0.1 Guided Policy Search

- *Algorithm:* Guided Policy Search (algo. 1)
- *Input:* Environment and Immediate Cost (Reward) function: $c(x_t, u_t)$
- *Complexity:* Subject to different cases
- *Data structure compatibility:* N/A
- *Common applications:* Do trajectory optimizaiton in dynamic systems, such as simulated robots in the swimming, hopping and walking tasks[1].

Problem. Guided Policy Search

Given the environment and immediate cost (reward) function, GPS do the interaction between controller and environment. The algorithm will gain a final policy that could "know" the optimal control through additional parameter added.

Description

As a popular algorithm in reinforcement learning, Guided Policy Search (GPS) has some special properties compared to other algorithms. It build model, or dynamics to the environment, so it could not only record, but also calculate. A better sample efficiency could be obtained compared to Model-Free algorithm. GPS will final give a parametric policy, such as neural network, so no online optimization needed when testing.

Problem Formulation

To formulate the problem, introduce the following formulation[2]

- setting: fixed time length task
- assumption: deterministic dynamics: $x_t = f(x_{t-1}, u_{t-1})$
- input: environment and immediate cost (reward) function: $c(x_t, u_t)$
- output: parametric policy

With the above settings, the output parametric policy will be specified as deterministic case in this project with frame work such that: Collect Data, Fit Dynamics, Optimization and Next Iteration.

Deterministic Policy Case

1. Collect data: Use controller to interactive with environment and get the trajectory dataset D $\mathcal{D} = \tau_i$, where the trajectory is represented through $\tau_i = x_{1i}, u_{1i}, \dots, x_{1T}, u_{1T}$.
2. Fit Dynamics. Use the dataset D to fit one linear model that would use different model for different input from different time, such that $x_{t+1} = f(x_t, u_t) = A_t x_t + B_t u_t + c_t$. The model is obtained through the τ_i data $(x_{t1}, u_{t1}, x_{t+1,1}), \dots, (x_{ti}, u_{ti}, x_{t+1,i})$ from one specific time t using linear regression.
3. Optimization.

   Controller As our goal is to minimize the cost while satisfying the dynamics constraint, the problem could be summarized as the equation

$$\min_{x_1, u_1, \dots, x_T, u_T} \sum_{t=1}^{T} c(x_t, u_t)$$
$$\text{s.t. } x_t = f(x_{t-1}, u_{t-1}) \quad t = 1, \dots, T$$

To solve this equation, when the cost is a quadratic mode, there is one optimization method called *Linear Quadratic Regulator (LQR)*. LQR could take the input:linear model(F_t, f_t)and quadratic cost(C_t, c_t) with the output K_t, k_t. Obtain the optimal control, which is the solution to the previous equation $u_t = K_t x_t + k_t$, $x_{t+1} = f(x_t, u_t)$.

If the cost function is not a quadratic model, we could use iterative LQR(iLQR) to get the solution, which will use linear and quadratic expansion then apply LQR.

Policy Then it comes to the special part of GPS, as one policy will be obtained to "know" how to do the optimal control. So one optimization variable θ and constraint $u_t = \pi_\theta(x_t)$ is added

$$\begin{aligned} \min_{x_1, u_1, \ldots, x_T, u_T, \theta} & \sum_{t=1}^{T} c(x_t, u_t) \\ \text{s.t. } & u_t = \pi_\theta(x_t) \quad t = 1, \ldots, T \\ & x_t = f(x_{t-1}, u_{t-1}) \quad t = 1, \ldots, T. \end{aligned}$$

The solution for the above equation using *Dual Gradient Descent(DGD)*, the brief procedure is that

- Write the corresponding Lagrangian function $\mathcal{L}(x, \lambda) = f(x) + \lambda C(x)$
- Find the x such that minimize the function $x^* = \text{argmin}_{\ x} \mathcal{L}(x, \lambda)$
- Use x^* into $\mathcal{L}(x, \lambda)$ to get the lower bound of the original question $g(\lambda) = \mathcal{L}(x^*, \lambda)$
- Update the lambda $\lambda = \lambda + \alpha \frac{\partial g}{\partial \lambda}$
- Return to step 2 to do iteration.

4. Next Iteration. Return to Collect Data step, which is that the new controller will interactive to the environment again and do the iteration.

Algorithm 1: Guided Policy Search [1]

Generate DDP solutions $\pi_{\varsigma_1}, \ldots, \pi_{\varsigma_n}$
Sample $\zeta_q, \ldots, \zeta_m$ from $q(\zeta) = \frac{1}{n} \sum_i \pi_{\varsigma_i}(\zeta)$
Initialize $\theta^* \leftarrow \arg\max_\theta \sum_i \pi_{\varsigma_i}(\zeta)$
Build initial sample set S from $\pi_{\varsigma_1}, \ldots, \pi_{\varsigma_n}, \pi_{\theta^*}$
for *iteration* $k = 1$ **to** K **do**
    Choose current sample set $\S_k \subset S$
    Optimize $\theta_k \leftarrow \arg \text{msx}_\theta \Phi_{S_k}(\theta)$
    Append samples from π_{θ_k} to S_k and S
    Optionally generate adaptive guiding samples
    Estimate the values of π_{θ_k} and π_{θ^*} using S_k
    if π_{θ_k} *is better than* π_{θ^*} **then**
        $\theta^* \leftarrow \theta_k$
        Decrease w_r
    end if
    else
        Increase w_r
        Optionally resample from π_θ
    end if
end for
return *the best policy* π_{θ^*}

References.

[1] Sergey Levine and Vladlen Koltun. "Guided Policy Search". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–1–III–9 (cit. on pp. 1, 2).

[2] meltycriss. *Summary of Guided Policy Search*. 2019. URL: https://www.meltycriss.com/2018/05/17/summary-gps/ (cit. on p. 1).

0.2 Spectral Clustering

- *Algorithm:* Unnormalized Spectral Clustering (algo. 2), Normalized Spectral Clustering (algo. 3)
- *Input:* Similarity matrix $\mathcal{S} \in \mathbb{R}^{n \times n}$, number k of clusters to construct
- *Complexity:* $\mathcal{O}(n^3)$
- *Data structure compatibility:* N/A
- *Common applications:* machine learning, exploratory data analysis, computer vision and speech processing.

Problem. Spectral Clustering

Spectral Clustering is an algorithm based on graph and matrix theories. The given data, treated as graph, is partitioned into k clusters.

Description

The similarity of this algorithm to the traditional spectral algorithm and clustering, such as K-means, could be recognized from the name. Spectral determines it will rely on principle components of an input matrix and cluster means it will rely on the data's clustering to make decision. It is originated from graph algorithm. Basically, it considered all the data as points in the space, and all the points could be connected through lines. The point with longer line connected will have lower weight. Then do the cut for the graph, to let the sum of weight in one subgraph as large as possible and let the sum of weight among different subgraphs as small as possible.

So the method used to construct the similarity graph, cut the graph and do the cluster need to be further specified. Some common approaches are listed.

Similarity Graph Construction [4]

There are several popular methods used to transform the data points $x_1, \dots, x_n$ into a graph.

- ϵ-neighborhood graph, usually considered as an unweighted graph. With a threshold ϵ, the distance measured by s_{ij} between x_i and x_j, such that $s_{ij} = ||x_i - x_j||_2^2$. Then each element in W matrix is defined as 0 when $s_{ij} \geq \epsilon$, otherwise defined as ϵ. So many of the information lost in this procedure as only two values are presented.
- k-nearest neighbor graphs. It will traverse all the sample points and use KNN algorithm to get the nearest k points of each data as neighbors. Only these neighbors have $w_{ij} > 0$.
- fully connected graph, because all the weight will be larger than 0. One common used function is Gaussian similarity function, which gives the same similarity graph and weighted graph

$$w_{ij} = s_{ij} = exp(-\frac{||x_i - x_j||_2^2}{2\sigma^2})$$

Cut Method

Min-cut might not give the optimal solution because the weights within one subgraph is not considered. So RatioCut[1] is introduced, which do not consider minimize $cut(A_1, A_2, ...A_k)$, but also maximize the number of points in each subgraph such that

$$RatioCut(A_1, A_2, ...A_k) = \frac{1}{2}\sum_{i=1}^{k}\frac{W(A_i, \overline{A}_i)}{|A_i|}$$

. Laplacian Matrix property is used to solve the equation and the dimension is reduced to finally obtain the result. So a bit information is lost and normally clustering on each line will be used.

