CS 234 Final Project: Electric Vehicle Smart Charging

Will McCloskey [1] **Oskar Triebe** [2] **Justin Luke** [2]

Abstract

Controlled "smart" charging of electric vehicles can support renewables integration objectives, minimize aging of grid assets, while still providing quality charging services to customers. This project contributes an open-source OpenAI Gym simulator and several reinforcement-learning based algorithms to experiment with charging of electric vehicles at a charging facility. The gym enables to re-simulate charging episodes based on real data. It provides high customizability with controller and environment configurations and utilizes a reward function which incentivizes vehicle charging at times of day with high renewable energy penetration while also minimizing peak load on the stations' shared transformer. Algorithms tested include Deep Q-Learning and SARSA, Double DQN, and Policy Gradient. Our experiments found variants of policy gradient to be most effective in maximizing solar charging while satisfying charge under a transformer constraint compared to the baseline uncontrolled charger. Policy gradient with a linear network reduced charging cost (non-renewable charging) by 18% with 93% charge quality in a single station setting and also provided 6% higher charge satisfaction in a multi-station setting with transformer limit. Future work will explore multi-agent frameworks, the benefits of coordination with on-site stationary storage and local solar PV arrays, and vehicle-to-grid charging scenarios.

1. Introduction

The International Energy Agency projects that the number of EVs on the road will grow from 3 million today to 125 million EVs by 2030 (T.Bunsen & J.Teter, 2018). As California sets ambitious EV and renewables adoption targets (5 million zero-emission vehicles by 2030; 100 percent renewables by 2045), "smart" controlled charging is crucial to balance the needs of drivers and the stability of the electricity grid. Shifting EV charging loads towards times of the day where there is a high supply of renewable energy sources such as solar, could help to meet Californias ambitious goals.

Our project contributes to the Smart Charging Infrastructure Planning Tool (SCRIPT) that is managed by the Grid Integration, Systems & Mobility (GISMo) group at SLAC. SCRIPT is an early stage project with limited technical work performed so far; thus our course project will be contributing original work. After designing our charging algorithm, we can validate its performance on the real-world EV charging stations at SLAC in mid-2019.

The reinforcement learning algorithms used in our project aim to control electric vehicle charging at a multi-station facility such that electricity cost in terms of carbon intensity is minimized, aging of the transformer supplying the facility is minimized, and charge quality is maximized. To achieve this, we developed a simulator to test and validate performance of these algorithms. We used policy gradient, deep q-learning, SARSA, and Double DQN in our experiments.

Section 2 of this paper will first provide background and related work in using reinforcement learning for load control in the energy domain. Section 3 provides an overview of the simulator created for this project and the algorithms used to test EV smart charging. Section 4 details the experiments performed under varying controller-environment configurations and discusses their results. Lastly, Section 5 discusses the impact of these results and future work.

2. Background & Related Work

Prior work on intelligent operation of electric loads with the grid have typically used mathematical optimization or control-theoretic frameworks (e.g. model-predictive control), though reinforcement learning (RL) approaches have now also been explored in the past years.

[1]Department of Electrical Engineering, Stanford University [2]Department of Civil and Environmental Engineering, Stanford University. Correspondence to: Justin Luke <justin.luke@stanford.edu>.

Chis et al. utilized fitted q-Iteration with a kernel-based function approximation of value iteration for a single simulated user in a MDP framework with objective to minimize cost of charging (Adriana Chis, 2015). Vandael et. al also used fitted q-Iteration for off-policy learning of an MDP defining EV charging behavior, but the EVs are following a heuristical charging policy and the EV data is synthesized from general (non-EV) transportation data (Stijn Vandael, 2018). In both works, deep learning function approximations were not explored. Jiang et al. used a multi-agent framework, considering charge station, power grid, traffic conditions, plug-in electric taxis, and passenger agents. They used Q-learning and Q(λ)-learning and ensured safer operation of power systems, but assumed simple Gaussian distributions for their simulated EV model (C.X. Jiang, 2018). Mocanu et al. use deep Q-learning and deep policy gradient techniques, however, for purposes of residential building energy optimization, not commercial EV charging stations (Elena Mocanu, 2018).

Thus, while there has been existing work on using RL and MDPs for charging while considering grid conditions, renewables, and prices, we have not found studies which have *jointly* used large-scale data-driven EV charging models, deep RL, and real-world experimental validation.

3. Approach

3.1. Data

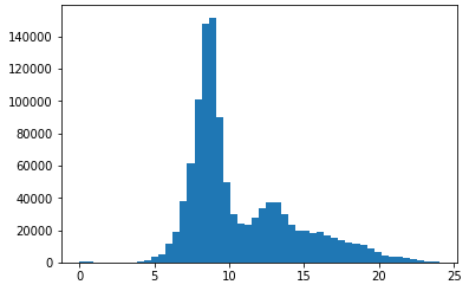

Figure 1. Frequency of session start times (hour of day) for the top 1000 stations. There is a large peak of arrivals at around 9 am and a smaller peak at 1 pm, similar to normal workplace parking.

We look at two different sets of workplace charging stations, both covering the last three years (2016 to 2018). The first dataset contains data for the 1000 busiest charging stations from various Bay Area locations, with 1,207,819 charging sessions. The second dataset contains data from the ZIP code with most busy stations (95014, Cupertino, CA), whereas we only use the 10 most and 10 least used stations at this location, with 28,516 and 10,677 charging session.

All charging sessions are filtered to be at least 5 minutes long and charge at least 1 kWh. We limit the duration to one day and the charge to 100kWh, corresponding to the largest mass-manufactured EV-battery. Further, we excluded stations which were coordinated to never have a fully charged car connected to them. These kinds of stations are the busiest in the network but are not representative of a typical workplace station.

For each charging session, we extract the station ID, start-datetime, total energy charged and duration of stay.

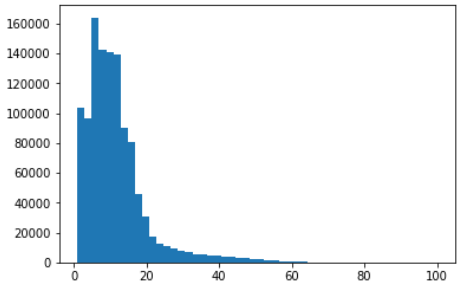

Figure 2. Frequency of energy amounts (kWh) charged during a session for the top 1000 stations.

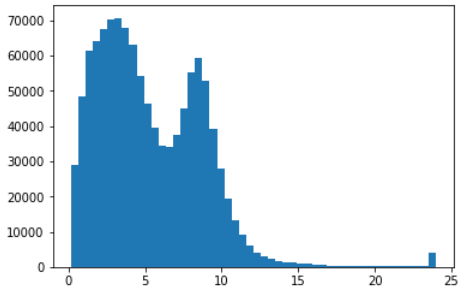

Figure 3. Frequency of different session durations (hours) for the top 1000 stations.

Split We divide our datasets into a training and an evaluation set with a ratio of 0.8 to 0.2. If not otherwise stated, we use the data from the busy stations at ZIP 95014. At training, we randomly select one or multiple stations. Hereby we fix the random number generator seed in order to ensure comparability among the controllers.

Each time when an episode ends or reaches its maximum length, a new start time is randomly sampled and a new episode extracted from the data.

3.2. Initial Simulator and Baseline

We first built our own simulator for an electric vehicle charging facility. Here we detail our initial work. A Simulator instance is utilized by an Experiment wrapper, which determines which Controller to use, which charging data and electricity price data to load, the number of stations at the facility, the power capacity limitation of the transformer serving the facility, the number of episodes to simulate, the length of an episode, the time step, and the weights for the reward terms in the reward function. For a given episode, the Simulator randomly samples a start time in the range of the loaded data and simulates for a fixed episode length.

The Simulator has the Controller pick an action for the current state and determines the reward and next state. The controller can use this (state, action, reward, next state) tuple to update controller parameters. By creating Controller objects separate from the Simulator, it is simple to plug-and-play different Controllers to compare their performance for a given Experiment instance.

We compare results the results of our models to a baseline (BaselineOne) controller, which is the status quo electric vehicle supply equipment (EVSE) charging strategy, deployed in present day. When a car arrives, the baseline controller simply charges at the maximum possible power rate until the car is fully charged or leaves the station. A plot of the baseline controller for a single station for one day is shown in Figure 4. We will compare smart charging reinforcement learning controllers against this baseline. For completeness, we also report the performance of two additional baseline controllers, do-not-charge controller (BaselineZero) and random-action controller (Random).

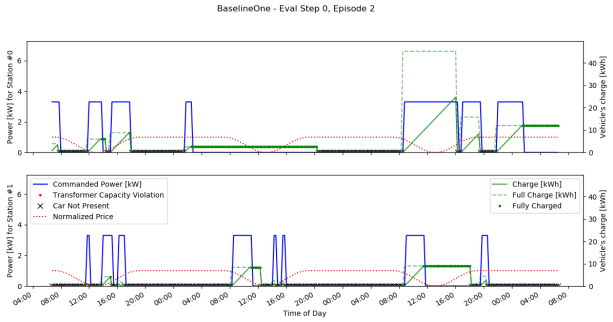

Figure 4. Example single-day episode on the Gym environment with the baseline controller. The baseline controller charges at the maximum rate whenever possible, and these rates are plotted in blue. In dotted green is the car's requested energy, and in solid green is the charged energy in the car. Finally, in dotted red is the price of electricity, which is cheapest at midday when renewables are most available.

3.3. OpenAI Gym Environment

One important product of our work is converting our simulator to a highly customizable standard OpenAI Gym environment for the open source community. We reformulated our Simulator and Experiment code into the OpenAI Gym environment template, we allow researchers interested in improving electric vehicle charging efficiency to run reinforcement learning algorithms on their own charging and electric price data. Many high quality open-source models work out of the box with OpenAI Gym environments. The environment contains many settings which are easily modified, as described below. We also allow a controller wrapper framework that plots episodes and calculates model statistics for those who write their own controllers. The simulator determines the next state based on the rates of charge at each station.

3.3.1. Overview

At a high level, the environment works as follows. To initialize, the environment takes in real-world data of charging sessions at one or multiple charging stations, where each session is a (start-datetime, total energy charged, duration) tuple, and the varying price of electricity. Each episode of the environment randomly samples a start time from the data and simulates over a fixed length of time (e.g. one week). At each time step – in our case every 15 minutes – the environment takes in an action, which is the rate of charge for each station.

There are multiple possible choices for featurizing the state space, selecting allowable actions, and determining when and how to reward the controller. All of these choices are supported and are easily obtained by selecting an existing configuration file.

3.3.2. Featurization of State Space for Observations

In general, the un-featurized state contains the current time as well as the information for each station – whether or not there is a car as well as the car's desired charge, its time since arrival, and its percentage charge. Note that the agents do *not* know the car departure times and must learn this from data. The state space needs to be featurized in order for function approximation to work. To featurize the state, one has to decide whether to make variables continuous or to discretize encode them.

If continuous, variables are converted into normalized floating point numbers. If discretized, variables are binned and one-hot encoded. For example, one can discretize time of day or percentage charged into bins and then one-hot-encode. Analogously, the day of the week can be represented as type of day (weekday or weekend) or one-hot

weekday.

It is also possible to combine both featurization methods. If not otherwise stated, we use the combined features (concatenated) for our experiments.

3.3.3. Action Space and Limits

Our environment also supports the option to discretize the action space. The action space, which is the rate of charge at each station, is naturally continuous. Continuous charge rates can allow a model to perform better than a discrete action space. Moreover, in the discrete case, the permutation of possible actions grows exponentially with the number of stations. Nonetheless, having a discrete action space is important as some models, such as DQN, are restricted to discrete action spaces. As default, we use a discrete action space represented as the permutation of all station-action combinations with 2 actions each (charge-max, not-charge).

We have also implemented vehicle-to-grid (V2G) capabilities in the Simulator to model two-way powerflow scenarios, in which EV fleets discharge to the grid in emergency events. In this scenario, the action space can include negative charge rates, though the simulator ensures the car cannot have negative state of charge.

In the real world, stations of a charging facility typically share a transformer, which limits their collective total charge rate. For example, if the transformer's rated capacity is insufficient, having multiple connected stations simultaneously charge at the maximum rate can cause damage.

We set an optional constraint on the action space to limit the total power draw. Our environment allows the option to impose a hard-cap transformer limit, which is set as a fixed fraction of the stations' collective maximum charge rate. Another option is to allow for violation of the rated transformer capacity, but with a negative reward term.

3.3.4. Reward Function

The reward function used by the simulator has three reward terms: charge reward r_{charge} (positive), electricity cost $-r_{cost}$ (negative), and transformer power capacity violation penalty $-r_{TC}$ (negative).

$$R(t) = \vec{w}^T(r_{charge}(t), -r_{cost}(t), -r_{CT}(t))$$

$$r_{charge}(t) = \frac{r_{max}}{t \cdot TC} \cdot \sum_{i=1}^{N} I_i(t) E_i(t)$$

$$r_{TC}(t) = \exp\Bigg(\log r_{max} \cdot \min\left[1,\ \frac{1}{TC} \cdot \max\left(0,\ \sum_{i=1}^{N} P_i(t) - TC\right)\right]\Bigg)$$

$$r_{cost}(t) = \frac{r_{max}}{t \cdot TC} \cdot price(t) \cdot \sum_{i=1}^{N} E_i(t)$$

where $\vec{w}$ is a unit norm weight vector, E_i is the Energy delivered to car i, P_i is the Power delivered to car i ($P_i = \frac{E_i}{t}$), I_i is the charge incentive (relative to charge state of car i) and TC is the transformer capacity.

The reward at a given timestep is a weighted sum of these rewards, where the relative weights of the three terms are Experiment hyper-parameters. The charge completion term is the product of the amount of energy delivered to the vehicle in that time step with a multiplier that is proportional to the inverse of the fraction of percentage charge completion, then summed across all stations at the facility.

The environment also has an option to defer the charge reward term and only give it to the agent when the car leaves, thus learning car departures, or only if the car is fully charged, thus encouraging complete charges.

The electricity cost term is the product of the sum of energy charged at all stations in that time step with the current electricity price. In California, the price is cheapest at peak-solar generation. Thus, charging at times with high renewable energy generation will lower electricity costs. As the motivation of this investigation is to allow for greater penetration of solar energy, we approximate the price as a full period of the cosine function from 7 am to 7 pm, where the price is 1 outside this window and zero at the center. However, the gym allows the electricity price to be loaded from historic data.

Lastly, the transformer capacity penalty term is zero when the total power draw from all stations is below the capacity threshold, but then grows exponentially by the amount of power violating the threshold. This reflects the expensive cost of overheating and aging the transformer serving the charging facility. The maximum penalty is reached when the draw reaches double the transformer capacity.

3.4. Methods

We implemented a number of reinforcement algorithms for the project. The three baseline algorithms are called BaselineOne, BaselineZero, and Random, which charge respectively always, never, and randomly. Since the current charging algorithm in the real world is BaselineOne, we are especially interested in comparing the performance off BaselineOne with our more advanced models. These more advanced models include various forms of Q-value approximation, such as SARSA and Q-Learning, as well as policy gradient methods.

We have multiple versions of Q-value approximation. The first is standard Q-learning with neural network function

approximation, which is the Q_{MLP} model. The Q-value is approximated by a neural network $Q(s, a; w)$ parametrized by w, and the gradient update is

$$\Delta \mathbf{w} = \alpha \Big(r + \gamma \max_a \hat{Q}(s_{t+1}, a; \mathbf{w}) - \hat{Q}(s_t, a_t; \mathbf{w}) \Big) \cdot \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t; \mathbf{w})$$

Rather than using the temporal-difference target $r + \gamma \max_a \hat{Q}(s_{t+1}, a; \mathbf{w})$, SARSA instead uses the target $r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \mathbf{w})$. Both methods can be improved via a second network. Double SARSA uses two networks, one parametrized by w_A and the other by w_B to improve training stability. The gradient update for double SARSA is

$$\Delta \mathbf{w}_A = \alpha \Big(r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \mathbf{w}_B) - \hat{Q}(s_t, a; \mathbf{w}_A) \Big) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t; \mathbf{w}_A)$$

and the roles of networks A and B are swapped each step with probability $\frac{1}{2}$. We also used double Q-learning which applies an analogous update.

The LinearQN and DeepQN networks apply methods such as experience replay and separate target network to stabilize training even more. These frameworks were taken from our class and modified for our problem setting, integrated into our controller framework. One major difference is the type of network: since our state space is a vector of features instead of an image, we replace the convolutional neural network with feed forward network. LinearQN is a single layer network. For DeepQN, we used a three-layer network and experimented with multiple layer sizes. The DQN network has layer sizes (512, 512, 256), DQN_{small} has layer sizes (128, 128, 64), and DQN_{nano} has layer sizes (64, 32, 16). We also experimented with the other parameters such as batch size, learning rate, and buffer size.

We also made use of policy gradient methods. In policy gradient, we parametrize a policy $\pi_\theta(s, a)$ and update θ by estimating the gradient of the expected return under π_θ. Again, the code was restructured from class into our controller framework. We use a baseline and normalize the advantages to improve training. The network PG is policy gradient with a discrete action space – where π_θ is a multinomial distribution – and the network PGC has a continuous output with multivariate normal distribution. The network sizes are analogous to those in the DQN setting. The PG_{long} newtwork was trained with 100 times smaller batch size, for 100 times more batches, with refined learning rate.

4. Experiment Results

All experiments were conducted with the environment setups described in table 4 in the appendices. There are Single and Multi2 (two) station environments. Charge reward was set to be delayed until departure. Single station environments end episode at departure. Multi station environments simulate episodes of three days.

Table 1. [Environment: SingleBal2] Comparison of evaluation metrics for different controllers on a single station EV charging environment.

Controller	Reward	Energy (total %)	Cost (per kWh)
BaselineZero	0.000 ± 0.0	0.000	0.000
BaselineOne	3.650 ± 0.1	**1.000**	0.565
Random	2.888 ± 0.1	0.784	0.557
LinearQN	3.372 ± 0.1	0.855	0.447
DeepQN$_{nano}$	3.329 ± 0.1	0.809	0.375
DeepQN$_{small}$	3.357 ± 0.1	0.824	0.390
DeepQN	3.445 ± 0.1	0.888	0.478
PG$_{linear}$	**3.668** ± 0.1	0.938	0.460
PG$_{nano}$	3.650 ± 0.1	1.000	0.565
PG$_{nano-long}$	3.510 ± 0.1	0.850	0.367
PG$_{small}$	3.650 ± 0.1	1.000	0.565
PG	3.650 ± 0.1	1.000	0.565
PGC$_{linear}$	3.589 ± 0.1	0.954	0.520
PGC$_{nano}$	2.902 ± 0.1	0.726	0.437
PGC$_{nano-long}$	2.794 ± 0.1	0.639	**0.262**
PGC$_{small}$	3.650 ± 0.1	1.000	0.565
PGC	3.650 ± 0.1	1.000	0.565
Q$_{MLP}$	2.685 ± 0.1	0.723	0.551
Sarsa$_{MLP}$	2.690 ± 0.1	0.730	0.564
Q$_{DoubleMLP}$	1.636 ± 0.1	0.460	0.620
Sarsa$_{DoubleMLP}$	0.818 ± 0.1	0.213	0.505

In the single-station environment, we find most algorithms do better than the baseline in terms of price, as seen in table 1. However, it always comes at the expense of the total energy supplied. As the controller does not know when cars leave, delaying charge, waiting for better prices, runs into the risk of the car leaving before being fully charged. Thus, Cost and Charge Completion are to be traded off. An example of this trade-off can be seen in figures 5 and 6 from the controller PG$_{linear}$. This controller is fairly aggressive in it's price optimization strategy, sometimes not reaching full charge before a car's departure. It does seem to learn to estimate the approximate departure time of cars, conditioned on the state.

The controller that achieved the best performance in terms of cost minimization is the PG algorithm with a small network, trained over many small batches. It does, however, only deliver 64 percent of the desired energy.

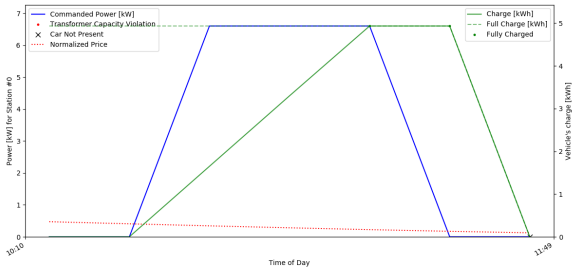

Figure 5. Sample session of controller PG_{linear}. First it defers charging and then starts charging as potential departure nears, and reaches completion before departure.

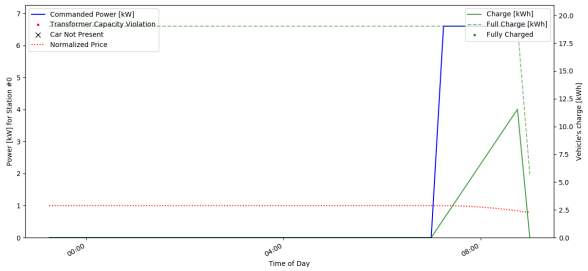

Figure 6. Sample session of controller PG_{linear}. While price is at high levels it defers charging and then starts charging as the price starts dropping, but does not reach completion before departure.

Some of the policy gradient algorithms don't learn intelligent actions and just stick to the same behavior of always charging, obtaining identical reward as the baseline. PG controllers with larger networks or discrete action space are more prone to this behavior.

The only controller which is able to improve over the baseline in terms of average reward is the linear PG network. It also achieves the best overall balance in terms of cost and charge completion. The confluence of best reward and best balance suggests that this environments reward function weights and gamma selection are well tuned for the given task. We report this controller's results on the other environment configurations in table 3.

Table 2 reports metrics for the scenario where controllers decide over actions for two stations, under a stringent transformer limit (0.5), we find that no controller is able to find a better balance than the baseline. Nevertheless, the linear PG controller achieves better average reward and is able to deliver more charge. An optimization for prize yields a large drop in delivered charge, as in the case of LinearQN.

The linear PG controller is able to minimize the cost at a lesser charge reduction in both the single and multi station

Table 2. [Environment: Multi2Bal2] Comparison of evaluation metrics for different controllers on a two station EV charging environment. There is a transformer limit that restricts charging output to half if both stations are commanded to charge.

Controller	Reward	Energy (total %)	Cost (per kWh)
BaselineZero	0.000 ± 0.0	0.000	0.000
BaselineOne	32.43 ± 0.3	0.794	0.589
Random	26.64 ± 0.3	0.653	0.587
LinearQN	15.24 ± 0.2	0.324	**0.381**
DeepQN_{small}	13.21 ± 0.2	0.314	0.551
PG_{linear}	**33.75** ± 0.4	**0.844**	0.615
PG_{small}	21.32 ± 0.3	0.540	0.615

Table 3. [Controller: PG_{linear}] Effects of the different environment settings on the controller performance.

Env	Reward	Energy (total %)	Cost (per kWh)
SingleBal1	2.020 ± 0.1	0.697	0.232
SingleBal2	3.668 ± 0.1	**0.938**	0.460
SinglePrice1	1.896 ± 0.1	0.591	0.144
SinglePrice2	1.068 ± 0.0	0.558	**0.114**
Multi2Bal1	24.08 ± 0.2	0.793	0.588
Multi2Bal2	33.75 ± 0.4	**0.844**	0.615
Multi2Charge	69.01 ± 0.7	0.792	0.574
Multi2Price	0.001 ± 0.0	0.000	**0.561**

environments, when optimized for price or balance, respectively. However a pure optimization for cost or energy did not manage to perform better than the baseline in the multi station case, as seen in table 3.

An analysis of the distribution of the prices at which the energy is delivered, shows that PG_{linear} is able to shift the charge timing towards mid-day, when price approaches zero. It still charges when the price is at its maximum, which is desired behavior, as many charging sessions are not or only little optimizable.

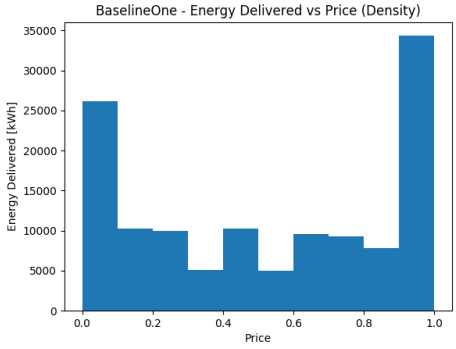

Figure 7. Distribution of energy delivered with respect to price. As seen on evaluation set for controller BaselineOne. [Environment: SingleBal2]

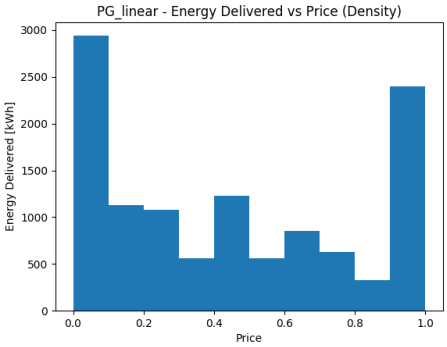

Figure 8. Distribution of energy delivered with respect to price. As seen on evaluation set for controller PG_{linear}. [Environment: SingleBal2]

A visualization of a smart charging algorithm deferring its charging after a vehicle arrival can be seen in Figure 9. The cars arriving before between 6AM and 10AM are not immediately charged as the controller has learned they will likely stay until noon and has opportunity to charge them at a cheaper price (higher renewables), shown by the red dotted line.

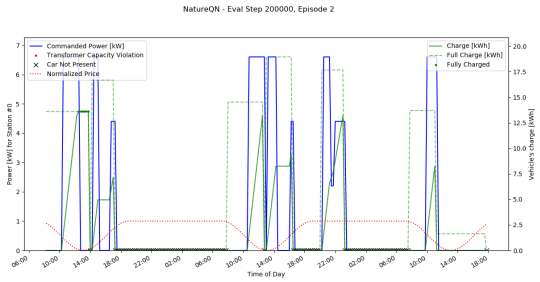

Figure 9. For EVs arriving in the morning, DeepQN controller defers their charge to late-morning/afternoon hours with greater solar penetration

Comparing the performance between the multi-layer perceptron (MLP) implementations of DQN with Q-learning and SARSA, SARSA tends to have more stable training and lower variance, as shown in Figure 10. This may be due to the off-policy nature of Q-learning, as compared to SARSA, which is on-policy. For both algorithms, their Double variant has better performance in terms of average reward, likely due to maximization bias in their standard implementations. Figure 11 in appendices shows the performance of DQN SARSA compared to the baseline for the same episode. The baseline figure has several instances of exceeding the transformer capacity rating whereas SARSA has successfully trained to balance power between the three stations and only increase charging rate when not all three cars need charging simultaneously.

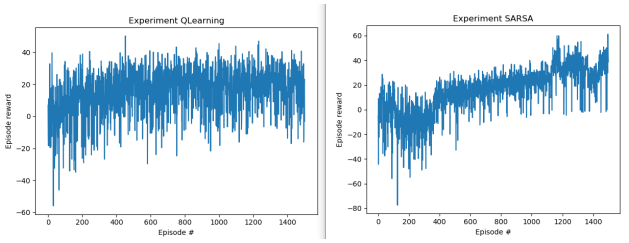

Figure 10. SARSA DQN (MLP) has faster and more stable training than Q-learning DQN (MLP)

5. Conclusion

Our results show reinforcement learning based charging of EVs may have potential to increase renewables generation and prolong lifetime of grid assets. Coignard et. al. have found one-way optimized charging of EVs provide services equivalent to 1.0GW stationary storage and $1.5 billion in value in California(Jonathan Coignard, 2018). This state-level optimization can be realized locally by smart charging algorithms at charging facilities to optimize for charging during peak renewables generation. Hilshey et. al. have found a 25kVA transformer will endure accelerated lifetime loss when supporting charging for beyond 10 EVs (Alexander D. Hilshey, 2011). Smart charging can be achieved through reward function design to mitigate transformer aging while supporting more cars, compared to the baseline naive charger. We also have learned through experimentation in this project that balancing the reward function and tuning controllers can be a difficult task in a multi-objective setting with competing goals.

The simulator environment, experiment testing wrapper, and controllers created in this project open many avenues for future work. A greater number of stations, beyond three tested in our project, can evaluate the scalability of our algorithms. Scenarios with on-site stationary battery storage to help shave peak demand and perform price arbitrage would also be of immediate practical value to EV charging stations in present day. Similarly, quantifying the economic benefit of having on-site solar generation when paired with smart-charging stations and stationary battery is also a priority of charging companies. With greater computing resources, the authors would like to extend this work to use a greater amount of the provided charging data set and create generalizable models for all geographic regions. Theoretically, the authors would like to explore a multi-agent framework, which may be well-suited to represent each station at one facility or each facility in a city. Additionally, branching may be useful to reduce the action space and better map each station's action to its individual reward contribution.

Acknowledgements

We would like to acknowledge the CS 234 teaching staff for their instruction and project support throughout the academic quarter. We would also like to thank the Grid Integration Systems & Mobility (GISMo) Lab at SLAC National Lab for providing access to their ChargePoint electric vehicle session data for this project.

References

Adriana Chis, Jarmo Lunden, V. K. Optimization of plug-in electric vehicle charging with forecasted price. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

Alexander D. Hilshey, Paul D. H. Hines, J. R. D. Estimating the acceleration of transformer aging due to electric vehicle charging. *2011 IEEE Power and Energy Society General Meeting*, 2011.

C.X. Jiang, Z.X. Jing, X. C. T. J. Q. W. Multiple agents and reinforcement learning for modelling charging loads of electric taxis. *Applied Energy*, 222:158–168, 2018.

Elena Mocanu, Decebal Constantin Mocanu, P. H. N. A. L. M. E. W. M. G. J. G. S. On-line building energy optimization using deep reinforcement learning. *IEEE Transactions on Smart Grid*, pp. 1–1, 2018.

Jonathan Coignard, Samveg Saxena, J. G. D. W. Clean vehicles as an enabler for a clean electricity grid. *Environmental Research Letters*, 13(5), 2018.

Stijn Vandael, Bert Claessens, D. E. T. H. G. D. Reinforcement learning of heuristic ev fleet charging in a day-ahead electricity market. *IEEE Transactions on Smart Grid*, 6 (4):1795 – 1805, 2018.

T.Bunsen, P.Cazzola, M. L. S. R. J. and J.Teter. Global ev outlook 2018. Technical report, International Energy Agency, 2018.

Code and Data

Our code base is primarily in Python (3704 lines). We used neural networks from tensorflow and scikit-learn libraries, numeric computation using NumPy, and data visualization using Matplotlib. The code base is made available on the following GitHub repositories:

- **gym_ev_charging**
- **controller_ev_charging**

The ChargePoint charging data cannot be shared, as it is owned by SLAC GISMo, however, the authors recommend

Pecan Street Data, available to the academic community, as an alternative source.

Contributions

Oskar designed and implemented the workflow for experiments, the Controller class, Configuration classes and their interfacing, including the command line interface, training and evaluation procedures. He integrated and trained the Controllers for Policy Gradient and Deep/Linear Q-Network architectures and some baselines. He created the test gym environment and extended many of the EV gym's functionalities, such as reward function, featurization and configuration options. Further, he took care of all the data preprocessing and transformation.

Will wrote and designed much of the original simulator and handled porting the simulator and experiment code into the OpenAI Gym environment framework. He wrote the code inside and outside the environment for easily recording various model statistics and metrics and reporting them for model evaluation. He also experimented with the reward function and various model and environment hyperparameters to improve performance.

Justin developed the DQN Q-Learning MLP and DQN SARSA MLP controllers and their Double variants,data visualization, experiment architecture and controller template on initial simulator, and data sampling. He also contributed to reward function design, evaluation statistics, experimenting and model tuning, implementing delayed reward function, and V2G features.

Appendices

Table 4. The different environment setups.

env	gamma	reward w. (c, p, t)	description
SingleBal1	0.99	(1,1,0)	balance
SingleBal2	0.95	(2,1,0)	balance
SinglePrice1	0.9	(1,1,0)	min price
SinglePrice2	0.98	(1,2,0)	min price
Multi2Bal1	0.99	(1,1,0)	balance, $TL = 0.5$
Multi2Bal2	0.95	(2,1,0)	balance, $TL = 0.5$
Multi2Charge	1.00	(1,0,0)	charge, $TL = 0.5$
Multi2Price	0.9	(1,1,0)	min price

Table 5. [Controller: LinearQN] Effects of the different environment settings on the controller performance.

Env	Reward	Energy (total %)	Cost (per kWh)
SingleBal1	1.787 ± 0.1	0.754	0.381
SingleBal2	3.668 ± 0.1	0.938	0.460
SinglePrice1	1.843 ± 0.1	0.623	0.218
SinglePrice2	0.933 ± 0.0	0.606	0.195
Multi2Bal1	3.971 ± 0.1	0.135	0.321
Multi2Bal2	15.246 ± 0.2	0.324	0.381
Multi2Charge	41.270 ± 0.5	0.470	0.447
Multi2Price	1.336 ± 0.0	0.101	0.389

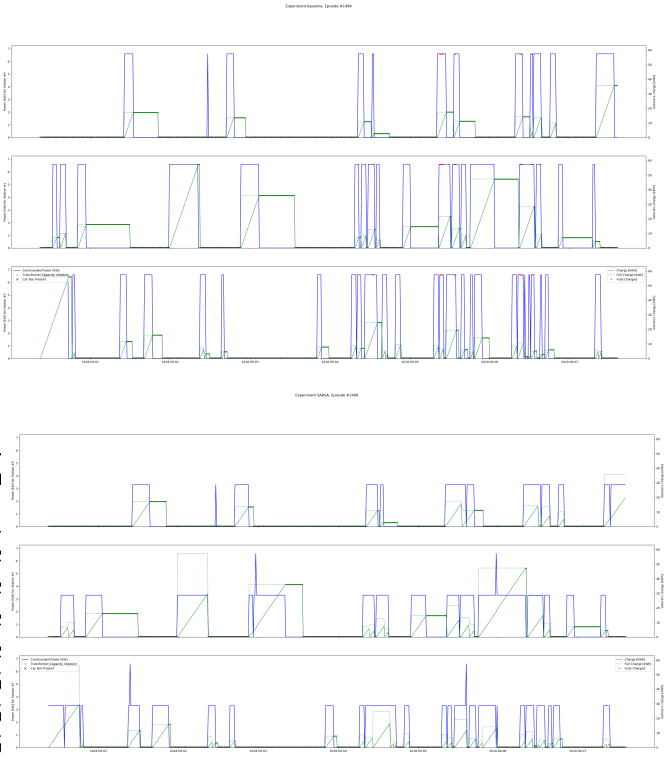

Figure 11. SARSA DQN (MLP) (bottom) learns to mitigate transformer aging by ensuring the charge action of the three stations never exceeds the transformer's rated capacity of 16.5kW, as compared to the Baseline controller (top) which has several instances of violation (red dots)