Stochastic Control of Electric Vehicle Charging

Kyle Anderson
CS 229 Machine Learning Final Project

Abstract*— This project attempts several methods to optimize charging schedules for electric vehicles on a constrained radial network using machine learning. In the first approach, electric vehicles act as independent agents in a Q-Learning framework, receiving negative rewards based on congestion charges calculated from their contribution to overloaded parts of the network. In this model, electric vehicles may take the action of increase or decrease the charging rate by a fixed amount at each time interval, but are subject to charging limits, battery capacity limits, and charging deadlines. Next, I consider a central aggregator who implements a supervised learning technique to recognize the discounted future cost of a network state. During the testing phase, the controller will greedily take actions that move to a better state. Finally, I consider another central aggregator scheme based on Linear Quadratic Regulation. I propose an (experimental) alteration to the LQR method to accommodate exponential action spaces by adding a polynomial number of quasi-time steps to allow the decision maker to perform many actions within a given time step.

I. Introduction

Electric vehicle charging is a unique electric load because it is deferrable, controllable, and deadline constrained. In other words, EV owners don't care when their vehicles are charged so long as they have enough charge when they need to drive. This type of load provides a unique opportunity to address three key issues in grid operation:

Reshaping - reshaping the **aggregate load curve** by charging during off-peak hours or even discharging during peak hours using a Vehicle-To-Grid system.

Demand Response - Reducing the required **reserve capacity** by providing ancillary service to the grid during unexpected increases or decreases in the load

Distribution Automation Providing **localized relief** to overloaded power lines or transformers on a constrained distribution grid in the case of faults or unexpected overloads

These benefits can only be obtained through the use of intelligent charging schemes. In fact, widespread uncontrolled electric vehicle charging could have disastrous consequences in many regions of the United States. Residential electricity usage tends to peak in the evening when people get home from work, turn on the TV, air conditioner, etc. (The same time that many drivers would, in theory, plug in their electric vehicle). Even deferred schemes such as charging at midnight or simplified schemes such as randomization could quickly degenerate as EV adoption increases. To put the magnitude of this problem in perspective, high powered electric vehicle chargers are rated as high as 50kW [Eaton], an order of magnitude higher than the average residential house load between 2kW and 4kW [JY]. Meanwhile, the distribution hardware in many areas of the United States is already operating above its nameplate ratings and approaching end of life [Weidmann]. Therefore, electric vehicles must be intelligent to avoid becoming the straw to break the camel's back.

II. Network Model

I obtained a set of sample distribution networks from Pacific Northwest National Labs Gridlab-D project [PNNL]. As it turns out, distribution networks in the United States are radial and often have a tree-like structure. In order to focus on the machine learning computational problem I decided to use my own Matlab script to generate my own networks of similar structure, allowing me to test my system on networks of various sizes and branching factors without focusing on building a robust parser.

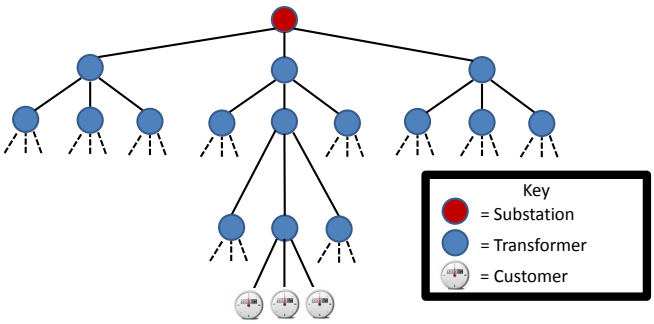


Figure 1. Sample Distribution Network

Figure 1 shows an example of one of my generated networks. The customers are connected as leafs of the tree, and the intermediate nodes are the constrained network elements (transformers, power lines, etc), which are subject to operational costs when they are overloaded. We define the operational cost of a network element as the percent by which it exceeds the average operational point of that network, scaled by a constant. Note that this cost is calculated on a per node basis using the per-node average load.

$$Cost = \alpha * \max\left[0, \frac{(Load - \mu_{load})}{\mu_{load}}\right]$$

In order to determine the operating state of an electrical network, one would typically solve the load flow problem. This problem consists of setting the P and Q values on load buses, P and |V| values on generation buses, and solving the following system of equations enforcing Kirchoff's laws to recover the unknown values.

$$0 = -P_i + \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\cos\theta_{ik} + B_{ik}\sin\theta_{ik})$$
$$0 = -Q_i + \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik})$$

This system of nonlinear equations can be solved using Newton's method[4].

