AUTONOMOUS PEV CHARGING SCHEDULING USING DEEP-Q NETWORK AND DYNA-Q REINFORCEMENT LEARNING

by

Fan Wang

Bachelor of Engineering, UOIT, 2017

A thesis

presented to Ryerson University

in partial fulfillment of

the requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2019

Author's Declaration For Electronic Submission Of A Thesis

AUTONOMOUS PEV CHARGING SCHEDULING USING DEEP-Q NETWORK AND DYNA-Q REINFORCEMENT LEARNING

Fan Wang
Master of Applied Science
Electrical and Computer Engineering
Ryerson University, 2019

Abstract

This paper proposes a demand response method that aims to reduce the long-term charging cost of a plug-in electric vehicle (PEV) while overcoming obstacles such as the stochastic nature of the user's driving behaviour, traffic condition, energy usage, and energy price. The problem is formulated as a Markov Decision Process (MDP) with unknown transition probabilities and solved using deep reinforcement learning (RL) techniques. Existing methods using machine learning either requires initial user behaviour data, or converges far too slowly. This method does not require any initial data on the PEV owner's driving behaviour and shows improvement on learning speed. A combination of both model-based and model-free learning called Dyna-Q algorithm is utilized. Every time a real experience is obtained, the model is updated and the RL agent will learn from both real data set and "imagined" experience from the model. Due to the vast amount of state space, a table-look up method is impractical and a value approximation method using deep neural networks is employed for estimating the long-term expected reward of all state-action pairs. An average of historical price is used to predict future price. Three different user behaviour without any initial PEV owner behaviour data are simulated. A purely model-free DQN method is shown to run out of battery during trips very often, and is impractical for real life charging scenarios. Simulation results demonstrate the effectiveness of the proposed approach and its ability to reach an optimal policy quicker while avoiding state of charge (SOC) depleting during trips when compared to existing PEV charging schemes for all three different users profiles.

Acknowledgments

Firstly, I would like to express my sincerest gratitude to my supervisor, Dr. Lian Zhao, for providing the invaluable assistance, guidance, and encouragement to my study and research. I am incredibly grateful to her willingness to steer me in the right direction, and not shy away from giving me constructive feedback when I need it the most. I firmly believe that her kind support and useful advice not only made this thesis possible, but will also prove to be beneficial in my future endeavors. I would also like to thank Dr. Jie Gao for his contributions to my research. This thesis would not otherwise be completed without Dr. Zhao and Dr. Gao's helpful academic support.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Background and Motivation

Electric vehicles (EV) use electric motors powered by a battery for propulsion instead of burning gasoline. EVs came into existence in the 1800s and has been the propulsion method for some trains and small vehicles. EVs are seeing a resurgence due to technological advancement in this field, and a world-wide focus on renewable energy. Plug-in electric vehicle (PEV) is a sub-class of EVs that can be plugged in at home, in public, or private charging stations. According to electrive.com, there are currently almost 5.6 million PEVs in the world. Since PEVs are able to recharge their batteries from renewable energy sources such as wind, solar, hydroelectric, or nuclear power sources, their greenhouse gas emission is effectively zero when charged with renewable energy [1]. It was found that all the electric cars on average produce less than half of the global warming emissions when compared with traditional gasoline vehicles, even when taken into consideration the higher emissions caused by electric car manufacturing [2]. Many governments are now giving incentives to purchasing PEVs as they are environmentally friendly. For instance, Transport Canada, a Canadian government agency, is now offering a purchase incentive program for electric vehicles. Any Canadian who purchase a PEV are eligible for an amount of $2500 to $5000. These types of programs are very common today for many countries.

Electric motors are very efficient at converting stored energy into driving the vehicle. PEVs do not consume energy when at rest or coasting [3]. It is shown that PEVs can have an efficiency of around 80%, in comparison with 15% for gasoline engines, and 20% for diesel engines [4]. Most electric cars have much lower cost per mile than traditional gasoline vehicles. For instance, the Nissan Leaf has a cost of 3.5 cents per mile, and the Chevrolet Volt has a cost of 3.8 cents per mile. Comparatively, gasoline cars such as the Toyota Corolla has a cost of 11.9 cents per mile, and the Hyundai Elantra has a cost of 13.1 cents per mile. PEVs drastically reduces cost of drivers due to their better efficiency [5].

PEVs also have a lower maintenance cost when compared to combustion vehicles, because electronic systems break down less often than mechanical systems. As PEVs do not run off of gasoline or diesel, they do not require oil changes, which reduces costs for owners [4]

Another very useful feature of PEVs is their utilization of vehicle-to-grid mode. In this mode, PEVs offer owners the ability to sell electricity stored in their batteries back to the power grid, which can help utilities operate more efficiently [6]. The vehicle-to-grid mode help balance loads by charging when the demand is low, and discharging when the demand is high. The PEV's batteries could also be used as backup power in case of an electricity outage.

Although there are downsides to PEVs, such as the limited range of the vehicle, cost of battery, charging time, and the availability of charging infrastructure, most of these problems are showing improvement with time. For instance, The cost of battery has decreased by 70% between 2008 and 2014.

Electric vehicles are expected to increase to 22% of the global share in 2030 [7], which means roughly 240 million cars in the world will be electrical. The average electric vehicle battery is 23kWh, which implies that there will be 5.5 Terawatt Hour of electric vehicle battery storage world wide. These are very conservative estimates and the reality of total electric vehicle battery storage is potentially much more.

If these battery storages are used in load demand response strategies, the power grid will be able to greatly alleviate peak power demand. However, currently, there are no customer incentive to participate in these strategies, therefore it becomes crucial that PEV owners get some benefit from participation. The most intuitive solution is to reduce PEV's charging cost from an owner's perspective. It is found that PEV owners can potentially earn hundreds of dollars depending on their daily drive by utilizing vehicle-to-grid mode [8].

However, this is likely far from optimal, and a PEV owner must manually sell the PEV's battery electricity. Manually taking part in the selling/buying of electricity causes a huge nuance for PEV owners. Therefore, more incentive can be given to PEV owners if the buying/selling of power is done automatically.

From this perspective, reinforcement learning techniques can be utilized to minimize a PEV owner's charging costs. Reinforcement learning (RL) has been gaining popularity due to advancement in research and computing power. RL has had a long history with its origin dating back to 1950s, when Richard Bellman described a problem of designing a controller to minimize a measure of a dynamical system's behaviour over time. This process has been introduced as a Markov Decision Process (MDP) [9]. Since then, RL has seen successes in beating the world's champion at Go, Chess, and Backgammon and has also achieved super human performances in video games [10].

The goal of RL is to discover an optimal policy in a dynamic environment. This problem formulation provides an excellent framework for PEV charging scheduling, as the PEV owner's behaviour and energy prices are both dynamic environments. In order to avoid owner participation in selling/buying power, a RL agent can "learn" to perform this task automatically by observing the owner's driving routines and fluctuating energy prices. Having a RL agent perform this task effectively means that a PEV owner can simply plug in his PEV at all times and his charging cost will be reduced and the PEV will have sufficient charges for the owner's trips.

1.2 Literature Review

Many methods to reduce the cost of charging PEV fleets have been proposed in literature [17–22]. An optimal charging scheduling strategy for charging a large PEV fleet to reduce cost and supplying the requested power to the grid is developed using partial differential equation in [17]. This method assumes prior knowledge of the large PEV fleet. A fully distributed solution for solving PEV's Cooperative Charging (PEV-CC) problem is proposed in [18]. A multi-agent distributed approach to solving the PEV-CC problem is developed in [19]. A game theoretic approach to finding optimal bidding strategies for electric vehicle aggregators with variable energy sources is formulated as a mathematical programming with equilibrium constraints in [20]. A bidding strategy with the objective of minimizing charging costs for a vehicle fleet is studied in [21], but it does not consider Vehicle-to-Grid mode. In [22], a decentralized real-time EV power allocation scheme is proposed. All of the above methods only take into consideration the PEV fleet charging problem and not from the perspectives of individual owners.

Numerous optimization methods to schedule PEV have been investigated in [23–29]. An optimization approach to reduce cost of charging based on battery degradation is proposed in [23]. Day-ahead charging scheduling using information gap decision theory based approach is introduced in [24]. In [25], an electric vehicle aggregator participating in electricity market, using a two-level optimization problem in the framework of model predictive control is analysed. It is found this method can reduce cost compared to the stochastic method and deterministic predictive method. Another two-stage optimization method for reducing PEV charging cost for the workplace is proposed in [26]. In [27], a real-time charging scheme is proposed, the problem is formulated as a binary optimization problem. A convex relaxation method is developed to reduce the computation complexity of the algorithm. In [28], a task time allocation and reward scheme for advertising PEV charging station information is proposed. PEV charging scheduling schemes can take advantage of the Hybrid Machine Learning Model proposed in [29] to predict the power consumption on trips. None of these approaches take the stochastic nature of PEV usage into account, as the problem is too complex due to variations in the driver's driving routine, traffic conditions, and energy price.

Methods using artificial intelligence such as reinforcement learning (RL) or artificial neural network to solve the challenges of single PEV charging scheduling are gaining popularity in research. Methods using existing user behaviour data to schedule a single PEV are examined in [30–32]. The authors of [30] use an artificial neural network trained using historical household power comsumption and EV energy demand data with two hidden layers to predict whether the PEV should charge or discharge. In [31], an off-line RL charging scheduling using fitted Q-iteration is proposed. The problem is formulated as a Markov Decision Process (MDP). This proposal achieved good results under the assumption that the user behaviour is known ahead of time. Two infinite horizon average cost MDP formulations are described for both hybrid vehicles and PEVs in [32]. The MDPs are built from historical data on vehicle usage. In the real world, user's previous driving behavior is often unknown and varies from owner to owner. These off-line methods cannot adapt if there are changes to the PEV owner's driving routine.

Model-free RL methods are developed in [33–35]. Q-Table based method where electricity price and time are discretized is utilized in [33] to discover an optimal demand response. However, this method becomes impractical when energy price and time have a large number of states. The authors of [34] solve the problem of having a large number of states by using a linear approximator to approximate Q-values. However, this method can only approximate linear functions, while such relationships are generally non-linear in a real life PEV charging scenario. In [35], the PEV charging scheduling problem is formulated as an MDP and an optimal charging policy is found using a model-free deep-Q network with memory replay. This method takes into account the user's stochastic behaviour and the fluctuating energy price, and is able to achieve good results. However, the simulation assumes that the PEV's battery SOC will never reach zero, and ignores the relationship between trip length and energy usage's effect on the PEV's battery SOC before and after trips. In addition, the suggested method does not take full advantage of the PEV owner's routinely behaviour. Under a more realistic scenario, the PEV is found to run out of battery during trips quite often. Furthermore, a pure model-free RL strategy takes a long time to discover an optimal charging policy.

1.3 Objectives

In this paper, a cost-efficient PEV charging/discharging scheduling strategy is proposed from the PEV owner's perspective. The proposed strategy fulfills the usage requirements of the PEV owner while taking full advantage of the fluctuating energy price to reduce charging cost or, in some cases, earn a profit. The problem is formulated as an MDP. A combination of model-based and model-free RL called Dyna-Q algorithm is utilized. A deep-Q network is utilized to decide the optimal charging action when given the current time of day, day of the week, current SOC of the PEV, current energy price, and the difference between current energy price and the average historical price for the next six hours. The approach considers a real-life scenario that takes into consideration the relationship of battery SOC and the variability of trips. The effectiveness of this approach is proven in simulations. The Dyna-Q algorithm can discover an optimal strategy much quicker than a pure model-free RL strategy. Furthermore, the simulation shows that this paper's problem formulation ensures that the PEV's battery SOC never decreases to zero during a trip, all the while reducing the PEV owner's charging cost. The method does not require any PEV owner's historical driving data. The model-based aspect of Dyna-Q algorithm is created and updated as experience is gained from the PEV owner's real-life driving.

1.4 Contributions

The contributions of this paper are as follows:

- A single PEV charging problem from the owner's perspective is formulated as an MDP. The owner's driving routine, the relationship between battery SOC and trips, and fluctuating energy price are taken into account.
- An optimal policy is discovered using a three-layer deep-Q network. The neural network can generate an optimal charging policy based on PEV owner's driving behaviour and fluctuating energy price while keeping the PEV battery SOC above zero during trips.
- A combination of model-based and model-free RL is utilized in training the deep-Q network. This ensures the an optimal charging policy is discovered quickly. The model for model-based RL is updated from real driving experience after its initial creation. The RL agent will learn from both real experience and experience generated from the model.

1.5 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 gives the background information needed on various machine learning techniques. This includes the optimization method used by most machine learning techniques, supervised learning algorithms, and how supervised learning algorithms is applied in reinforcement learning.

Chapter 3 outlines the system model for single PEV charging scheduling. The state, action, reward space for reinforcement learning are discussed in detail.

Chapter 4 investigates the various reinforcement learning methods that can be applied to the PEV charging problem. On-line, off-line, on-policy, off-policy, model-free, model-base, and combinations of any of the above methods are discussed.

Chapter 5 provides the model-based aspect of our proposed reinforcement learning approach. The user's driving behaviour is modeled and updated every time an real experience occurs.

Chapter 6 presents the simulation results comparing various approaches to single PEV charging scheduling.

Finally, chapter 7 concludes the thesis and discusses future works.

Chapter 2

Machine Learning Fundamentals

This thesis relies heavily on machine learning techniques, and this section outlines the algorithms utilized.

Machine learning algorithms usually belong to three classes, and are listed as follows:

- Supervised learning: attempts to predict a target (dependent variable) from a given set of features (independent variables). This class of machine learning algorithms usually requires massive amount of sample data.
- Unsupervised learning: attempts to cluster data into different groups, without any target to predict.
- Reinforcement learning: attempts to learn a policy by making specific decisions. This class of machine learning algorithms trains itself continuously within an environment through trial and error.

The proposed solution to PEV charging scheduling in this thesis makes use of supervised learning and reinforcement learning extensively, and these techniques are discussed in-depth throughout the rest of this thesis. The PEV charging scheduling scheme's long term expected reward are estimated using a neural network, which is a form of supervised learning.

2.1 Loss Function

The most common class of machine learning algorithms is supervised learning. This class of algorithms usually tries to minimize an objective function, typically known as the loss function. The loss function is usually the difference between a predicted value and the true value, more specifically shown as:

$$J = y - \hat{y}, \tag{2.1}$$

where J is the loss function, $\hat{y}$ is the predicted value, and y is the true value. The predicted value $\hat{y}$ is usually obtained using a value approximation method. The actual optimization method are discussed in following chapters.

Traditionally, statistical methods such as linear regression are used to make predictions. With the recent advancement in computing technology, machine learning methods such as neural networks and XGboost are shown to drastically improve accuracy when compared to traditional statistical methods [36].

The two categories of supervised learning methods are regression and classification [37], which are discussed in the following subsection. The regression aspect of supervised learning is used to estimate the long term expected reward of a PEV charging scheduling policy.

2.1.1 Regression

Regression attempts to generalize and estimate the relationship between a dependent and independent variable. This is very useful technique to generalize patterns based on existing data, which is used in the PEV charging scheduling scheme.

The most commonly used loss function for regression is the mean squared error [38]. It is the average of the squared difference between predictions and actual values. The loss function for mean squared error is shown as follows:

$$J = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}, \tag{2.2}$$

where y is the true value, $\hat{y}$ is the predicted value using a value approximation function, and n is the total number of samples or data points. The neural network used for the proposed method uses the mean squared error as its loss function, the details of which are discussed in the following chapters.

In some cases, another commonly used loss function called the mean absolute error is used. It is the measurement of the average of sum of absolute differences between predictions and the true value, and is shown below:

$$J = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}, \tag{2.3}$$

where $\hat{y}_i$ is the predicted value, and y_i is the true value.

2.1.2 Classification

Classification tries to assign observations to two or more categories.

The most common loss function for classification is the cross entropy loss function [39], and is shown as follows:

$$J = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)), \tag{2.4}$$

where $\hat{y}_i$ is the predicted value, and y_i is the true value. These values take on a range between $(0, 1)$ and are probability values. More specifically, the predicted value will be the probability of it belonging to class 1. The cross entropy loss function will penalize the predictions that are confident and wrong more heavily. Although classification is not used in PEV charging scheme, it was considered as an option for policy gradient reinforcement learning methods.

These loss functions are very commonly used in machine learning and statistics and the following subsections discusses the most commonly used optimization algorithms to minimize loss functions.

2.2 Iterative Optimization

This section outlines frequently used iterative optimization algorithms in machine learning. Iterative optimization algorithms are widely used to optimize supervised learning methods such as neural networks through a process called back-propagation [40]. The back-propagation technique is utilized extensively in solving the PEV charging scheduling problem.

2.2.1 Gradient Descent

Gradient descent is a crucial algorithm in supervised learning. It is the foundation of back-propagation in deep neural networks, which has applications in many areas in our modern world [41]. A variation of the gradient descent algorithm will be used to optimize the neural network utilized in the PEV charging scheme.

The goal of gradient descent is to minimize a function. In machine learning, the most common function to minimize is the loss function J, with respect to weights θ. The weights are used in a value approximation function, $f(x)$, to predict a value given input values x. The input values x are usually a vector of features, which has more than one value. The loss function J can be minimized with respect to the weights θ, which reduces the loss of the value approximation function $f(x)$.

The gradient descent algorithm computes the gradient of the loss function, $\nabla J(\theta)$ with respect to the weight, θ, and moves the weight in the gradient descent direction. This process is done iteratively until the loss function stops showing improvement. The gradient descent algorithm is shown as follows:

$$\theta \leftarrow \theta - \eta \nabla J(\theta), \tag{2.5}$$

where η is the learning rate, θ are the weights, and $\Delta J(\theta)$ is the gradient of the error function with respect to the weights. Learning rate is the step size, which means how much to move the weights in the direction of gradient descent per step.

Gradient descent computes the loss of the entire data set, and is usually computationally heavy [42]. A variation of the gradient descent, called stochastic gradient descent, is more commonly used as it generally reaches convergence faster and requires less computation [43].

Stochastic gradient descent computes the loss function for a single data point, and updates the weight θ with the gradient of that loss function. The problem with stochastic gradient descent is that it could “overshoot” the minimum due to frequent fluctuations [44]. The variation of gradient descent that is most widely used is the mini-batch gradient descent [45]. Instead of using the whole data set or a single data point to compute the loss function, a fraction of the whole data points are used instead. This method can reduce computation time, leads to faster convergence, and solves the overshoot issue of stochastic gradient descent. A mini-batch sampling method will be used in the PEV charging scheduling scheme.

One of the biggest issues of gradient descent and its variations is the potential to be “trapped” in a sub-optimal local minima. When this issue occurs, it is difficult to overcome, and will usually require resetting the weights θ. The following approach is an attempt to overcome this issue.

2.2.2 Momentum

The momentum algorithm is similar to gradient descent, except it adds a ”momentum” update vector [46]. The update vector is essentially a fraction of the previous update, and is shown as:

$$V(t) = \gamma V(t-1) + \eta \nabla J(\theta), \tag{2.6}$$

where γ is the strength of the momentum, V is the update value, η is the step size, and $\Delta J(\theta)$ is the gradient of the loss function with respect to the weights.

The update of the weights θ is therefore:

$$\theta \leftarrow \theta - V(t) \tag{2.7}$$

Momentum algorithm usually leads to faster and more stable convergence and reduces overshooting and oscillations. The momentum update is included in the optimization algorithm used for PEV charging scheduling.

2.2.3 Adagrad

There can be many parameters, i in the weights θ_i. Some parameters are used infrequently and some are used more frequently. For this reason, an algorithm that attempts to update the more infrequently used parameters with a bigger learning rate η is proposed as the Adagrad algorithm [47]. The Adagrad algorithm is as follows:

$$\theta_i^{(t+1)} = \theta_i^t - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}, \tag{2.8}$$

where $g_{t,i}$ is the gradient of the loss function at step t with respect to the specific parameter of weight, shown as $\theta_{t,i}$, and θ are the weights of the value approximation function.

This update algorithm takes into consideration the different frequency of parameters of the weight θ. The previously mentioned algorithms will be utilized as a foundation to the optimization algorithm used for the actual PEV charging scheduling scheme.

2.3 Supervised Learning

Supervised learning is the most commonly used class of machine learning. It is found in many areas of artificial intelligence, such as self driving cars, computer vision, and credit card fraud detection. The supervised learning algorithm requires massive amounts of data for the machine to "learn" specific patterns between inputs and outputs [48].

Supervised learning falls into two categories: regression, and classification. These were outlined in the beginning of this chapter.

There are many supervised learning algorithms, the following is a list of the most commonly seen:

- Support Vector Machines
- Linear Regression
- Logistic Regression
- Naive Bayes
- Bagging and Boosting Forest

- k-nearest Neighbor
- Neural Networks
- Similarity Learning

The most popular and effective supervised learning algorithm today is neural networks, which will be discussed in the following subsections. Neural networks is a very important aspect of the PEV charging scheme.

2.3.1 Artificial Neural Network

Neural networks have uses in many applications such as: computer vision, natural language processing, and time series analysis. The PEV charging scheduling problem uses an artificial neural network to approximate values which are used for reinforcement learning.

Neural networks are inspired by human brain's biological neural networks, in where neurons fire when a certain activation threshold is achieved. Each neuron is connected with other neurons in the brain, forming a complex network [49].

An artificial neural network architecture is made up of 3 crucial parts:

- Neurons/Nodes
- Connections
- Weights

A neural network can consist of many layers, with a parallel number of nodes per each layer. Each of these nodes have a set of weight values, in which each input of the neural network is multiplied by, and summed together with the bias. This value is then passed to an activation function, and the output of that is fed into another layer.

Figure 2.1 shows an example of a single neuron. In the figure, $x_1, x_2, .., x_n$ is multiplied by weights $w_1, w_2, .., w_n$. The bias b is then added to this value and passed through an activation function to produce an output. To summarize:

$$a_1 = A(\sum_{i=1}^{n} x_i w_i + b), \tag{2.9}$$

where a_1 is a single output, A is the activation function, x are input features, and w are neuron weights.

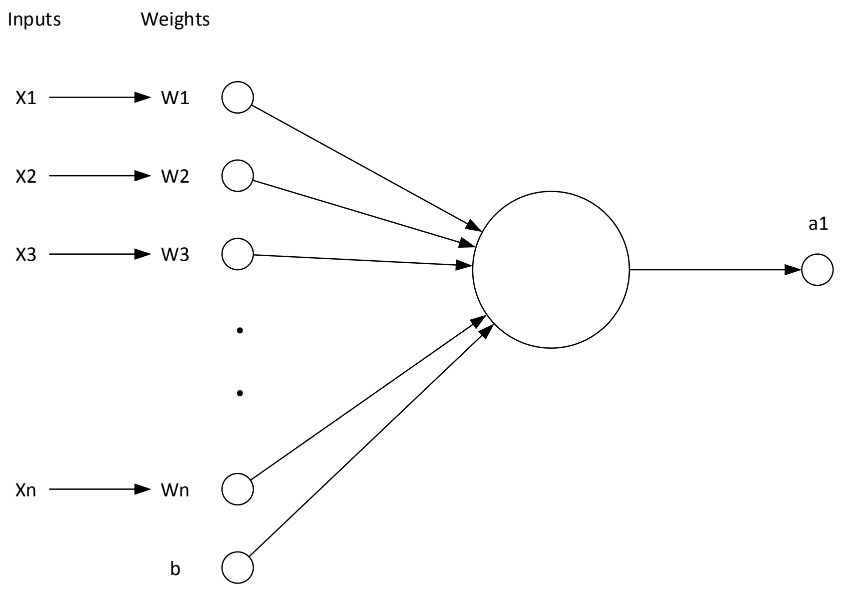

Figure 2.1: Single Neuron

There are many activation functions, and they are explained more in depth in the following subsections.

2.3.1.1 Activation Functions

A neural network can have no activation functions, and in the earliest version of the neural network, there were in fact no activation functions. Without an activation function, the network is not able to predict any non-linearity in the data, and real-world data is often non-linear. This caused neural networks to be ineffective in solving most real-world problems.

The mostly used activation function for binary classification is the Sigmoid function [50], and is shown as the following:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.10}$$

The output of this function takes on the range of $(0, 1)$, which is ideal for binary classification (1s or 0s). A decision threshold needs to be set, which is usually set as 0.5. More specifically, if the output $S(x) > 0.5$, then the classifier will decide that the output belongs to class 1, and conversely, if the output $S(x) < 0.5$, then the classifier will decide that the output belongs to class 0.

The curve of the activation function is shown in 2.2

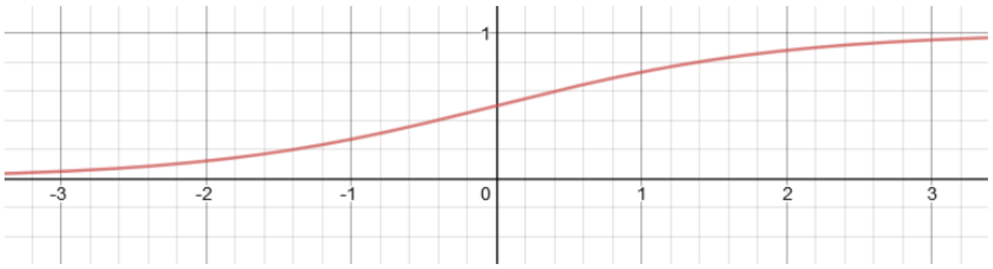

Figure 2.2: Sigmoid Activation Curve

The advantage of the sigmoid function is that it is continuously differentiable across all values. This is a very beneficial property when we use any of the above mentioned optimization algorithms, which requires the gradient of the function.

Another activation function that is commonly used is the hyperbolic tangent activation function, or otherwise called the tanh function. The advantage of this function over the existing Sigmoid function shown in (2.10) is that it has a stronger gradient, as the function ranges from $(-1, 1)$ [51]. The tanh function is shown as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.11}$$

The curve of the tanh function is shown as Figure 2.3. As seen, the function converges to 1 as x approaches infinity, and 0 when x approaches negative infinity.

The issue with both the sigmoid and tanh function is the vanishing gradient problem. When the input values are very large or very small, the gradient of the function approaches 0. This is unwanted as a non-zero gradient is needed to optimize the neural network. Although binary classification is not used in the final PEV charging scheme, it was considered as an option for a policy gradient reinforcement learning method.

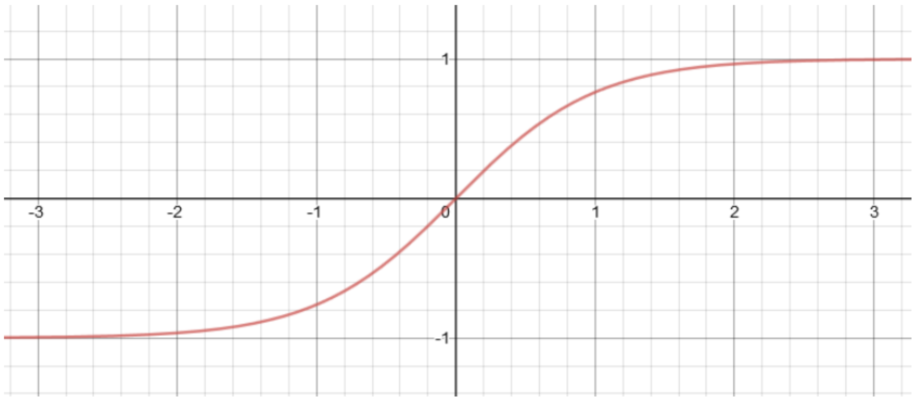

Figure 2.3: TanH Activation Curve

The sigmoid and tanh activation functions are commonly used in binary classification, however, they become insufficient when solving a multi-class classification problem. The softmax function allows multi-class classification [52].

The activation function is as follows:

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \tag{2.12}$$

where J is the total number of classes and i is the ith specific class score. The softmax function also suffers from the vanishing gradient problem.

The rectified linear (Relu) function can mitigate the vanishing gradient problem the other activation functions have. The Relu function is shown as:

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{2.13}$$

The curve for the relu activation function is shown in Figure 2.4. The relu function is most commonly used in the hidden layers of the neural network. There are many advantages of the relu activation function, shown as follows [53]:

- Computational simplicity: the relu function is linear and does not require exponential calculation
- Representational sparsity: the relu function can output true zero values. When the input is negative, the output is 0.
- Linear behavior: This avoids the vanishing gradient problem, as all input values will have a desired gradient

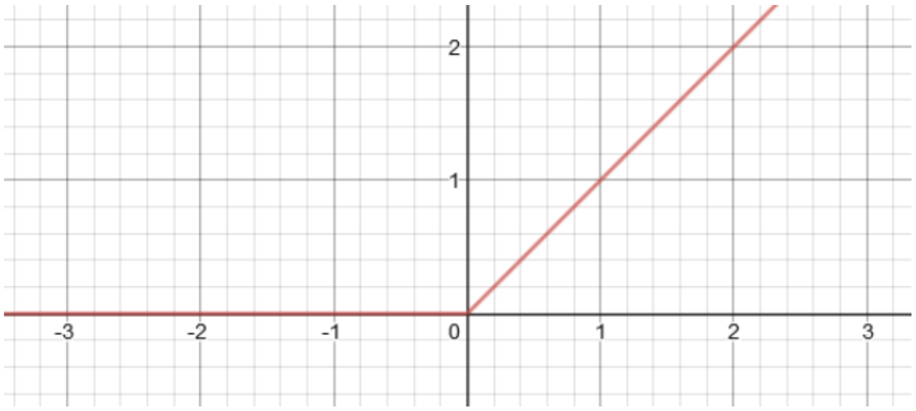

Figure 2.4: Rectified Linear Activation Curve

The relu activation function will be used for the hidden layers of a deep-q network, which is utilized to solve the PEV charging scheduling problem.

If negative values are of importance in the neural network, then the relu activation function will produce a gradient of 0 whenever the input is negative. A special case of relu activation function named leaky relu, shown in Figure 2.5, is created to combat this issue [54]. Leak relu activation function is shown as follows:

$$f(x) = \begin{cases} 0.01x & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{2.14}$$

This adds a small gradient for when the output of the activation function is negative.

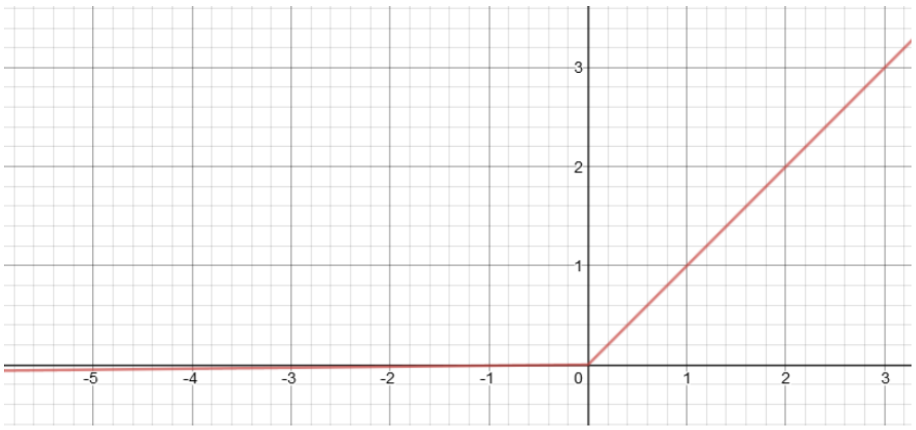

Figure 2.5: Leaky Rectified Linear Activation Curve

2.3.1.2 Back-propagation

A fully connected neural network has every node of a layer connected with every node of the next layer. An example of a single hidden layer neural network is shown in Figure 2.6.

The neural network attempts to minimize a loss function through back-propagation. The minimization is done using any of the iterative optimization methods mentioned earlier in this chapter. For example, referring to Figure 2.6, if the input layer is represented as x, the weights of the hidden layer represented as

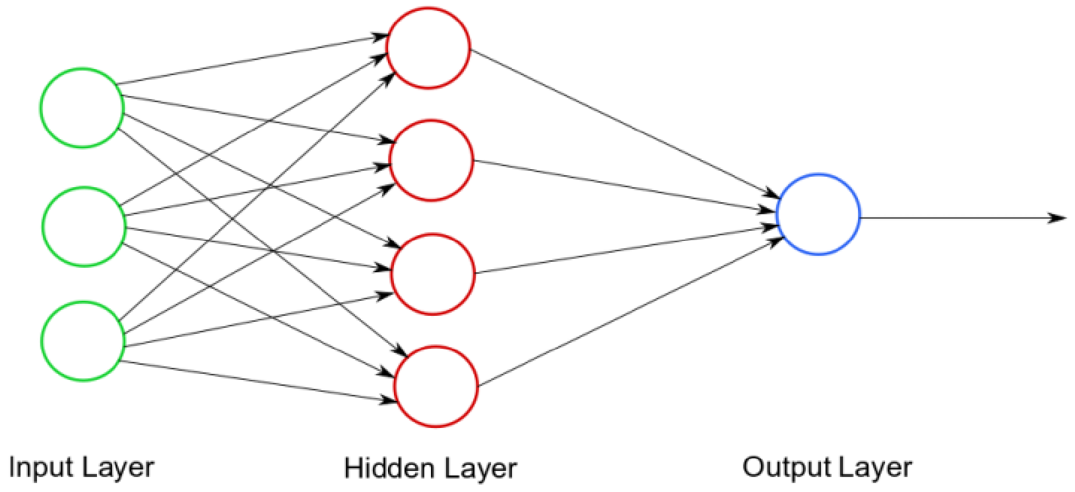

Figure 2.6: Neural Network

w_{ij}, the output of the hidden layer after going through an activation function is o_j, the summation of the hidden layer before an activation function is represented as S_j, and the error function is represented as E, then the back-propagation is done as follows:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial S_j}\frac{\partial S_j}{\partial w_{ij}} \tag{2.15}$$

When the gradient is found, the optimization function will move in the opposite direction of the gradient, which reduces the error. The artificial neural network is a crucial component of the PEV charging scheduling solution, and will be discussed in depth in a later chapter.

Chapter 3

System Model

The problem of finding a PEV charging strategy to minimize the overall charging cost for the PEV owner is formulated as a finite MDP with discrete time steps and unknown transition probabilities.

MDP is an effective method for decision making under uncertainty [55]. An MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $\mathcal{P}$ is a set of transition probabilities that reflects the probability of arriving at a specific next state S_{t+1}, after taking an action a in a previous state S for all states and actions; and $\mathcal{R}$ is a set of rewards that contains all the immediate reward after transitioning from any one state to any next state from taking any actions.

3.1 MDP State Space

The state space for the proposed MDP formulation is defined as follows: a state s encapsulates six variables: $S = [d\ m\ c\ p\ \Delta P\ k]$. The variables $d\ \epsilon\ [1, .., 7]$ and $m\ \epsilon\ [0, .., 1439]$ represents the day of the week, and minute of the day, respectively.

There are a total of 1440 minutes in a day. The variable c reflects the state of the PEV battery, which ranges from 0 to 1, with 1 being full SOC and 0 being empty SOC. The variables p and ΔP represent the current and predicted future energy price, p is the current electricity price in dollars per kWh; and ΔP is the difference of current energy price and the estimated energy price of the next six hours, averaged over every hour. Price prediction will not be the focus of this paper.

There are existing methods in literature, such as the long short-term memory neural network proposed in [35] that is more accurate for price prediction than using historical averages. However, the method of price prediction has no impact to the viability of the proposed charging strategy when compared with other charging strategy, as a better price prediction will reduce the various method's charging cost by a similar ratio. The variable k is a binary variable that represents whether the PEV has ran out of battery on a given trip, with 1 representing that the PEV depleted its battery during a trip, and 0 representing the battery was sufficient for the trip.

The estimated energy price is taken from historical data of the same month and date of previous years. There are no state representations for when the PEV is unplugged. Specifically, if the PEV becomes unplugged (when owner takes a trip), the next state S_{t+1} will be when the PEV is plugged in again.

There are predictable patterns in the user's weekly driving routines, and with the six variable state space, the PEV owner's driving behavior should be sufficiently represented. Since the MDP has unknown transition probabilities, $\mathcal{P}$ will be "learned" from real-life experience using RL techniques. The transition probability is taken into consideration implicitly by the deep-Q network when deciding on the optimal policy, which will be discussed in later sections.

3.2 MDP Action Space

The action space A consists of these actions: discharge, idle, or charge, represented by $a = -1$, $a = 0$, and $a = 1$ respectively. The PEV cannot exceed its maximum battery capacity when charging, or its minimum battery capacity when discharging, shown as the following constraint:

$$0 \leq c + V_{\text{charge}} \leq c_{\text{max}}, \tag{3.1}$$

where c is the current battery SOC; V_{charge} is the amount of electricity charged into or discharged from the battery, which can be positive or negative; and C_{max} is the battery capacity. Because of the above constraint, there are only two valid actions in the states with $c = 0$ or $c = 1$.

3.3 MDP Reward

The immediate reward of an action is the cost of charging or discharging, shown as:

$$R = -p \cdot a \cdot r, \tag{3.2}$$

where p is the current price of electricity in dollars per kWh; a is the action taken; and r is the minutely charging/discharging rate of the PEV. The reward has a negative sign because if a charging action is taken, ie. $a = 1$, then the reward is negative, as a cost has incurred for charging the PEV. Similarly, discharging the vehicle will yield a positive reward. If the PEV owner takes a trip, and has insufficient energy to reach destination, then a penalty $-P$ plus the cost of recharging during a trip $-C$ is obtained as the reward. To summarize:

$$R = \begin{cases} -p\cdot a\cdot r, & \text{if } d_{t+1} - d_t = 0 \text{ and} \\ & m_{t+1} - m_t = 1; \text{or} \\ & d_{t+1} - d_t = 1 \text{ and } m_{t+1} = 0 \\ -P - C, & \text{if} m_{t+1} - m_t \neq 0 \text{ and k} = 1;\ \text{or} \\ & d_{t+1} - d_t \neq 0 \text{ and k} = 0 \end{cases} \tag{3.3}$$

3.4 MDP State Transition

When the PEV is plugged in at both state S_t and the next state S_{t+1}, then the day, time and the PEV's battery SOC are fully deterministic. The day and the minute of the next state S_{t+1} are:

$$d_{t+1} = \begin{cases} d_t, & \text{if } t = 1439 \\ d_t + 1, & \text{else} \end{cases} \tag{3.4}$$

$$m_{t+1} = \begin{cases} m_t + 1, & \text{if } t \neq 1439 \\ 0, & \text{else} \end{cases} \tag{3.5}$$

where the number 1439 is the last minute of the day, and a new day starts when t reaches 1440. The battery SOC of the next state is:

$$c_{t+1} = \begin{cases} 0, & \text{if } c_t = 0, a = 0 \\ c_{\max}, & \text{if } c_t = c_{max}, a = 0 \\ c_t + a \cdot r, & \text{else} \end{cases} \tag{3.6}$$

The electricity price of the next state is unknown, therefore the transition probability **p** is also unknown.

If the PEV owner takes a trip, then the next state S_{t+1} is no longer deterministic, as knowledge of the duration of the trip and the battery charges used are unknown. Due to uncertainties in road conditions, nature of user's trips, and energy price, it becomes difficult to determine S_{t+1}. This challenge is resolved using deep-Q network which is discussed in Section IV.

3.5 State-Action Quality

Since the transition probability of the MDP is unknown and there exists a large state-action space, we use a deep reinforcement learning strategy to discover the optimal charging policy.

A selection criteria is needed to choose an action for a given state. Knowledge of the short term reward R is insufficient when choosing the optimal action. Therefore a long-term expected reward is required. The Bellman equation represents the long-term expected reward when following a policy as:

$$Q^{\Pi}(S_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ..], \tag{3.7}$$

where Π is the PEV charging/discharging policy, which selects an action for each state; γ is the discount factor; and R is the immediate reward following the action at time step t. The Q value is therefore the expected sum of the discounted reward of every time step t with a discount factor γ. The variable γ represents the weight of future rewards, and takes a range from 0 to 1. Having a discount factor of 1 means that the future and present has the same importance, and conversely, having a discount factor of 0 means that the future is of no importance. The goal of reinforcement learning is therefore to find the policy that maximizes $Q(s, a)$ at every time step:

$$\Pi'(s) = \mathrm{argmax}_{a \epsilon A} Q(s, a) \tag{3.8}$$

Chapter 4

Reinforcement Learning Methods

This chapter analyzes the many different reinforcement learning methods and their viability and their pros and cons for single PEV charging scheduling.

MDPs are discussed more in-depth as they are the foundation building block of reinforcement learning.

4.1 Markov Decision Process

An MDP can formally describe an environment for reinforcement learning, where the environment is fully observable [56]. If the environment is not fully observable, then other methods must be employed, such as partially observable MDP (POMDP) [57]. For the PEV charging scheduling problem, the environment can be considered as fully observable, and an MDP is sufficient for modeling the problem.

Almost all RL problems can be formalized as MDPs, an example of a problem solved using MDPs is optimal control, and it primarily deals with continuous MDPs.

For any given state S to be considered Markov, it must capture all relevant information from history, in other words, the future of the state is independent of the past given the present. If the current state S is known, then all history regarding how S is reached can be discarded, as the state itself already contains all the relevant information about its history.

There exists a transition probability of being in one state S_t at time t, and transitioning into the next state S_{t+1} at time $t + 1$.

The formal definition of is as follows: a state S_t is Markov if and only if

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t], \tag{4.1}$$

in other words, the probability of being in state S_t and transitioning into state S_{t+1}, is the same as the probability being in state S, and knowing all the transitions that ultimately resulted in being state S, and transitioning into S_{t+1}.

For a single state S_t transitioning into another single state S_{t+1}, the transition probability is defined by:

$$P_{S_t S_{t+1}} = P[S_{t+1}|S_t] \tag{4.2}$$

A state transition matrix, $\mathbf{P}$, can define the transition probabilities of all state S transitioning into all next states S_{t+1}, and takes the form of:

$$\mathbf{P} = \begin{bmatrix} P_{11} & ... & P_{1n} \\ \\ P_{n1} & ... & P_{nn}, \end{bmatrix} \tag{4.3}$$

where P_{11} is the probability of being in state S_1 and transitioning back into state S_1, and P_{1n} is the probability of being in state S_1 and transitioning into state S_n.

A Markov process is therefore a memoryless random process, in other words, a sequence of random states $S_1, S_2, ..$, all with the Markov property.

A Markov Process is also known as a Markov Chain, and it is a tuple of $< S, \mathbf{P} >$, where S is a finite set of states, and $\mathbf{P}$ is the state transition probability matrix.

Figure 4.1 shows an example of what a Markov chain looks like. Each node represents a state of the Markov Chain, and each arrow represents the probability of transitioning into another state. For example, the probability of being in state 1 and transitioning into state 2 is 1/4. The formal definition of the transition probability matrix $\mathbf{P}$ for the Markov Chain shown in Figure 4.1 is as follows:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/4 & 1/4 & 0 & 0 \\ 0 & 1/3 & 0 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1, \end{bmatrix} \tag{4.4}$$

where the first row and first column represents the probability of being in state 0 and transitioning into state 0, more precisely, $P[S_0|S_0]$. From 4.4, state 0, 4, and 5 are observed to be absorbing states, defined to be a state that once entered, cannot be left.

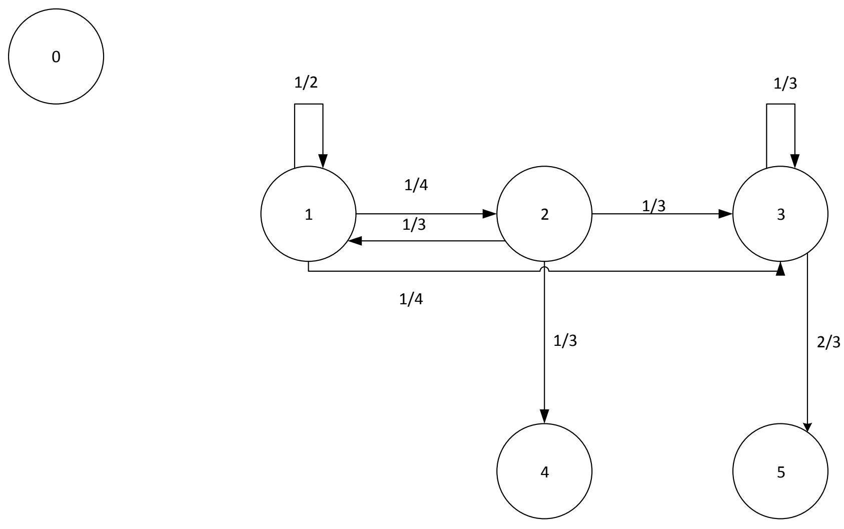

Figure 4.1: Markov Chain.

An Markov reward process (MRP) is a Markov chain with “rewards” [58]. In other words, there are rewards for reaching certain states. The Markov reward process is a tuple of $< S, \mathbf{P}, R, \gamma >$, where S is a finite set of states, $\mathbf{P}$ is the state transition probability matrix, R is a reward function, and γ is a discount factor.

Figure 4.2 shows a Markov reward process. The state S and transition probability matrix $\mathbf{P}$ behaves the same as a Markov chain. However, there is now an reward associated with transitioning from one state to another. For instance, by reaching the state "Pass", a reward of 10 is obtained. This formulation allows being in some states to be more favourable than other states.

The discount factor γ is a penalty given to future states. There are many reasons to include a discount factor γ, some reasons are: uncertainty about the future, future value discounts in finance, and animal instinctive drive to seek immediate rewards.

