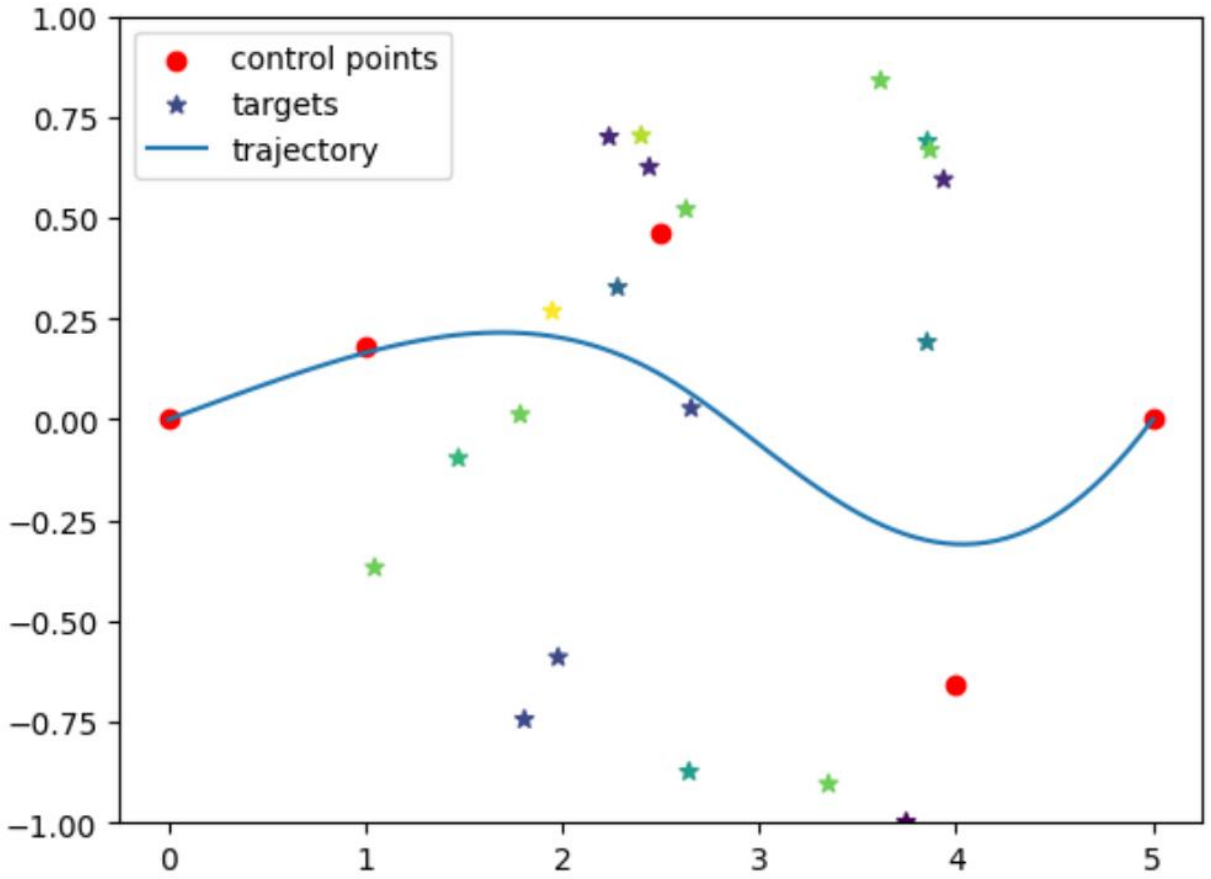
# Project3 Trajectory Planning

## Hints

by 11911012 Lishuang Wang




1. Given n targets, each target has its location and features.

2. A target can be distinguish to a class. Each class has its score.

3. We can control a curve by setting the location of three control points.

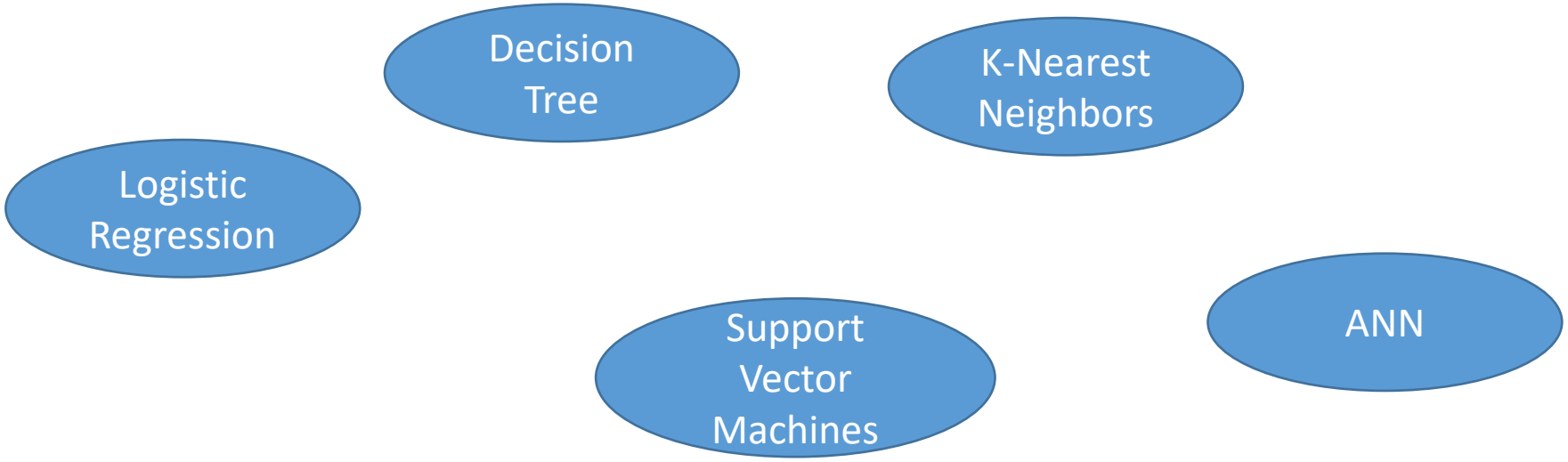
4. When the curve closed to a target, it means we can get the score of the target's class.

