Build Model

Import Libraries

In [1]:
```
import pandas as pd
from matplotlib import image
from matplotlib import pyplot as plt
from math import cos, asin, sqrt, pi, atan2, log
import numpy as np
import torch
import torch.utils.data as data
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
import torch.nn.functional as F
```

Loading Prepared Data

1. Loading Data CSV

In [2]:
```
dataset = pd.read_csv("part_1_output/final_prepared_dataset.csv")
datasetdf=pd.DataFrame(dataset)
datasetdf.head()
```

Out[2]:

	startX	startY	velocity_value_1	velocity_value_2	velocity_value_3	velocity_value_4	velocity
0	86.878209	216.274132	1.608839	0.448801	0.265213	0.072890	
1	87.313688	216.385327	0.448801	0.265213	0.072890	0.000000	
2	87.168528	216.162941	0.265213	0.072890	0.000000	0.072421	
3	87.241094	216.162941	0.072890	0.000000	0.072421	2.113376	
4	87.241094	216.162941	0.000000	0.072421	2.113376	0.826399	

5 rows × 62 columns

1. Load Floorplan Graph Dataset

In [3]:
```
graph = pd.read_csv("part_1_output/floor_plan_graph.csv")
graphdf=pd.DataFrame(graph)
graphdf.head()
```

Out[3]:

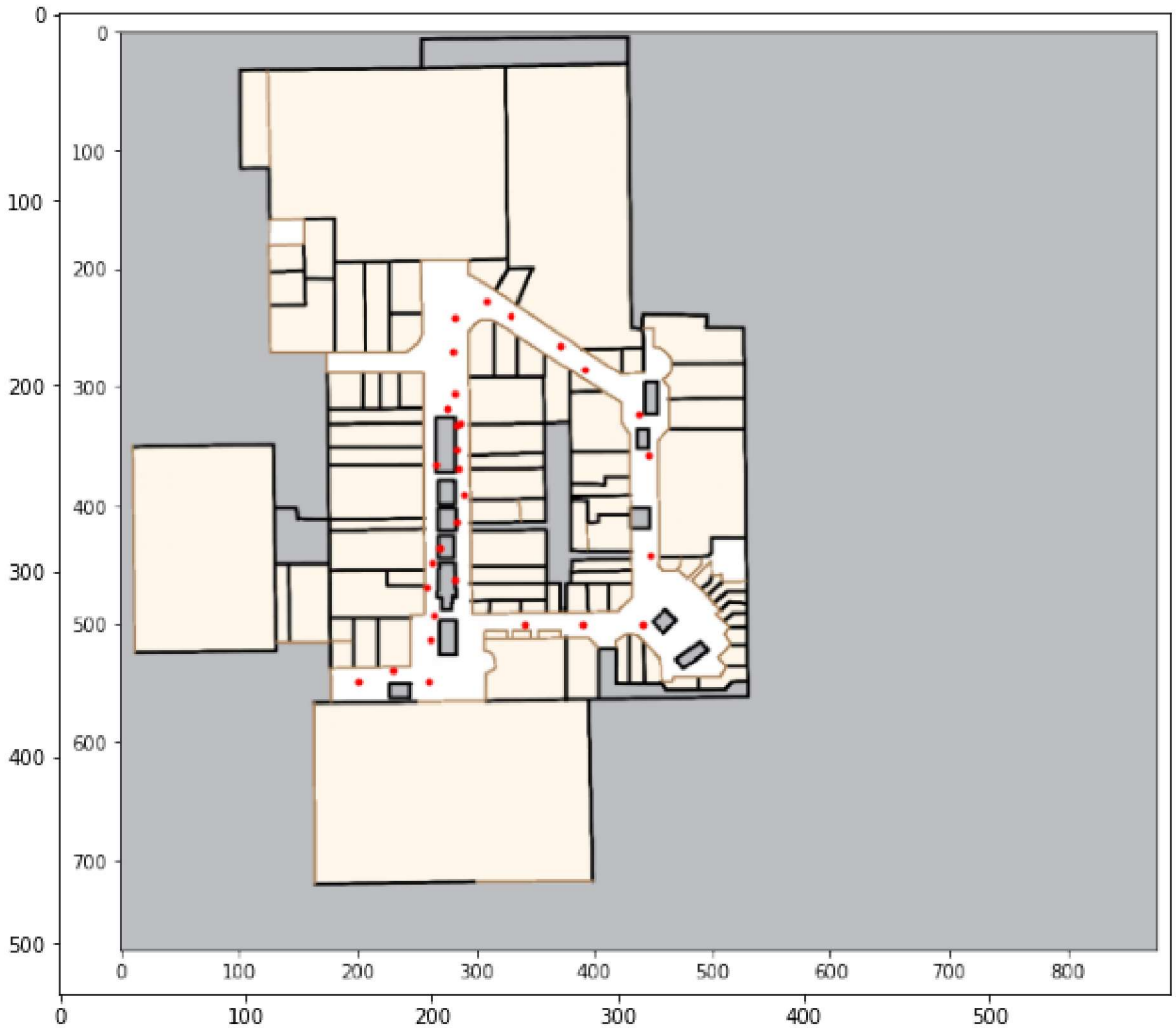
	nodeid	x_dir_pixels	y_dir_pixels	connected_graph_id
0	0	392	285	G1
1	1	283	353	G1
2	2	445	357	G1
3	3	447	442	G1
4	4	280	270	G1

In [4]:
```
graph_dict=graphdf.T.to_dict().values()
graph_dict
```

Out[4]:
```
dict_values([{'nodeid': 0, 'x_dir_pixels': 392, 'y_dir_pixels': 285, 'connected_graph
_id': 'G1'}, {'nodeid': 1, 'x_dir_pixels': 283, 'y_dir_pixels': 353, 'connected_graph
_id': 'G1'}, {'nodeid': 2, 'x_dir_pixels': 445, 'y_dir_pixels': 357, 'connected_graph
_id': 'G1'}, {'nodeid': 3, 'x_dir_pixels': 447, 'y_dir_pixels': 442, 'connected_graph
_id': 'G1'}, {'nodeid': 4, 'x_dir_pixels': 280, 'y_dir_pixels': 270, 'connected_graph
_id': 'G1'}, {'nodeid': 5, 'x_dir_pixels': 308, 'y_dir_pixels': 228, 'connected_graph
_id': 'G1'}, {'nodeid': 6, 'x_dir_pixels': 283, 'y_dir_pixels': 415, 'connected_graph
_id': 'G1'}, {'nodeid': 7, 'x_dir_pixels': 260, 'y_dir_pixels': 550, 'connected_graph
_id': 'G1'}, {'nodeid': 8, 'x_dir_pixels': 265, 'y_dir_pixels': 493, 'connected_graph
_id': 'G1'}, {'nodeid': 9, 'x_dir_pixels': 269, 'y_dir_pixels': 436, 'connected_graph
_id': 'G1'}, {'nodeid': 10, 'x_dir_pixels': 261, 'y_dir_pixels': 513, 'connected_grap
h_id': 'G1'}, {'nodeid': 11, 'x_dir_pixels': 258, 'y_dir_pixels': 470, 'connected_gra
ph_id': 'G1'}, {'nodeid': 12, 'x_dir_pixels': 390, 'y_dir_pixels': 500, 'connected_gr
aph_id': 'G1'}, {'nodeid': 13, 'x_dir_pixels': 341, 'y_dir_pixels': 500, 'connected_g
raph_id': 'G1'}, {'nodeid': 14, 'x_dir_pixels': 263, 'y_dir_pixels': 449, 'connected_
graph_id': 'G1'}, {'nodeid': 15, 'x_dir_pixels': 266, 'y_dir_pixels': 366, 'connected
_graph_id': 'G1'}, {'nodeid': 16, 'x_dir_pixels': 275, 'y_dir_pixels': 319, 'connecte
d_graph_id': 'G1'}, {'nodeid': 17, 'x_dir_pixels': 282, 'y_dir_pixels': 463, 'connect
ed_graph_id': 'G1'}, {'nodeid': 18, 'x_dir_pixels': 282, 'y_dir_pixels': 306, 'connec
ted_graph_id': 'G1'}, {'nodeid': 19, 'x_dir_pixels': 440, 'y_dir_pixels': 500, 'conne
cted_graph_id': 'G1'}, {'nodeid': 20, 'x_dir_pixels': 286, 'y_dir_pixels': 331, 'conn
ected_graph_id': 'G1'}, {'nodeid': 21, 'x_dir_pixels': 283, 'y_dir_pixels': 332, 'con
nected_graph_id': 'G1'}, {'nodeid': 22, 'x_dir_pixels': 285, 'y_dir_pixels': 369, 'co
nnected_graph_id': 'G1'}, {'nodeid': 23, 'x_dir_pixels': 230, 'y_dir_pixels': 540, 'c
onnected_graph_id': 'G1'}, {'nodeid': 24, 'x_dir_pixels': 438, 'y_dir_pixels': 323,
'connected_graph_id': 'G1'}, {'nodeid': 25, 'x_dir_pixels': 289, 'y_dir_pixels': 390,
'connected_graph_id': 'G1'}, {'nodeid': 26, 'x_dir_pixels': 329, 'y_dir_pixels': 240,
'connected_graph_id': 'G1'}, {'nodeid': 27, 'x_dir_pixels': 200, 'y_dir_pixels': 550,
'connected_graph_id': 'G1'}, {'nodeid': 28, 'x_dir_pixels': 371, 'y_dir_pixels': 265,
'connected_graph_id': 'G1'}, {'nodeid': 29, 'x_dir_pixels': 282, 'y_dir_pixels': 242,
'connected_graph_id': 'G1'}])
```

1. Floorplan Graph Image

In [5]:
```
floorplan = image.imread('part_1_output/floor_plan_graph.png')
plt.figure(figsize = (10,10))
plt.imshow(floorplan)
plt.show()
```



Prepare Train Test Set

Split inputs and targets

In [6]:
```
inputsdf=datasetdf.iloc[:,0:42]
inputsdf.head()
```

Out[6]:

	startX	startY	velocity_value_1	velocity_value_2	velocity_value_3	velocity_value_4	velocity
0	86.878209	216.274132	1.608839	0.448801	0.265213	0.072890	
1	87.313688	216.385327	0.448801	0.265213	0.072890	0.000000	
2	87.168528	216.162941	0.265213	0.072890	0.000000	0.072421	
3	87.241094	216.162941	0.072890	0.000000	0.072421	2.113376	
4	87.241094	216.162941	0.000000	0.072421	2.113376	0.826399	

5 rows × 42 columns

In [7]:
```
targetsdf=datasetdf.iloc[:,42:]
targetsdf.head()
```

Out[7]:

	state_1	state_2	state_3	state_4	state_5	state_6	state_7	state_8	state_9	state_10	state_11	s
0	23	23	23	23	23	23	23	23	23	23	23	
1	23	23	23	23	23	23	23	23	23	23	23	
2	23	23	23	23	23	23	23	23	23	23	23	
3	23	23	23	23	23	23	23	23	23	23	23	
4	23	23	23	23	23	23	23	23	23	23	23	

In [8]:
```
inputs=inputsdf.to_numpy()
print(inputs.shape)
inputs
```

Out[8]:
```
(1534, 42)
array([[ 8.68782086e+01,  2.16274132e+02,  1.60883934e+00, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 8.73136883e+01,  2.16385327e+02,  4.48800823e-01, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 8.71685279e+01,  2.16162941e+02,  2.65212732e-01, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       ...,
       [ 8.22330918e+01,  2.14495019e+02,  1.70368594e-01, ...,
        -3.05667663e+00,  3.08490689e+00,  3.14159265e+00],
       [ 8.21604954e+01,  2.14606212e+02,  1.43114696e-01, ...,
         3.08490689e+00,  3.14159265e+00,  3.06877167e+00],
       [ 8.21604954e+01,  2.14717411e+02,  1.06922051e-01, ...,
         3.14159265e+00,  3.06877167e+00,  3.14159265e+00]])
```

In [9]:
```
targets=targetsdf.to_numpy()
print(targets.shape)
targets
```

Out[9]:
```
(1534, 20)
array([[23, 23, 23, ..., 23, 23, 23],
       [23, 23, 23, ..., 23, 23, 23],
       [23, 23, 23, ..., 23, 23, 23],
       ...,
       [27, 27, 27, ..., 27, 27, 27],
       [27, 27, 27, ..., 27, 27, 27],
       [27, 27, 27, ..., 27, 27, 27]], dtype=int64)
```

