Electric Vehicle Charge Scheduling Using Monte Carlo Tree Search

Arec Jamgochian

***Abstract*—Integrating electric vehicles (EVs) safely into the electricity grid will require smart algorithms for scheduling charging. These algorithms will have to reason about car state-of-charge, grid constraints, and uncertainty in future electricity pricing. In this paper, we formulate the problem of charging a large number of EVs with future price uncertainty as a continuous-state Markov Decision Process (MDP). We use feedforward and recurrent neural networks to perform system identification offline to learn price transition models. We incorporate these models with Monte Carlo Tree Search with Double Progressive Widening to solve for best actions online. Finally, we turn our methods into a closed-loop controller, which we evaluate in three pricing scenarios.**

I. Introduction

ELECTRIC vehicle (EV) charging requires significant electricity consumption. Utilities are already observing a trend of sharper electric demand ramp-ups in shorter time periods. To meet unanticipated electricity demand, utilities often have to rely on "peaker" power plants, which typically use dirtier fuel sources for electricity production. As an increasing number of EVs are adopted, an increasingly larger burden will be placed on the electricity grid, both in terms of net power draw and in terms of stochastic unanticipated need. Mitigating these infrastructural issues is necessary for widespread EV adoption. This motivates a need for smart charging algorithms that can schedule the charging of a large number of electric vehicles.

Minimizing the cost of charging while considering constraints from the grid and the objective of charging all cars in a timely fashion requires information about the state of charge for each vehicle, how those charge levels evolve given charging infrastructure, and the current and future costs of electricity. While state-of-charge evolution may be known a priori, the evolution of electricity costs may not. Specifically, we consider the Real Time Pricing (RTP) scheme in which the cost of electricity is proportional to the total demand a grid experiences. In this work, we assume charging does not have enough share in total power consumption to affect this total demand, but this assumption can be relaxed easily.

In this paper, we formulate the problem of charging a set of electric cars given constraints on the grid and uncertainties in future cost as a Markov Decision Process (MDP). Given the assumption that charging does not affect price, all we have to do to determine when to charge is to predict future price. We do this by using system identification offline to learn different transition models for price. Specifically, we evaluate the use of autoregressive transition models in which we use a) feedforward neural networks (NN) and b) long short-term memory (LSTM) to map past prices to a Gaussian distribution over next-step price.

Since price and state-of-charge are continuous, we must rely on approximate solution methods to choose the action in any given state. This is done by implementing Monte Carlo Tree Search (MCTS) with Double Progressive Widening (DPW) [1]. While progressive widening on the state space would be sufficient, adding progressive widening on the action space would allow scaling up to charging a large number of n cars, in which there are 2^n possible decisions.

The main contributions of this paper can be summarized as:

- Formulating electric vehicle charge scheduling as a MDP
- Performing system identification offline using a NN and LSTM to learn a transition model
- Implementing MCTS w/ DPW with our learned transition models to solve for best actions online

The remainder of this section introduces necessary background, Section II formulates EV charge scheduling MDP, Section III describes our experiments, and we conclude in Section IV.

A. Background

1) EV Charge Scheduling: The common problem formulation of charging EVs considers the binary action for whether to charge each car at each time step given grid constraints, a cost function, and a charging goal. Much work has been done considering EV charge scheduling as a convex optimization problem, even work including uncertainty in different grid factors [2]–[6]. Unfortunately, the charge profile of an electric profile is non-convex, rendering the problem non-convex. An example charging profile depicting power draw as a function of time for a Nissan Leaf can be seen in Fig. 1. Typically, the power draw per is a function decreasing in a car's state-of-charge (SOC), and includes a very low-draw "trickle mode" at the end of charging. This simple non-convexity makes the use of control scheduling algorithms that rely on convex optimization (e.g. Model Predictive Control) intractable.

2) Markov Decision Processes: An additional level of complexity comes from taking into account the price of charging. Naturally, it makes sense to schedule charging when the cost of electricity is cheap, since this minimizes user costs and avoids grid overloads. In a real-time pricing scheme (RTP), the cost of charging is proportional to the total demand on the grid. It thereby becomes essential to forecast total demand in order to make better decisions about when to charge.

Given the non-convexities and uncertainty in the problem, it is appropriate to make an alternative formulation as a Markov

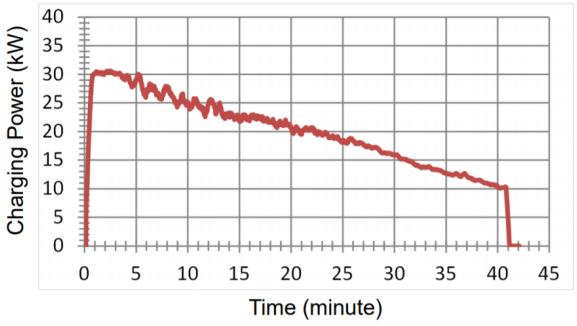

Fig. 1. An example charging profile. Power drawn during charging is depicted as a function of time for a Nissan Leaf.

Decision Process (MDP). MDPs are fully described by a state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ describing the reward emitted with each decision, and the transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ describing the probability of transitioning between states. The goal of such an undiscounted, finite-horizon MDP is to find a policy π of actions to take that maximizes expected reward, or value at a state $V(s) = \mathbb{E}[\sum_{t=0}^{T} R(s_t, a_t = \pi(s_t), s'_t) \mid s_0 = s]$. Discrete state-action MDPs can be solved exactly offline using dynamic programming, while continuous-state MDPs can be solved approximately.

3) Monte Carlo Tree Search with Double Progressive Widening: Often times, an optimal action for the entire state space is not necessary. If all we care about is an action at a particular state s_0, then we only need to concern ourselves with states reachable from s_0. Monte Carlo Tree Search (MCTS) is an online approximate solution method that takes advantage of this. MCTS creates a search tree of states and actions up to a max depth D. It keeps track of the number of times a particular node has been visited, and running average of the value at each node given reward obtained from a generator (s', r) $G(s, a)$. In the Upper Confidence Trees (UCT) implementation of MCTS [7], the action chosen at each node that has been visited previously is $a = \arg\max_a Q(s, a) + c\sqrt{\frac{\log \sum_a N(s,a)}{N(s,a)}}$, where $Q(s, a)$ is the tabular form of expected value, $N(s, a)$ is the count for vising a certain state-action node pair, and c is an exploration constant. Searching through actions this way can be seen as striking a balance between exploration and exploitation.

Note that UCT applied to continuous state or action spaces would make for very shallow trees, as the probability of visiting a state twice is zero. This is allayed by progressively widening on the state and/or action space (referred to as "double progressive widening" (DPW) when done on both). The idea of progressive widening [8] done on the state space for example is only sample a new state from a node if the value $k_s N(s)^{\alpha_s}$ is larger than the current number of branches at that node, otherwise sample from those branches. Note that k_s, α_s, and likewise k_a and α_a are hyperparameters determining the widening on the state and action spaces respectively. DPW has been shown to be very effective at solving problems with continuous state-action spaces.

II. Problem Formulation

In this paper, we cast the problem of scheduling the charging of C cars as a continuous-state Markov Decision Process (MDP). At each time step, we make the binary to decision to charge each car: $\mathcal{A} = \{\{0, 1\}^C\}, \mid \mathcal{A} \mid = 2^C$. Our state space is the cartesian product of the state-of-charge of each car and all the variables required to derive price, $\mathcal{S} = \mathcal{S}_{SOC} \times \mathcal{S}_{Price}$, where $\mathcal{S}_{SOC} = \{(0, 1)^C\}$.

If the decision is made to charge car c, then the level of charge in the car soc_c increases by $f(soc_c)$, where f is determined by the charging profile, known a priori. The level of charge in each car is fully observed, and it is assumed for this project that each car has the same charging profile. The power drawn off the grid in a single step is assumed proportional to $soc'_c - soc_c$.

The reward function we use to calculate true instantaneous reward is a multi-objective function maximizing the final charge of each car, the cost of electricity, and the infrastructural cost of charging many cars simultaneously:

$$R(s, a, s') = R_{elec} + \lambda R_{grid} + \beta R_{charge} \tag{1}$$

$$R_{elec} = -p \sum_c (soc'_c - soc_c) \tag{2}$$

$$R_{grid} = -(\sum_c a_c)^2 \tag{3}$$

$$R_{charge} = \begin{cases} 0 & s' \notin s_{terminal} \\ -\sum_c (0.95 - soc'_c) & s' \in s_{terminal} \end{cases}, \tag{4}$$

where p is the price of electricity in the current state and can be derived from $s_{Price} \in \mathcal{S}_{Price}$. The first term measures the immediate electricity cost of charging, the second term punishes the charger for charging many cars simultaneously (acting as a grid constraint), and the third term punishes the scheduler for cars that do not have full charge at the end of the sequence (determined if s' is a terminal state). The most practical implementation of terminal states is to include time in the state space, increment it by one at each time step, and terminate once the simulation end time is reached. λ and β are hyperparameters that trade off the relative importance of each of the three terms in our reward function.

Given the fully-defined MDP above, we can solve for the best action from a given state online using Monte Carlo Tree Search w/ Double Progressive Widening [8]. While technically we only need progressive widening on the state space, progressive widening on the action space would allow us to scale to a larger number of cars, as $\mid \mathcal{A} \mid = 2^C$. MCTS run once will tell us a best-case, open-loop string of actions to run from an initial state s_0. We can formulate a closed-loop method for charging in a fixed time window T as follows:

1) Run MCTS-DPW to depth T with learned transition model to obtain best action a from state s
2) Take action a, observe next price p, and use this information to update true state s
3) Decrement T by one and return to 1)

We note that p evolves independently from our charging actions, and our process as such is non-Markovian, resulting in

variable reward at each state. This is why we must incorporate electricity cost into the MDP state, and explore various ways of making predictions on future cost. Since we have time series data of electricity demand (and thereby price), we can perform this system identification offline, and apply our learned models as a transition function. In this paper, we evaluate three different methods of system identification: a naive approach, an autoregressive feedforward neural network, and a recurrent neural network. The implications of each on the price state space are described below.

A. Naive Price Transitions

The most basic price transition model would be predicting that next state price is equal to current price. This is also equivalent to predicting that the price will never change, as the predicted price at depth D would also be equal to the current price. Implementing this require s_{Price} to contain only one value, the current price p. Each iteration of MCTS-DPW works with a single price until the very end. Then s_{Price} gets updated with the true next-step price in the update step of the closed-loop controller.

B. Feedforward Neural Network

A better method would be incorporating the past n prices into the state at t, and learning a neural network to map those costs to values that parameterize a distribution over cost at $t+1$. A neural network is a model which alternates between taking linear combinations of nodes in a given layer to make a new layer, and applying nonlinear activation functions on the new layers. Neural networks are frequently used today because of their ability to approximate any function. An simple example can be seen in Fig. 2.

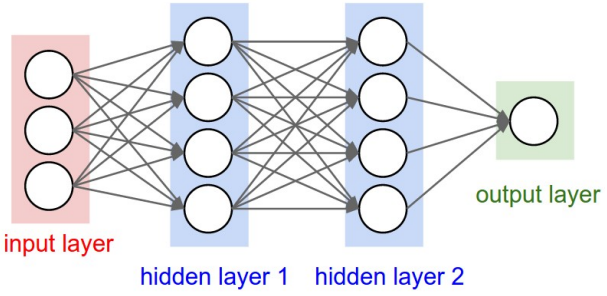

Fig. 2. An example neural network with two hidden layers and one output. Our method has two outputs, the mean and standard deviation of a Gaussian distribution over next price.

One can use a neural network to predict next state by learning a mapping from n past prices to a mean and standard deviation of a Gaussian distribution over next price. This mapping must minimize the expected negative log likelihood of the data. Learning can done by calculating the negative log likelihoods of batches of the data given fixed model weights (i.e. the likelihood of the true next price being drawn from a Gaussian with mean and standard deviation derived from the neural network), and then using first order methods to update the model weights accordingly.

To implement this, we need to store the past n prices in s_{Price}. For our transition model, the next price would be drawn from a Gaussian distribution defined by mean and standard deviation obtained by running s_{Price} through the neural network and the next state s'_{Price} would shift the prices appropriately. In the update step of the closed-loop controller, we simply add the new true price and delete the oldest price to s_{Price}

C. Recurrent Neural Network

A final method would be to use a recurrent neural network (RNN) such as long short term memory (LSTM) [9] to keep track of a hidden state that could map to parameters of a next step price distribution. A recurrent neural network updates a hidden state based on input it receives, a hidden state which can easily be mapped to a set of outputs. LSTM propogates a hidden state h and a cell state c to allow for a slower type of memory and to enable better gradient flow through long sequences. LSTM the architecture for which can be seen in Fig. 3, has proven especially useful for natural language processing.

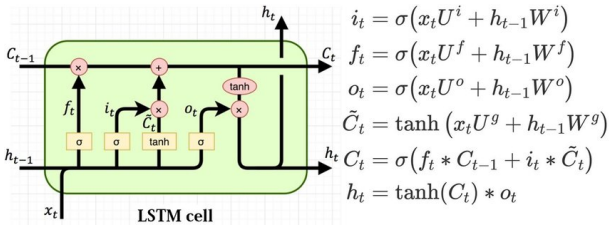

Fig. 3. Long short-term memory cell architecture.

This model could be learned offline in a method similar to the one described in the last section. The output layers would map the hidden state to a mean and standard deviation that parameterize a Gaussian distribution over next-state price. At each step, the LSTM would take as input the current price. One could then learn model weights to minimize the expected negative log likelihood of the data, that is the likelihood of the true next price given the predicted mean and standard deviation.

To implement a learned LSTM model with MCTS-DPW, we need to store current price, along with the current hidden and cell states in s_{Price}. For our transition model, the next price would be drawn from a Gaussian distribution defined by mean and standard deviation obtained by running the current hidden state through the mapping from hidden state to distribution parameters. The next state would consist of this sampled next price, and the updated hidden and cell states from running the sampled price through the LSTM. In the update step of the closed-loop controller, we would need to add the true next price, and run it through the LSTM to get the true next hidden state and cell state.

III. Experiments

In our experiments, we first take to offline system identification to formulate the feedforward neural network and the long short-term memory transition models. With each of these models, $T(s_{Price}, a, s'_{Price}) =$

$\mathcal{N}(s'_{Price}; \mu(s_{Price}; \theta), \sigma(s_{Price}; \theta))$, where θ encompasses each models parameters. Again, to learn each model, we compute the mean negative log likelihood given fixed model weights for a batch of data, and use gradient methods to update these weights. We use Adam [10] as our weight optimizer. We use LeakyReLU's as our nonlinear activations. To ensure the models can learn valid (positive) standard deviations, we add a softplus layer before the standard deviation outputs.

To model prices, we use a representative electricity demand dataset, namely the OpenEI residential load dataset [1] which contains average hourly load data at different locations across the United States for the period of one year. We specifically look to data from the past 6 hours to predict the next one. After some model parameter tuning which we evaluate using a holdout dataset, we choose the fix the following model architectures:

- NN: 3 hidden layers, each with 40 units
- LSTM: hidden/cell state size of 40, output neural network has 2 hidden layers each with 20 units

We report the mean absolute percentage errors (MAPE) of the next-step mean prices obtained from both methods on a holdout set in Table I. MAPE is defined as

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|,$$

where A_t is the actual value and F_t is the forecast value.

TABLE I
NEXT-STEP PREDICTION ACCURACY

Method	MAPE (%)
Naive	13.5
Neural Network	3.8
Long Short-Term Memory	1.5

Once the transition models are learned, we implement MCTS-DPW with each transition model, for which we use the solver implementations in POMDPs.jl and MCTS.jl [11]. We use a state-of-charge transition function $f(soc_c) = .15/(1 + \exp(10(soc_c - .8)))$, which describes the increment to each car's state-of-charge if the charge action is selected. This function is loosely based on charging profiles obtained from a Tesla forum[2].

To test our methods, we sample three 20-step price scenarios from our electricity consumption dataset. The scenarios that were sampled can be seen in Fig 4. We run our closed-loop controllers on five cars that all start empty using the different transition and state update models on each scenario. We run five trials for each scenario-method combo, and report the average reward obtained, and time per trial in Table II. We also report the reward if we were to take a naive charge-on-arrival policy of charging each car to full immediately. A list of hyperparameters used can be seen in Table III. The value of $\beta = 200$ artificially constrains all cars to be full by the end of the simulation.

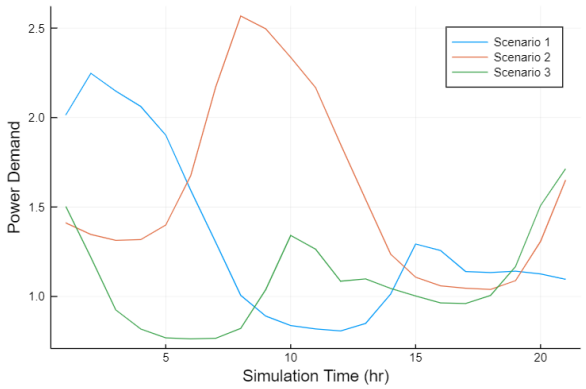

Fig. 4. The three price scenario used in the experiments.

TABLE II
SIMULATION RESULTS

Method	Sim. Time (s)	Scen. 1 Reward	Scen. 2 Reward	Scen. 3 Reward
Charge-on-arr.	0.0005	-116.3	-116.5	-114.3
MCTS-Naive	1.4	-85.3 ± 1.7	-88.0 ± 0.9	-83.4 ± 1.3
MCTS-NN	21.5	-86.0 ± 1.1	-89.5 ± 3.2	-86.1 ± 3.2
MCTS-LSTM	54.1	-86.1 ± 1.2	-89.0 ± 1.0	-85.6 ± 1.8

TABLE III
PARAMETER VALUES

Name	Description	Value
C	Number of cars in simulation	5
λ	Grid Constraint Weight	0.5
β	Final Car Charge Weight	200
N	Number of Iterations of MCTS	1000
c	Exploration Constant	10
k_s, k_a	DPW Scale Factors	10
α_s, α_a	DPW Exponents	0.5

The first thing one notices is that even though all MCTS policies perform better than the charge-on-arrival policy, there is not much improvement between them. To show this behavior further, the models were run additionally with $C = 1$car,$\lambda = 0.0$, so that the solver only had to fully charge one car while only considering price. These experiments showed the same behavior, with minimal differences between the policies for the three MCTS methods.

This is counter-intuitive since there is a significant improvement in next-step price prediction accuracy between the models. The most likely explanation for this is that even though next-step errors are low, many-step errors are high as the errors compound. The price estimate at $t + 10$ obtained by sampling from an autoregressive neural network 10 times is just as bad as predicting $p_{t+10} = p_t$. Possible solutions to this are discussed in the following section.

[1] openei.org/datasets/files/961/pub/RESIDENTIAL_LOAD_DATA_E_PLUS_OUTPUT/

[2] teslamotorsclub.com/tmc/threads/updated-model-3-charging-profiles-durations.145054/

IV. Discussion and Future Work

Integrating a large number of electric vehicles into our electricity grid safely requires the implementation of smart charge scheduling algorithms. These algorithms must determine how to charge cars given grid constraints, different charging patterns, and the cost of electricity. In a scheme where future electricity price is not known a priori, we must predict it to make the best decisions. Real Time Pricing schemes price electricity proportional to total demand on a grid, so in this paper we focus on predicting demand.

Charging is usually framed as an optimization problem. However because of uncertainties and non-convex dynamics, it makes sense to model it as a Markov Decision Process. In this paper, we formulated electric vehicle charging as an MDP, taking into account the state of charge in each car and a price state that could be used to predict future prices. We examined performing system identification offline to learn a transition model between prices using feedforward neural networks and long short-term memory. We took to solving the resulting continuous-state large-action MDP online by using Monte Carlo Tree Search with Double Progressive Widening.

We found that although there were significant improvements in next-time price predictions in our models learned offline, there were not significant improvements in overall closed-loop controller performance. The most likely reason for this is that predicting good next-step prices doesn't help you much in the long run, as errors still compound. A better approach would be to simultaneously predict all future prices. This could be done by keeping a belief over possible future price curves, and updating that belief as prices are observed. This would warrant turning the problem into a mixed-observability Markov Decision Process (MOMDP), where the true price state is the future price curve and it is only partially observed.

An alternative approach may be folding the time into state space more directly, and making better predictions of price based on time. Preliminary experiments show that priced depends strongly on time-of-day, day-of-week, and month-of-year, all information that we are excluding in our models but can fold in very easily. If we made baseline predictions from just the time, then we could could allow the models mentioned in this paper to focus exclusively on learning the residuals. This can be viewed as focusing on learning a distribution over how far away we will be from the normal price (which would already be known).

References

[1] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *Learning and Intelligent Optimization (LION)*, 2011, pp. 433–445.

[2] J. C. Mukherjee and A. Gupta, "A review of charge scheduling of electric vehicles in smart grid," *IEEE Systems Journal*, vol. 9, no. 4, pp. 1541–1553, 2015.

[3] O. Sundström and C. Binding, "Optimization methods to plan the charging of electric vehicle fleets," in *International Conference on Control, Communication and Power Engineering*, 2010, pp. 28–29.

[4] O. Sundstrom and C. Binding, "Flexible charging optimization for electric vehicles considering distribution grid constraints," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 26–37, 2012.

[5] T. K. Kristoffersen, K. Capion, and P. Meibom, "Optimal charging of electric drive vehicles in a market environment," *Applied Energy*, vol. 88, no. 5, pp. 1940–1948, 2011.

[6] M. E. Khodayar, L. Wu, and M. Shahidehpour, "Hourly coordination of electric vehicle operation and volatile wind power generation in SCUC," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1271–1279, 2012.

[7] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European Conference on Machine Learning (ECML)*, 2006, pp. 282–293.

[8] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 433–445.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "POMDPs.jl: A framework for sequential decision making under uncertainty," *Journal of Machine Learning Research*, vol. 18, no. 26, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-300.html