In class Pytorch Excercises

Basira Daqiq

2/19/2026

```
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
#Generate data for training a neural network for XOR
def genXORData(n=100, noise=0):
  A = np.random.randint(0, 2, size=n)
  B = np.random.randint(0, 2, size=n)
  C = A ^ B
  return np.stack((A+np.random.normal(0, noise, size=n), B+np.random.normal(0,
  noise, size=n)), axis=1), C
```

```
def train(model, X, y, eta=0.1, steps=100, l2=0.0):
  X = torch.tensor(X, dtype=torch.float32)
  y = torch.tensor(y, dtype=torch.float32).view(-1, 1) #convert to a column vector

  opt = torch.optim.SGD(model.parameters(), lr=eta, weight_decay=l2)
  BCE = nn.BCEWithLogitsLoss()

  for step in range(steps):
    model.train()
    opt.zero_grad()

    #run forward propagation and collect logits on the full data set
    #how does this change for mini batches?
    logits = model(X)
    #determine the loss
    loss = BCE(logits, y)
    #run backward propagation
    loss.backward()
    #update parameters
    opt.step()

    if step>0 and step%100==0:
      model.eval()
      with torch.no_grad():
        preds = torch.sigmoid(logits)
        yHat = (preds >= 0.5).float()
        acc = (yHat.eq(y)).float().mean().item()
      print('Step: ', step, 'Loss: ', loss, "Acc: ", acc)

  return model
```

```
def accuracy(model, X, y):
    X_t = torch.tensor(X, dtype=torch.float32)
    y_t = torch.tensor(y, dtype=torch.float32).view(-1, 1)

    model.eval()
    with torch.no_grad():
        logits = model(X_t)
        preds = (torch.sigmoid(logits) >= 0.5).float()
        acc = (preds.eq(y_t)).float().mean().item()
    return acc
```

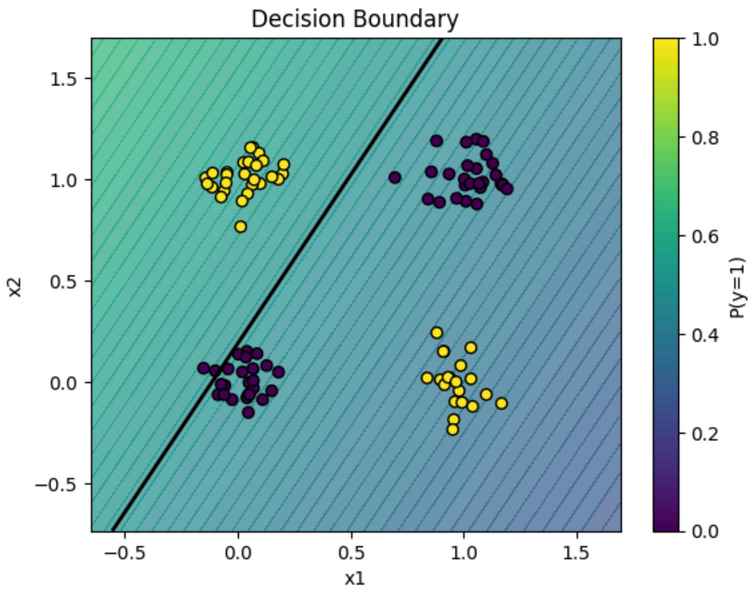
```
def plotBoundary(model, X, y, steps=100):
  model.eval()
  x_min, x_max = X[:,0].min()-0.5, X[:,0].max()+0.5
  y_min, y_max = X[:,1].min()-0.5, X[:,1].max()+0.5

  xx, yy = np.meshgrid(np.linspace(x_min, x_max, steps), np.linspace(y_min, y_max, steps))
  grid = np.c_[xx.ravel(), yy.ravel()]
  grid_t = torch.tensor(grid, dtype=torch.float32)

  with torch.no_grad():
    logits = model(grid_t).view(-1)
    probs = torch.sigmoid(logits)

  Z = probs.reshape(xx.shape)

  plt.contourf(xx, yy, Z, levels=50, vmin=0, vmax=1, alpha=0.7)
  plt.contour(xx, yy, Z, levels=[0.5], colors='black', linewidths=2)
  plt.scatter(X[:,0], X[:,1], c=y, edgecolor="k")
  plt.title("Decision Boundary")
  plt.xlabel("x1");
  plt.ylabel("x2")
  plt.colorbar(label="P(y=1)")
  plt.show()

#Extract the activation values of a particular layer in the network
#Adapted with assistance from ChatGPT
def getActivations(model, layer, X):
  acts = []

  def hook(module, inp, out):
    acts.append(out.detach().cpu())

  handle = layer.register_forward_hook(hook)
  model.eval()
  with torch.no_grad():
    _ = model(torch.tensor(X, dtype=torch.float32))
  handle.remove()
  return acts[0].numpy()

#Embed the activations of a particular layer in 2d using pca and visualize
#Adapted with assistance from ChatGPT
def plotLayerFeatures(model, layer, X, y):
  A = getActivations(model, model[layer], X)
  Z = PCA(n_components=2).fit_transform(A)

  plt.scatter(Z[:,0], Z[:,1], c=y, edgecolor="k")
  plt.title("Layer "+str(layer))
  plt.xlabel("feat-1"); plt.ylabel("feat-2")
  plt.show()
```

1. XOR problem with zero hidden layers.

The model does pretty bad with only one layers. it has an accuracy of about 70% however all the wrong samples only come from one group based on the decision boundary. Even with trying different learing rate values, and different steps the model performence does not approve single.

```
X, y = genXORData(n=100, noise=0.1)

#Create layers to be added to the network
#Typically, these will alternate between layer type and non-linear activation functions
layers = []
layers.append(nn.Linear(2, 1)) #input dimension, output dimension, changed from 2 to 1
model = nn.Sequential(*layers)
model = train(model, X, y, steps=1000)
plotBoundary(model, X, y)
```

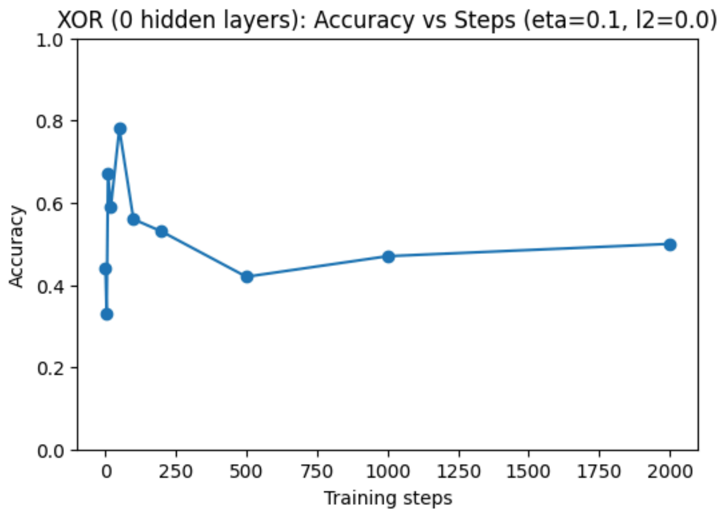
```
Step:  100 Loss:  tensor(0.6853, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.800000011920929
Step:  200 Loss:  tensor(0.6841, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.800000011920929
Step:  300 Loss:  tensor(0.6837, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.800000011920929
Step:  400 Loss:  tensor(0.6835, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.800000011920929
Step:  500 Loss:  tensor(0.6834, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7900000214576721
Step:  600 Loss:  tensor(0.6834, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7900000214576721
Step:  700 Loss:  tensor(0.6833, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7900000214576721
Step:  800 Loss:  tensor(0.6833, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7799999713897705
Step:  900 Loss:  tensor(0.6833, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7799999713897705
```



Double-click (or enter) to edit

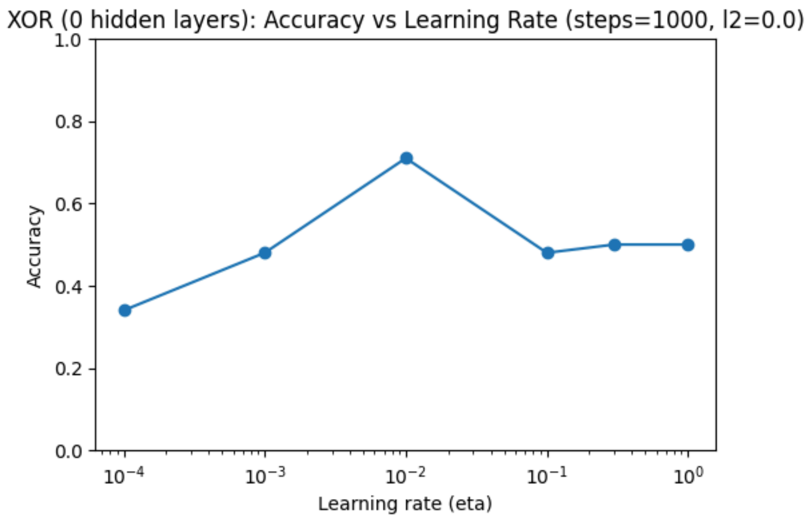
```
# Data
X, y = genXORData(n=100, noise=0.1)

# Hyperparams for this sweep
eta_fixed = 0.1
l2_fixed = 0.0
step_list = [1, 5, 10, 20, 50, 100, 200, 500, 1000, 2000]

accs_steps = []

for s in step_list:
    model = nn.Sequential(nn.Linear(2, 1))
    model = train(model, X, y, eta=eta_fixed, steps=s, l2=l2_fixed)
    accs_steps.append(accuracy(model, X, y))

plt.figure(figsize=(6,4))
plt.plot(step_list, accs_steps, marker='o')
plt.xlabel("Training steps")
plt.ylabel("Accuracy")
plt.title(f"XOR (0 hidden layers): Accuracy vs Steps (eta={eta_fixed}, l2={l2_fixed})")
plt.ylim(0, 1)
plt.show()
```

```
Step:  100 Loss:  tensor(0.6888, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5400000214576721
Step:  100 Loss:  tensor(0.6875, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5400000214576721
Step:  200 Loss:  tensor(0.6842, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.4699999988079071
Step:  300 Loss:  tensor(0.6831, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3499999940395355
Step:  400 Loss:  tensor(0.6828, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3799999952316284
Step:  100 Loss:  tensor(0.6966, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5400000214576721
Step:  200 Loss:  tensor(0.6878, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5400000214576721
Step:  300 Loss:  tensor(0.6847, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.4000000059604645
Step:  400 Loss:  tensor(0.6835, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3199999928474426
Step:  500 Loss:  tensor(0.6830, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.33000001311302185
Step:  600 Loss:  tensor(0.6828, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3499999940395355
Step:  700 Loss:  tensor(0.6827, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.4099999964237213
Step:  800 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.41999998688697815
Step:  900 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.46000000834465027
Step:  100 Loss:  tensor(0.6901, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5400000214576721
Step:  200 Loss:  tensor(0.6856, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5099999904632568
Step:  300 Loss:  tensor(0.6839, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.33000001311302185
Step:  400 Loss:  tensor(0.6832, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3199999928474426
Step:  500 Loss:  tensor(0.6829, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  600 Loss:  tensor(0.6827, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.38999998569488525
Step:  700 Loss:  tensor(0.6827, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.41999998688697815
Step:  800 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.4399999976158142
Step:  900 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.4699999988079071
Step:  1000 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  1100 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  1200 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.49000000953674316
Step:  1300 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.49000000953674316
Step:  1400 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  1500 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  1600 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  1700 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  1800 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  1900 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
```



```
# Hyperparams for this sweep
steps_fixed = 1000
l2_fixed = 0.0
eta_list = [1e-4, 1e-3, 1e-2, 1e-1, 3e-1, 1.0]

accs_eta = []

for eta in eta_list:
    model = nn.Sequential(nn.Linear(2, 1))
    model = train(model, X, y, eta=eta, steps=steps_fixed, l2=l2_fixed)
    accs_eta.append(accuracy(model, X, y))

plt.figure(figsize=(6,4))
plt.plot(eta_list, accs_eta, marker='o')
plt.xscale("log")
plt.xlabel("Learning rate (eta)")
plt.ylabel("Accuracy")
plt.title(f"XOR (0 hidden layers): Accuracy vs Learning Rate (steps={steps_fixed}, l2={l2_fixed})")
plt.ylim(0, 1)
plt.show()
```

```
Step:  100 Loss:  tensor(0.7099, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3499999940395355
Step:  200 Loss:  tensor(0.7097, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3499999940395355
Step:  300 Loss:  tensor(0.7096, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3499999940395355
Step:  400 Loss:  tensor(0.7095, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3499999940395355
Step:  500 Loss:  tensor(0.7093, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  600 Loss:  tensor(0.7092, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  700 Loss:  tensor(0.7091, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  800 Loss:  tensor(0.7090, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  900 Loss:  tensor(0.7088, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  100 Loss:  tensor(0.7122, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  200 Loss:  tensor(0.7113, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  300 Loss:  tensor(0.7104, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  400 Loss:  tensor(0.7095, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  500 Loss:  tensor(0.7087, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  600 Loss:  tensor(0.7080, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  700 Loss:  tensor(0.7072, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  800 Loss:  tensor(0.7065, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  900 Loss:  tensor(0.7059, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  100 Loss:  tensor(0.7225, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.6000000238418579
Step:  200 Loss:  tensor(0.7061, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.6700000166893005
Step:  300 Loss:  tensor(0.6974, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.75
Step:  400 Loss:  tensor(0.6928, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7699999809265137
Step:  500 Loss:  tensor(0.6902, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7699999809265137
Step:  600 Loss:  tensor(0.6887, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7799999713897705
Step:  700 Loss:  tensor(0.6877, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7799999713897705
Step:  800 Loss:  tensor(0.6870, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.7799999713897705
Step:  900 Loss:  tensor(0.6864, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.75
Step:  100 Loss:  tensor(0.6852, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3100000023841858
Step:  200 Loss:  tensor(0.6838, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3100000023841858
Step:  300 Loss:  tensor(0.6832, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3199999928474426
Step:  400 Loss:  tensor(0.6829, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3400000035762787
Step:  500 Loss:  tensor(0.6827, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.3799999952316284
Step:  600 Loss:  tensor(0.6827, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.41999998688697815
Step:  700 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.4300000071525574
Step:  800 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.46000000834465027
Step:  900 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  100 Loss:  tensor(0.6829, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.46000000834465027
Step:  200 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.47999998927116394
Step:  300 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.49000000953674316
Step:  400 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  500 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  600 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  700 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  800 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  900 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  100 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.49000000953674316
Step:  200 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  300 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  400 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  500 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  600 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  700 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  800 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
Step:  900 Loss:  tensor(0.6826, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.5
```



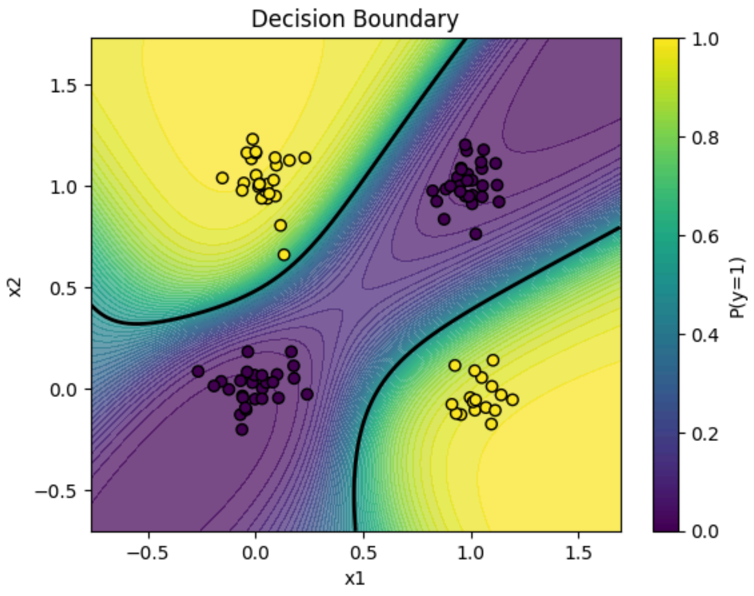
Start coding or generate with AI.

2. Single Hidden layer

The model perfomrnvr improves significantly with an additional of one hidden layer. The model is able to reach 100% trainng accuracy withing 500 iteration. The model stays consistently performing well even with varrying learning rate value and total number of iteration.

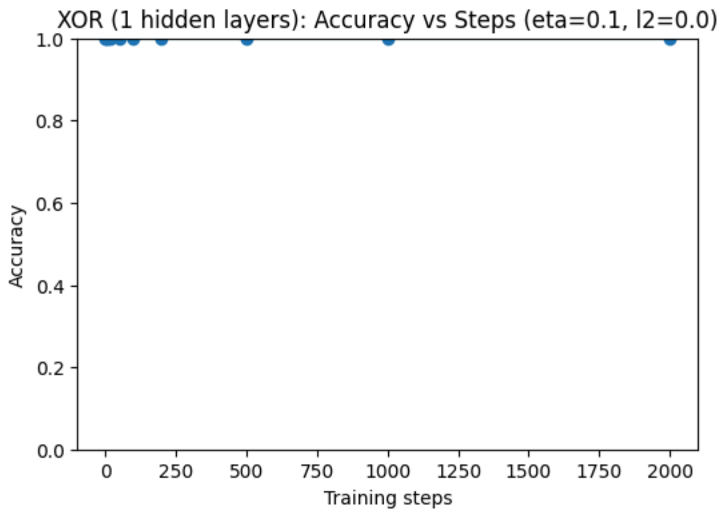
```
layers1 = []
layers1.append(nn.Linear(2, 16)) #input dimension, output dimension, changed from 2 to 1
layers1.append(nn.Tanh())
layers1.append(nn.Linear(16, 1))

model1 = nn.Sequential(*layers1)
model1 = train(model1, X, y, steps=1000)
plotBoundary(model1, X, y)
```

```
Step:  100 Loss:  tensor(0.6347, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.8100000023841858
Step:  200 Loss:  tensor(0.5450, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.8199999928474426
Step:  300 Loss:  tensor(0.3994, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.8299999833106995
Step:  400 Loss:  tensor(0.2496, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9900000095367432
Step:  500 Loss:  tensor(0.1534, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.1018, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0733, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0560, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0448, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
```



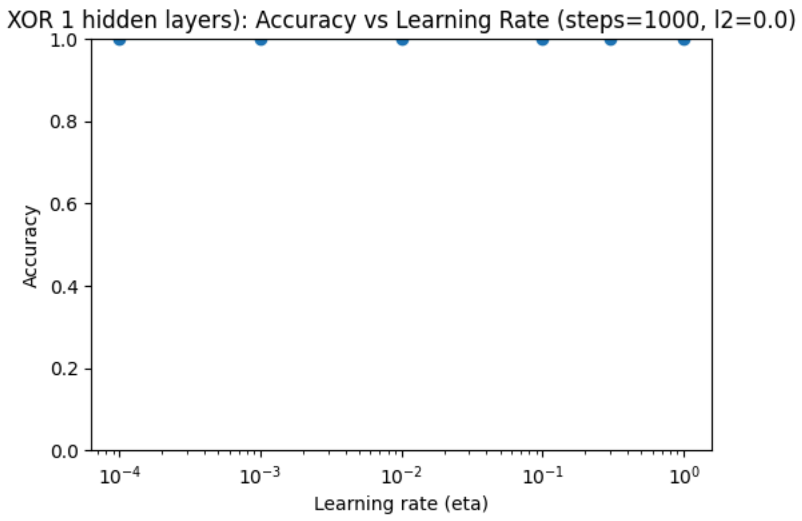
```
X, y = genXORData(n=100, noise=0.1)

# Hyperparams for this sweep
eta_fixed = 0.1
l2_fixed = 0.0
step_list = [1, 5, 10, 20, 50, 100, 200, 500, 1000, 2000]

accs_steps = []

for s in step_list:
    model1 = train(model1, X, y, eta=eta_fixed, steps=s, l2=l2_fixed)
    accs_steps.append(accuracy(model1, X, y))

plt.figure(figsize=(6,4))
plt.plot(step_list, accs_steps, marker='o')
plt.xlabel("Training steps")
plt.ylabel("Accuracy")
plt.title(f"XOR (1 hidden layers): Accuracy vs Steps (eta={eta_fixed}, l2={l2_fixed})")
plt.ylim(0, 1)
plt.show()
```

```
Step:  100 Loss:  tensor(0.0006, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0005, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1000 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1700 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  1900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
```



```
steps_fixed = 1000
l2_fixed = 0.0
eta_list = [1e-4, 1e-3, 1e-2, 1e-1, 3e-1, 1.0]

accs_eta = []

for eta in eta_list:
    model1 = train(model1, X, y, eta=eta, steps=steps_fixed, l2=l2_fixed)
    accs_eta.append(accuracy(model1, X, y))

plt.figure(figsize=(6,4))
plt.plot(eta_list, accs_eta, marker='o')
plt.xscale("log")
plt.xlabel("Learning rate (eta)")
plt.ylabel("Accuracy")
plt.title(f"XOR 1 hidden layers): Accuracy vs Learning Rate (steps={steps_fixed}, l2={l2_fixed})")
plt.ylim(0, 1)
plt.show()
```

```
Step:  100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0004, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  100 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  200 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  300 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  400 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  500 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  600 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  700 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  800 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
Step:  900 Loss:  tensor(0.0003, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  1.0
```

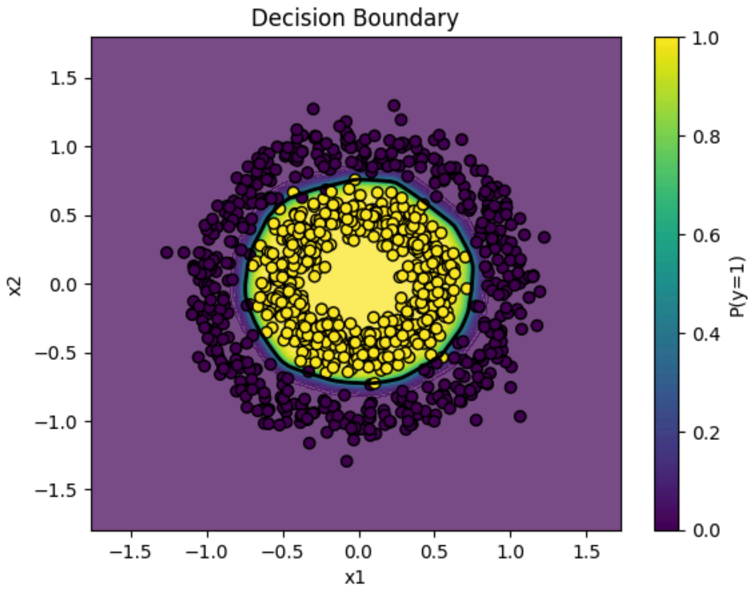


3. Explore different network depths and hidden layer widths for the concentric circle dataset.

Deeper networks preform better because diffrent nodes specialize at getting different areas of the inpute distrubution correct. So the

```
X, y = make_circles(n_samples=1000, noise=0.1, factor=0.5)

#Create layers to be added to the network
#Typically, these will alternate between layer type and non-linear activation functions
layers = []
layers.append(nn.Linear(2, 32))
layers.append(nn.ReLU())
layers.append(nn.Linear(32, 64))
layers.append(nn.ReLU())
layers.append(nn.Linear(64, 10))
layers.append(nn.ReLU())
layers.append(nn.Linear(10, 1))
#layers.append(nn.Sigmoid)

model = nn.Sequential(*layers)

model = train(model, X, y, steps=1000)

layer = 0
plotBoundary(model, X, y)
plotLayerFeatures(model, layer, X, y)
```

```
Step:  100 Loss:  tensor(0.6536, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.8600000143051147
Step:  200 Loss:  tensor(0.4642, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9010000228881836
Step:  300 Loss:  tensor(0.1034, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9900000095367432
Step:  400 Loss:  tensor(0.0456, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9909999966621399
Step:  500 Loss:  tensor(0.0322, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9909999966621399
Step:  600 Loss:  tensor(0.0265, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9909999966621399
Step:  700 Loss:  tensor(0.0233, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9909999966621399
Step:  800 Loss:  tensor(0.0213, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9909999966621399
Step:  900 Loss:  tensor(0.0198, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>) Acc:  0.9919999837875366
```


Decision Boundary
x1
x2
P(y=1)