Convert To Tensor

In [10]:
```
inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)
print(inputs)
print(targets)
```

```
tensor([[ 8.6878e+01,  2.1627e+02,  1.6088e+00,  ...,  0.0000e+00,
          0.0000e+00,  0.0000e+00],
        [ 8.7314e+01,  2.1639e+02,  4.4880e-01,  ...,  0.0000e+00,
          0.0000e+00,  0.0000e+00],
        [ 8.7169e+01,  2.1616e+02,  2.6521e-01,  ...,  0.0000e+00,
          0.0000e+00,  0.0000e+00],
        ...,
        [ 8.2233e+01,  2.1450e+02,  1.7037e-01,  ..., -3.0567e+00,
          3.0849e+00,  3.1416e+00],
        [ 8.2160e+01,  2.1461e+02,  1.4311e-01,  ...,  3.0849e+00,
          3.1416e+00,  3.0688e+00],
        [ 8.2160e+01,  2.1472e+02,  1.0692e-01,  ...,  3.1416e+00,
          3.0688e+00,  3.1416e+00]], dtype=torch.float64)
tensor([[23, 23, 23,  ..., 23, 23, 23],
        [23, 23, 23,  ..., 23, 23, 23],
        [23, 23, 23,  ..., 23, 23, 23],
        ...,
        [27, 27, 27,  ..., 27, 27, 27],
        [27, 27, 27,  ..., 27, 27, 27],
        [27, 27, 27,  ..., 27, 27, 27]])
```

In [11]:
```
ds = TensorDataset(inputs, targets)
ds[0:3]
```

Out[11]:
```
(tensor([[ 8.6878e+01,  2.1627e+02,  1.6088e+00,  4.4880e-01,  2.6521e-01,
           7.2890e-02,  0.0000e+00,  7.2421e-02,  2.1134e+00,  8.2640e-01,
           6.9784e-02,  1.0297e-01,  0.0000e+00,  1.0858e-01,  1.2667e-01,
           4.1078e-01,  4.3377e-01,  4.8899e-01,  5.7149e-01,  2.8022e-01,
           5.1010e-01,  7.6066e-02,  3.8021e-01,  2.5000e-01, -2.1491e+00,
           0.0000e+00,  0.0000e+00,  3.1416e+00,  2.0338e+00, -1.5708e+00,
           3.1416e+00, -1.5708e+00,  0.0000e+00, -9.9256e-01,  0.0000e+00,
           2.9734e-01,  0.0000e+00,  0.0000e+00,  3.6581e-01,  0.0000e+00,
           0.0000e+00,  0.0000e+00],
         [ 8.7314e+01,  2.1639e+02,  4.4880e-01,  2.6521e-01,  7.2890e-02,
           0.0000e+00,  7.2421e-02,  2.1134e+00,  8.2640e-01,  6.9784e-02,
           1.0297e-01,  0.0000e+00,  1.0858e-01,  1.2667e-01,  4.1078e-01,
           4.3377e-01,  4.8899e-01,  5.7149e-01,  2.8022e-01,  5.1010e-01,
           7.6066e-02,  2.1347e-01,  2.5000e-01, -2.1491e+00,  0.0000e+00,
           0.0000e+00,  3.1416e+00,  2.0338e+00, -1.5708e+00,  3.1416e+00,
          -1.5708e+00,  0.0000e+00, -9.9256e-01,  0.0000e+00,  2.9734e-01,
           0.0000e+00,  0.0000e+00,  3.6581e-01,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00],
         [ 8.7169e+01,  2.1616e+02,  2.6521e-01,  7.2890e-02,  0.0000e+00,
           7.2421e-02,  2.1134e+00,  8.2640e-01,  6.9784e-02,  1.0297e-01,
           0.0000e+00,  1.0858e-01,  1.2667e-01,  4.1078e-01,  4.3377e-01,
           4.8899e-01,  5.7149e-01,  2.8022e-01,  5.1010e-01,  7.6066e-02,
           2.1347e-01,  1.4161e-01, -2.1491e+00,  0.0000e+00,  0.0000e+00,
           3.1416e+00,  2.0338e+00, -1.5708e+00,  3.1416e+00, -1.5708e+00,
           0.0000e+00, -9.9256e-01,  0.0000e+00,  2.9734e-01,  0.0000e+00,
           0.0000e+00,  3.6581e-01,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00]], dtype=torch.float64),
 tensor([[23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23,
          23, 23],
         [23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23,
          23, 23],
         [23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23,
          23, 23]]))
```