5. Our goal is to maximum the total score we obtained.

We can split this work into two parts.

1. Train a classifier to distinguish the class of a target. (the input parameters is targets' features and classes' scores, we need to obtain the targets' scores)

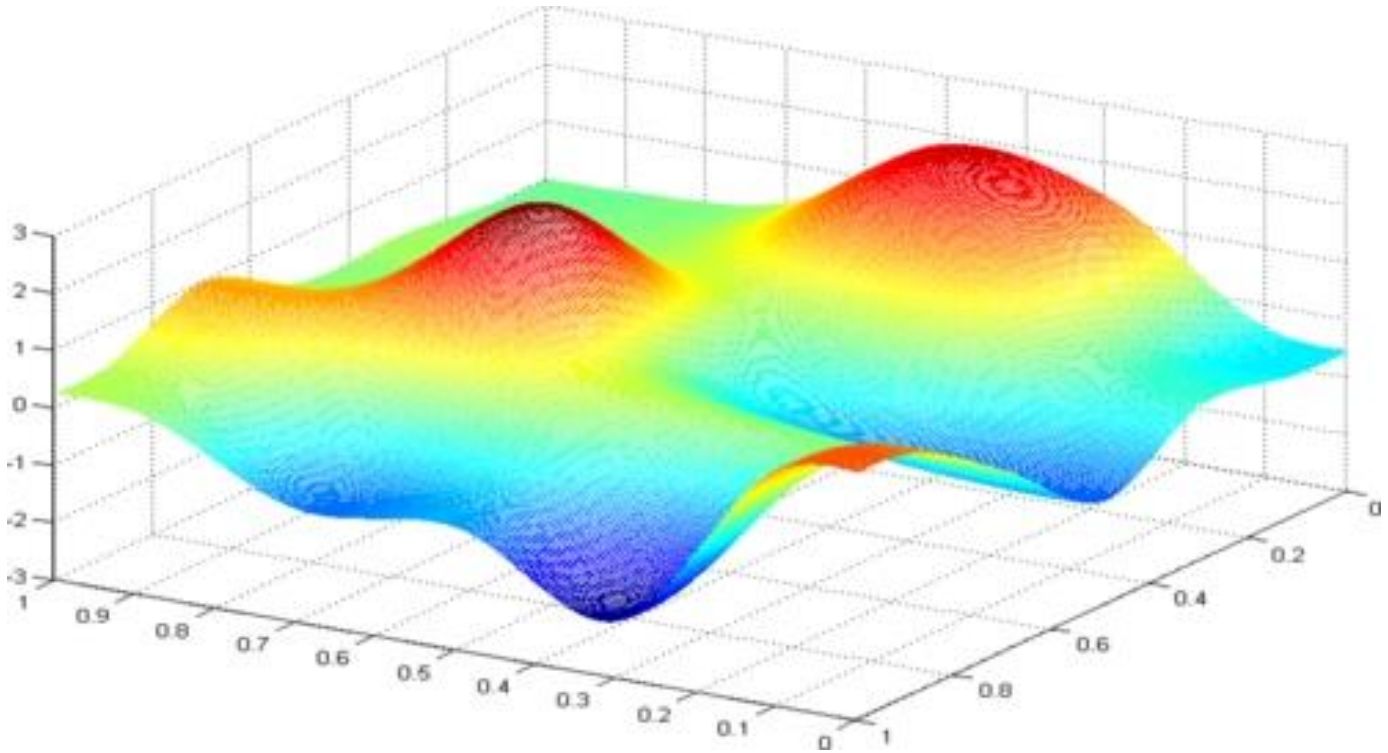
2. Once we obtain such classifier, we can evaluate our control points by ourselves.



Now, we can treat this problem as a optimization problem.
We need to Maximize f(x):
f means the total score we get, x means the coordinateds of the control points.

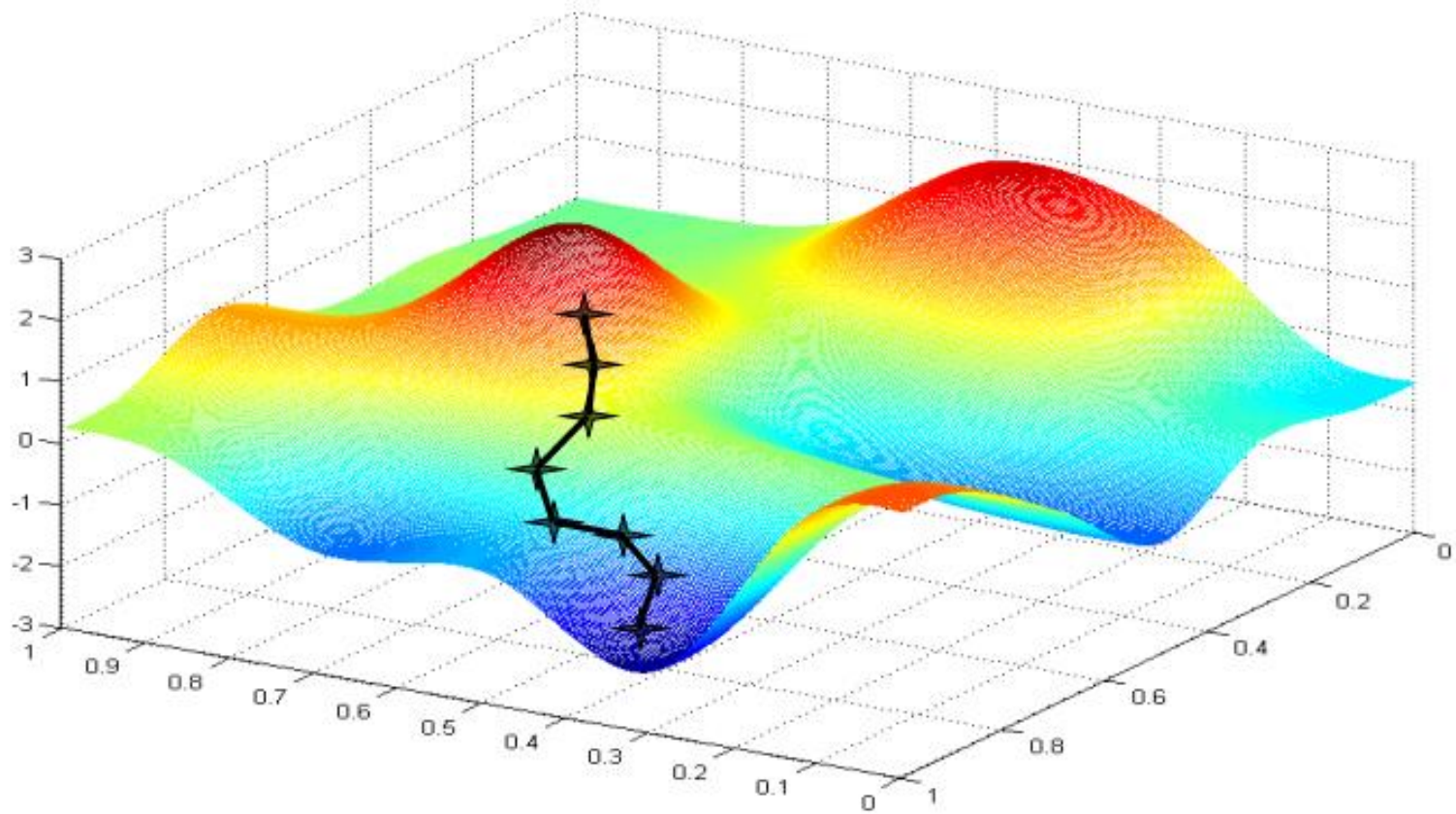
We have learn some black box optimization algorithm before, like genetic algorithms. But, in this work, we have only 0.3s for each question.

Out time is too tight to genarate a solution population run the algorithms for iterations.



However, we still have some methods to optimize our solution, that is gradient descent.

We can obtain the gradient on our current solution, and change the solution along with its gradient to optimize the solution.



# How to obtain the gradient?

We can take advantage of the autograd funtion of pytorch.

For example,

```python
import torch

x = torch.as_tensor(1.)
y = torch.as_tensor(2.)

x.requires_grad = True
y.requires_grad = True

z = x * x + y * y

z.backward()
print(f"x.grad: {x.grad}, y.grad: {y.grad}")
```

```
x.grad: 2.0, y.grad: 4.0
```

We can examine the gradient by hand :)

In our project, we can also get the gradient by autograd.

```
ctps_inter = torch.rand((N_CTPS - 2, 2)) * torch.tensor([N_CTPS - 2, 2.]) + torch.tensor([1., -1.])
ctps_inter.requires_grad = True
score = evaluate(compute_traj(ctps_inter), target_pos, class_scores[target_cls], RADIUS)
score.backward()
print(f"solution's grad: \n{ctps_inter.grad}")
```

However, when we want to use above code to obtain the gradient of the solution, we get such exceptions:

```
Traceback (most recent call last):
  File "C:\Users\cvgcv\Desktop\AIProject3\eval.py", line 28, in <module>
    ctps_inter = agent.get_action(target_pos, target_features, class_scores)
  File "C:\Users\cvgcv\Desktop\AIProject3\agent.py", line 47, in get_action
    score.backward()
  File "C:\Users\cvgcv\anaconda3\envs\gym_env\lib\site-packages\torch\_tensor.py", line 487, in backward
    torch.autograd.backward(
  File "C:\Users\cvgcv\anaconda3\envs\gym_env\lib\site-packages\torch\autograd\__init__.py", line 197, in backward
    Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn
```

# Why?

notice the original given function evaluate

```
def evaluate(traj: torch.Tensor, target_pos: torch.Tensor, target_scores: torch.Tensor,
             radius: float, ) -> torch.Tensor:
    cdist = torch.cdist(target_pos, traj)  # see https://pytorch.org/docs/stable/generated/torch.cdist.html
    d = cdist.min(-1).values
    hit = (d < radius)
    value = torch.sum(hit * target_scores, dim=-1)
    return value
```

When we modified above code to

```
def evaluate(traj: torch.Tensor, target_pos: torch.Tensor, target_scores: torch.Tensor,
             radius: float, ) -> torch.Tensor:
    cdist = torch.cdist(target_pos, traj)  # see https://pytorch.org/docs/stable/generated/torch.cdist.html
    d = cdist.min(-1).values
    hit = (d < radius)
    print(hit)
    hit.requires_grad = True
    value = torch.sum(hit * target_scores, dim=-1)
    return value
```

We can get

```
Traceback (most recent call last):
  File "C:\Users\cvgcv\Desktop\AIProject3\eval.py", line 28, in <module>
    ctps_inter = agent.get_action(target_pos, target_features, class_scores)
  File "C:\Users\cvgcv\Desktop\AIProject3\agent.py", line 46, in get_action
    score = evaluate(compute_traj(ctps_inter), target_pos, class_scores[target_cls], RADIUS)
  File "C:\Users\cvgcv\Desktop\AIProject3\src.py", line 54, in evaluate
    hit.requires_grad = True
RuntimeError: only Tensors of floating point and complex dtype can require gradients
tensor([False, False,  True, False, False, False,  True, False,  True, False,
        False, False, False,  True, False,  True,  True, False, False,  True,
        False,  True, False,  True, False, False, False,  True, False,  True,
        False, False, False,  True,  True, False, False,  True, False,  True])
```

So, we need to change the evaluation function to avoid boolean opearants.

```python
def evaluate_modified(traj: torch.Tensor, target_pos: torch.Tensor, target_scores: torch.Tensor, radius: float,
                      ) -> torch.Tensor:
    cdist = torch.cdist(target_pos, traj)  # see https://pytorch.org/docs/stable/generated/torch.cdist.html
    d = cdist.min(-1).values
    hit = (d < radius)
    d[hit] = 1
    d[~hit] = 0
    value = torch.sum(d * target_scores, dim=-1)
    return value
```

```python
ctps_inter = torch.rand((N_CTPS - 2, 2)) * torch.tensor([N_CTPS - 2, 2.]) + torch.tensor([1., -1.])
ctps_inter.requires_grad = True
# score = evaluate(compute_traj(ctps_inter), target_pos, class_scores[target_cls], RADIUS)
score = evaluate_modified(compute_traj(ctps_inter), target_pos, class_scores[target_cls], RADIUS)
score.backward()
print(f"solution's grad: \n{ctps_inter.grad}")
```

This time, we can correctly obtain the gradients of the solution.

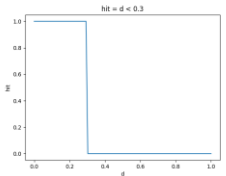
```
solution's grad:
tensor([[0., 0.],
        [0., 0.],
        [0., 0.]])
```

Why zeros? Something wrong?

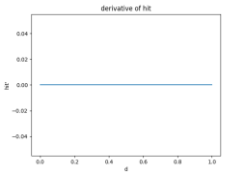
No, that is exactly the right gradient.
Notice the line in your code:

```python
def evaluate_modified(traj: torch.Tensor, target_pos: torch.Tensor, target_scores: torch.Tensor, radius: float,
                      ) -> torch.Tensor:
    cdist = torch.cdist(target_pos, traj)  # see https://pytorch.org/docs/stable/generated/torch.cdist.html
    d = cdist.min(-1).values
    hit = (d < radius)
    d[hit] = 1
    d[~hit] = 0
    value = torch.sum(d * target_scores, dim=-1)
    return value
```

this function “hit”, which funcion graph is:



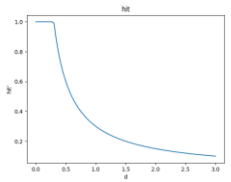

And its derivative is:



So, we also need to design another function to replace the origin hit function.

For example, we use:

```
hits = d <= radius
d[hits] = 1
d[~hits] = radius / d[~hits]
```



Its derivative is not equal to zero when d > 0.3.

Using, such surrogate function, we can successfully get the gradient of the solution.

```
solution's grad:
tensor([[ 2.8480, 16.4638],
        [ 1.8182, 15.2240],
        [-0.5492, 15.9569]])
```

Then, we can optimize our solution using the gradient.

```
ctps_inter = torch.rand((N_CTPS - 2, 2)) * torch.tensor([N_CTPS - 2, 2.]) + torch.tensor([1., -1.])
ctps_inter.requires_grad = True
lr = 1

for it in range(10):
    gra_score = evaluate_modified(compute_traj(ctps_inter), target_pos, class_scores[target_cls], RADIUS)
    real_score = evaluate(compute_traj(ctps_inter), target_pos, class_scores[target_cls], RADIUS)
    print(it, real_score.item())
    gra_score.backward()
    ctps_inter.data = ctps_inter.data + lr * ctps_inter.grad / torch.norm(ctps_inter.grad)
```

After 10 iterations, the score of such solution rise from -58 to -9.

```
0 -33
1 -8
2 -19
3 -5
4 -3
5 0
6 0
7 0
8 0
9 0
```

# What can be done to obtain the further performance?

1. use multiple initial points.
2. design another surrogate function.
3. use advanced optimizer, like SGD, Adam.
4. shedule the learning rate in iterations.
5. .....

That’s all. Hope this can be helpful. Thx :).