Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell →Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [192]:
```
NAME = "Nopphawan Nurnuansuwan"
ID = "122410"
```

AT82.03 Machine Learning Aug 2021: Final Examination

Happy Wednesday! This is the midterm for Machine Learning in the August 2021 semester.

This exam is 2.5 hours long. Once the exam starts, you will have exactly 2.5 hours to finish your work and upload your notebook to Google Classroom.

Please fill in this notebook with your code and short answers. Be sure to put all of your code in the cells marked with

```
# YOUR CODE GOES HERE
```

and please put your answers to the short answer questions exactly where you see the remark

You answer goes here.

Be complete and precise in your answers! Be sure to answer the question that's being asked. Don't dump random information in the hope that it'll give you partial credit. I give generous partial credit, but I will deduct points for answers that are not on point.

Also beware that if I discover any cheating, I will give you a 0 for the entire exam, or worse, and you will likely fail the class. Just don't do it!

OK, that's all for the advice. Relax, take a deep breath, and good luck!

Question 1: Data exploration plots (10 points)

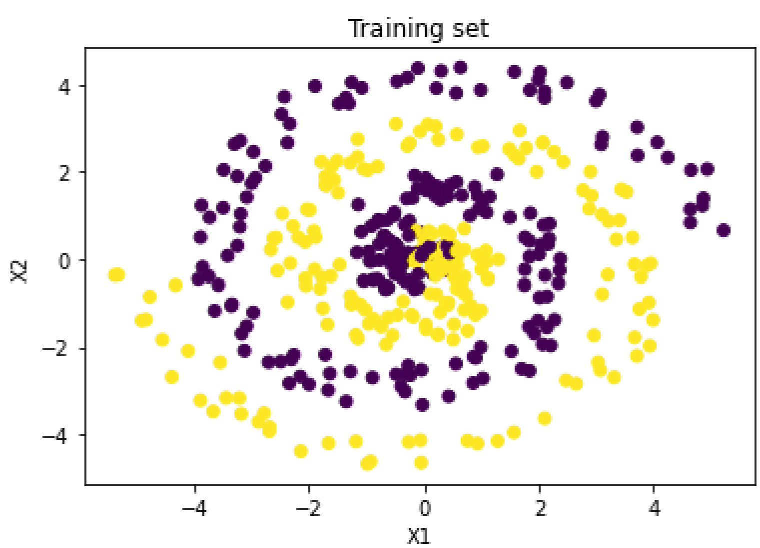
Consider the dataset below consisting of two input variables/featues and a single target variable:

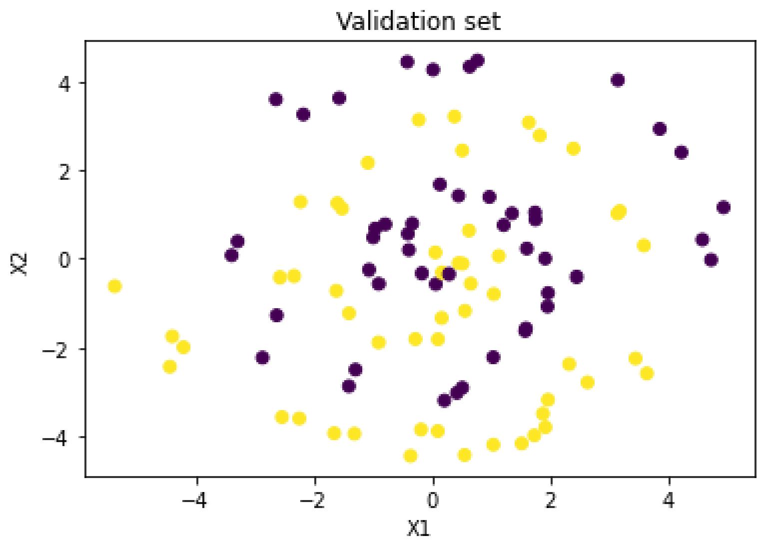
In [193]:
```
data = [[-0.04854449138505812, 0.212806014853555, 1], [0.054809748314162826, 0.1394
```

In the cell below, write code to split the data into training and validation sets in an 80%/20% ratio, then make two scatterplots, each showing the training set or validation set, with the two classes in different colors.

