Autonomous Household Energy Consumption Using Deep Reinforcement Learning

Nathan Tsang, Collin Cao, Serena Wu, Zilin Yan, Ashkan Yousefi, Alexander Fred-Ojala, Ikhlaq Sidhu
University of California, Berkeley
Berkeley, USA

***Abstract*—With the massive growth of smart meters and availability of data, there is a unique opportunity to offer novel services to utility customers. The emergence of new technologies can be used to learn a customer's electrical consumption patterns and to offer customized solutions for minimizing electricity costs. One such technology is deep reinforcement learning (DRL), which is explored in this paper. The proposed approach for autonomous household energy management utilizes deep Q-learning with a novel method to deal with potential scalability issues that involves grouping dependent electrical devices. The defined reward function addresses the problems of previous researches by adding more scenarios for the penalty and reward. Transfer learning using an adviser agent is also utilized to improve training time.**

***Index Terms*—artificial intelligence, autonomous systems, energy management, load management, neural networks**

I. Introduction

With the recent improvements in technology in the building and infrastructure sectors, more buildings are utilizing smart electronic devices to reduce both their environmental footprint as well as their utility costs. This paper is motivated by the idea that optimal controllers can be installed in residential buildings to reduce carbon emissions and utility costs without perturbing the occupants' comfort or lifestyle. While energy management systems are common for newer commercial developments, they are uncommon in residential buildings. Instead of relying on smart controllers, devices are manually switched on and off by humans which results in energy inefficiencies and higher costs for home-owners and tenants.

A wide variety of methods have been proposed for use in a household smart controller such as dynamic programming, game theory, and Markov-Decision Processes. While these are effective in optimizing a building's energy use, they typically require a significant amount of information about the building to be optimized. A disadvantage of these methods is that they must be recomputed every time an optimization is required or its underlying assumptions have changed. This can be time consuming and limit the benefit of such systems in complex, changing environments. To solve the issues of computation time and generalization, we propose that such systems instead utilize deep reinforcement learning (DRL) algorithms. DRL has the ability to provide faster solutions and can better adapt to the changing environments without needing modifications to the underlying algorithms. In this paper, we focus on methods such as Q-learning and Policy Gradients with the goal of reducing a household's daily electricity costs.

II. Related Works

For classic energy systems, the solution of using basic algorithms to optimize energy consumption is very mature [1] and have been successful in reducing energy cost [2]. Much of this research is focused on how to optimize large scale energy systems [3] because the introduction of renewable energy or other generation sources to the grid introduces new challenges to the existing system [4], [5]. To resolve or optimize these systems, recent research have utilized reinforcement learning (RL) methods in addition to traditional algorithms [6].

There are also efforts to use RL to reduce energy cost in buildings [7]-[9]. Using DRL could further resolve many of the issues of the current framework [8]. However, the proposed methods are hard to implement and reproduce without further given data. One observation of the current DRL models is that they are unstable and need simulated environments to train many trials before testing in real energy systems.

To further optimize a household energy cost with DQN and DPG, we build a simulation environment and the designed algorithms to validate the current research. We also further expand the research with transfer learning methods in reinforcement learning [10] to speed up the training process and more adaptable to real world scenarios. Also using multi-agent system to resolve the problem of scalability.

III. Problem Formulation

The task of optimizing a building's energy use can be formulated into a RL problem because it involves sequential decision-making. Like all RL problems, a simulation environment (similar to OpenAI gym) is needed to evaluate the performance of the model before introducing it to real systems.

Our environment was constructed to represent a typical residential building. As such, the simulation includes N electrical devices (e.g. AC, lights, receptacles), each with

specific daily requirements. The observation of each device at time *t* is represented by a tuple represented as

$$obs = (t, s, e, d, l) \tag{1}$$

where [*s, e*] is the permitted start and end operating interval, *l* is the electrical load requirement in kW, and *d* is the required daily usage duration. For example, a device that is permitted to operate between 6am–4pm, has a power requirement of 1 kilowatt per hour, and is must run for at least 4 hours per day would have starting state as *obs* = (*0, 6, 16, 1, 4*).

To optimize energy use, the state space includes hour of the day *t* and electricity cost λ_t, in addition to the device requirements. Using these six features, the agent must ensure that the accumulated usage for each device is equal to its required usage by the end of each day. At each hour the agent decides which devices to turn on/off to satisfy these requirements and minimize total cost.

A. Cost Minimization Problem

Minimizing total electricity cost *J* while satisfying the requirements outlined above is achieved by solving the following optimization problem.

$$\begin{aligned}
\min \quad & J = \sum_{t=1}^{24} \sum_{i=1}^{N} a_{d,t} P_i \lambda_t \\
\text{s.t.} \quad & \sum_{t=s_i}^{e_i} P_i \Delta t \geq E_i, \quad \forall\, i \in 1, 2, ..., N \\
& \lambda_t \geq 0, \quad \forall\, t \in 1, 2, ..., 24 \\
& P_i \geq 0, \quad \forall\, i \\
& a_{i,t} \in [0, 1], \quad \forall\, i, t \\
& 1 \leq s_i \leq l_i \leq 24, \quad \forall\, i \in 1, 2, ..., N
\end{aligned} \tag{2}$$

Here, λ_t is the utility cost at time *t*, $a_{i,t}$ is the action of device *i* wherein *a = 0* means the device is off and *a = 1* means the device is on, P_i is the power requirement and E_i is the daily energy requirement for device *i*. While is assumed that the *s*, *e* and *d* remain unchanged throughout day, they vary between days. Thus, the agent must learn a policy that can be generalized across different constraints.

By convention, RL agents are constructed to maximize, not minimize a function. Thus, our reward function was equal to the negative of the cost function *J*.

IV. Background and Preliminaries

A. Reinforcement learning

RL refers to an area of machine learning where the goal is to train an agent to interact with an environment to maximize some reward. An agent (e.g. robot, autonomous vehicle, energy management controller) can be any system or device that makes sequential decisions (i.e. whether to take action A or B) based on information received from an environment. The agent learns the optimal decisions, known as a policy, by first randomly taking actions and receiving feedback from the environment in the form of a reward. These rewards may be received immediately or they may be received after a time delay. The success of each action is measured by the discounted sum of its immediate and future rewards. These key components are conventionally represented in RL as:

- *S* is the set of permissible states.
- *A* is the set of permissible actions.
- *R* is the set of permissible rewards.

The interaction between an agent and the environment is depicted by a tuple (*s, a, r, s'*). This tuple represents an agent that begins in state *s*, takes an action *a*, receives a reward *r*, and transitions to the next state *s'*.

The goal of RL is to find the optimal policy, and the process of doing this varies by RL algorithm. Using Q-learning, the optimal policy is derived from the value of being in each state and taking each action. Using Policy Gradients, the optimal policy is learned more directly.

Unlike Markov Decision Processes, RL is model-free and makes no assumption about the transition function, from (*s, a*) to (*r, s'*). In other words, after enough training, the agent can act optimally in the environment by following its learned policy without having full knowledge of the environment. Using neural networks, the agent can approximate value functions, meaning that it will take optimal actions in states it has not yet visited. RL algorithms that utilize neural networks and deep learning can be applied at scale to environments that have infinite states.

B. Neural networks

Neural networks are a class of machine learning algorithms in which the architecture is made up of layers of neurons. A neuron calculates the linear sum of its inputs and then performs a non-linear transformation of that sum based on its activation function. There are several types of activation functions including Rectified Linear Units (ReLU), sigmoid, or hyperbolic tangents. In the first layer, the inputs are the data itself, while in the successive layers the inputs are the outputs from the previous layer. The output of the final layer represents the model's prediction. There are several types of neural network architectures, such as feed-forward, recurrent, and convolutional. Each of these may be considered deep neural networks (DNN) if they consist of two or more layers, known as hidden layers, between the input and output layers.

In both DQN and DPG, the inputs to the network are features that represent the agent's state. In DQN, the outputs are the Q-values for each action given that state. In DPG, the outputs are the probability of taking each action given that state. In both cases, the network is a value function approximator and informs the agent about which action it should take. An example of a deep feed-forward network with two hidden layers is shown in Fig. 1, 2.

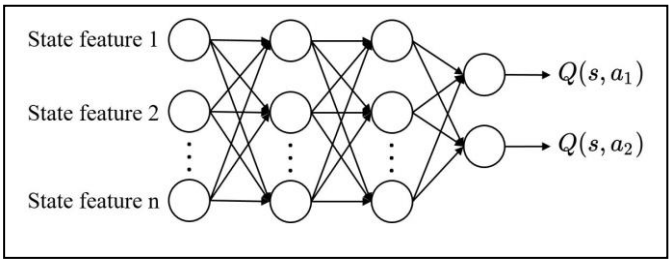

Figure 1. DNN configuration for DQN with two hidden layers

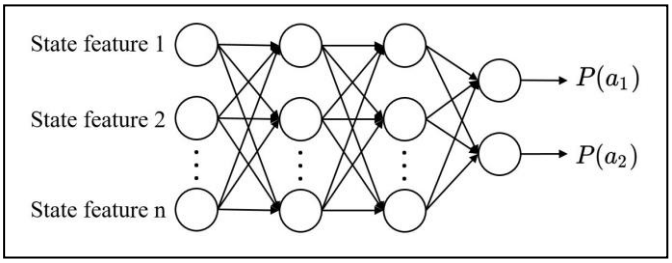

Figure 2. DNN configuration for DPG with two hidden layers

C. Deep Q-Network (DQN)

DQN is a machine learning algorithm in RL that combines the power of DNN with traditional (tabular) Q-learning. In Q-learning, the Value Function $Q^{\pi}(s, a)$ represents the total expected value of being in state s and taking action a based on policy π. After enough training, the agent learns $Q(s, a)$ and therefore which actions are optimal based on the current state. While tabular Q-learning stores the Q-values for each state in a matrix, DQN instead utilizes a DNN with parameter θ($Q(s, a, \theta)$ to approximate the Q-value function. By using a function approximator, DQN is scalable and can be used for complex environments, while tabular Q-learning is restricted to small state-spaces.

DQN is optimized by calculating loss as the mean-squared error in Q-values as shown in (1) and adjusting parameters θ based on the gradient (2). There are two types: component headings and text headings.

$$\mathcal{L} = \mathbb{E}[(r + \gamma max_{a_t+1} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta))^2] \quad (3)$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \mathbb{E}[(r + \gamma max_{a_t+1} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta)) \frac{\partial Q(s_t, a_t, \theta)}{\partial \theta}] \quad (4)$$

One key problem of DQN is that the neural network oscillates and diverges during training due to the model is sequential nature. To avoid this problem, DQN often use experience replay to uncouple sequential states and improve performance. Experience reply is utilized by recording past state transitions (s_t, a_t, r_t, s_{t+1}) and storing them in a memory bank M. During the training process, transitions are randomly sampled from memory bank instead of the most recent transitions. The DQN algorithm used in this paper is illustrated in Appendix A.

D. Deep Policy Gradient (DPG)

It is suggested that the DPG can achieve a faster rate of convergence and requires less time compare to DQN. DPG does this by maximizing the expected reward by directly take the gradient on the policy (5).

$$\nabla \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}[r(\tau) \nabla log(\pi_\theta(\tau))] \quad (5)$$

To solve for the above equation the algorithm need samples x_i, $p(x_i \mid \theta)$ to calculate the estimate gradient,

$$\nabla_i^\theta = r(x_i) \nabla log(p(x_i|\theta))) \quad (6)$$

as moving the direction towards the gradient will increase the log probability of the sample x_i with reward $r(x_i)$.

Therefore, we take the gradient at the end of the game with the samples that are collected during the game as trajectory,

$$\tau = (s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}) \quad (7)$$

and the policy gradient compute the gradient based on the sample trajectory by calculating the derivative of log probability of the trajectory as in (8).

$$\frac{\partial}{\partial \theta} log(p(\tau|\theta)) = \frac{\partial}{\partial \theta} \sum_{t=0}^{T-1} log(\pi(a_t|s_t, \theta)) \quad (8)$$

The final gradient update for τ is (9).

$$\mathcal{R}_\tau \frac{\partial}{\partial \theta} \sum_{t=0}^{T-1} log(\pi(a_t|s_t, \theta)) \quad (9)$$

In contrast to DQN, the DNN is used to estimate the probability of the action based on given θ. The agent evaluates the output of the DNN as the probability of taking each action. The given probability gives the randomness that sometime the model can explore undiscovered states whereas the DQN in contrast only give discrete decisions. DQN is required to plan exploration strategy such as ε-greedy. The DPG algorithm used in this paper is illustrated in Appendix B.

V. Implementation Details and Setup

A. Environment

In this project, we simulated two types of environments, one with a single device and another with multiple devices. In the single device environment, the agent only needs to decide a binary action – to turn the device either on or off at each hour. The purpose for this simple environment is to test the performance of our RL algorithms and verify whether the construction of our environment and reward function is appropriate and conducive for training an agent.

Next, we utilized a multiple device environment which is more realistic for a residential building. In this environment, the agent must make multiple decisions at each hour to minimize total daily cost, increasing our state space by 2^N, where N is the number of devices.

Each day, the schedule start s_i, schedule stop e_i, and required daily duration r_i were randomly generated for each device i. An example of such constraints for a given day is shown in Table

I. In this paper, it was assumed that electricity costs are deterministic and follow the schedule outlined in Table II.

TABLE I. EXAMPLE OF DEVICE CONSTRAINTS

Load, d	*s*	*f*	*l*	*r*
1	8	20	6	4
2	12	13	1	2
3	11	14	3	1
4	1	24	5	3

TABLE II. ELECTRICITY COST SCHEDULE

Time, t	*Cost, λ*
1-12	5
13-14	12
15-18	5
19-21	10
22-24	5

To help with the training procedure, the reward function was constructed to include an additional factor $u_{i,t}$, which incentivizes devices that are on during the permitted time window and a penalizes devices that are on outside this window. More formally:

$$u_{i,t} = \begin{cases} 10, & \text{if } a_{d,t} = 1, \text{ and } t \in [s_i, e_i] \\ -10, & \text{if } a_{i,t} = 1, \text{ and } t \notin [s_i, e_i] \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

Using this reward function, each device is designed to have maximum reward of 110 and having 90 or higher rewards means the basic device requirements are satisfied.

1) Space Complexity: One of the disadvantages of using DQN in this paper is that for multiple device environments the state space of the simulation grows exponentially. The scalability of DPG is much better compared to DQN. The parameters only grow linearly as the device increase. For three devices, the DPG's DNN output only has three neurons whereas the DQN has 2^3 neurons.

2) Space Complexity: In a multiple device environment, it's important to note that devices may dependent or independent to each other. When the devices are independent, turning on or off one of the devices will not affect the reward or action of other devices. In scenario can be resolved by using multiple DQN agents controlling each individual device or device group. When the devices are dependent to each other, multiple devices must be turned on together to distribute the task to achieve the requirements. The latter case is much more complicated and the DQN will require extra resources to learn the optimal strategy. Again, the DPG is sufficiently good for the task of managing multiple devices. The implementation in this paper considers dependent devices.

B. Network Architecture and Training Process

To ensure the different algorithms have a fair comparison, they both utilize same DNN and have similar learning rate and hyperparameters. The learning rate and discount factor was set to 0.01 and 0.95, respectively. Each model was trained with 1,500 episodes where each episode runs for one day (i.e. 24 time steps). For both DQN and DPG, the experiments use DNN with two fully-connected hidden layers of [32, 16] neurons and ReLU as the activation function. The size of the memory bank *M* was 2,000 observations and the batch size was 256 observations. All experiments were run on the same machine to control for differences in processing speed.

VI. RESULTS AND DISCUSSION

A. Algorithm Type

The results of running DQN and DPG with a single-device and a three-device environment are shown in Fig. 3, 4. For a single device, DPG only requires few episodes to converge, while DQN requires about 800 episodes to achieve similar performance. DPG will converge very fast to the near optimal state where it tends to stay at a reward of around 90. DQN slowly improves to a higher reward and is stable at a reward of 102. DPG fails to find the optimum reward whereas DQN will continue to search for the global optimum. By directly calculating the gradient, DPG has a fast convergence speed in the single device environment scenario.

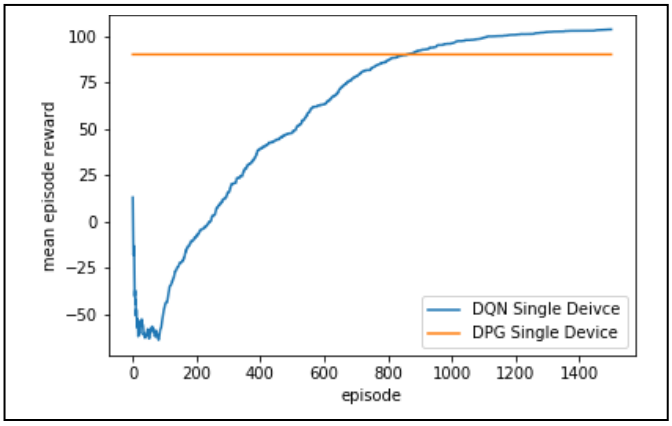

Figure 3. Mean reward during training process for DQN and DPG in a single-device environment.

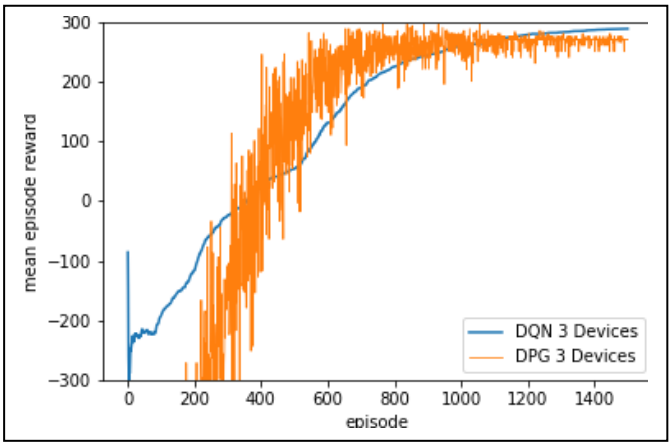

Figure 4. Mean reward during training process for DQN and DPG in a three-device environment.

In the 3-device environment, the reward of the randomly initialized models improves over time as both models learn a near-optimal policy. In general, DPG converges faster but some parameter initialization caused its reward to be stuck at a negative value. In contrast, DQN converges at a slower rate but eventually find the optimal loading strategy and has mean reward that exceeds that of DPG. Since DPG is unstable in some instances and generally fails to achieve a higher mean reward than DQN, our study focused on improving the DQN model.

B. *Number of Devices vs. DQN Performance*

To compare the performance of different number of devices, we averaged the mean reward per device. As the previous section discussed, the vanilla DQN use 2^N number of output neurons. The learning curve of different number of devices is plotted in Fig. 5. As the number of devices increases, the mean reward decreases slightly as a result of increasing state complexity. Due to the exponential nature of the DQN, the learning speed declines significantly between episodes 400 and 800.

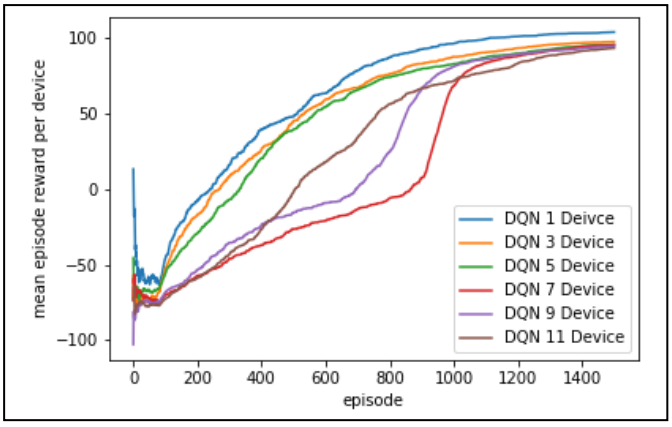

Figure 5. Learning curve and time required for training based on different number of devices.

C. *Scalable Device Environments*

Notice that in Fig. 6, the original (ungrouped) DQN training time increase exponentially as the number of devices increase. We proposed an idea using grouped agents to resolve this issue.

In most of the cases, devices are independent to each other. As a novel approach to solve this real-life scenario, we will implement multiple DQN agents at once, each of which being responsible for a group of dependent devices. By splitting independent devices, the training process is significantly faster. Moreover, as the number of groups increases, training time increase linearly instead of exponentially, as shown in Fig. 6.

Another major benefit of each group being independent is that we can parallel the training process and deploy the training in a distributed computing system.

D. *Impact of Transfer Learning*

One observation from the training is that the DRL methods, especially DQN, use significant amount of time to learn how to satisfy the user's requirement. To reduce training time, we utilized a method called transfer learning that transfer knowledge (in the form of an advised action) from one agent to another agent using an adviser.

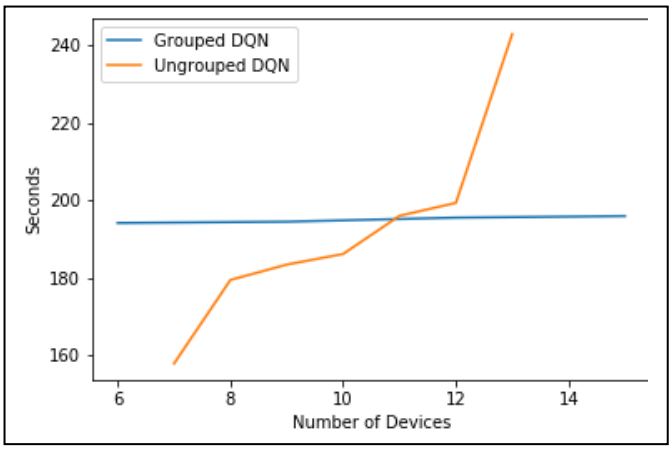

Figure 6. Training time of grouped agents and original approach.

We append the advised action to the state observation as a,

$$(t, s, e, d, l) \rightarrow (t, s, r, d, l, a) \tag{11}$$

In transfer learning, one of the agents is advised by giving the trained model's prediction to another other agent. While two different agents might have different requirements, the trained agent is able to make reasonable actions based on past experience.

The advised agent learns 100-200 episode faster than our original agent and achieved a mean reward of 105, compared to the normal agent's reward of 102 (Fig. 7). Thus, the advised agent has 25% better performance in energy saving (compared to a baseline of 90, where below 90 means agents are violating requirements). This method could be critical to a case when a requirement has changed and a new agent needs to be trained with limited time.

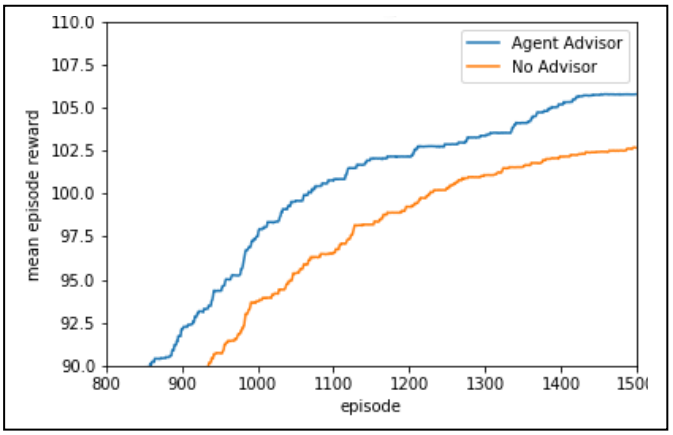

Figure 7. Mean reward during training with an adviser and without an adviser.

VII. Conclusion

In this paper we successfully developed a DRL model to optimize the energy consumption of a single household building. We found that while DQN and DPG could both achieve a reasonable mean reward, DQN was more stable and had a generally better reward. DQN was successful at finding an optimal policy when considering multiple devices, although its performance per device slightly decreased. Due to the vanilla DQN's exponential nature, we utilized multiple agents that were each responsible for a group of dependent devices. This method is preferred because it scales linearly with the number of devices. To minimize training time, we utilized two methods to transfer knowledge from one agent to another.

Appendix A

Algorithm 1 DQN with experience replay

Initialize replay buffer D
Initialize Q as DNN with random weights θ
for each episode **do**
  Record initial state as s_1
  for t in each time step **do**
    Choose random a_t with probability ε
    Else choose $a_t = argmax_a Q(\phi(s_t), a, \theta)$
    step the environment and observe r_t, x_{t+1}
    $s_{t+1} \leftarrow [s_t, a_t, s_{t+1}]$
    Add s_{t+1} to the replay buffer D
    s_i, a_i, r_i, s_{i+1} = sample batch from D
    if episode end at i **then**
      $y_i = r_i$
    else
      $y_i = r_i + max(\hat{Q}(s_{i+1}, a', \theta))$
      Perform Gradient Descent on Q
  end for
end for

Appendix B

Algorithm 2 DPG

Initialize hyper-parameters
Initialize DNN with random weights θ
for each time step **do**
  Sample actions with DNN
  Calculate probabilities by $p(a|\theta, s)$ to A
  Record values of hidden layers of DNN to H
  Record s to S
  Step the environment to s'
  Record reward r to R
  if episode is ended **then**
    Compute gradient ∇ from $\tau = R, A, S, H, \theta$
    update θ by gradient ∇
    clear R, A, S, H and reset the environmnet
    $s \leftarrow s'$
end for

References

A. J. Wood and B. F. Wollenberg, *Power Generation, Operation and Control.* New York: Wiley, 2003.

B. R. Parekh, A. T. Davda, B. Azzopardi, and M. D. Desai, "Dispersed generation enable loss reduction and voltage profile improvement in distribution network-case study, Gujarat, India," IEEE Trans. Power. Syst., vol. 29, no. 3, pp. 1242-1249, May 2014.

C. W. Gellings and W. M. Smith, "Integrating demand-side management into utility planning," Proc. of IEEE, vol. 77, no. 6, pp. 908-918, June 1989.

J. M. Carrasco et al., "Power-electronic systems for the grid integration of renewable energy sources: A survey," IEEE Trans. Ind. Electron., vol. 53, no. 4, pp. 1002-1016, June 2006.

S. Kouro, J. I. Leon, D. Vinnikov and L. G. Franquelo, Grid-connected photovoltaic systems: An overview of recent research and emerging PV converter technology, IEEE Ind. Electron. Mag., vol. 9, no. 1, pp. 4761, March 2015.

S. Lakshminarayana, T. Q. S. Quek, and H. Vincent, "Poor cooperation and storage tradeoffs in power-grids with renewable energy resources." IEEE Trans. On Ind. Electron. Mag., vol 32, no. 1, pp.47-61, 2015.

T. Remani, E. A. Jasmin and T. P. I. Ahamed, "Residential load scheduling with renewable generation in the smart grid: a reinforcement learning approach," IEEE Systems Journal, pp. 1-12, 2018.

J. E. Mocanu, D. C. Mocanu, P. H. Nguyen, A. Liotta, M. E. Webber, M. Gibescu and J. G. Slootweg, "On-line building energy optimization using deep reinforcement learning," IEEE Trans. Smart Grid, 2018.

K. Dalamagkidis and D. Kolokotsa, "Reinforcement learning for building environmental control", in *Reinforcement Learning: Theory and Applications.* Vienna: I-Tech Education and Publishing, 2008, pp. 283-294.

M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: a survey," Journal of Machine Learning Research, vol. 10, pp. 1633-1685, 2009.