Split Train Test Set Randomly

In [12]:
```
# Random split
train_set_size = int(len(ds) * 0.8)
test_set_size = len(ds) - train_set_size
train_set, test_set = data.random_split(ds, [train_set_size, test_set_size])
```

Train DataLoader

In [13]:
```
batch_size = 5
train_dl = DataLoader(train_set, batch_size, shuffle=True)
len(train_dl)
```

Out[13]:
```
246
```

Test DataLoader

In [14]:
```
batch_size = 5
test_dl = DataLoader(test_set, batch_size, shuffle=True)
len(test_dl)
```

Out[14]:
```
62
```

Define CRF Functions

Nearest State

In [15]:
```
def calcNearestState(x,y):
    currentNearestStateID=None
    minDistance=float('inf')

    for points in graph_dict:

        stateid=points["nodeid"]
        val1=points['y_dir_pixels']/2.5-y
        val1*=val1
        val2=points['x_dir_pixels']/2.5-x
        val2*=val2
        val3=val1+val2
        distanceToState=val3

        if distanceToState<=minDistance:
            minDistance=distanceToState
            currentNearestStateID=stateid
    return currentNearestStateID
```

Connection Checking Function

In [16]:
```
def isConnected(state_id_t,state_id_t_minus_1):
    state_t=graphdf[graphdf["nodeid"]==state_id_t]
    state_t_minus_1=graphdf[graphdf["nodeid"]==state_id_t_minus_1]
    gid1,gid2=state_t["connected_graph_id"].values[0],state_t_minus_1["connected_graph
    if gid1==gid2:
        return 1
    else:
        return 0
```

Function 1

In [17]:
```
def f1(velocity_value_at_t,std_of_velocity_values,state_id_t,state_id_t_minus_1):
    state_t=graphdf[graphdf["nodeid"]==state_id_t]
    state_t_minus_1=graphdf[graphdf["nodeid"]==state_id_t_minus_1]
    x2,y2=state_t["x_dir_pixels"].values[0],state_t["y_dir_pixels"].values[0]
    x1,y1=state_t_minus_1["x_dir_pixels"].values[0],state_t_minus_1["y_dir_pixels"].va

    euc_dis=sqrt((y2-y1)**2+(x2-x1)**2)

    val= log(1/(std_of_velocity_values*sqrt(2*pi)))-((velocity_value_at_t-euc_dis)**2/
    return val*isConnected(state_id_t,state_id_t_minus_1)
```

Function 2

In [18]:
```
def f2(velocity_angle_at_t,std_of_velocity_angles,state_id_t,state_id_t_minus_1):
    state_t=graphdf[graphdf["nodeid"]==state_id_t]
    state_t_minus_1=graphdf[graphdf["nodeid"]==state_id_t_minus_1]
    x2,y2=state_t["x_dir_pixels"].values[0],state_t["y_dir_pixels"].values[0]
    x1,y1=state_t_minus_1["x_dir_pixels"].values[0],state_t_minus_1["y_dir_pixels"].va

    orientation=atan2(y2-y1,x2-x1)

    val= log(1/(std_of_velocity_angles*sqrt(2*pi)))-((velocity_angle_at_t-orientation)
    return val*isConnected(state_id_t,state_id_t_minus_1)
```

Combine Both

In [19]:
```
def F(velocity_value_at_t,velocity_angle_at_t,std_of_velocity_values,std_of_velocity_a

    return w1*f1(velocity_value_at_t,std_of_velocity_values,state_id_t,state_id_t_minu
```

Calculate std

In [20]:
```
def std(list1):
    return torch.std(list1)
```

Define Viterbi Algorithms

Viterbi Recursion

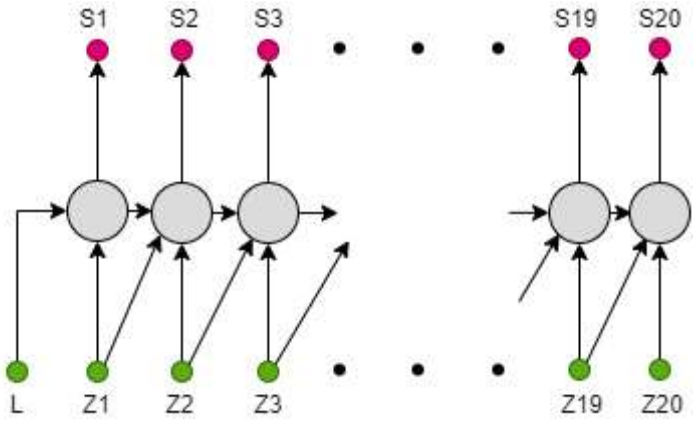
In [21]:
```
def ViterbiRecursion(starting_state,lstate,velocity_value_at_t,velocity_angle_at_t,std

    if i==0:
        return F(velocity_value_at_t,velocity_angle_at_t,std_of_velocity_values,std_of
    else:
        maxval=-float('inf')
        argmax=None
        for j in graphdf["nodeid"]:
            key="delta-i,l="+str(i-1)+","+str(j)
            if key in cache.keys():
                val1 = cache[key]
```

```
            else:
                val1 = ViterbiRecursion(starting_state,j,velocity_value_at_t,velocity_
            val2 = F(velocity_value_at_t,velocity_angle_at_t,std_of_velocity_values,st
            val=val1+val2
            if(val>maxval):
                maxval=val
        cache["delta-i,l="+str(i)+","+str(lstate)]=maxval
        return maxval
```

Define Model

Detailed Model Diagram



Layer 1

In [22]:
```
#ViterbiRecursion(starting_state,lstate,velocity_value_at_t,velocity_angle_at_t,std_of

class Layer1(torch.nn.Module):
    def __init__(self):

        super().__init__()
        self.w1 = torch.nn.Parameter(torch.randn(()))
        self.w2 = torch.nn.Parameter(torch.randn(()))
        self.deltas=[]
        self.gammas=[]


    def forward(self, x):
        startx=x[:,0]
        starty=x[:,1]
        v_values=x[:,2:22]
        v_angles=x[:,22:42]
        startingState=[]
        for iteration in range(len(startx)):
            startingState.append(calcNearestState(startx[iteration],starty[iteration])
        startingState=torch.tensor(startingState)
        std_v_values=std(v_values)
        std_v_angles=std(v_angles)

        for datano in range(5):
            cache={}
```

```
            single_argmax=[]
            for t in range(1,21):
                maxval=-float('inf')
                argmax=None
                for state in range(30):
                    val = ViterbiRecursion(startingState[datano].item(),state,v_values
                    if val>maxval:
                        maxval=val
                        argmax=state
                single_argmax.append(argmax)
            print(single_argmax)
            self.gammas.append(single_argmax)

        return self.gammas
```

In [23]:
```
model = torch.nn.Sequential(
    Layer1()
)
```

In [24]:
```
list(model.parameters())
```

Out[24]:
```
[Parameter containing:
 tensor(-0.4729, requires_grad=True),
 Parameter containing:
 tensor(-2.5907, requires_grad=True)]
```

Training

In [25]:
```
loss_fn = torch.nn.MSELoss()
opt = torch.optim.SGD(model.parameters(), lr=1e-5)
```

In [26]:
```
# Utility function to train the model
def fit(num_epochs, model, loss_fn, opt, train_dl):

    # Repeat for given number of epochs
    for epoch in range(num_epochs):

        # Train with batches of data
        for xb,yb in train_dl:

            # 1. Generate predictions
            pred = model(xb)

            # 2. Calculate loss
            loss = loss_fn(pred, yb)

            # 3. Compute gradients
            loss.backward()

            # 4. Update parameters using gradients
            opt.step()

            # 5. Reset the gradients to zero
            opt.zero_grad()

        # Print the progress
        if (epoch+1) % 10 == 0:
            print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, loss.item(
```

In [ ]:
```
fit(100, model, loss_fn, opt, train_dl)
```

In [27]:
```
for xb,yb in train_dl:
    pred = model(xb)
    print(pred)
    break
```

```
[27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5]
[5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27]
[5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27]
[27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5]
[5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27]
[[27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5], [5, 27, 5, 2
7, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27], [5, 27, 5, 27, 5, 27, 5,
27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27], [27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 2
7, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5], [5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27, 5, 27,
5, 27, 5, 27, 5, 27]]
```

In [ ]: