XOR

In [1]:

```
import numpy as np
import torch
import torch.utils.data as tud
import torch.nn as tnn
```

In [2]:

```
import matplotlib.pyplot as plt
```

In [3]:

```
class xor_dataset(tud.Dataset):
  def __init__(self, size=10000):
    # purposely getting dataset from np as random values might be different for torch
    np.random.seed(0)
    self.data = torch.from_numpy(2 * np.random.uniform(size=(size, 2)) - 1)
    # as suggested on teams, using two output nodes
    labels = torch.zeros((size, 1))
    labels[self.data[:, 0]*self.data[:, 1]<0, 0]=1
    self.labels=labels

  def __len__(self):
    return len(self.data)

  def __getitem__(self, idx):
    return {'point': self.data[idx], 'label': self.labels[idx]}
```

In [4]:

```
train_set, val_set, test_set = tud.random_split(xor_dataset(), [7000,1500,1500], generato
r=torch.Generator().manual_seed(0))
```

In [5]:

```
train_loader = tud.DataLoader(train_set, batch_size=16, shuffle=True, drop_last=True)
val_loader = tud.DataLoader(val_set, batch_size=1500, shuffle=True, drop_last=True)
test_loader = tud.DataLoader(test_set, batch_size=1500, shuffle=True, drop_last=True)
```

In [6]:

```
class my_nn_1(tnn.Module):
  def __init__(self, h, d_in=2, d_out=1):
    super(my_nn_1, self).__init__()
    self.linear1 = tnn.Linear(d_in, h)
    self.linear2 = tnn.Linear(h, d_out)
    self.sig = tnn.Sigmoid() # since loss cant be calculated for ouputs exceeding 1

  def forward(self, x):
    h_relu = self.linear1(x).clamp(min=0)
    z_pred = self.linear2(h_relu)
    y_pred = self.sig(z_pred)
    return y_pred
```

In [7]:

```
# cross entropy loss isn't valid for binary classification
# hence binary cross entropy is used
crit_bce = tnn.BCELoss()
crit_mse = tnn.MSELoss()
```

In [8]:

```
from copy import deepcopy as dcopy
```

In [9]:

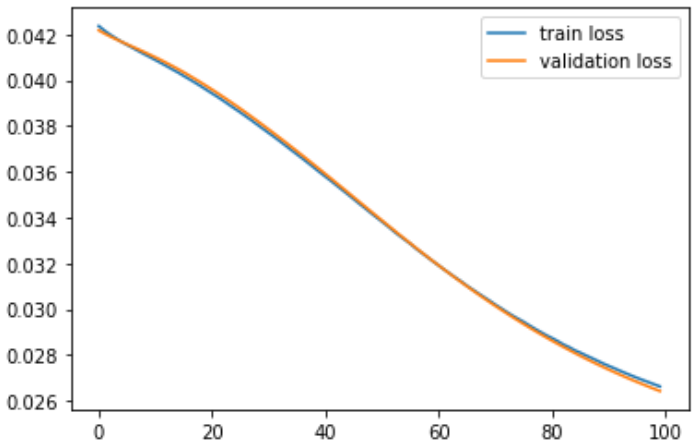
```
def fit(h=4, epochs=10, criterion=tnn.BCELoss(), lr=1e-3):
  model = my_nn_1(h)
  opt = torch.optim.SGD(model.parameters(), lr=lr)
  tlh=[]
  tah=[]
  vlh=[]
  vah=[]
  vl_min=10**10
  for ep in range(epochs):
    model.train()
    training_loss=0
    acc=0
    for (idx, b) in enumerate(train_loader):
      xb=b["point"].float()
      yb=b["label"]
      yb_pred = model(xb)
      loss=criterion(yb_pred, yb)
      training_loss+=loss
      acc+=torch.sum(torch.round(yb_pred)==yb)
      opt.zero_grad()
      loss.backward()
      opt.step()
    tah.append(acc/7000)
    tlh.append(training_loss/7000)
    model.eval()
    with torch.no_grad():
      val_loss=0
      acc=0
      for (idx, b) in enumerate(val_loader):
        xb=b["point"].float()
        yb=b["label"]
        yb_pred = model(xb)
        loss=criterion(yb_pred, yb)
        acc+=torch.sum(torch.round(yb_pred)==yb)
        val_loss+=loss
      vah.append(acc/1500)
      vl=val_loss/16
      if vl<vl_min:
        best_model=dcopy(model)
        vl_min=vl
      vlh.append(vl)
  return best_model, tlh, tah, vlh, vah
```

In [10]:

```
m, tlh, tah, vlh, vah = fit(epochs=100, criterion=crit_bce, lr=1e-3)
```

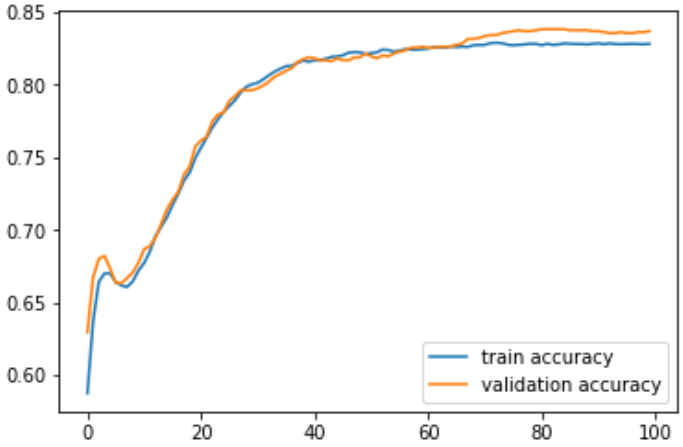
In [11]:

```
plt.plot(tlh, label="train loss")
plt.plot(vlh, label="validation loss")
plt.legend()
plt.show()
```



In [12]:

```
plt.plot(tah, label="train accuracy")
plt.plot(vah, label="validation accuracy")
plt.legend()
plt.show()
```

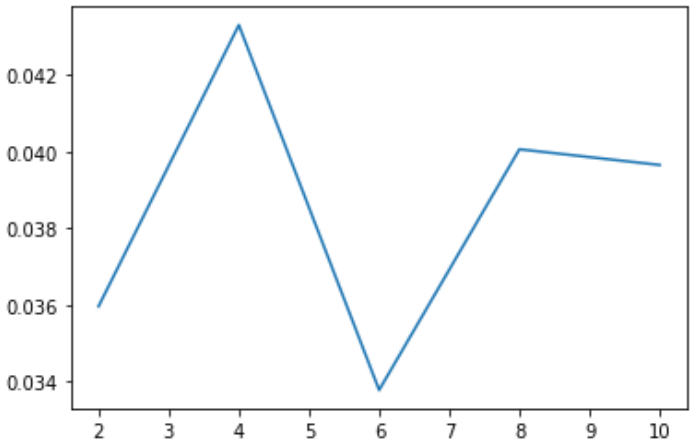


In [17]:

```
hl_sizes=[2,4,6,8,10]
best_vl=[]
for hl in hl_sizes:
  _, tlh, tah, vlh, vah = fit(h=hl, epochs=50, criterion=crit_bce, lr=1e-3)
  best_vl.append(np.min(vlh))
```

In [18]:

```
plt.plot(hl_sizes, best_vl)
plt.show()
```



In [19]:

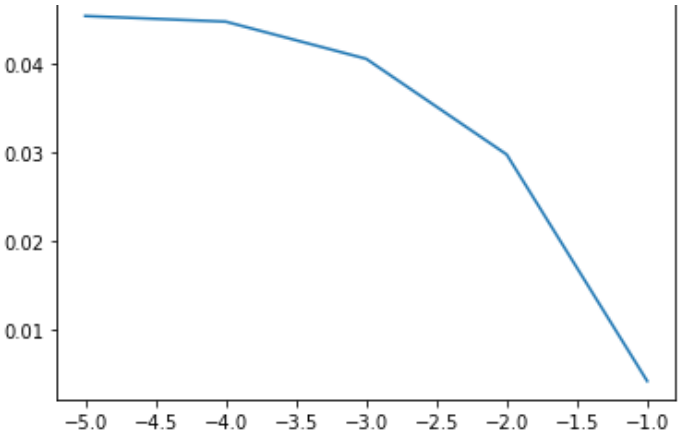
```
# h=6 gives best results
```

In [20]:

```
lr_vals=[10**(i-5) for i in range(5)]
best_vl=[]
for lr in lr_vals:
  _, tlh, tah, vlh, vah = fit(epochs=20, criterion=crit_bce, lr=lr)
  best_vl.append(np.min(vlh))
```

In [21]:

```
plt.plot(np.log10(lr_vals), best_vl)
plt.xlabel = "Learning rate as a power of 10"
plt.show()
```



In [22]:

```
# lr=0.1 gives best results
```

In [24]:

```
best_m, tlh, tah, vlh, vah = fit(h=6, epochs=100, criterion=crit_bce, lr=0.1)
```

In [25]:

```
full_test_loader = tud.DataLoader(test_set, batch_size=1500)
for pts in full_test_loader:
  test_pts=pts
```

In [ ]:

```
# mean loss
label_pred = best_m(test_pts["point"].float())
crit_bce(label_pred, test_pts["label"])/16
```

Out[ ]:

```
tensor(0.0026, grad_fn=<DivBackward0>)
```

In [ ]:

```
label_pred[label_pred<0.5]=0
label_pred[label_pred>=0.5]=1
```

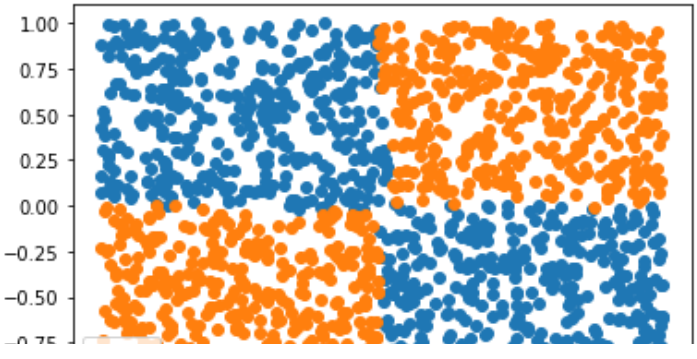
In [ ]:

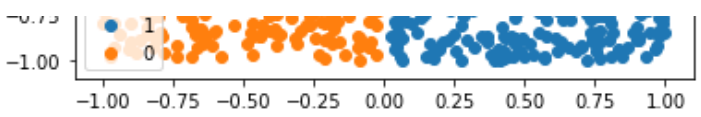
```
# accuracy
print(torch.sum(label_pred==test_pts["label"])/(label_pred.shape[0]))
```

```
tensor(0.9853)
```

In [ ]:

```
x=test_pts["point"][:, 0].reshape(-1,1)
y=test_pts["point"][:, 1].reshape(-1,1)
plt.scatter(x[label_pred==1], y[label_pred==1], label="1")
plt.scatter(x[label_pred==0], y[label_pred==0], label="0")
plt.legend()
plt.show()
```





In [ ]:

```
# near perfect prediction
```

In [ ]:

```

```