Appendix

3

(a)

```
class Net(nn.Module):
    def __init__(self, d):
        super(Net, self).__init__()
        # TO IMPLEMENT Part (a)
        # here you should declare functions for layers with parameters
        self.fc1 = nn.Linear(d, 10*d, bias=True)
        self.fc2 = nn.Linear(10*d, 1, bias=True)

    def forward(self, x):
        # TO IMPLEMENT Part (a)
        # here you should trace the forward computation of the network
        # using functions declared in __init__
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x.float()

```

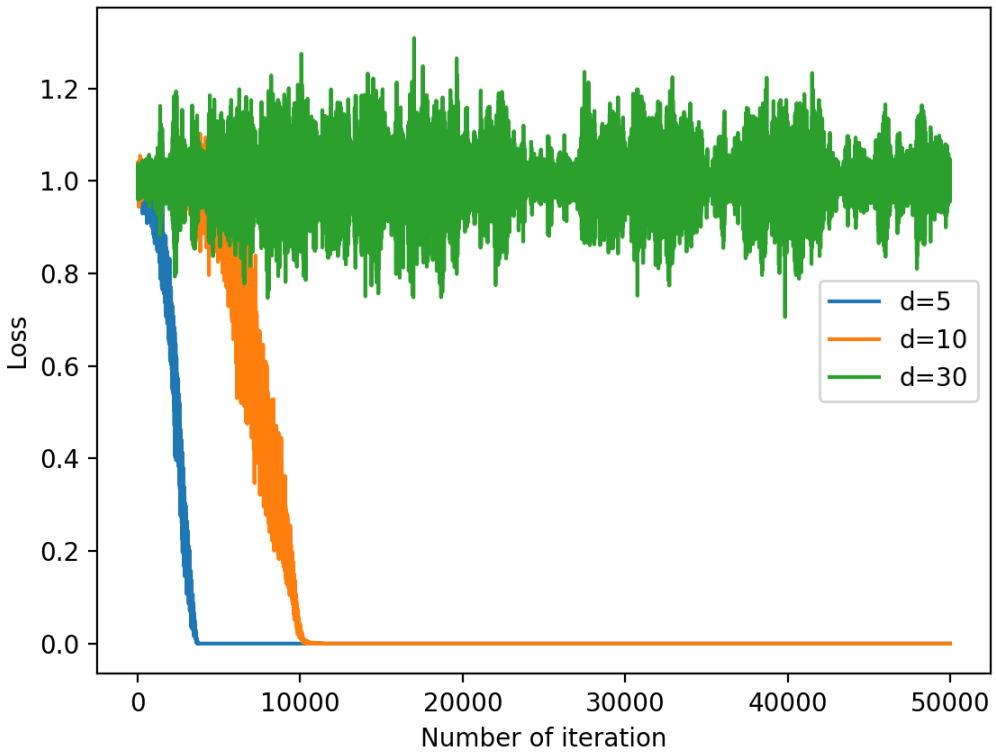
```
# Initializing the data generation process
d = 5
data = DataGen(d)

### Defining network parameters by hand

# Initializing the network
net = Net(d)

# Defining and setting parameters for perfect approximation
# TO IMPLEMENT Part (a)
v = data.v_star
w = np.zeros((10*d, d))
for i in range(10*d):
    if i <= 3*d/2-1:
        w[i,:] = v
b = np.zeros((10*d, 1))
alpha = np.zeros((1, 10*d))
for j in range((d//2)+1):
    b[3*j,0] = -(2*j-0.5)
    b[3*j+1,0] = -2*j
    b[3*j+2,0] = -(2*j+0.5)
    alpha[0,3*j] = 4
    alpha[0,3*j+1] = -8
    alpha[0,3*j+2] = 4
beta = np.array([-1])
set_parameters(d, w, b, alpha, beta, net)

# Checking that network perfectly approximates 10,000 examples
inputs, labels = data.get_batch(10000)
assert sum(net(inputs).reshape(-1) - labels.float()) == 0
```

(b)

```python
# Defining the loss
def myLoss(outputs, labels):
    # TO IMPLEMENT Part (b)
    # should return 1/b * sum_{i=1}^b loss(f_p(x_i),y_i)
    outputs = outputs.view(-1,1)
    labels = labels.view(-1,1)
    n = outputs.size()[0]
    ones = torch.ones(n, 1, dtype=float, requires_grad=True)
    arg = torch.mul(outputs, labels)
    arg = torch.mul(arg, -1)
    arg = torch.add(arg, 1)
    l = F.relu(arg)
    ml = torch.sum(l)
    ml = torch.div(ml, n)
    return ml
```

```python
d_list = [5, 10, 30]
for d1 in d_list:
    net1 = Net(d1)
    data1 = DataGen(d1)
    loss_function1 = myLoss
    optimizer1 = optim.SGD(net1.parameters(), lr=0.001, momentum=0.9)
    batch_size1 = 100
    num_iter1 = 5*10**4
    loss_over_iteration1 = []
    for m in range(num_iter1):
        inputs1, labels1 = data1.get_batch(batch_size1)
        optimizer1.zero_grad()
        outputs1 = net1(inputs1)
        loss1 = loss_function1(outputs1, labels1)
        loss1.backward()
        optimizer1.step()
        loss_over_iteration1.append(loss1.item())
        if m%200 == 0:
            print('[%5d] loss: %.3f' % (m, loss1.item()))
    print('Finished Training d=%2d' % d1)
    plt.plot(np.array(loss_over_iteration1))
plt.legend(('d=5', 'd=10', 'd=30'))
plt.xlabel('Number of iteration')
plt.ylabel('Loss')
plt.show()
```

Plot


1.2
1.0
0.8
0.6
0.4
0.2
0.0
Loss
0
10000
20000
30000
40000
50000
Number of iteration
d=5
d=10
d=30