In [194]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from IPython.display import clear_output
import matplotlib.pyplot as plt


data_arr = np.array(data)
X = data_arr[:,:-1]
y = data_arr[:,-1]

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=20/100)

plt.scatter(X_train[:,0], X_train[:,1], c=y_train)
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Training set")
plt.show()

plt.scatter(X_val[:,0], X_val[:,1], c=y_val)
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Validation set")
plt.show()
```





Question 2: RBF SVM model (30 points)

Use the RBF SVM code (Gaussian kernel) we developed in lab using cvxopt to find a good model that performs well on the validation set. Visualize the result, showing the validation set with the -1/+1 regions of the input space shown in different colors.

You may normalize the data if you find it necessary.

In [195]:

```
import cvxopt

def cvxopt_solve_qp(Q, c, A=None, B=None, E=None, d=None):
    Q_new = (Q+Q.T)/2
    args = [cvxopt.matrix(Q_new), cvxopt.matrix(c)]
    if A is not None:
        args.extend([cvxopt.matrix(A), cvxopt.matrix(B)])
        if E is not None:
            args.extend([cvxopt.matrix(E, (1, c.shape[0]), 'd'), cvxopt.matrix(d)])
    sol = cvxopt.solvers.qp(*args)
    if sol is not None and 'optimal' not in sol['status']:
        return None
    x = sol['x']
    return np.array(x).reshape(-1)

def gauss_kernel(X):
    sigma = 0.2
    m = X.shape[0];
    K = np.matrix(np.zeros([m,m]));
    for i in range(0,m):
        for j in range(0,m):
            K[i,j] = (X[i,:] - X[j,:]).reshape(1,-1) @ (X[i,:] - X[j,:]).reshape(-1,1)
    K = np.exp(-K/(2*sigma*sigma))
    return K;

def get_alpha_star(X, y, K, C):
    m, n = X.shape
    diag_y = np.diag(np.array(y).reshape(-1))
    Q = diag_y*K*diag_y
    c = -np.ones((m,1))
    A = np.vstack((-np.eye(m), np.eye(m)))
    B = np.hstack((np.zeros(m), np.ones(m) * C))
    E = y.reshape(1,-1)
    d = np.zeros(1)

    alpha_star = (cvxopt_solve_qp(Q, c, A, B, E, d))
    return alpha_star

def get_wb(X, y, alpha, K):
    # Find the support vectors
    S = alpha > 1e-2
    XS = X[S]
    yS = y[S]
    alphaS = alpha[S]
    alphaSyS = np.multiply(yS.T, alphaS)
    w = np.array(alphaSyS@XS).reshape(-1)
    # Find b
    KS = K[S,:][:,S]
    NS = len(alphaS)
    # Normalize w,b
    scalef = np.sqrt(np.sum(w**2))
    w = w/scalef
    b = (np.sum(yS) - np.sum(alphaSyS*KS))/NS/scalef
    return w,b

def predict(x, X, y, alpha):
```

```
    s = []
    sigma = 0.2
    for j in range(x.shape[0]):
        ss = 0
        for i in range(X.shape[0]):
            ss += alpha[i]*y[i]*np.exp((-(X[i]-x[j])@(X[i]-x[j]))/(2*sigma*sigma))
        s.append(ss)
    s = np.array(s)
    s[s >= 0] = 1
    s[s < 0] = -1
    return s
```

In [196]:

```
C = 3

K = gauss_kernel(X_train)
alpha_star = get_alpha_star(X_train, y_train, K, C)
w,b = get_wb(X_train, y_train, alpha_star, K)
y_pred = predict(X_val, X_val, y_val, alpha_star)
acc = (np.sum(y_val == y_pred)/y_val.size)
```

```
     pcost       dcost       gap    pres   dres
 0: -5.2187e+01 -2.6165e+03  3e+03  1e-14  2e-15
 1: -1.2743e+02 -4.7140e+02  3e+02  1e-14  9e-16
 2: -1.5812e+02 -2.1711e+02  6e+01  5e-15  5e-16
 3: -1.6236e+02 -1.7215e+02  1e+01  4e-15  4e-16
 4: -1.6329e+02 -1.6510e+02  2e+00  7e-15  5e-16
 5: -1.6352e+02 -1.6383e+02  3e-01  2e-15  4e-16
 6: -1.6357e+02 -1.6359e+02  2e-02  1e-14  5e-16
 7: -1.6358e+02 -1.6358e+02  4e-04  4e-15  5e-16
 8: -1.6358e+02 -1.6358e+02  8e-06  1e-14  4e-16
Optimal solution found.
```

In [197]:

```
print("Accuracy:", acc)
```

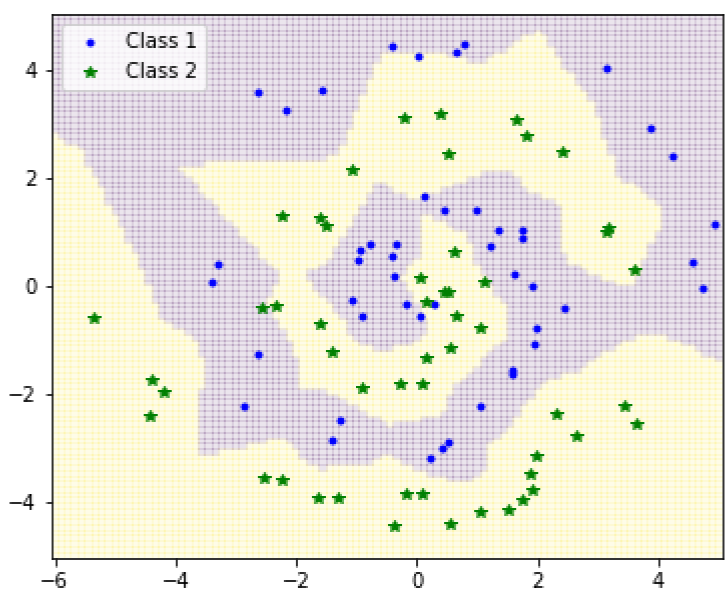
```
Accuracy: 0.96
```

In [198]:

```
def plot_data(X1, X2):
    ax = plt.axes()
    plt.title('Sample data for classification problem')
    plt.grid(axis='both', alpha=.25)
    plt.plot(X1[:,0],X1[:,1],'b.', label = 'Class 1')
    plt.plot(X2[:,0],X2[:,1],'g*', label = 'Class 2')
    #plt.legend(loc=2)
    ax.set_aspect('equal', 'datalim')
    return ax
```

In [199]:

```
X_calss1 = X_val[y_val.reshape(-1)==-1]
X_calss2 = X_val[y_val.reshape(-1)==1]

res = 100

plt.figure(figsize=(6, 5))

x_lim = [np.floor(np.min(X_val[:,0])), np.ceil(np.max(X_val[:,0]))]
y_lim = [np.floor(np.min(X_val[:,1])), np.ceil(np.max(X_val[:,1]))]
x_series = np.linspace(x_lim[0], x_lim[1], res)
y_series = np.linspace(y_lim[0], y_lim[1], res)

x_mesh, y_mesh = np.meshgrid(x_series, y_series)

x_mesh = x_mesh.reshape(-1, 1)
y_mesh = y_mesh.reshape(-1, 1)