Newton Update Rule:

$$\begin{bmatrix}\Delta\theta \\ \Delta|V|\end{bmatrix} = -J^{-1}\begin{bmatrix}\Delta P \\ \Delta Q\end{bmatrix} J = \begin{bmatrix}\frac{\partial\Delta P}{\partial\theta} & \frac{\partial\Delta P}{\partial|V|} \\ \frac{\partial\Delta Q}{\partial\theta} & \frac{\partial\Delta Q}{\partial|V|}\end{bmatrix}$$

Where:

$$\Delta P_i = -P_i + \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\cos\theta_{ik} + B_{ik}\sin\theta_{ik})$$
$$\Delta Q_i = -Q_i + \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik})$$

My initial Q-Learning reward function used MatPOWER[5], a Matlab-based solver to determine the load on each node in the network, and the resulting operational cost. This method turned out to be a bottleneck as it limited the speed of my algorithms. I resorted to an approximation where the load on any node in the tree network is simply the recursive sum of all the child loads. This approximation is mostly accurate for tree networks.

III. Customer & EV Models

In addition to acquiring network models from PNNL, I found some smart meter data from a PG&E Pilot program [7] that provides a realistic estimate of the non-EV base load for my models. Additionally, I was able to acquire traffic data regarding driving patterns in the United States from National Highway Traffic Safety Administration [8]. The ultimate goal of this project would be to prove the performance of my algorithms on a unified model using these datasets. However, for now, I generate the customer loads using Matlab in order to focus on the ML problem at hand. This model provides sample customer data including the arrival time, departure time, amount of charge required, and battery sizes for each electric vehicle on a given operational day. These data generation scripts were inspired by the datasets described above, but are easier to work with because they are not missing data points. See Appendix A for a detailed description of these models.

IV. Optimal Solution

The algorithms discussed in this paper assume we do not know the future arrival times of EVs or the amount of charge that will be required by an EV before it arrives. Additionally, the algorithms do not assume any future knowledge the customer's base load. The algorithms do assume that once an electric vehicle is plugged in, its deadline and charge required are known. If we were clairvoyant, we could generate an optimal policy that minimizes the network operational costs while ensuring hard deadlines for all electric vehicles by solving the following linear program.

$$\min \alpha * \sum_t P + \sum_t \sum_i [\beta * P_{i,t}] + \sum_t \sum_i \sum_j [\chi * P'_{i,j,t}] + \sum_t \sum_i \sum_j \sum_k [\delta * P''_{i,j,k,t}]$$

s.t.

$$-RampLimit_c \le \text{Rate}_{c,t} - Rate_{c,t-1} \le RampLimit_c \qquad \forall t, \forall customers$$
$$\text{SOC}_{c,t} = SOC_{c,t-1} + Rate_{c,t-1} \qquad \forall t, \forall customers$$
$$\text{Rate}_{c,t} = 0 \qquad \forall t \text{ when customers not home}, \forall customers$$
$$\text{SOC}_{c,t} = \text{SOC}_{c,t-1} - \text{ChargeRequired}_c \qquad \forall t \text{ when customers drives away}, \forall customers$$
$$P_t \ge \sum_i \sum_j \sum_k \sum_l Load_{i,j,k,l,t} - \frac{1}{T}\sum_t \sum_i \sum_j \sum_k \sum_l Load_{i,j,k,l,t} \qquad \forall t$$
$$P'_{i,t} \ge \sum_j \sum_k \sum_l Load_{i,j,k,l,t} - \frac{1}{T}\sum_t \sum_j \sum_k \sum_l Load_{i,j,k,l,t} \qquad \forall t, \forall i$$
$$P''_{i,j,t} \ge \sum_k \sum_l Load_{i,j,k,l,t} - \frac{1}{T}\sum_t \sum_k \sum_l Load_{i,j,k,l,t} \qquad \forall t, \forall i, \forall j$$
$$P''_{i,j,k,t} \ge \sum_l Load_{i,j,k,l,t} - \frac{1}{T}\sum_t \sum_l Load_{i,j,k,l,t} \qquad \forall t, \forall i, \forall j, \forall k$$
$$0 \le SOC_{c,t} \le BatterySize_c \qquad \forall t, \forall customers$$
$$-RateMax_c \le Rate_{c,t} \le RateMax_c \qquad \forall t, \forall customers$$

This solution provides a metric against which can compare the performance of our algorithms, as well as generate a "score" for network states to use as training data for the supervised learning method discussed later.

V. MultiAgent Q-Learning

My first attempt was a multi agent Q-Learning model in which each customer agent maintains its own discrete state space and keeps track of the rewards it has seen.

$\text{Rate} \in \{-2,-1,0,1,2\}$

$\text{Priority}^* \in \{-2,-1,0,1,2\}$

$\text{L1Load} \in \{under, average, over, critical\}$

$\text{L2Load} \in \{under, average, over, critical\}$

$\text{L3Load} \in \{under, average, over, critical\}$

$L4Load \in \{under, average, over, critical\}$

$\text{Time} \in \{\text{5pm-5:30pm,5:30-6pm, 6pm-6:30pm,6:30-7pm,7pm-8pm,other}\}$

*Priority = ChargeRequired/Deadline, scaled by Battery Size

$$\text{R(s)} = \text{Rate} * \left[\frac{(L1Load - \mu_{l1\_load})}{\mu_{l1\_load}} + \frac{(L2Load - \mu_{l2\_load})}{\mu_{l2\_load}} + \frac{(L3Load - \mu_{l3\_load})}{\mu_{l3\_load}} + \frac{(L4Load - \mu_{l4\_load})}{\mu_{l4\_load}} \right]$$

The agents do not keep track of the state of other agents, and only have knowledge of the aggregate load on the nodes that are on their path to the root. The state transition probabilities are not deterministic from the perspective of each agent because they depend on the actions of other agents in the system as well as on the future base loads in the system. For this reason, I have modeled this decision using Q-Learning, where the agent tries to learn the expected utility of an action value pair without modeling state transitions.

The epoch for training an EV is a single day with 96 15-minute time steps. On each training day, the customer models generate new data for arrival time, departure time, and charge required, but they maintain their same statistics to generate this data from day to day. At the end of each day, the Q values are updated according to:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha_t(s_t, a_t) * (R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Since the system dynamics from the perspective of a single EV are changing over time as other customers refine their policy, we do not want the agent to stop "exploring" possible states. Therefore, we probabilistically select the next action using the Boltzmann rule:

$$p(a \mid s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')\tau}}$$

It is important to note here that if the optimal action was not selected, the Q value will not back propagate to the previous state. This problem is more difficult than a traditional application of Q-learning because the state action reward depends on the actions of other EVs that are not visible from the perspective of a single EV. Furthermore, in order to keep the state space small enough to have a workable model, I was forced to do course discretization.

In the initial process, all electric vehicles entered the training phase with Q matrix set to zero. In this case, the operating policies did not seem to converge at all. I received better results when I "enabled" the EVs on the network one at a time. This allowed each EV to find a stable operating policy given the new network, and future EVs to learn their policy on a stable network. This method may, in fact, more accurately reflect how such a system would be implemented in practice since it is unlikely that all EV owners will purchase a new vehicle at the same time. In the end, this method yielded mediocre results (See "Results").

VI. Central Aggregator Model

In this central aggregator model, I assume there is a central decision maker who has full network knowledge at the current time step, but does not have knowledge of future customer base loads or arrivals of new EVs. I implemented a supervised learning method of training a neural network to recognize "good" network states. I generate random initial starting conditions for the network, and use the linear program to solve for optimal solutions. I use the LP solution to calculate the discounted cost of operating the network over the upcoming 8-hour period given the random initial state, and assuming it follows the optimal policy solved by the LP.

In order to reduce the dimentionality of a single NN, I train a neural network for each node and for each time period. Thus, there are a total of 96*(#nodes) distinct neural networks. Training samples are 8-input vector indicating the summed EV charging schedules for each node in the network. For example, if the LP decided that one customer would would charge at 3kW for the next 5 hours, and another customer would charge 2kW for the next 3 hours, the training input for the parent node (transformer) would be:

Input: [5 5 5 3 3 0 0]

Output: estimated discounted cost

If all charging operations were deferred for 1-hour, the vector would be

Input: [0 5 5 5 3 3 0]

Output: estimated discounted cost

Note that the input vectors represent the EV load associated with plugged in customers only, while the estimated discounted cost would be a function of the base load and new EV customers as well. Thus, the neural network is implicitly predicting the base load and arrival of new EVs.

Intuitively, having a different neural network for each time period allows us to capture the effect that deferring a load at a certain time of day may be more costly at a particular node than at a different time of day.

During the test/operation phase, we greedily find the customer action (increase rate, stay same, decrease rate), which results in the best increase to the overall system

cost. (i.e. the weighted sum of the outputs from the NNs for each node in the network). We continue selecting actions that decrease the total system cost until there are no more such actions, at which point we advance to the next time step.

VII. Linear Quadratic Regularization (Experimental)

I have attempted an implementation of Linear Quadratic Regularization with an experimental modification to the algorithm from class, but it is not yet working (I would love some feedback here).

I am trying to apply LQR to this problem using the "central aggregator" approach, where the states of each customer and each node (transformer) are concatenated to form a large state vector (~1500 dimensions). The actions available at each time step are to increase the charging rate, decrease the charging rate, or hold the charging rate for each customer. This would yield 3^n potential actions, which makes the algorithm from class intractable. In order to decompose the action space, I create intermediate time steps that do not correspond to actual time steps within the system. Precisely, I add a "quasi-timesteps" between each real time step for each customer in the system, where the system operator has the ability to take an action, but the system does not otherwise change. This method gives the operator the ability to select any combination of actions in between actual system time steps. Precisely, the A_t matrices corresponding to the real time steps will be calculated from training data and reflect the probabilistic changes to the system as time advances (such as new EVs arriving, deadlines decreasing, and base loads changing).

$$s_{t+1} = A_t s_t + B a_t + w_t$$
$$w_t \sim N(0, \Sigma_t)$$
$$\min_A \sum_{i=1}^{m} \sum_{t=1}^{T-1} \left\| s_{t+1} - (A_t s + B_t a_t) \right\|^2$$

On the other hand, the A_t matrices corresponding to the intermediate steps will be the identity matrix, since the only changes to the state vector on the quasi-time steps will be a deterministic function of the selected action.

Furthermore, in both real time steps and quasi-time steps, the B_t matrixes are known since the actions have a fully specified effect on the state vector (the loads at specific locations are increased/decreased).

With this modification as follows, we can follow the dynamic programming procedure discussed in class. Specifically, we recursively calculate the value function for each state using

$$V_t^*(s_t) = \max_{a_t} R^t(s_t, a_t) + E_{s_{t+1} \sim P_{sa}}[V_{t+1}^*(s_{t+1})]$$

Now, the reward function, R, can be calculated deterministically since the system operator knows the full dynamics, and the P_{sa} will be estimated from sample data for the real time steps, and known exactly for the "quasi-timesteps".

I chose a random state vector at the end of the epoch as the base case.

$$V_T^*(s_T) = -s_T^T U_T s_T$$

I select an epoch of 3 days so we can clip out a 24-hour period to find a "stationary" day, where the randomly selected "base case" does not have significant impact.

Then, the action at each time step corresponds to

$$a_t = ((B_t^T \Phi_{t+1} B_t - V_t)^{-1} B_t \Phi_{t+1} A_t) * s_t$$

VIII. Results And Conclusion

In this study, the simplest approach using supervised learning worked best. I quantify these results by showing the average overuse charge for each node for an operational day. The penalty contribution of each node is weighted by the number of customers beneath the node.

$$Performance = \frac{1}{96} \frac{1}{|N|} \sum_{t=1}^{96} \sum_{n \in N} (\#Customers)_n * \max\left[0, \frac{(Load_{n,t} - \mu_n)}{\mu_n}\right]$$

When calculating the performance, I ran the model from a random starting position for 48 hours, and used the last 24 hours as the performance data in an attempt to create a "stationary" day, since random initial starting points could be particularly bad. Under this method, the Linear Program would, in general, perform perfectly. Note that the LP used to generate our supervised learning training data starts from random (i.e. "bad") states, in which case it may have significant discounted costs.

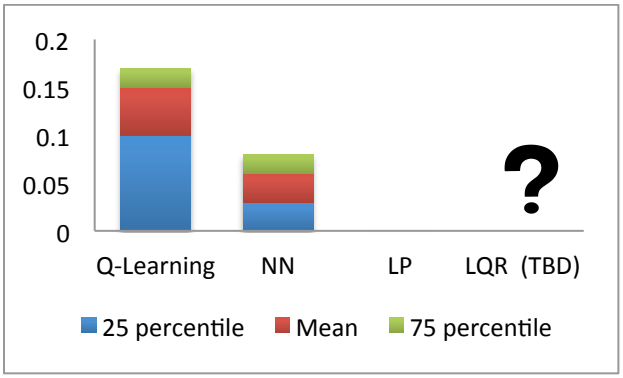
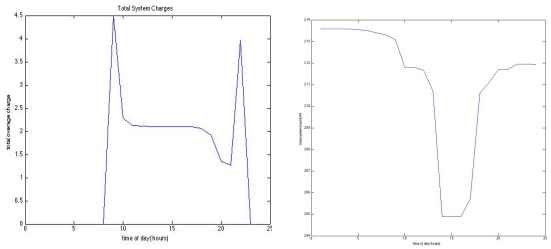


Figure 2. Performance

The plot below (left) shows the average overuse in the network over time for selected operation day using the supervised learning approach. The plot below (right) shows the total system load over time, which stays within 10% margin throughout the day (scaled axis).



IX. Appendix A (Customer Data Generation)

A. Initialization

I use a Matlab script to create a set of "realistic" customers, $C_{i,j,k,l,,}$ located beneath transformers i,j,k,l in the network from Figure 1. During initialization, each customer generates statistics that it will use to generate its outputs during operation.

$bounds\_loadScaling_{1\ldots96,\ 1..2} \leftarrow Traingular[0,4]$

$BatterySize \leftarrow Triangular[10,70]$

$Bounds\_leaveForWork_{1..2} \leftarrow Triangular[20,44]$

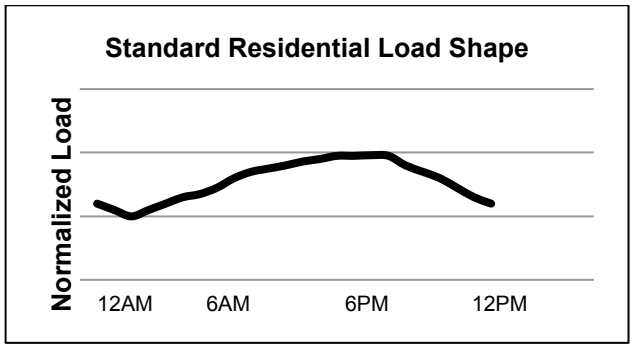
$Bounds\_arriveHome_{1..2} \leftarrow Triangular[48,92]$

$Bounds\_chargeRequired_{1..2} \leftarrow Tri[-BattSize,BattSize]$

B. Operation

During operation, the customer model will generate a baseload for each 15-minute interval by scaling the normalized standard residential load shape shown below (96 intervals per day).

$$baseLoad_{1\ldots96} \leftarrow loadScaling_{1\ldots96} .* Tri[C_{i,j,k,l}.bounds\_loadScaling_{\ldots96}]$$



The customer's load at each interval, t, will be the sum of the EV charging rate and the base load.

$$Load_{1\ldots96} \leftarrow baseLoad_{1\ldots96} .+ Rate_{1\ldots96}$$

The customer will increment the state of charge for his battery at each time step

$$SOC_{t+1} \leftarrow SOC_t + Rate_t$$

During each day of operation, the customer will generate a new leaveForWork, arriveHome, and chargeRequired value based on the statistics it generated during initialization.

$$leaveForWork \leftarrow Triangular[Bounds\_leaveForWork_{1..2}]$$

$$arriveHome \leftarrow Triangular[Bounds\_arriveHome_{1..2}]$$

$$chargeRequired \leftarrow Triangular[Bounds\_chargeRequired_{1..2}]$$

The model requires its state of charge to be greater than chargeRequired at time leaveForWork.

$$SOC_{leaveForWork} > chargeRequired_{leaveForWork}$$

At time leaveForWork, the stateOfCharge will be reduced by chargeRequired. The model will force its charging rate to increase if it is in danger of missing this deadline.

$$SOC_{leaveForWork+1} \leftarrow SOC_{leaveForWork} - chargeRequired_{leaveForWork}$$

The model also requires the rate to be zero between the times leaveForWork and arriveHome.

$$Rate_{leaveForWork\ldots arriveHome} == 0$$

X. Acknowledgements

I would like to thank Professor Abbas El Gamal and Han-I Su for the contributions in helping formulate this problem and discussing potential techniques.

References

[1] http://www.smartgrid.com/wp-content/uploads/2012/05/Eaton-DCQC.pdf
[2] http://nj.gov/emp/facts/
[3] http://www.weidmann-solutions.cn/zhenduan/condition_based_strategies.pdf
[4] http://en.wikipedia.org/wiki/Power_flow_study
[5] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," *Power Systems, IEEE Transactions on*, vol. 26, no. 1, pp. 12-19, Feb. 2011.
[6] http://en.wikipedia.org/wiki/Electric_power_transmission
[7] PG&E Smart Meter Pilot Program.
[8] http://www.nhtsa.gov/NCSA
[9] http://en.wikipedia.org/wiki/Q-learning