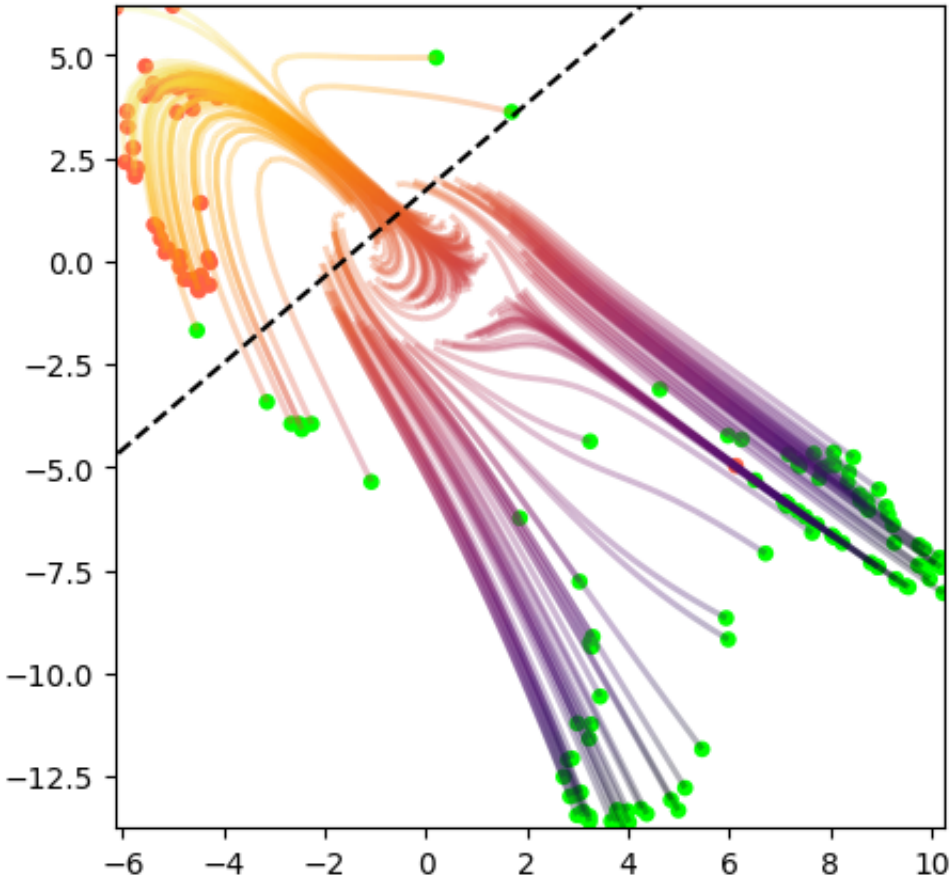
```
500 0.045606400817632675
```




```
label = (yh[1,:,0] >= 0.5).long()
```

```
print((torch.abs(label)-torch.abs(ytrain)).sum()/1024)
```

```
tensor(0.0156)
```

[Questions]

You can see that the neural ODEs fail to achieve 100% accuracy. This is somewhat odd considering that the simple shallow network could achieve 100% accuracy and NODEs are an advanced version of ResNets.

[3 pts] Can you explain why NODEs (by design) fail and are not suitable for performing this task?

Answer: In the case of NODE, the movement of one point can not overlap with the trajectory of another point. At the point of overlap, the vectors of both the points will follow the same trajectory. In the case of our data set, the Circular Annuli Data, one set of data points are enclosed within another. This might be causing the overlap in trajectories and hence NODEs are not able to classify.

[4 pts] now implement variants of NODEs that overcome the issue of NODEs shown above. Any variant of NODE will be accepted. The resulting model should achieve 100% accuracy. Following is the one such variant. See the implementation below (Not all parts of the implementation is revealed; I removed one important block myDepthCat.)

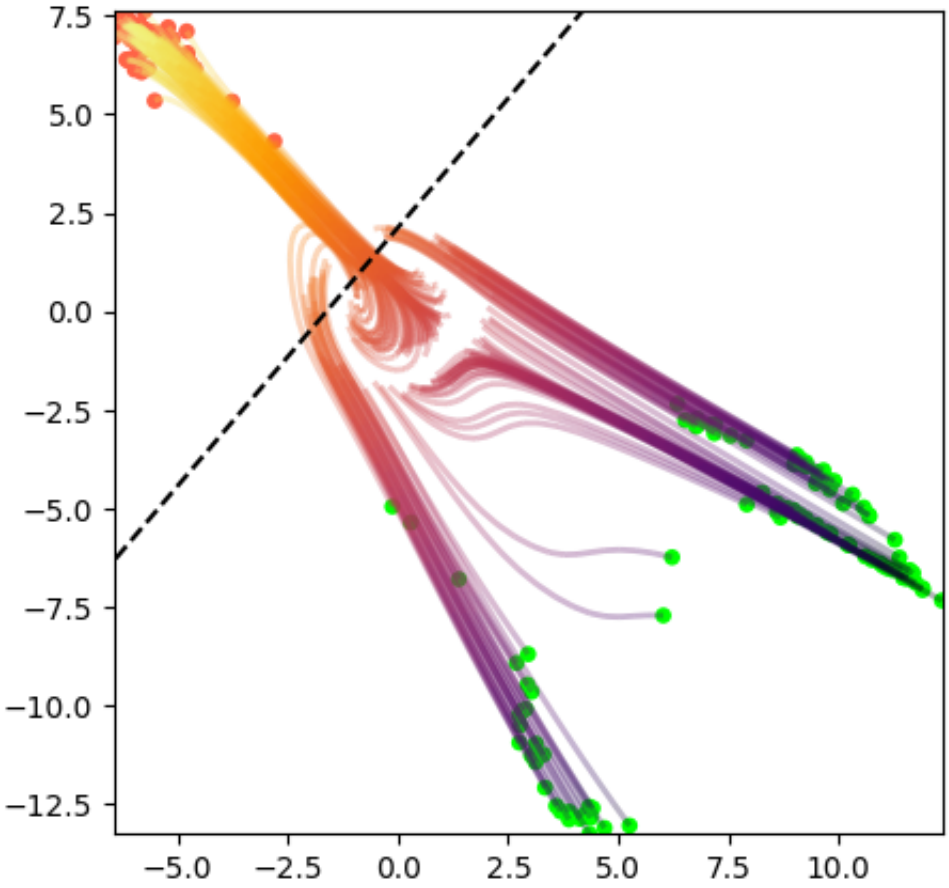
```
class ODEFunc(nn.Module):

    def __init__(self):
        super(ODEFunc, self).__init__()

        hdim = 32
        self.net = nn.Sequential(
            myDepthCat(1),
            nn.Linear(2+1, hdim),
            nn.Tanh(),
            myDepthCat(1),
            nn.Linear(hdim+1, hdim),
            nn.Tanh(),
            nn.Linear(hdim, 2)
        )

        #for p in self.net[-1].parameters(): torch.nn.init.zeros_(p)

    def forward(self, t, y):
        for _, module in self.net.named_modules():
            if hasattr(module, 't'):
                module.t = t
        return self.net(y)


#Concatenate depth info to the input
class myDepthCat(nn.Module):
    def __init__(self, additional_dim):
        super(myDepthCat, self).__init__()
        self.t = None
        self.dim_add = additional_dim
    def forward(self, x):
        time = torch.ones(x.size(0), self.dim_add).to(x.device) * self.t  #Concat
        return torch.cat((x, time), dim=1)
```

```
class Model(nn.Module):
    def __init__(self,odefunc,device="cpu"):
        super(Model, self).__init__()

        self.odefunc = odefunc
        self.linear_layer = nn.Linear(2,1)
        for p in self.linear_layer.parameters(): torch.nn.init.zeros_(p)

    def forward(self, y):
        t_span = torch.linspace(0., 1., 2)
        pred_y = odeint(self.odefunc, y, t_span).to(device)
        yhat = self.linear_layer(pred_y)
        return yhat



from matplotlib.collections import LineCollection

ii = 0
odefunc = ODEFunc().to(device)
model = Model(odefunc).to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-6)

for itr in range(1, 501):
    optimizer.zero_grad()
    #x0, y = sample_annuli(device=device, n_samples=1024)
    yh = model(Xtrain)
    loss = nn.BCELoss()(torch.sigmoid(yh[-1]), ytrain[:,None].float())
    print(itr,loss.item())
    loss.backward()
    optimizer.step()

    if itr % 100 == 0:
        with torch.no_grad():
            x0, ys = sample_annuli(n_samples=200) ; s = torch.linspace(0, 1, 20)
            xS = odeint(model.odefunc, x0, s).to(device)
            r = 1.05*torch.linspace(xS[:,:,-2].min(), xS[:,:,-2].max(), 2)
            pS = torch.cat([model.linear_layer(xS[:,i,-2:].to(device))[None,:,:]

            fig, ax = plt.subplots(1, 1, figsize=(5,5), sharex=True, sharey=True)
            for i in range(len(x0)):
                x, y, p = xS[:,i,-2].numpy(), xS[:,i,-1].numpy(), model.linear_la
                points = np.array([x, y]).T.reshape(-1, 1, 2) ; segments = np.con
```

```
            norm = plt.Normalize(pS.min(), pS.max())
            lc = LineCollection(segments, cmap='inferno', norm=norm, alpha=.3
            lc.set_array(p) ; lc.set_linewidth(2) ; line = ax.add_collection(
        pS_ = model.linear_layer(xS[-1,:,-2:].to(device)).view(-1).detach().c
        colors = ['lime','tomato']
        for i in range(200):
            ax.scatter(xS[-1,i,-2], xS[-1,i,-1], c=colors[ys[i]], edgecolor='

        ax.plot(r, dec_bound(model, r), '--k')
        ax.set_xlim(xS[:,:,-2].min(), xS[:,:,-2].max()) ; ax.set_ylim(xS[:,:,
        plt.show()
```

```
1 0.6931472420692444
2 0.6931009888648987
3 0.6930516362190247
4 0.6929983496665955
5 0.692940890789032
6 0.6928789019584656
7 0.6928120255470276
8 0.6927399635314941
9 0.6926623582839966
10 0.6925788521766663
11 0.6924890875816345
12 0.6923923492431641
13 0.6922881603240967
14 0.6921758651733398
15 0.6920547485351562
16 0.6919237375259399
17 0.6917822360992432
18 0.6916290521621704
19 0.6914631724357605
20 0.6912834644317627
21 0.6910887360572815
22 0.6908777356147766
23 0.6906490325927734
24 0.6904011964797974
25 0.6901324987411499
26 0.6898413896560669
27 0.6895262002944946
28 0.6891847848892212
29 0.6888154745101929
30 0.688416063785553
31 0.6879847049713135
32 0.6875191926956177
33 0.6870173215866089
34 0.6864770650863647
35 0.6858961582183838
36 0.6852725744247437
```

```
491 0.007249310612678528
492 0.007099114824086428
493 0.00702727260068059
494 0.006929709576070309
495 0.006885241251438856
496 0.006831398233771324
497 0.006734044756740332
498 0.006691489368677139
499 0.006618768908083439
500 0.006551876664161682
```




```
label = (yh[1,:,0] >= 0.5).long()
print((torch.abs(label)-torch.abs(ytrain)).sum()/1024)
```

```
tensor(0.)
```

```
accuracy = (label==ytrain).sum()/1024
print("Model Accuracy is: ", accuracy.item())
```

```
Model Accuracy is:  1.0
```