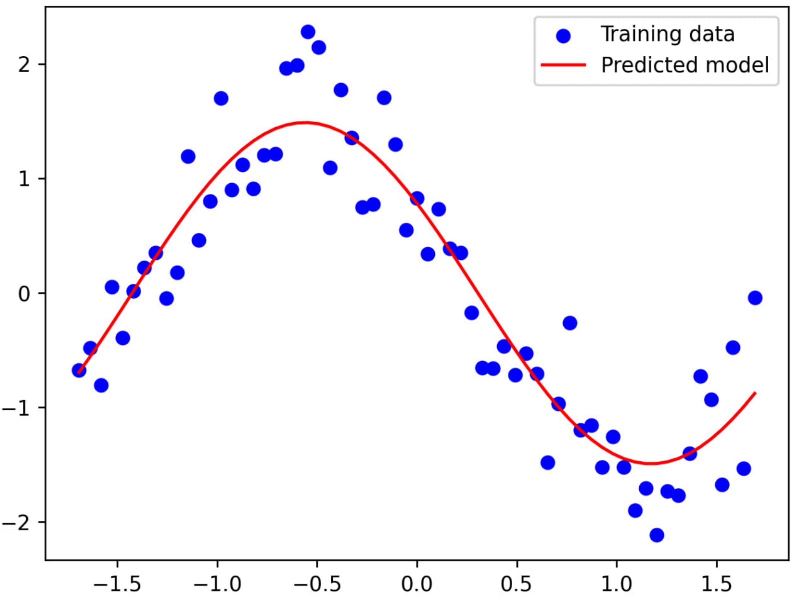
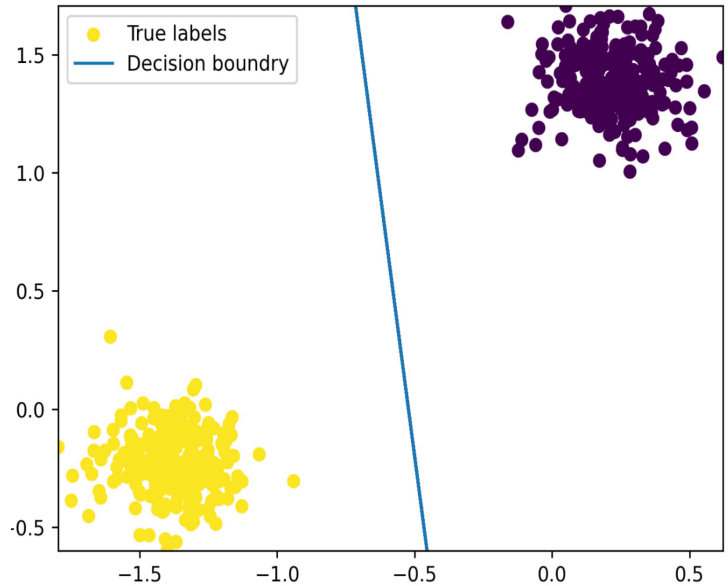


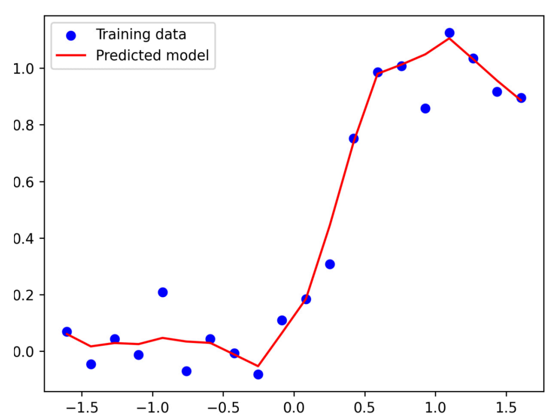



```
class Sinw(nn.Module):
    def __init__(self):
        super().__init__()
        self.b =Parameter( torch.randn((1,1),requires_grad=True) )
        self.a = Parameter( torch.randn((1,1),requires_grad=True) )
        self.a0 = Parameter( torch.randn((1,1),requires_grad=True) )
        self.b0 =Parameter( torch.randn((1,1),requires_grad=True) )

    def forward(self,inp):

        return self.a0*torch.sin(self.a*inp+self.b)+self.b0
model = torch.nn.Sequential(Sinw())

criterion = nn.L1Loss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for i in range(5000):
    y_pred = model(X_train)
    loss = criterion(y_pred.view(-1), y_train)
    if i%100==0:
        print(loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

model.eval()
y_pred = model(X_train)

plt.scatter(X_train.numpy(), y_train.numpy(), color='blue', label='Training data')
plt.plot(X_train.numpy(), y_pred.detach().numpy(), color='red', label='Predicted model')
```

```
model = torch.nn.Sequential(torch.nn.Linear(2,1))
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001

for i in range(100):
    y_pred = model(X_train)
    loss = criterion(y_pred.view(-1), y_train)
    if i%100==0:
        print(loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

model.eval()
X_test = torch.arange(0, 5, 0.01).view(-1, 1)
y_pred = model(X_train)
```

For the above data, I fitted a sin because the data looks like a nosy sin(x). I minimized the absolute distance between x2 and a0*sin(ax+b)+b0.
For the classification problem, I separated the data with a simple line x1w1+x2w2+b and threshold 0.5. data looks like two circles.

```
y_pred[y_pred>=0.5]=1.0
y_pred[y_pred<0.5]=0.0

p =list(model.parameters())
y = (-X_train[:,0]* p[0][0,0]-p[1] + 0.5 )/p[0][0,1]
plt.scatter(X_train[:,0].numpy(),X_train[:,1].numpy(),c=y_pred.detach().numpy()[:,0], label='True labels')
plt.plot(X_train[:,0].numpy(),y.detach().numpy(), label='Decision boundry')
plt.xlim([X_train[:,0].min(),X_train[:,0].max()])
plt.ylim([X_train[:,1].min(),X_train[:,1].max()])
plt.legend()
plt.show()
```




```
model = torch.nn.Sequential(torch.nn.Linear(1,128),torch.nn.ReLU(),torch.nn.Linear(128,1))
criterion = nn.L1Loss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for i in range(1000):
    y_pred = model(X_train)
    loss = criterion(y_pred.view(-1), y_train)
    if i%100==0:
        print(loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

model.eval()
y_pred = model(X_train)

plt.scatter(X_train.numpy(), y_train.numpy(), color='blue', label='Training data')
plt.plot(X_train.numpy(), y_pred.detach().numpy(), color='red', label='Predicted model')
```

For this problem, I use a two-layer NN with L1 loss (MAE). the number of data is low so this model has overfitted with such many parameters.

Majid Sharghi Foroushani, M.N. 23193769, ix05ogym