mesh = np.append(x_mesh, y_mesh, axis=1)
y_pred = predict(mesh, X_val, y_val, alpha_star)

x_mesh = x_mesh.reshape(res, res)
y_mesh = y_mesh.reshape(res, res)
y_pred = y_pred.reshape(res, res)

plt.plot(X_calss1[:,0],X_calss1[:,1],'b.', label = 'Class 1')
plt.plot(X_calss2[:,0],X_calss2[:,1],'g*', label = 'Class 2')
plt.pcolormesh(x_mesh, y_mesh, y_pred, cmap='viridis', shading='auto', alpha=0.1)
plt.legend()

plt.show()
```



Question 3: Neural network model (30 points)

In our deep learning lab, you already found neural networks in the Tensorflow Playground capable of learning this dataset. Using the code developed in lab, find a good multilayer neural network model that performs well on the validation set. Visualize the result, showing the validation set with the -1/+1 regions of the input space shown in different colors.

You may normalize the data if you find it necessary.

In [200]:

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [201]:

```
# hyperparameters
epochs = 200
batch_size = 8
learning_rate = 0.003

class Network(nn.Module):

    def __init__(self):
        super(Network, self).__init__()
        self.l1 = nn.Linear(2, 16)
        self.l2 = nn.Linear(16, 4)
        self.l3 = nn.Linear(4, 2)
        self.tanh = nn.Tanh()

    def forward(self, x):
        x = self.l1(x)
        x = self.tanh(x)
        x = self.l2(x)
        x = self.tanh(x)
        x = self.l3(x)
        return F.log_softmax(x)
```

In [202]:
```python
net = Network()
print(net)
```

```
Network(
  (l1): Linear(in_features=2, out_features=16, bias=True)
  (l2): Linear(in_features=16, out_features=4, bias=True)
  (l3): Linear(in_features=4, out_features=2, bias=True)
  (tanh): Tanh()
)
```

In [203]:
```python
optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
loss_func = nn.CrossEntropyLoss()
```

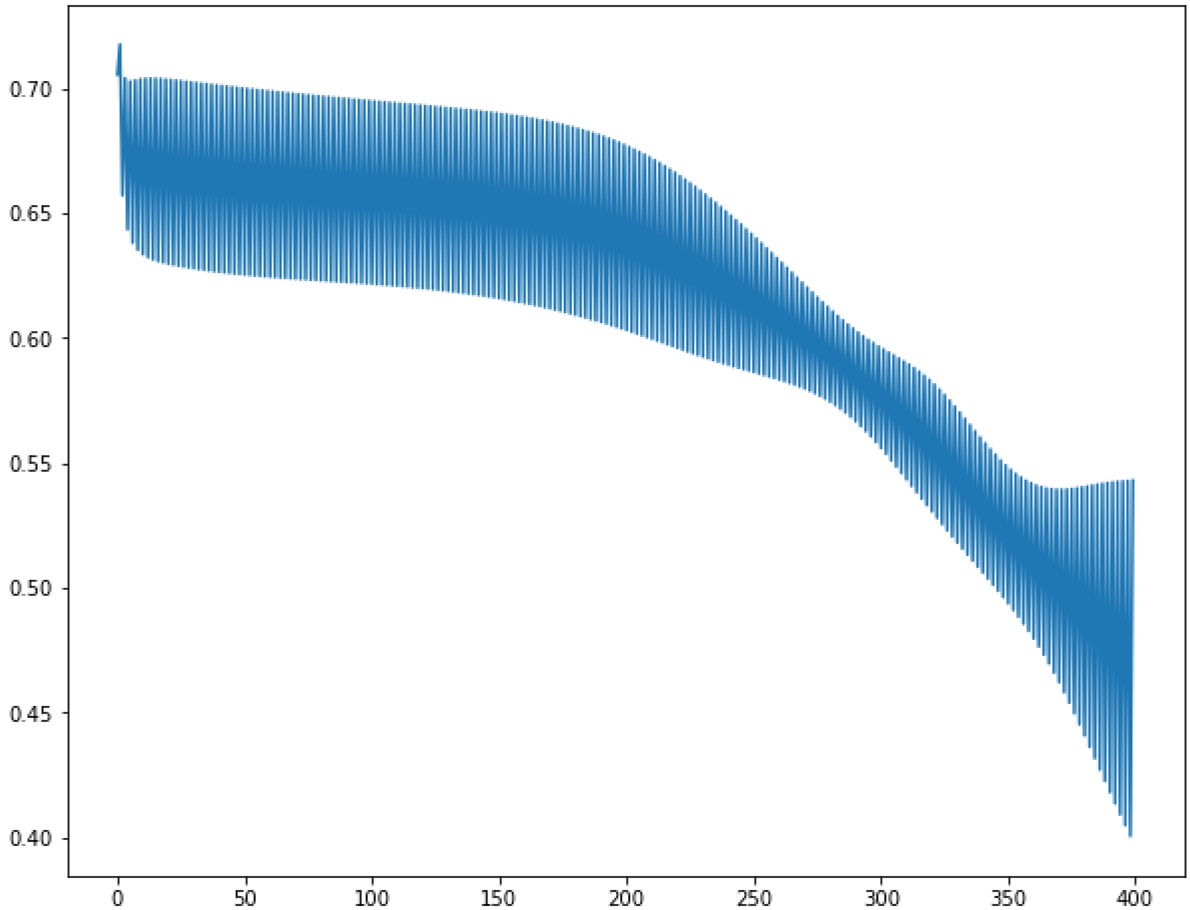
In [204]:
```python
loss_log = []

