Coordination of Distributed Energy Resources without Power Grid Models using Reinforcement Learning

Thomas Navidi
School of Electrical Engineering
Stanford University
Email: tnavidi@stanford.edu

***Abstract*—In this paper, I explore the implementation of a deep deterministic policy gradient (DDPG) learning agent to the domain of controlling distributed energy resources in a distribution network. This domain has traditionally been dominated by optimal control strategies such as [1], but these strategies can be slow for very large power networks, and they all require the knowledge of grid parameters, which is often unobtainable in practice. The usage of DDPG allows the storage units to learn a policy which minimizes electricity operating costs through performing energy arbitrage, and promotes network reliability by charging or discharging power to balance the system. These two features are key to promoting the increase of renewable energy generation. This work is an extension of my previous quarter's CS-238 final project which aims to solve the same problem. This paper builds on that work in 4 key ways: (i) expanding the state and action space to consider an entire day's worth of data. (ii) Training the actor and critic networks with labeled data in a supervised learning approach before performing the DDPG algorithm. (iii) Data augmentation was used in training to increase the number of examples to cover a broader range of possibilities. (iv) Most notably, the rewards were generalized to fit any price or power grid structure. These generalizations of the capabilities of the learning algorithm were realized without a loss in performance.**

I. Introduction

A grand challenge of the future electric grid is how to best coordinate distributed energy resources (DERs) to achieve grid reliability while allowing renewables to ultimately replace fossil fuel sources. Coordinating millions of connected devices in the real world, however, is challenging due to a combination of cyber-physical constraints including electric power grid physics, device communication delays, and data privacy concerns. My previous work has demonstrated the capabilities of a two-layer model predictive control (MPC) architecture that coordinates DERs while respecting these constraints [1]. However, this algorithm requires a model of the physical distribution grid, which is often unavailable in the real world, and involves the computation of a lengthy optimization problem, which can take long for large power networks. No known algorithms are able to effectively manage grid reliability through coordination of DERs without knowledge of the grid; however, my final project will attempt to learn a policy for control of DERs without having knowledge of the grid. Execution of this policy will also be much faster than the computation of an optimization problem.

II. Problem Formulation

In the problem, there are thousands of homes connected together by the grid, and several of these homes are equipped with rooftop solar arrays, and controllable battery storage devices. The battery storage devices can charge, which increases net power consumption, or discharge to decrease net power consumption. The power consumption of homes across the grid affects the voltages at each home and can cause them to become too high or too low. Therefore, it is up to the battery storage devices to adjust the net power of each home in a way that ensures voltage is within operating bounds. Unfortunately, when the power grid structure is unknown, the effects of the power demand on grid reliability are also unknown. They can only be known by taking measurements on the system after the battery storage decision has been determined. This problem can be framed as a reinforcement learning problem, where battery charging decisions will be made, and the implications of those actions will be determined and effect future decisions. In order to avoid confusion, neural networks will be referred to as neural nets or network, and the power network will be called the power grid or grid.

A. State and Action Space

The state and action space for our reinforcement learning problem are both in the continuous domain with the power demanded throughout the grid as the state, and the power charged or discharged by the battery as the action. Each node in the grid consumes power that varies throughout the day, which is discretized hourly to make 24 points in a day. Each node and daily power profile are vectorized to make a vector of length $24N_{nodes}$ where N_{nodes} is the number of nodes in the power grid. This is the state vector because the power demanded in each hour is what determines the voltage levels. The action vector is the daily battery charging power profile. This vector has a length of $24N_{storage}$ where $N_{storage}$ is the number of storage nodes in the power grid. This is an improvement over the previous implementation where only a single hour was considered at a time for two main reasons. First, when considering only a single hour, actions must be greedy because future states of the system are unknown. In this problem, greedy actions perform significantly worse than those that consider a broader horizon. Second, the state of charge

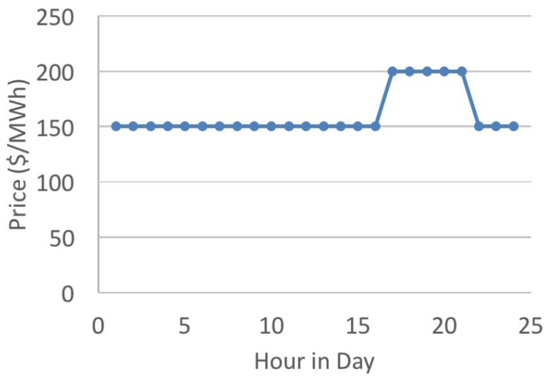

Fig. 1. Time of use prices throughout the day

of the battery does not need to be included as one of the state variables, since the battery can easily complete a cycle over the day and return to the same state of charge. In the previous implementation the state of charge was included to prevent the storage units from charging while full or discharging while empty. This simplifies the problem and generalizes the capability of the controller. The use of data augmentation and supervised learning enabled the expansion of the state and action space without loss of performance.

B. Objective

There are two metrics of success to optimize over when choosing the best charging strategy. The first objective is to minimize the cost of electricity. This is done through energy arbitrage, which is buying electricity at a low price and selling it at a high price. Buying electricity occurs when the storage unit charges power from the grid, and selling occurs when the unit discharges power onto the grid. The value of arbitrage profit is shown in equation (1) where p is the vector of electricity prices throughout the day and u is the vector of battery charging and discharging. Figure 1 shows the price structure used for this paper; however, any time varying price structure would work.

$$ARB = p^T u \tag{1}$$

The second metric of success is related to the electric power quality, particularly with respect to voltage variations, which occur when the voltage magnitude at any house, substation, or other node in the network is above or below the nominal voltage by greater than 5 percent. This metric is meant to support grid reliability. The penalty metric for voltage violations for a single node and time step is the sum of squared deviations of the voltage from the 5 percent bounds.

Time is discretized over T time steps and is indexed by $t \in [1 : T]$. Each node in the network is indexed by $i \in [0 : N]$. Equation 2 gives the formulation where V_{it} is the per unit voltage at node i and time t. Figure 2 shows the metric for a single bus and time step. The choice for this metric reflects our desire to curb voltage violations caused by the introduction of RDGs.

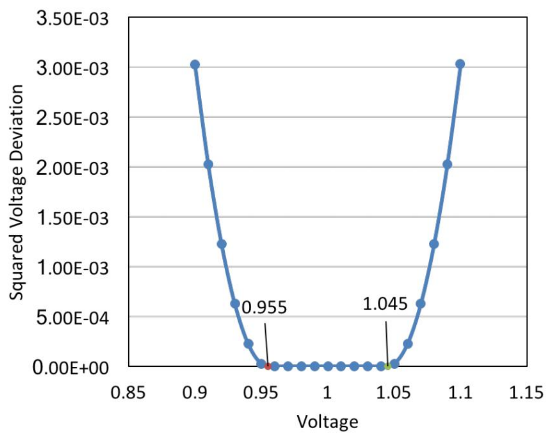

Fig. 2. Lack of electric power quality metric measured as squared voltage deviations

$$\sum_{i=0}^{N}\sum_{t=1}^{T}(\max(V_{it} - 1.05, 0) + \max(0.95 - V_{it}, 0))^2 \tag{2}$$

These two metrics can be combined to create a measure of total performance as shown

$$\begin{aligned}&\frac{1}{2}\sum_{\tau=1}^{24} p_\tau \cdot \Re(s_{0\tau}) - \\ 150 &\sum_{i=0}^{N}\sum_{\tau=1}^{24}(\max(V_{i\tau} - V_{\text{tol}+}, 0) + \max(V_{\text{tol}-} - V_{i\tau}, 0))^2\end{aligned} \tag{3}$$

Equation (3) will be used as the reward function for the critic network throughout the learning process. This extends on my previous work by providing a single constant reward function to the learning algorithm regardless of the underlying price structure. My previous project hand tuned the reward function hourly in order to take advantage of human known features of the price structure. This generalized formula is an improvement because it allows the networks to learn on any price structure. By incorporating data augmentation and supervised learning this generalization comes at no cost to performance.

Finally, the total performance metric will be compared to an optimal controller where the network topology and all other variables are known ahead of time. Also, the performance can be compared to a system that does not incorporate coordinated control where each battery only performs its own local optimization. This is the baseline heuristic that will be used for supervised learning as described in the next section. The system will hopefully be only slightly worse than the fully deterministic optimal case, and much better than the uncoordinated case.

C. *Case Study*

The distribution grid experimented on has 7 nodes with 4 houses, a transformer, a commercial building, and a substation. One of the houses has a rooftop PV system and a battery storage system. The solar data was obtained from an NREL dataset and scaled to match typical rooftop PV systems, while the building demands were obtained from PG&E. The battery storage charging and maximum capacity were chosen to match approximately the capabilities of the Tesla PowerWall 2 for realism.

III. Initial Supervised Learning

In order to give the random exploration process a solid starting point when searching for battery charging profiles, the actor network was trained using supervised learning before starting the DDPG reinforcement learning algorithm. However, there are 2 major challenges involved with acquiring and labeling data for the supervised learning approach.

A. *Data*

The first challenge comes from acquiring a sufficient amount of data. The dataset I received comes from a PG&E database containing the power demanded of thousands of homes over the course of 5 months recorded at an hourly time resolution. Unfortunately, 5 months of data only provides 150 days, which is not enough training examples to train our neural net to have sufficient generalizing capability. Therefore, the data was augmented by adding Gaussian random noise with 0 mean and standard deviation equal to 20% of the original datas standard deviation. This noise provides a realistic representation of actual power data, which varies approximately within this range. After data augmentation, the total number of training examples is 15000 days. The 150 days of actual data are split into 75 days for dev and 75 days for testing.

The second challenge involves labeling our data with the optimal control strategy. Since the optimal algorithm in [1] takes too long to compute for large power grids, I must use a heuristic to develop the control strategy for the training data. Also, I want to avoid using the power grid models, which are needed to run [1]. Fortunately, there is a heuristic method that runs quickly and does not use any grid parameters to run. The heuristic maximizes arbitrage profit, but mostly ignores voltage violations. This is able to achieve performance somewhat close to optimal. The training data will be labeled with the output of the heuristic control algorithm, and reinforcement learning will push performance of the heuristic closer to the true optimal.

B. *Training and Performance*

The goal of the initial supervised training is to train the actor network to imitate the heuristic controller. The neural net is a fully connected feed foreword architecture with 2 hidden layers and an output layer. The 2 hidden layers have 100 and 50 units respectively. The activation function chosen is the exponential linear unit due to its shown ability to train faster for deep networks than the rectified linear unit [2]. The output layer has 24 units with an activation function of tanh. This activation was chosen because the battery control power is bounded between -1 and 1 where 1 represents maximum charging power. The network was implemented as a fully connected architecture instead of an RNN since all future data is considered in the input, and the input space is not prohibitively large. The loss function chosen is the mean squared error or l2 norm between the output and the heuristic charge profile. There is also a l2 regularizer to prevent overfitting to the training data, which consists entirely of augmented data. The regularization hyperparameter was tuned to 0.1. The optimizer is the Adam optimizer due to its robustness and speed. The learning rate was selected to be 0.0001. The neural net was trained over 4000 epochs. The code was implemented in Python using Tensorflow. Table I shows the final mean squared error loss on the 3 sets after training. It also shows the performance for the metrics compared to the optimal and heuristic controllers when evaluated on the real data in the dev and test. Clearly, the actor network is able to do a good job of imitating the heuristic controller, partially due to the simplicity of the heuristic control strategy, which is often the same for many data points. The dev and test mean squared error are lower than the training error because the training data is made of augmented data from the dev and test sets, also, the regularizer has done a good job of preventing over fitting. When evaluating the 3 performance metrics the actor network does an excellent job of mimicking the heuristic, but there is still a lot of room for improvement before reaching optimal performance.

TABLE I
Mean Squared Error for Training the Actor Network and Overall Performance of the Actor Network

Set	Mean Squared Error
Train	5.702E-4
Dev	2.154E-4
Test	1.99E-4

Method	Arbitrage	Voltage	Total
Neural Net	209.414	0.864	-24.893
Heuristic Control	209.992	0.878	-26.704
Optimal Control	209.577	0.295	60.539

Figure 3 show a sample output from the actor network along with the charging action from both the heuristic and optimal controllers. Here we see the actor is able to mimic the heuristic, but there is still a noticeable difference from what is considered optimal.

IV. Reinforcement Learning

The next step is to improve the actor network using a reinforcement learning approach called deep deterministic policy gradients (DDPG) adapted from [3]. This is an actor-critic method for continuous state and actions spaces. This paper takes the deterministic policy gradients method from [4] and applies deep neural networks and other recent advances to ensure the neural network training process is stable. The recent advances to make stable Q networks are adapted from

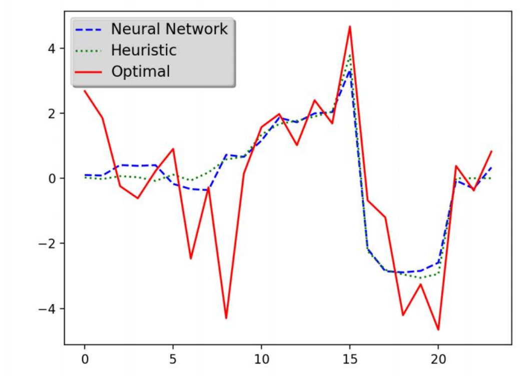

Fig. 3. Sample charge profile output from the actor network compared to the heuristic and the optimal controller.

[5] where a deep Q network is used to train an agent to play Atari video games. The two novel advances are the usage of separate target networks to get the target value of the next state when training the main Q-network, and the use of an experience replay buffer to reduce correlation between training examples for the Q-network.

A. Critic Network

A second fully connected neural network was used to implement the critic portion of the DDPG algorithm. The critic takes as input both the state space, like the actor network, and the action space from the actor network. The output is a scalar value for that state action input pair. The first hidden layer for the state space input is made of 100 units and is shared with the first hidden layer of the actor network. This helps training because the features extracted from the state space are useful to both the actor and critic networks. Another hidden layer of 50 units is used for the action space input. These two layers are concatenated and connected to another hidden layer of 25 units. Each hidden layer his an ELU activation function. The output layer is made up of a single unit with no activation function for predicting the value. The loss function minimized was the mean squared error between the label and the predicted value. Also included was a l2 regularizer with a hyperparameter of 0.1. The Adam optimizer was used for training with a learning rate of 0.001.

Before starting DDPG, the critic network is trained using supervised learning similarly to the actor network. The augmented data is used as input to the actor network which makes a prediction on the action to take in that state. These state action pairs are used as the input training data. The label is the value of the total performance metric (3) for the corresponding state and action. I assume that each day is a terminal state since the storage units return to the same capacity at the end of the day. This dramatically simplifies the value function, which helps improve training. Immediately after the critic network is trained, the DDPG algorithm begins.

B. DDPG Implementation

The main loop of the algorithm begins with the observation of the initial state. The state is fed to the actor network which gives an action and noise is added to this action from a random exploration process. This new action is executed by the storage units, and the reward is observed from the environment. The state, action, and reward tuple is stored in the experience replay buffer. Then, $M = 256$ tuples are randomly sampled from the experience replay buffer and used as a batch to train the actor and critic networks. However for the first 20000 training steps, the actor network is not trained because it has already been trained to mimic the heuristic controller. The critic network needs to learn of unvisited states through random exploration before it can be used to train the actor network. Otherwise, the actor network performance will deviate from the heuristic controller in an undesirable way

C. Exploration Random Process

The exploration random process chosen was an Ornstein-Uhlenbeck process as in the DDPG paper to generate noise that is correlated in time and provides exploration efficiency in physical control problems with inertia. This takes advantage of the fact that our system has inertia in the state of charge of the battery systems. It was found to perform better than an independent identically distributed Gaussian random processes. The exploration random noise was weaned down linearly over the first 30000 training steps. Training continued after exploration stopped to allow the networks to finish training.

D. Results

After 40500 training days, the actor network was able to output a charging policy that performed substantially better than the original heuristic controller. Table II shows the best achieved performance compared to the heuristic and the optimal controller.

TABLE II
MEAN SQUARED ERROR FOR TRAINING THE ACTOR NETWORK

Method	Arbitrage	Voltage	Total
DDPG Actor	175.000	0.449	20.035
Heuristic Control	209.992	0.878	-26.704
Optimal Control	209.577	0.295	60.539

Figure 4 shows a sample new actor output compared to the heuristic and the optimal controller. On hour 8, it can be seen the actor learned to discharge there in order to avoid a over voltage penalty. This learning is due to the random exploration process added to the heuristic that achieved better performance on that day.

V. Conclusion and Future Work

In conclusion, the reinforcement learning agent is able to reduce some of the voltage violations without any knowledge of the power grid. This is simply unachievable using an optimal control approach. However, many more case studies

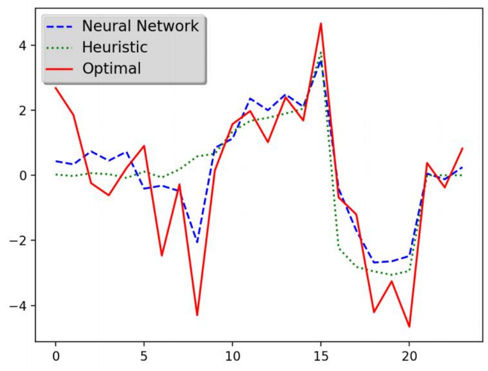

Fig. 4. Sample charge profile output from the actor network after training with DDPG.

and much larger networks with several storage units will have to be tested to confirm the success of the DDPG algorithm in this domain.

There is much future work to be done to refine the reinforcement learning approach for controlling battery storage systems in a distribution network. Perhaps the biggest improvement could come from using a more complex exploration strategy. For example, using information about the time a voltage violation occurs as well as whether it was an over or under voltage could inform the direction of exploration. This could improve performance and reduce the number of examples needed before a successful strategy is found. Another problem to be addressed occurs during training when the policy would sometimes suddenly perform very poorly after learning a good policy. This is known to occur in deep learning and can be mitigated by using trust region policy optimization as described in [6]. Additional work involves testing much larger networks. However, there are many challenges associated with expanding the state space. Particularly, the amount of training examples needed to express a representative set of possible states increases dramatically.

VI. Code

Link to Github repository:
https://github.com/tnavidi1/cs230final.git

Data files are not included in the repository. Code adapted from github user IgnacioCarlucho for DDPG implementation of Mountain Car from AI gym.

References

[1] K. Anderson, R. Rajagopal, and A. E. Gamal, "Coordination of distributed storage under temporal and spatial data asymmetry," *IEEE Trans. on Smart Grid.*, 2017.

[2] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *Int. Conf. Learning Representations*, 2016.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *International Conference on Learning Representations*, 2016.

[4] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," *International Conference on Machine Learning*, 2014.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, and et. al., "Human-level control through deep reinforcement learning," *Nature*, 2015.

[6] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimizatin," *International Conference on Machine Learning*, 2015.