o = self.l2(h)
    return o

x = [
     [0., 0.],
     [0., 1.],
     [1., 0.],
     [1., 1.]
]

x = torch.tensor(x)
y = [
     [0.],
     [1.],
     [1.],
     [0.]
]
y = torch.tensor(y)

Ir_all = [0.001, 0.01, 0.1, 1, 10, 100]
max_epoch = 30000

for Ir in Ir_all:
  model = MLP(2, 3, 1)
  optimizer = torch.optim.SGD(model.parameters(), lr=Ir)
  criterion = nn.MSELoss()
  params = list(model.parameters())

  history = []
  history_l1 = []
  history_l2 = []

  for epoch in range(max_epoch):
    pred = model(x)
    error = criterion(y, pred)
    history.append(error.item())
    model.zero_grad()
    error.backward()
    optimizer.step()

    history_l1.append(torch.max(params[0].grad).data)
    history_l2.append(torch.max(params[2].grad).data)

  history = np.array(history, dtype=np.float32)
  history_l1 = np.array(history_l1, dtype=np.float32)
  history_l2 = np.array(history_l2, dtype=np.float32)
  epochs = np.arange(1, max_epoch+1)

  fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12, 4))
  axes[0].plot(epochs, history)
  axes[0].set_title('Ir␣=␣%f,␣loss' % (Ir))
  axes[0].set_xlabel("epochs")
  axes[0].set_ylabel("loss")
  axes[1].plot(epochs, history_l1)
  axes[1].set_title('max␣grad(l1)')
  axes[1].set_xlabel("epochs")
  axes[1].set_ylabel("max␣grad")
  axes[2].plot(epochs, history_l2)
  axes[2].set_title('max␣grad(l2)')
  axes[2].set_xlabel("epochs")
  axes[2].set_ylabel("max␣grad")
  plt.show()
```

学習率が 0.001, 0.01 のとき，loss が 0 付近まで到達していない．逆に学習率が 1 以上のときは loss が増加して

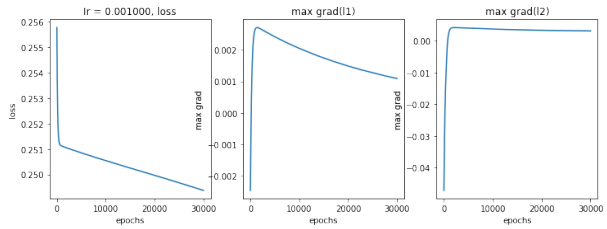

図 1: 学習率 0.001 のとき

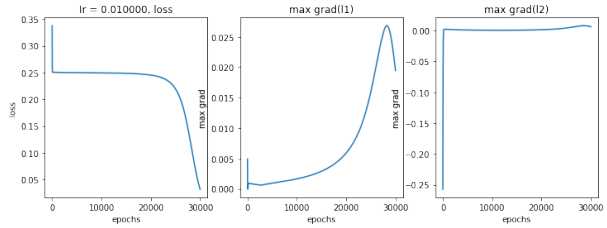

図 2: 学習率 0.01 のとき

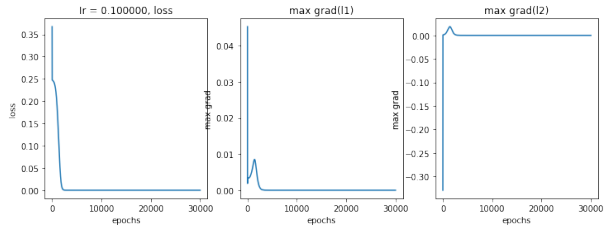

図 3: 学習率 0.1 のとき

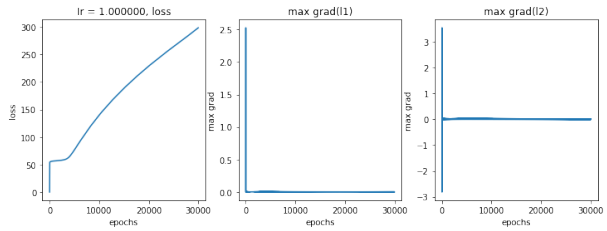

図 4: 学習率 1 のとき

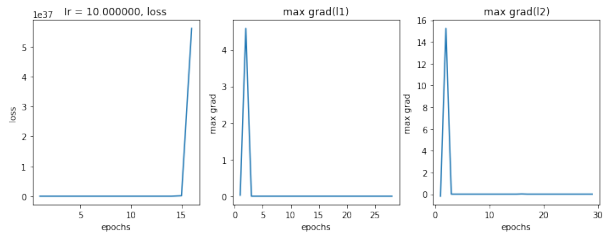

図 5: 学習率 10 のとき

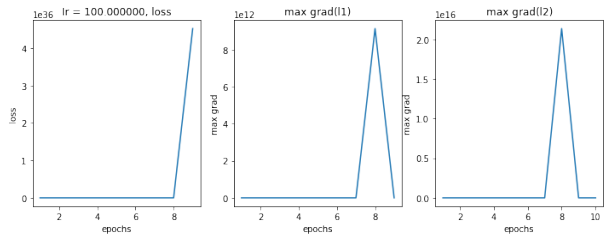

図 6: 学習率 100 のとき

いる．今回調べた 6 種類の中では 0.1 が一番効率よく学習できている．

2 パリティビット

Listing 2 に作成したソースコードを示す．また図 7 に loss の変化を示す．

Listing 2: パリティビット

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn

class MLP(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super(MLP, self).__init__()

    self.l1 = nn.Linear(input_size, hidden_size)
    self.l2 = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    h = torch.sigmoid(self.l1(x))
    o = self.l2(h)
    return o

x = torch.tensor([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]], dtype=torch.float)
y = torch.tensor([0, 1, 1, 0, 1, 0, 0, 1])

model = MLP(3, 3, 2)
optimizer = torch.optim.SGD(model.parameters(), lr=1.0e-1)
criterion = nn.CrossEntropyLoss()

history = []
max_epoch = 100000

for epoch in range(max_epoch):
  pred = model(x)
  error = criterion(pred, y)
  history.append(error.item())
  model.zero_grad()
  error.backward()
  optimizer.step()

```

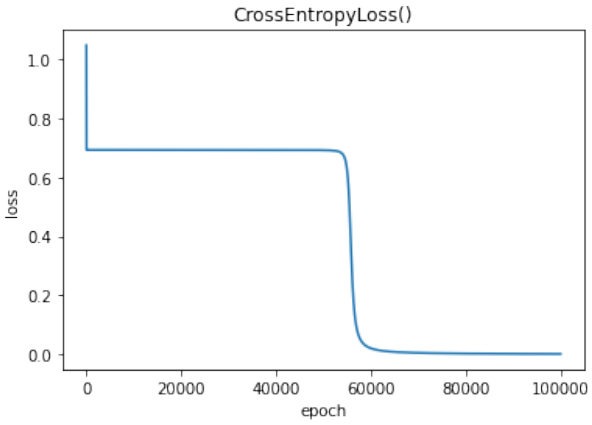

図 7: 交差エントロピーを誤差関数としたときの loss の推移

```
36 history = np.array(history, dtype=np.
       float32)
37 epochs = np.arange(1, max_epoch+1)
38
39 plt.plot(epochs, history)
40 plt.title("CrossEntropyLoss()")
41 plt.xlabel("epoch")
42 plt.ylabel("loss")
43 plt.show()
44
45 for x_data in x:
46   x_in = torch.tensor(x_data)
47   y_out = model(x_in)
48   index = torch.argmax(y_out)
49   print("[{},␣{},␣{}]␣=>␣{}".format(
         x_data[0], x_data[1], x_data[2],
         index))
```

Listing 2 の最後の for 文で学習データを入力したときに正しく解が得られるか確認した．図 8 にその結果を示す．

```
[0.0, 0.0, 0.0] => 0
[0.0, 0.0, 1.0] => 1
[0.0, 1.0, 0.0] => 1
[0.0, 1.0, 1.0] => 0
[1.0, 0.0, 0.0] => 1
[1.0, 0.0, 1.0] => 0
[1.0, 1.0, 0.0] => 0
[1.0, 1.0, 1.0] => 1
```

図 8: Listing 2 の最後の for 文の出力結果