X_train_tensor = (torch.tensor(X_train)).float()
y_train_tensor = (torch.tensor(y_train)).float()
for i in range(len(y_train)):
    if y_train[i]==-1: y_train_tensor[i]=0

for e in range(epochs):
    for i in range(0, X_train_tensor.shape[0], batch_size):
        x_mini = X_train_tensor[i:i + batch_size]
        y_mini = y_train_tensor[i:i + batch_size]

        # Not necessary in recent PyTorch versions
        # x_var = Variable(x_mini)
        # y_var = Variable(y_mini)

        optimizer.zero_grad()
        net_out = net(x_mini)

        loss = loss_func(net_out, y_mini.long())
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            loss_log.append(loss.item())

    clear_output(wait=True)
    print('Epoch: {} - Loss: {:.6f}'.format(e, loss.item()))
```

```
Epoch: 199 - Loss: 0.487605
```

In [205]:

```python
plt.figure(figsize=(10,8))
plt.plot(loss_log)
```

Out[205]: [<matplotlib.lines.Line2D at 0x1671dea9a90>]



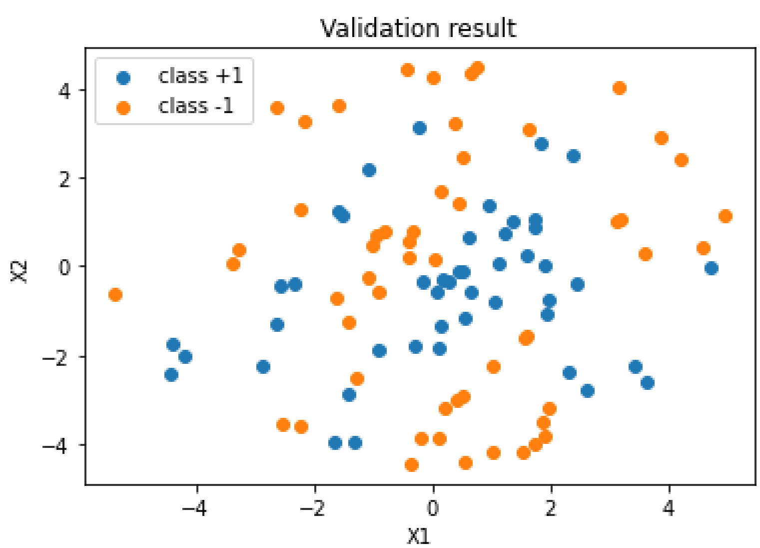
In [206]:

```python
test = torch.FloatTensor((X_val).tolist())
test_var = Variable(test)

net_out = net(test_var)

y_test_pred = torch.max(net_out.data, 1)[1].numpy()

for i in range(len(y_test_pred)):
    if y_test_pred[i]==0: y_test_pred[i]=-1

print(y_test_pred)
```

```
[-1  1  1  1  1 -1  1 -1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1  1  1 -1  1
 -1  1  1 -1 -1  1 -1 -1 -1 -1  1 -1  1 -1  1  1 -1  1 -1 -1 -1 -1  1  1
 -1 -1  1 -1  1  1 -1  1  1 -1 -1 -1 -1 -1  1  1  1 -1  1  1 -1  1  1 -1
 -1  1 -1  1  1 -1  1  1 -1 -1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1
  1  1  1  1]
```

In [207]:

```python
X_val_calss1 = X_val[y_test_pred==1]
X_val_calss2 = X_val[y_test_pred==-1]

plt.scatter(X_val_calss1[:,0], X_val_calss1[:,1], label='class +1')
plt.scatter(X_val_calss2[:,0], X_val_calss2[:,1], label='class -1')
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Validation result")
plt.legend()
plt.show()
```



Question 4: Robot maze RL (30 points)

In class, we developed policies using Q learning and SARSA for multiple grid worlds.

Construct a maze in a 10x10 grid with a starting location in the lower left and ending location in the upper right.

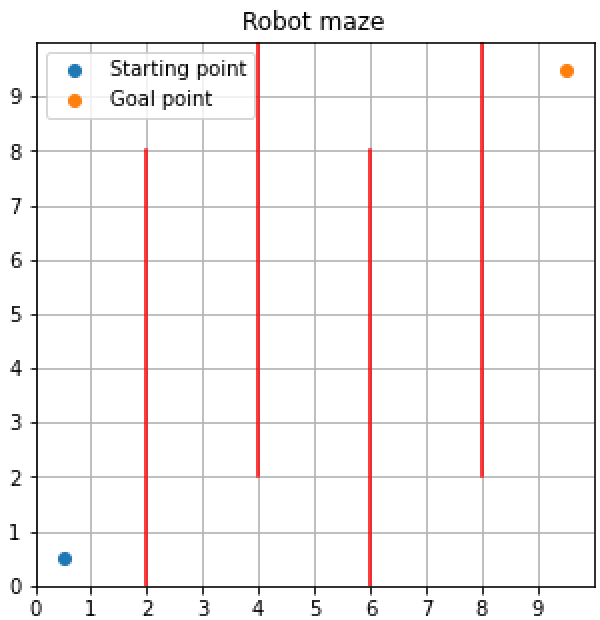
Train your Q learning or SARSA agent to find the goal. Show the resulting policy in a grid representation similar to what we developed in class.

In [208]:

```
gamma = 1
x_env, y_env = 10, 10
s_initial = (0, 0)
s_terminal = (9, 9)
# (x, y)
wall = [[(2,0),(2,8)],
        [(4,2),(4,10)],
        [(6,0),(6,8)],
        [(8,2),(8,10)]]
wall_arr = np.array(wall)
# actions
action_names = ['U', 'D', 'R', 'L']
n_act = len(action_names)
def actions(a):
    move = [[0,1],[0,-1],[1,0],[-1,0]]
    return move[a]

def env(s, a):
    a_move = (actions(a))
    s_new = np.array(s)+np.array(a_move)

    s_new[0] = max(min(s_new[0],9),0)
    s_new[1] = max(min(s_new[1],9),0)

