In [12]:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import copy
```

In [13]:

```python
hexa = pd.read_csv('FeedForward_Data_hexa.csv', names=['dim1','dim2','label'])
```

In [14]:

```python
def plot_data(X, y, filename):
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
    plt.savefig(filename)
    plt.close()

def plot_decision_boundary(clf, X, y, filename):
    # Set min and max values and add padding as required
    x_min, x_max = -1.0, 1.0
    y_min, y_max = -1.0, 1.0
    h = 0.01
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Predict the function value for the whole gid
    #Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    X_out = clf(torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype = torch.float))
    Z = X_out.data.max(1)[1]
    # Z.shape
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral, s = 1)
    plt.savefig(filename)
    plt.close()
```

In [15]:

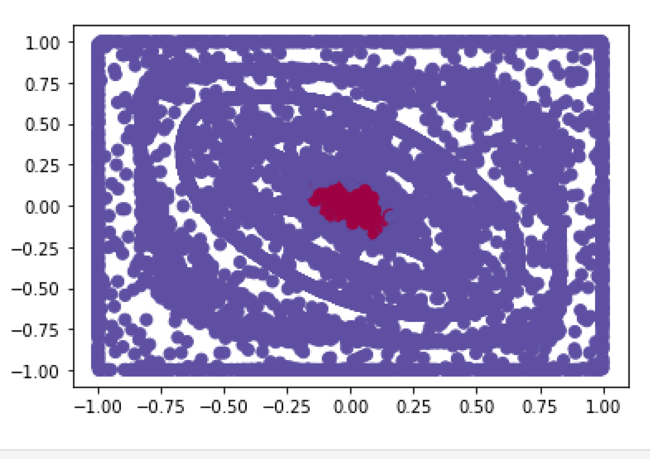
```python
X_hexa = hexa.values[:, 0:2]  # Take only the first two features.
X_hexa = torch.tensor(X_hexa, dtype = torch.float)
y_hexa = hexa.values[:, 2]
y_hexa = torch.tensor(y_hexa, dtype = torch.long)
```

In [16]:

```python
plt.scatter(X_hexa[:, 0], X_hexa[:, 1], c=y_hexa, cmap=plt.cm.Spectral, s = 50)
```

Out[16]:

```
<matplotlib.collections.PathCollection at 0x2534b8f05e0>
```



In [17]:

```python
class Net_a(nn.Module):

    def __init__(self):
        super(Net_a, self).__init__()
        self.fc1 = nn.Linear(2, 15)
        torch.nn.init.uniform_(self.fc1.weight, a=-2, b=2)

        self.fc2 = nn.Linear(15, 15)
        torch.nn.init.uniform_(self.fc2.weight, a=-2, b=2)

        self.fc3 = nn.Linear(15, 15)
        torch.nn.init.uniform_(self.fc3.weight, a=-2, b=2)

        self.fc4 = nn.Linear(15, 15)
        torch.nn.init.uniform_(self.fc4.weight, a=-2, b=2)

        self.fc5 = nn.Linear(15, 15)
        torch.nn.init.uniform_(self.fc5.weight, a=-2, b=2)

        self.fc6 = nn.Linear(15, 2)
        torch.nn.init.uniform_(self.fc6.weight, a=-2, b=2)


    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = F.relu(self.fc5(x))
        x = self.fc6(x)
        return F.log_softmax(x)
        #return F.softmax(x)
```

In [18]:

```python
previous_loss = 1

#%% train
net_a = Net_a()

# create a stochastic gradient descent optimizer
learning_rate = .02
optimizer = torch.optim.SGD(net_a.parameters(), lr=learning_rate, momentum=0.9)


# create a loss function
#criterion = nn.CrossEntropyLoss()
criterion = nn.NLLLoss()

#Stopping conditions
stopping_crit = np.power(1/10,6)
nepochs = 20000

data, target = X_hexa, y_hexa
for epoch in range(nepochs):

    optimizer.zero_grad()
    # forward propagate
    net_out = net_a(data)
    # compute loss
    loss = criterion(net_out, target)

    # backpropagate
    loss.backward()
    # update parameters
    optimizer.step()
    # print out report

    if epoch % 100 == 0:
        print('Epoch ', epoch, 'Loss ', loss.item())
        net_out = net_a(data)
        pred = net_out.data.max(1)[1]  # get the index of the max log-probability
        correctidx = pred.eq(target.data)
        ncorrect = correctidx.sum()
        accuracy = ncorrect.item()/len(data)
        print('Training accuracy is ', accuracy)
        if accuracy >= 0.97:
            break

    if abs(previous_loss - loss.item()) < stopping_crit:
        break

    previous_loss = copy.copy(loss.item())

#%% compute accuracy on training data
net_out = net_a(data)
pred = net_out.data.max(1)[1]  # get the index of the max log-probability
correctidx = pred.eq(target.data)
ncorrect = correctidx.sum()
accuracy = ncorrect.item()/len(data)
print('Training accuracy is ', accuracy)
```

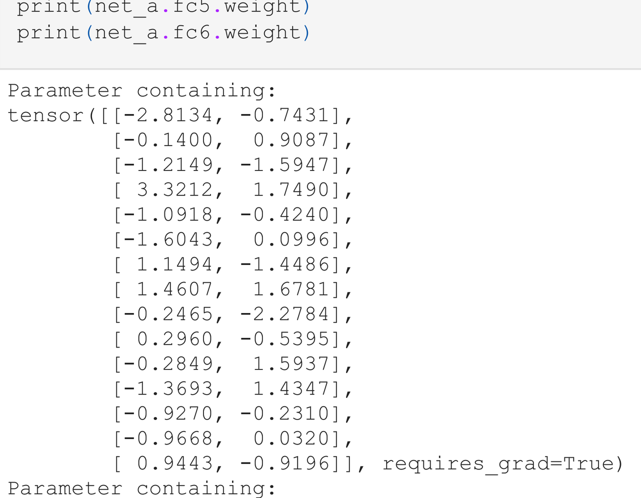
```
<ipython-input-17-732c73c9d00f>:31: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  return F.log_softmax(x)
Epoch  0 Loss  20.884859085083008
Training accuracy is  0.9405799278846154
Epoch  100 Loss  0.1759939044713974
Training accuracy is  0.9405799278846154
Epoch  200 Loss  0.16694723069667816
Training accuracy is  0.9405799278846154
Epoch  300 Loss  0.15456463396549225
Training accuracy is  0.9405799278846154
Epoch  400 Loss  0.1535159796476364
Training accuracy is  0.9405799278846154
Epoch  500 Loss  0.1627575308084488
Training accuracy is  0.9405799278846154
Epoch  600 Loss  0.15054650604724884
Training accuracy is  0.9405799278846154
Epoch  700 Loss  0.1530572474002838
Training accuracy is  0.9405799278846154
Epoch  800 Loss  0.1507023274898529
Training accuracy is  0.9405799278846154
Epoch  900 Loss  0.14822515845298767
Training accuracy is  0.9405799278846154
Epoch  1000 Loss  0.14909633994102478
Training accuracy is  0.9405799278846154
Epoch  1100 Loss  0.15268570184707642
Training accuracy is  0.9405799278846154
Epoch  1200 Loss  0.15150310099124908
Training accuracy is  0.9405799278846154
Epoch  1300 Loss  0.1510932594537735
Training accuracy is  0.9405799278846154
Epoch  1400 Loss  0.1531142294406891
Training accuracy is  0.9405799278846154
Epoch  1500 Loss  0.15063579380512238
Training accuracy is  0.9405799278846154
Epoch  1600 Loss  0.15320692956447 6
Training accuracy is  0.9405799278846154
Epoch  1700 Loss  0.14856640994548798
Training accuracy is  0.9405799278846154
Epoch  1800 Loss  0.1551642268896103
Training accuracy is  0.9405799278846154
Epoch  1900 Loss  0.14954222738742828
Training accuracy is  0.9405799278846154
Epoch  2000 Loss  0.1498364359140396
Training accuracy is  0.9405799278846154
Epoch  2100 Loss  0.1518077552318573
Training accuracy is  0.9405799278846154
Epoch  2200 Loss  0.1479649692773819
Training accuracy is  0.9405799278846154
Epoch  2300 Loss  0.14816322922706604
Training accuracy is  0.9405799278846154
Epoch  2400 Loss  0.1470182090997696
Training accuracy is  0.9405799278846154
Epoch  2500 Loss  0.14880245923995972
Training accuracy is  0.9405799278846154
Epoch  2600 Loss  0.14857104420661926
Training accuracy is  0.9405799278846154
Epoch  2700 Loss  0.1542617231607437
Training accuracy is  0.9405799278846154
Epoch  2800 Loss  0.16089704632759094
Training accuracy is  0.9405799278846154
Epoch  2900 Loss  0.15431243181228638
Training accuracy is  0.9405799278846154
Epoch  3000 Loss  0.15187621116638184
Training accuracy is  0.9405799278846154
Epoch  3100 Loss  0.14810438454151154
Training accuracy is  0.9405799278846154
Epoch  3200 Loss  0.14676420390605927
Training accuracy is  0.9405799278846154
Epoch  3300 Loss  0.14788338541984558
Training accuracy is  0.9405799278846154
Epoch  3400 Loss  0.14830511808395386
Training accuracy is  0.9405799278846154
Epoch  3500 Loss  0.1500018835067749
Training accuracy is  0.9405799278846154
Epoch  3600 Loss  0.15337088704109192
Training accuracy is  0.9405799278846154
Epoch  3700 Loss  0.14679771661758423
Training accuracy is  0.9405799278846154
Epoch  3800 Loss  0.1500265747308731
Training accuracy is  0.9405799278846154
Epoch  3900 Loss  0.1456853449344635
Training accuracy is  0.9405799278846154
Epoch  4000 Loss  0.14666929841041565
Training accuracy is  0.9405799278846154
Epoch  4100 Loss  0.14597059786319733
Training accuracy is  0.9405799278846154
Epoch  4200 Loss  0.15132766962051392
Training accuracy is  0.9405799278846154
Epoch  4300 Loss  0.1524515002965927
Training accuracy is  0.9405799278846154
Epoch  4400 Loss  0.14893785119056702
Training accuracy is  0.9405799278846154
Epoch  4500 Loss  0.14982107281684875
Training accuracy is  0.9405799278846154
Epoch  4600 Loss  0.1480204164981842
Training accuracy is  0.9405799278846154
Epoch  4700 Loss  0.145859494805336
Training accuracy is  0.9405799278846154
Epoch  4800 Loss  0.15279439091682434
Training accuracy is  0.9405799278846154
Epoch  4900 Loss  0.147756889462471
Training accuracy is  0.9405799278846154
Epoch  5000 Loss  0.14989760518074036
Training accuracy is  0.9405799278846154
Epoch  5100 Loss  0.14922019839286804
Training accuracy is  0.9405799278846154
Epoch  5200 Loss  0.14940524101257324
Training accuracy is  0.9405799278846154
Epoch  5300 Loss  0.1530437022447586
Training accuracy is  0.9405799278846154
Epoch  5400 Loss  0.14672408998012543
Training accuracy is  0.9405799278846154
Epoch  5500 Loss  0.14702707529067993
Training accuracy is  0.9405799278846154
Epoch  5600 Loss  0.14914588630199432
Training accuracy is  0.9405799278846154
Epoch  5700 Loss  0.14579521119594574
Training accuracy is  0.9405799278846154
Epoch  5800 Loss  0.14842286705970764
Training accuracy is  0.9405799278846154
Training accuracy is  0.9405799278846154
```

In [23]:

```python
plt.scatter(X_hexa[:, 0], X_hexa[:, 1], c=correctidx, cmap=plt.cm.Spectral, s = 50)
```

Out[23]:

```
<matplotlib.collections.PathCollection at 0x2534b9afa00>
```



In [24]:

```python
print(net_a.fc1.weight)
print(net_a.fc2.weight)
print(net_a.fc3.weight)
print(net_a.fc4.weight)
print(net_a.fc5.weight)
print(net_a.fc6.weight)
```

```
Parameter containing:
tensor([[-2.8134, -0.7431],
        [-0.1400,  0.9087],
        [-1.2149, -1.5947],
        [ 3.3212,  1.7490],
        [-1.0918, -0.4240],
        [-1.6043,  0.0996],
        [ 1.1494, -1.4486],
        [ 1.4607,  1.6781],
        [-0.2465, -2.2784],
        [ 0.2960, -0.5395],
        [-0.2849,  1.5937],
        [-1.3693,  1.4347],
        [-0.9270, -0.2310],
        [-0.9668,  0.0320],
        [ 0.9443, -0.9196]], requires_grad=True)
Parameter containing:
tensor([[-0.3258,  0.6418, -0.4129,  0.2181, -0.9987, -1.8935, -1.4370,  1.3377,
         -0.2427,  0.8839, -0.0413, -1.9882,  1.5102,  0.4049, -1.7717],
        [-1.4635, -1.9229,  0.3100, -0.7933,  1.8278, -0.5095,  1.5423,  0.2787,
         -1.8755, -1.3205, -0.1961, -0.5270,  1.1449, -0.7772, -1.6659],
        [-1.7951, -0.1731, -1.5055, -1.9075,  0.9809, -0.6134, -3.1843,  0.4802,
          0.0482, -1.0638,  1.1587,  0.1352, -0.2615, -1.3792, -1.0573],
        [ 1.2697, -1.3275,  0.6771,  2.1575, -1.0979,  2.4923,  1.8484, -0.6916,
         -1.9612,  2.3986,  0.2399, -1.1118, -0.0174, -0.5360, -0.3041],
        [ 0.0470,  1.1131, -0.8054,  0.9824,  1.1632,  4.3513,  4.7267,  1.7661,
          1.3207,  1.1203, -0.9051, -1.1512,  0.3297, -2.3044,  2.9651],
        [ 2.5319, -2.6292, -1.9667, -1.8847,  1.6145, -0.6934, -1.8957, -0.1858,
         -1.4301, -2.2553, -1.6495, -0.7355,  0.7596,  0.0195, -2.2610],
        [-1.4753, -1.0549, -1.9096, -0.9419,  0.4213, -0.9752, -0.2578,  1.9877,
         -1.5981,  1.7498,  0.3334, -3.3863,  1.9155, -3.4066, -1.2008],
        [-0.3542, -1.0843, -1.8663, -0.7271, -1.4527, -0.9505, -0.4115,  0.6152,
          1.6589,  1.1464, -0.5448, -2.3581,  1.7517, -0.7161,  1.4946],
        [-1.1066,  0.4330, -0.5088,  1.9684,  1.5765,  0.4874, -1.0581,  0.7233,
          0.3142, -0.4628,  0.5439, -0.6338,  1.2089, -2.7438,  0.1437],
        [-1.1468,  1.0090,  0.8086, -0.9852, -0.1429, -0.4420, -2.1554, -2.5512,
          0.5793, -0.0300, -1.6544,  1.0020,  1.8273, -1.6405, -0.3684],
        [-1.7076, -0.3863, -0.3295,  0.3357,  1.0623,  1.1462, -1.9752,  1.7369,
          0.1855,  0.2186,  1.3389, -1.7475,  1.6436,  0.8827,  1.8383],
        [-0.7676, -0.0261, -1.7648,  0.9281,  0.8114,  0.6687, -0.7881,  0.8301,
          1.3412, -2.0757,  1.9333, -1.6591, -1.4319,  0.4503, -0.4765],
        [ 1.1202,  1.8845, -1.2107, -0.8066, -0.0769, -1.4365, -1.5029, -0.5378,
          1.6329,  1.9238,  0.3860, -1.1791, -1.4675, -0.1419, -0.1366],
        [-1.5370,  2.6788,  1.2921, -1.6760,  1.5715,  2.7262,  3.5003, -0.0576,
          0.0237,  2.4601,  0.3474, -2.0585,  0.4208, -0.1376, -0.8313],
        [-1.3693, -0.7611, -1.6348, -0.0463, -0.5726,  1.8773,  2.9445,  1.6059,
          1.7363,  1.1329,  1.4336, -0.2790,  1.3754, -0.3423,  0.5400]],
       requires_grad=True)
Parameter containing:
tensor([[ 3.2359, -0.0905, -0.4468,  1.9938,  1.0282,  2.8489,  0.9762, -1.0290,
         -1.5485, -0.0735,  0.4664, -0.8913,  1.4123,  1.9741, -0.3941],
        [-1.4043,  0.9785, -0.0709,  1.5547, -1.1050, -1.8058, -0.3811, -1.5182,
          0.5579, -1.7065,  0.1702, -1.1235,  0.2223,  0.4112, -1.4715],
        [-0.0680, -0.8464, -0.1478,  0.7782, -0.5362,  0.6399, -0.9438, -0.3640,
          0.2496, -1.6597, -1.5623,  0.9264,  0.4078,  0.4466, -1.2605],
        [ 0.1972, -1.3418,  1.3590, -0.2077, -2.3503,  1.2375,  0.3936, -1.6564,
         -1.9579, -2.4254,  0.7287,  0.3483, -1.4760, -2.7025, -1.0949],
        [ 2.0152, -0.4608, -1.2156,  2.7244,  0.3939, -0.3206, -0.3418,  0.2456,
         -1.1478,  1.5999, -1.6352, -1.8542,  1.3660,  2.5399,  0.8923],
        [-0.1542, -0.1068, -0.4444, -1.3904, -2.1821, -0.6563,  0.7880, -1.5591,
          1.0106, -1.9360, -1.1852, -1.9320, -1.5427, -3.3587,  0.3202],
        [ 1.2121, -0.2172, -0.7338,  1.6919, -1.4765, -1.4369, -0.8861, -0.9590,
         -0.6418,  0.0557, -0.9657, -0.4240,  1.9753, -1.7481,  0.4161],
        [-1.8801,  1.3535, -0.6703, -0.5143, -3.4616, -0.8251,  1.1939,  0.9372,
          0.9845, -3.5543, -3.1492, -0.3388,  0.6621, -0.6024, -1.4825],
        [-0.4525, -1.0630, -0.4404,  0.6875, -0.3364,  0.7605,  0.8251, -1.2959,
          0.8828, -0.6376,  1.1484,  1.2192,  0.9338, -1.3920, -2.2570],
        [ 0.1047,  1.2767,  0.4696, -1.3360, -0.5766,  0.2454, -1.4822, -0.4569,
          1.1616, -1.2227,  0.5518, -0.4050,  0.4948, -0.7429,  0.7093],
        [-0.9181, -1.7003,  1.2769, -1.8895,  0.3511, -0.5896,  0.4993,  0.9411,
         -1.6152, -0.6023, -1.2564,  0.2942, -0.9454,  0.7312, -0.1085],
        [-3.1789, -1.5332, -0.9711, -3.6548, -2.6599,  0.1068,  0.5229, -0.3496,
         -1.6747, -3.5000,  0.8131,  0.2602, -0.8152, -2.1812, -2.5234],
        [ 1.0432,  1.2890,  1.0153, -0.6750,  0.4359,  1.0250,  0.3861, -0.4970,
         -0.2726, -0.5607, -1.5787,  0.6039, -1.2051, -0.9648, -1.2036],
        [ 0.2307, -1.7628, -0.1367,  1.8407, -0.0752,  1.8724, -1.9176,  1.7148,
         -1.5341,  1.0579,  0.4120,  0.3667, -1.3071,  1.1189, -1.0402],
        [ 1.2722, -1.3582, -1.0965,  2.6477,  0.5344,  1.3185,  0.3593,  1.6011,
          0.1246,  3.3047,  1.9421, -1.1171,  1.9087,  3.9727, -0.1675]],
       requires_grad=True)
Parameter containing:
tensor([[-0.3084, -0.2555,  1.4090, -0.2914,  1.2993, -1.7041,  0.3165, -0.4003,
         -1.6437, -0.5023,  1.3849,  0.4337, -1.6525, -0.1279, -1.9572],
        [ 1.3326, -1.4118, -1.9764, -1.8662, -0.9480,  1.4149, -1.0754, -0.6339,
         -1.0238, -1.1197,  0.0318,  1.3012,  0.8538, -0.9158,  0.0501],
        [ 3.1595, -1.1649,  1.1355,  1.5354,  4.3099,  3.2213,  1.8962,  4.2616,
          0.7889, -0.6594,  0.7322, -0.9548, -0.1740,  1.5406, -1.6967],
        [-0.8139,  1.7330,  0.9884, -0.0477, -1.3935, -2.9852,  1.4943, -1.7995,
         -0.5314,  1.4180, -1.4988, -1.2449, -1.1517, -0.6075, -1.7538],
        [-1.2265,  1.0638, -1.7740, -1.2554, -1.8779, -1.7449,  1.8518,  0.6301,
          0.3939,  1.6983,  0.7379, -1.3247,  1.4253, -2.4026, -2.0566],
        [ 0.4987,  0.8599,  0.9032, -1.0002, -1.8161,  1.3678, -0.3394, -0.6310,
          1.3922, -1.8166,  0.9602, -1.7827, -0.1526, -1.8623,  0.4486],
        [-2.7513, -1.9001, -0.5585,  1.0054, -4.3457, -3.0176, -1.9098, -2.3320,
         -0.2579,  0.2766, -0.2849,  0.3590, -0.1865, -6.4060, -0.9019],
        [ 1.4901, -1.7411, -0.7889, -0.9898, -0.6713,  0.5143, -1.1251, -0.3081,
          0.8246,  0.1546,  0.3667, -0.5386, -0.8691,  0.9019, -1.0503],
        [-0.5459, -0.4445,  1.2736,  1.2178, -1.4111, -0.1779,  1.4821, -0.3175,
          0.2505,  1.3450,  0.0487, -1.7831,  1.1170, -0.4439, -0.8695],
        [-1.2335, -1.0848,  1.4950, -1.0107,  4.2271,  0.0410, -0.1810,  2.7649,
         -1.7104, -0.0848,  2.0460, -0.8654,  0.5333,  3.3684,  0.9763],
        [-0.8158,  0.2619, -0.2558, -1.4338, -0.7110,  0.1343, -0.4776, -1.2352,
         -0.0549, -0.3086,  0.7566,  0.9631,  0.0713, -0.3146, -0.7208],
        [ 1.7358,  0.4544,  0.9534, -0.8852, -1.5849, -1.4977,  0.8849, -1.7430,
          0.9142,  1.8490, -0.9809,  0.3078,  1.0008, -1.9829,  0.1466],
        [-0.3136, -1.1792, -0.4101, -1.2354, -0.9658,  0.7597,  0.1294, -0.6640,
         -1.2870, -1.0167,  0.7576, -0.6427,  1.4408, -1.1853, -1.2400],
        [ 1.5794,  1.2390,  0.0429, -1.2859,  6.2223,  2.8149, -0.5968,  2.3724,
         -1.4787,  1.9665,  1.0820,  0.0770,  1.2358,  5.1054,  2.0142],
        [-1.9289,  0.9036, -1.1466,  1.7901, -1.0724,  0.2929,  0.0919, -0.0362,
         -0.4279, -1.9669, -0.5779, -1.0357, -0.0295, -0.1702,  0.4915]],
       requires_grad=True)
Parameter containing:
tensor([[ 1.6858e+00, -3.8833e-01,  8.5158e-01, -1.0717e+00, -1.0578e+00,
          1.1713e+00,  2.1774e-01,  1.3075e-01, -1.3831e+00, -2.0808e-01,
         -1.5632e-01,  8.0687e-01, -7.1675e-01,  6.2656e-02, -1.7944e+00],
        [-1.9314e+00, -1.0708e+00, -9.4300e-01, -3.6603e-01, -1.1020e+00,
          2.6703e-01, -1.2882e-01, -5.8048e-01, -1.6931e+00,  7.2824e-01,
         -1.8108e+00,  6.2375e-01, -1.2342e+00, -1.7003e+00, -1.5935e+00],
        [-1.4719e+00, -1.2602e+00,  1.0172e+00,  1.5335e+00, -7.7794e-01,
          5.3700e-01,  1.9255e+00, -1.9520e+00, -6.4925e-03,  1.8729e+00,
          7.9388e-01,  3.9182e-02,  1.6912e+00, -8.7431e-01, -1.6671e+00],
        [ 1.1489e+00,  7.6405e-01,  2.1018e-01,  1.5554e-01, -2.3772e-01,
         -2.8262e-01, -7.2732e-01,  4.9800e-01,  1.3750e+00,  2.7830e+00,
          6.2998e-01, -4.6607e-02,  1.4049e+00,  2.1078e+00, -9.0890e-01],
        [ 4.8536e-01,  1.0649e+00,  6.0672e-01, -8.3231e-01,  1.8866e+00,
         -3.0274e-01, -1.0483e+00, -1.4318e+00,  1.6252e+00,  5.9879e-01,
         -1.9415e+00,  2.0483e-01,  1.6692e+00,  4.9444e-01,  1.1035e+00],
        [ 1.8116e+00,  7.0342e-01,  2.5192e-01,  1.6866e+00,  1.0775e+00,
          1.1351e+00, -1.6433e+00, -9.5674e-01,  4.3114e-01,  5.5713e-01,
          1.5788e+00, -3.8501e-01, -8.8615e-02, -1.4738e+00,  1.9773e+00],
        [-9.4988e-01,  5.0892e-01,  1.1964e+00,  7.5331e-01, -5.1099e-01,
         -1.9825e+00, -1.9824e+00, -4.1693e-01, -9.7547e-01, -1.2495e+00,
         -9.2755e-01,  9.6521e-02,  5.4154e-01,  5.1194e-01, -7.2844e-01],
        [-2.6910e-01,  7.6905e-01,  1.7674e-01,  1.6091e+00,  2.9285e-01,
          3.5022e-01, -1.3494e+00, -1.8660e+00,  3.4500e-01, -1.2140e+00,
          7.1069e-01, -1.3457e+00,  1.1026e+00,  5.2942e-01,  1.7005e+00],
        [ 2.3942e-01,  4.9636e-01, -2.5817e+00, -2.5212e+00, -9.1223e-01,
         -5.1677e-01, -6.6720e+00, -1.2070e+01, -1.3714e+00, -2.1280e+00,
         -1.7735e+00, -1.8075e+00, -1.8781e+00, -5.1069e+00,  1.0671e-01],
        [ 1.1796e+00,  1.7455e+00,  1.5204e+00, -6.5927e-01,  1.5148e+00,
         -9.0277e-01, -6.5567e-01, -1.3977e+00, -3.2441e-01, -1.2447e+00,
          4.4508e-01,  1.7832e+00, -1.9325e+00,  8.8776e-01, -1.2101e+00],
        [-3.7011e-01,  1.7540e+00, -1.1809e+00, -3.0051e-01,  1.9662e+00,
          1.2891e+00,  2.9132e+00,  4.4053e+00, -1.1347e+00,  1.2985e+00,
         -8.2771e-01,  1.5601e-01,  1.5577e+00, -5.0376e-01,  2.4956e-01],
        [-1.5910e+00, -2.7797e-01,  3.8454e-01,  1.2339e+00,  1.0455e+00,
          8.2528e-01,  3.4869e-02, -5.9562e-01,  9.6099e-01, -1.2313e+00,
         -1.0003e+00, -1.3296e+00,  1.8612e+00, -2.4059e+00,  1.1077e+00],
        [ 9.5810e-01, -1.3496e+00, -1.5110e+00, -1.2281e+00, -1.1186e+00,
         -1.5013e+00, -3.8608e+00, -2.5684e+00, -3.1742e-01, -2.6852e+00,
          1.8728e+00,  2.5523e-01, -1.3645e+00, -7.0871e-01, -1.1594e+00],
        [-1.5359e+00, -7.9611e-01,  3.5657e-01, -3.9457e-01,  9.5916e-01,
         -4.8731e-01,  1.4660e+00, -4.9668e-01,  8.1310e-01,  1.1119e+00,
          1.6977e+00,  2.1296e-02, -1.9863e+00,  9.8990e-01, -3.9306e-01],
        [ 3.6754e-01, -8.7493e-01, -1.5850e+00,  3.6457e-01,  8.1989e-01,
          8.7183e-01,  2.3773e-01, -1.6348e+00, -6.7257e-01, -6.1742e-01,
         -1.0652e+00, -1.2322e+00,  1.6019e+00, -1.1257e+00, -9.0614e-01]],
       requires_grad=True)
Parameter containing:
tensor([[-0.0478, -1.5857, -0.6739,  1.1525, -1.3065,  0.7879,  0.8809, -1.0783,
         -0.4482, -1.7350, -3.6953, -8.6137, -0.7232, -4.6883,  1.4580],
        [ 1.7139,  1.6940,  1.1659, -1.3100,  0.0623, -0.5980, -1.4996, -0.6447,
          0.6811, -1.2990,  1.9884,  7.5678,  1.6736,  7.9107, -1.7080]],
       requires_grad=True)
```

In [ ]: