HW 4 Report

2016112083 김연웅

About My Model Design

```
# Define DNN Model
class MyModel(torch.nn.Module):

    def __init__(self):
        super(MyModel,self).__init__()
        self.l1=torch.nn.Linear(100,30,bias=True)
        self.l2=torch.nn.Linear(30,15,bias=True)
        self.l3 = torch.nn.Linear(15,1,bias=True)


        self.Sigmoid = torch.nn.Sigmoid()
        self.ReLU=torch.nn.ReLU()

        self.dropout = torch.nn.Dropout(p=0.2)

    def forward(self,x):
        x = self.dropout(self.ReLU((self.l1(x))))
        x = self.dropout(self.ReLU((self.l2(x))))
        y_pred = self.Sigmoid(self.l3(x))
        return y_pred
```

Model Layers

1. **Input layer:**

- Input dimension: 100
- output dimension: 30

2. **Frist hidden layer:**

- Input dimension : 30
- output dimension : 15
- activation function: ReLU
- dropout: 0.2

3. **Second hidden layer:**

- input dimension: 15
- output dimension: 1
- activation function: ReLU
- dropout: 0.2

4. **output :**

- output : 1
- activation function: sigmoid

Number of Parameters

```
The number of parameters: 3511 elements
MyModel(
  (layer1): Sequential(
    (0): Linear(in_features=100, out_features=30, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.2, inplace=False)
  )
  (layer2): Sequential(
    (0): Linear(in_features=30, out_features=15, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.2, inplace=False)
  )
  (layer3): Sequential(
    (0): Linear(in_features=15, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

The number of parameters: 3511 elements.

Weighted F1 Score I obtained

In [15]:
```
# Load best performance model
checkpoint = torch.load('./bestModel.pt')
trained_model = MyModel()



trained_model.load_state_dict(checkpoint['state_dict'])

# evaluate
trained_model.eval()
with torch.no_grad():

    # load test dataset
    test_x = torch.from_numpy(np.loadtxt('./HW4_testX.csv',delimiter=',',dtype=np.float32))
    test_y = torch.from_numpy(np.loadtxt('./HW4_testY.csv',delimiter=',',dtype=np.float32).reshape(-1,1))

    # evaluate using weighted f1 score.
    y_pred = trained_model(test_x)
    yp = y_pred.detach().numpy()
    yp = [1.0 if x > 0.5 else 0.0 for x in yp]

    print("weighted F1:", f1_score(test_y, yp, average='weighted'))
```

```
weighted F1: 0.8329938841330211
```

Hyper parameter setting

```
# Hyper parameter setting
batch_size = 1000
learning_rate =0.01
weight_decay = 0.02
num_epoch = 500
momentum = 0.95
# Create Loss function and optimizer
criterion = torch.nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate, weight_decay=weight_decay, momentum = momentum)


# Create datsets and dataloader
trainset = HW4_trainDataset()
testset = HW4_testDataset()
train_loader =DataLoader(trainset, batch_size = batch_size, shuffle = True)
test_loader= DataLoader(testset, batch_size = batch_size, shuffle = False)
```

mini-batch size : 1000

num_epoch : 500

loss function: torch.nn.BCELoss()

optimizer : torch.optim.SGD()
(learning_rate:0.01, weight decay: 0.02, momentum:0.95)

---

comments

unlike my conventional belief, larger mini-batch size such as 1000, yielded better f1 score. I`m tryint to figure out the reason.

Random Seeds Configuration

```
# Configure Random seeds
torch.manual_seed(777)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(777)
```