    if hit_wall(s,s_new):
        return s
    else:
        return tuple(s_new)

def hit_wall(s,s_new):
    for w in wall:
        if (w[0][0]==s[0]) and (w[0][0]==s_new[0]+1):
            if s[1] in range(w[0][1], w[1][1]):
                # print(a, s,s_new, range(w[0][1], w[1][1]))
                return True
        if (w[0][0]==s_new[0]) and (w[0][0]==s[0]+1):
            if s[1] in range(w[0][1], w[1][1]):
                # print(a, s,s_new, range(w[0][1], w[1][1]))
                return True
    return False

def reward(s):
    if s == s_terminal:
        return 0
    else:
        return -1
```

In [209]:

```
wall_arr = np.array(wall)

plt.figure(figsize=(5, 5))
plt.title('Robot maze')
plt.xlim(0, x_env)
plt.ylim(0, y_env)
plt.xticks(np.arange(0, x_env, 1))
plt.yticks(np.arange(0, y_env, 1))
for i in range(wall_arr.shape[0]):
    plt.plot((wall_arr[i].T)[0],(wall_arr[i].T)[1], 'r')

plt.scatter(s_initial[0]+0.5,s_initial[1]+0.5, label="Starting point")
plt.scatter(s_terminal[0]+0.5,s_terminal[1]+0.5, label="Goal point")
plt.legend()

plt.grid()
```



In [210]:

```
eps = 0.1
alp = 0.5

def epsilon_greedy(Q, s, eps):
    if np.random.uniform() < eps:
        return np.random.randint(n_act)
    else:
        return np.argmax(Q[s[0], s[1]])
```

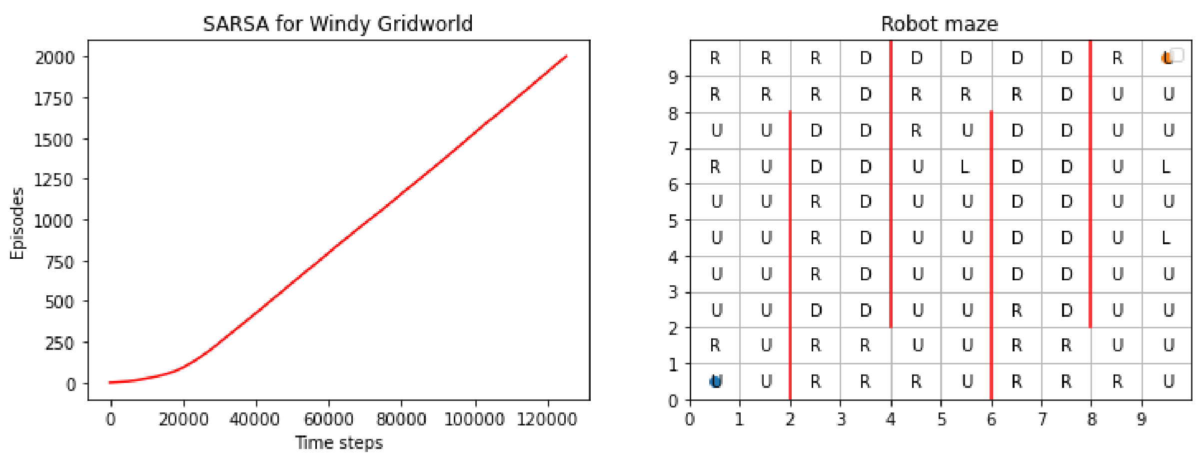
In [211]:

```
n_episodes = 2000
steps_list = [0]
steps = 0
# Difine Q table
Q_table = np.zeros((x_env, y_env, n_act))

for episode in range(n_episodes):
    s = s_initial
    a = epsilon_greedy(Q_table, s, eps)
    # steps = 0
    while s != s_terminal:
        s_next = env(s, a)
        R = reward(s_next)
        a_next = epsilon_greedy(Q_table, s_next, eps)
        Q = Q_table[s[0], s[1], a]
        Q_next = Q_table[s_next[0], s_next[1], a_next]
        Q_table[s[0], s[1], a] = Q + alp*(R + gamma*Q_next - Q)
        s = s_next
        a = a_next
        steps += 1
    steps_list.append(steps)

```

In [212]:

```python
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.title('SARSA for Windy Gridworld')
plt.plot(steps_list, range(n_episodes+1), 'r-')
plt.xlabel('Time steps')
plt.ylabel('Episodes')
# plt.ylim(0, 170)

ax = plt.subplot(1, 2, 2)
plt.title('Robot maze')
plt.xlim(0, x_env)
plt.ylim(0, y_env)
plt.xticks(np.arange(0, x_env, 1))
plt.yticks(np.arange(0, y_env, 1))
for i in range(wall_arr.shape[0]):
    plt.plot((wall_arr[i].T)[0],(wall_arr[i].T)[1], 'r')
plt.scatter(s_initial[0]+0.5,s_initial[1]+0.5)
plt.scatter(s_terminal[0]+0.5,s_terminal[1]+0.5)
plt.legend()
plt.grid()

for y in range(y_env):
    for x in range(x_env):
        s = (x, y)
        a = np.argmax(Q_table[s])
        plt.text(x+0.4, y+0.35, action_names[a])
plt.show()
```

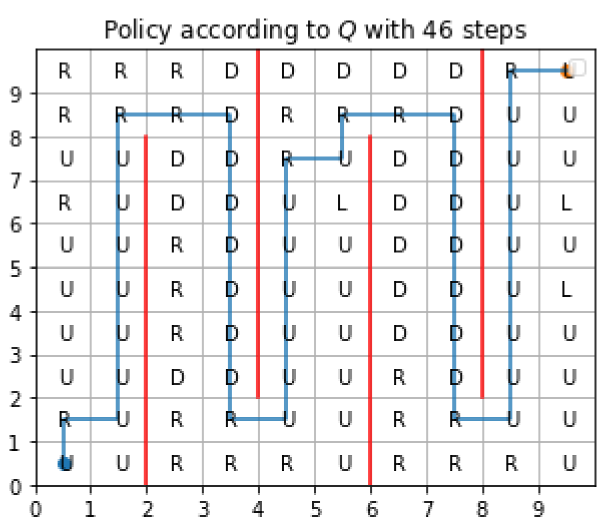
No handles with labels found to put in legend.



In [213]:

```python
state_list = []
s = s_initial
a = epsilon_greedy(Q_table, s, 0.001)
state_list.append(s)
while s != s_terminal:
    s_next = env(s, a)
    s = s_next
    a = epsilon_greedy(Q_table, s, 0.001)
    state_list.append(s)

state_list_array = np.transpose(np.array(state_list))+0.5

plt.figure(figsize=(5, 4))

plt.xlim(0, x_env)
plt.ylim(0, y_env)
plt.xticks(np.arange(0, x_env, 1))
plt.yticks(np.arange(0, y_env, 1))
for i in range(wall_arr.shape[0]):
    plt.plot((wall_arr[i].T)[0],(wall_arr[i].T)[1], 'r')
plt.scatter(s_initial[0]+0.5,s_initial[1]+0.5)
plt.scatter(s_terminal[0]+0.5,s_terminal[1]+0.5)
plt.legend()
plt.grid()

for y in range(y_env):
    for x in range(x_env):
        s = (x, y)
        a = np.argmax(Q_table[s])
        plt.text(x+0.4, y+0.35, action_names[a])

plt.plot(state_list_array[0], state_list_array[1])
plt.title('Policy according to $Q$ with '+str(len(state_list)-1)+' steps')
plt.show()
```

No handles with labels found to put in legend.



In [ ]:

```

```