Another cut method is Ncut, which will not count the number of points in one subgraph but the $vol(A_i)$

$$NCut(A_1, A_2, ...A_k) = \frac{1}{2}\sum_{i=1}^{k}\frac{W(A_i, \overline{A}_i)}{vol(A_i)}$$

Cluster

The most common final cluster method is $K-$means cluster[2], which has k centroids that it uses to define clusters. The points are divided into one cluster once the distance it is to this cluster's centroid is smaller than any other centroids.

Algorithm

The most common common spectral clustering algorithm is introduced for unnormalized [4];

Algorithm 2: Unnormalized Spectral Clustering

Input : Similarity matrix $\mathcal{S} \in \mathbb{R}^{n\times n}$, number k of clusters to construct
Output: Cluster $A_1, \dots, A_k$ with $A_i = \{j | y_j \in C_i\}$

Construct a similarity graph by one of the ways described above. Let $\mathcal{W}$ be its weighted adjacency matrix
Compute the unnormalized Laplacian $\mathcal{L}$
Compute the first k eigenvectors $u_1, \dots, u_k$ of $\mathcal{L}$
Let $\mathcal{U} \in \mathbb{R}^{n\times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ as columns
For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i-$th row of $\mathcal{U}$
Cluster the points $(y_i)_{i=1,\dots,n}$ in $\mathbb{R}^k$ with the $k-$means algorithm into clusters $C_1, \dots, C_k$
return *Cluster* $A_1, \dots, A_k$ *with* $A_i = \{j | y_j \in C_i\}$

As the most popular methods used for similarity matrix construction, cut and clustering are fully constructed graph based on the Gaussian function, Ncut and $K-$means, the algorithm is introduced [4]

Algorithm 3: Normalized Spectral Clustering

Input : Similarity matrix $\mathcal{S} \in \mathbb{R}^{n\times n}$, number k of clusters to construct
Output: Cluster $A_1, \dots, A_k$ with $A_i = \{j | y_j \in C_i\}$

Construct a similarity graph. Let $\mathcal{W}$ be its weighted adjacency matrix
Compute the unnormalized Laplacian $\mathcal{L}$
Compute the first k eigenvectors $u_1, \dots, u_k$ of the generalized eigenproblem $\mathcal{L}\sqcap = \lambda Du$
Let $\mathcal{U} \in \mathbb{R}^{n\times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ as columns
For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i-$th row of $\mathcal{U}$
Cluster the points $(y_i)_{i=1,\dots,n}$ in $\mathbb{R}^k$ with the $k-$means algorithm into clusters $C_1, \dots, C_k$
return *Cluster* $A_1, \dots, A_k$ *with* $A_i = \{j | y_j \in C_i\}$

Time Complexity[3]

Most expensive step is computing the eigenvalues/eigenvectors for Laplacian matrix, which is $\mathcal{O}(n^3)$. n is the number of input points. To construct the similarity matrix costs $\mathcal{O}(n^2)$. And the final $K-$means will cost $\mathcal{O}(nldk)$, where l is the number of $k-$means iteration, d is the dimensionality and k is the final number of clusters.

So the total time complexity is $\mathcal{O}(n^3)$

In conclusion, Spectral Clustering is suitable for sparse data as it only needs the similarity matrix, which is difficult for $K-$means. However, different similarity matrix might give different result.

References.

[1] Maxim Panov. *Spectral clustering: an overview*. 2015. URL: www.cis.upenn.edu/~cis515/cis515-15-spectral-clust-chap5.pdf (cit. on p. 4).

[2] Chris Piech. *K Means, Stanford CS221*. 2012. URL: https://stanford.edu/~cpiech/cs221/handouts/kmeans.html (cit. on p. 4).

[3] Serafeim Tsironis, Mauro Sozio, Michalis Vazirgiannis, and LE Poltechnique. "Accurate spectral clustering for community detection in mapreduce". In: *Advances in Neural Information Processing Systems (NIPS) Workshops*. Citeseer. 2013 (cit. on p. 4).

[4] Ulrike Von Luxburg. "A tutorial on spectral clustering". In: *Statistics and computing* 17.4 (2007), pp. 395–416 (cit. on pp. 3, 4).