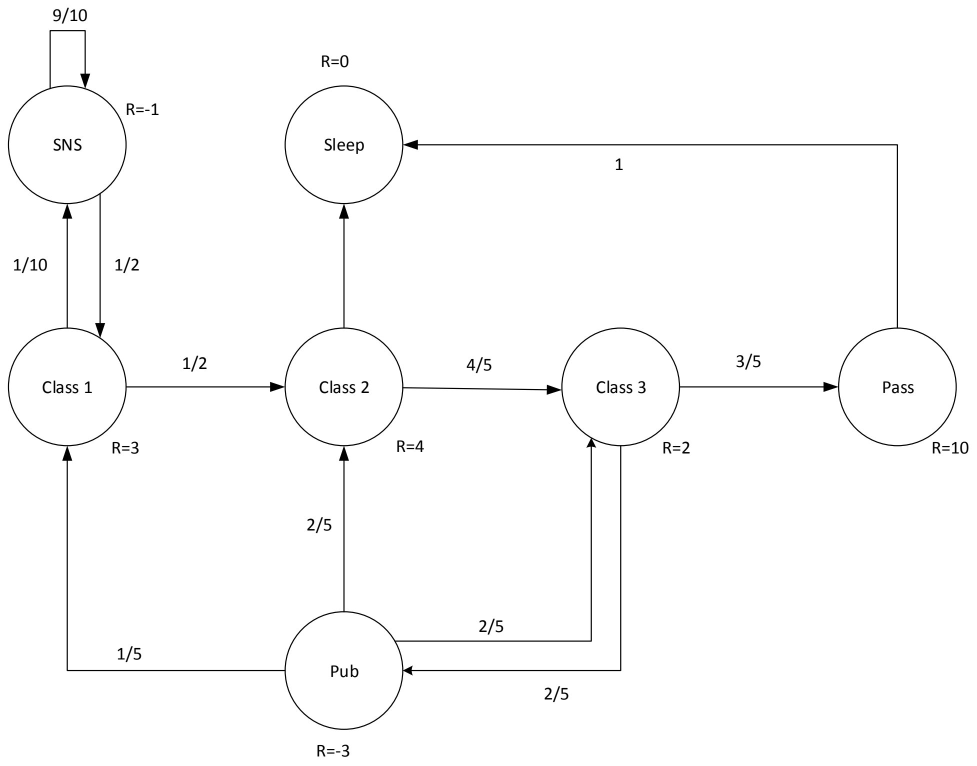

Figure 4.2: Markov Reward Process.

There needs a return that summarizes the “goodness” of an experience. The return is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + .. = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{4.5}$$

where γ is the discount and it takes on a range between 0 and 1, and R is the immediate reward of each step. Having a discount factor of 1 means that there are no discounts to future steps, and every step is as important as the current one. Conversely, having a discount of 0 means that only the immediate step is of value, and no future steps are taken into consideration.

Having only the return is insufficient to represent the whole MRP, as long-term return for any given state S is needed.

A value function, $v(S)$ represents the expected value of state S and is given as follows:

$$v(S) = E[G_t | S_t = S], \tag{4.6}$$

where $v(S)$ is the expected return of state S.

The value function can further be broken down into the following:

$$v(S) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = S], \tag{4.7}$$

where R_{t+1} is the reward of the immediate next step, and $v(S_{t+1})$ is the expected long-term value of the next state. This intuitively makes sense, as the long term value of the current state is the immediate reward of moving to the next state, plus the expected reward of the next state.

However, this only holds true if the next state is deterministic, but if the next state is probabilistic, as in, there are different probabilities of transitioning into different states, then another solution is needed, and is as follows:

$$v(S) = \mathbf{R} + \gamma \sum_{S' \epsilon S} \mathbf{P}_{SS'} v(S'), \tag{4.8}$$

where $\mathbf{R}$ is the immediate reward of transitioning out of the current state S, and $\mathbf{P}_{SS'}$ is the probability transition matrix that summarizes the probability of transitioning from state S into any next state S'. The value function $v(S')$ is the expected long-term value of any next state $v(S')$.

The above equation, when summarized, gives the Bellman equation, and is expressed concisely using matrices, as follows:

$$\mathbf{v} = \mathbf{R} + \gamma \mathbf{P} \mathbf{v}, \tag{4.9}$$

where $\mathbf{v}$ is the value vector, which contains the expected value of all states, and $\mathbf{R}$ is the reward vector, which contains the reward of all states. The probability transition matrix is $\mathbf{P}$, which contains the transition probability of going from state S to a next state. To express the Bellman Equation more clearly, the following is the expanded vectors and matrices form:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} \dots P_{1n} \\ P_{21} \dots P_{2n} \\ \vdots \\ P_{n1} \dots P_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{4.10}$$

Since the Bellman equation is linear, it can be solved as the following:

$$\mathbf{v} = (\mathbf{I} - \gamma \mathbf{P})^{-1} \mathbf{R} \tag{4.11}$$

However, solving the above equation is computationally expensive, as inverting a matrix is a $O(N^3)$ operation. The only time that the above equation is solvable is when the state space is small. For PEV charging scheduling, the state space is much too vast to solve linearly using the above method.

A Markov decision process is very similar to an MRP, except it has an action. A MDP is a tuple of $< S, A, \mathbf{P}, R, \gamma >$, where S is a finite set of states, A is a finite set of actions, $\mathbf{P}$ is a state transition probability matrix, R is a reward function, and γ is a discount factor. Refer to 4.3 for what a MDP looks like.

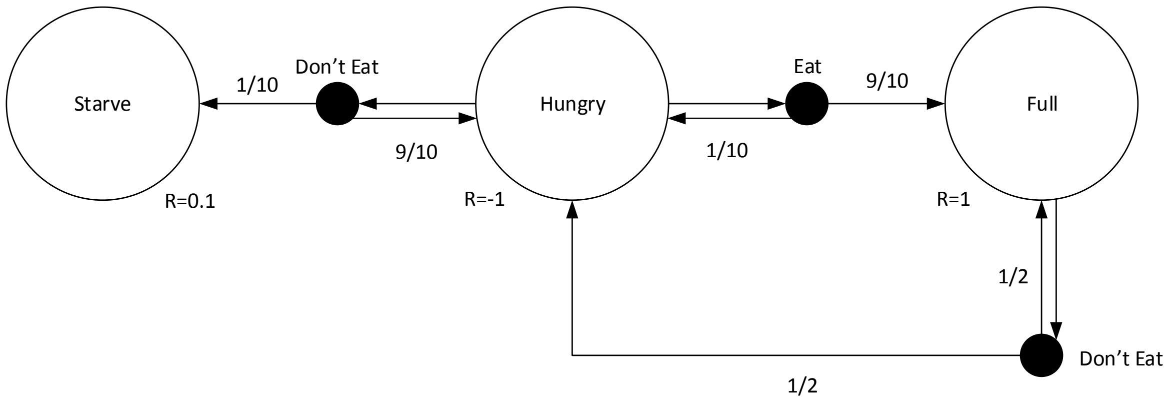

Figure 4.3: Markov Decision Process.

In the above figure, if current state is "Hungry", then there are two actions which can be taken, either "Don't Eat" or "Eat". If the action "Eat" is taken, there is a 0.9 probability the next state reached is "Full" and a reward of 1 is obtained. There is also a 0.1 probability that the state "Hungry" is transitioned back into, in which case a reward of -1 is obtained.

The probability transition matrix, P ,is now expressed as follows, since there are actions that can be taken which results in different state transitions:

$$\mathbf{P}^a_{SS'} = P[S_{t+1} = S' | S_t = S, A_t = a] \tag{4.12}$$

And the reward function is also affected by actions, and is expressed as follows:

$$\mathbf{R}^a_S = E[R_{t+1} | S_t = S, A_t = a] \tag{4.13}$$

To choose a set of actions, a policy is needed, and is defined as the probability of choosing an action when given a state. A policy π as follows:

$$\pi(a|S) = P[A_t = a|S_t = S] \tag{4.14}$$

Therefore it can be said that given an MDP and a policy, the state and reward sequence becomes a MRP, as actions will be determined by the policy, or in other words:

$$\mathbf{P}^{\pi}_{SS'} = \sum_{a \epsilon A} \pi(a|s)\mathbf{P}^{a}_{SS'} \tag{4.15}$$

$$\mathbf{R}^{\pi}_{S} = \sum_{a \epsilon A} \pi(a|s)\mathbf{R}^{a}_{S} \tag{4.16}$$

To discover an optimal policy, an expected long term return function that takes action into account is needed. Similar to the value function for MRPs $v(S)$, an action-value function $q_\pi(s, a)$ is shown as follows:

$$q_\pi(S, a) = E[G_t|S_t = S, A_t = a] \tag{4.17}$$

The action-value function can then be expressed as the Bellman expectation equation, and is as follows:

$$q_\pi(S, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, a_{t+1})|S_t = S, A_t = a] \tag{4.18}$$

Similar to MRPs, if the next transitioned state after taking an action is not fully deterministic, then a probability transition matrix $\mathbf{P}$ is required. The Bellman expectation equation then becomes:

$$q_\pi(S, a) = R^a_S + \gamma \sum_{S' \epsilon S} \mathbf{P}^a_{SS'} v_\pi(S'), \tag{4.19}$$

where $\mathbf{P}^a_{SS'}$ is the probability transition matrix for taking action a in state S and transitioning into S'. More concisely, in vector and matrices form, the Bellman expectation equation can be expressed as:

$$\mathbf{v}_\pi = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}_\pi, \tag{4.20}$$

where $\mathbf{v}$ is the value function for policy π, $\mathbf{R}^\pi$ is the reward vector for following policy π, and $\mathbf{P}^\pi$ is the transition probability matrix for following policy π.

Similarly to Bellman's equation, the Bellman's expectation equation can be solved linearly as:

$$\mathbf{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{R}^\pi \tag{4.21}$$

This is similar to the Bellman equation solution, and it also has a computation complexity of $O(n^3)$.

To solve the Bellman expectation with an iterative method, a dynamic programming approach can be used and is discussed in the following section.

4.2 Dynamic Programming

If we have sufficient usage history of the PEV user's driving behavior and electricity pricing is deterministic (for instance in Toronto), a dynamic programming approach can be used to find the long term value function for every state action pair.

Using the user's driving history, we can build a probability transition matrix. For instance, if we know the probability of the user going to work at 8 am on a Monday is 95%, and we know the probability of the user being home at 5 pm is 95%, we can then imply that P_{M8M5} is 95%.

An entire probability transition matrix, $\mathbf{P}$, can then be built for every single day and every single minute for the user.

For a dynamic programming approach to work, we must have full knowledge of the MDP. With the MDP fully known, we can do planning iteratively [9].

To discover the value function of any given policy π, we can iterate over the Bellman expectation equation as following:

$$v_{t+1}(S) = \sum_{a \epsilon A} \pi(a|S)(R_S^a + \gamma \sum_{S' \epsilon S} P_{SS'}^a v_t(S')) \tag{4.22}$$

For simplicity, the above equation can be expressed in vector and matrices form as the following:

$$\mathbf{v}_{t+1} = (\mathbf{R}^\pi - \gamma \mathbf{P}^\pi)\mathbf{R}^t \tag{4.23}$$

To improve a policy π, we must first evaluate the value function $v_\pi(S)$, which is shown as:

$$v_\pi(S) = E[R_{t+1} + \gamma R_{t+2} + ...|S_t = S] \tag{4.24}$$

After evaluating the value function of the policy, we can then act greedily according to the value function, more specifically, choosing to take actions that results us landing in the state which has the highest value. A greedy policy is shown below:

$$\pi' = greedy(v_\pi) \tag{4.25}$$

In other words, we first evaluate the value function of the policy, update the policy by taking greedy actions. This step is repeated as many times as possible until there are no more changes in the policy.

This process is called policy iteration, and it always converges to the optimal policy π^*, as long as sufficient iterations are ran.

We can also improve the policy directly using a method called policy improvement. With policy improvement, instead of evaluating the value function, and then changing the policy based on the greedy method, the policy is directly updated using the state-action value, $q_\pi(S, a)$.

The policy can be improved by acting greedily with respect to the action, and is shown as follows:

$$\pi'(S) = argmax_{a \epsilon A} q_\pi(s, a) \tag{4.26}$$

The above equation will allow the agent to choose the most greedy action given any state-action pair. The above will improve the current policy by taking the action in a state that will yield the maximum q-value, and the following will hold true:

$$q_\pi(S, \pi'(s)) \geq argmax_{a \epsilon A} q_\pi(s, a) \tag{4.27}$$

This means that the new policy will be taking the action that yields the highest q-value in state S, and this policy will yield a higher or at worst, the same q-value as the previous policy. Consequently, the value function is improved as well, as in $v_{\pi'}(S) \geq v_\pi(S)$.

When improvements stop, we will know that the current policy will yield the best state-action value, and the Bellman optimality equation, shown as:

$$v_\pi(S) = max_{a \epsilon A} q_\pi(S, a), \tag{4.28}$$

has been satisfied.

The downside of using this method to schedule PEV charging, are as follows:

- The user's driving history must be known, but is often unknown in real-world scenarios.
- Future electricity price must be known, and in real-world scenarios this is only possible for cities which have static electricity price schedules.
- The policy is learnt off-line, and if the user's driving behaviour changes, the algorithm must be restarted using newly obtained data.

Due to these limitations, a lot of real-world PEV charging scenarios cannot be solved using dynamic programming. To solve the off-line learning problem, or more specifically, when a user changes his behaviour, an on-line method using model-free reinforcement learning can be used. This is discussed in the following section.

4.3 Model-Free Policy Evaluation

Model-free RL is able to learn a policy without any prior data, and obtains data through sampling or observing experiences. There are many variations of Model-Free RL, and a few are discussed in the following subsections.

4.3.1 Monte-Carlo Learning

Monte-Carlo policy evaluation uses sampling, and obtains an empirical mean return. Therefore, instead of assuming the MDP is known, in which case we can calculate expected return using dynamic programming, an empirical mean return, obtained from sampling, is used to update the state value [59]. The First-Visit Monte-Carlo policy algorithm is often used and is shown in Algorithm 1.

The first-visit policy is used when only the first visit to a state during an episode is important, and the rest of the visits are unimportant or redundant. For instance, traversing out of a maze, where landing back into the same location will have no impact on future actions. If repeated visits to a state is important, then the every-visit policy is used, and is shown in 2.

Similarly to the first-visit policy, instead of only updating a state value function during the first visit within an episode, every visit to that state will have a state value function update. For PEV charging scheduling, every-visit is important, and Algorithm 2 would be used to figure out the value function of the policy.

The value function update can be simplified into the following:

$$V(S_t) \leftarrow V(S_t) + 1/N(S_t)(G_t - V(S_t)), \tag{4.29}$$

Algorithm 1 First-Visit Monte-Carlo

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
1: Initialize increment counter: $N(S) = 0$ for all states
2: Initialize state visit counter: $M(S) = 0$ for all states
3: Initialize total return: $T(S) = 0$ for all states
4: Initialize value function: $V(S) = 0$ for all states
5: while $V(S) \neq v_\pi(S)$ do
6:     Start episode following policy $\pi$
7:     if $M(S) = 0$ then
8:         $N(S) \leftarrow N(S) + 1$
9:         $T(S) \leftarrow T(S) + G_t$
10:        $V(S) = T(S)/N(S)$
11:    else
12:        Do nothing
13:    end if
14: end while
```

Algorithm 2 Every-Visit Monte-Carlo

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
1: Initialize increment counter: $N(S) = 0$ for all states
2: Initialize total return: $T(S) = 0$ for all states
3: Initialize value function: $V(S) = 0$ for all states
4: while $V(S) \neq v_\pi(S)$ do
5:     Start episode following policy $\pi$
6:     $N(S) \leftarrow N(S) + 1$
7:     $T(S) \leftarrow T(S) + G_t$
8:     $V(S) = T(S)/N(S)$
9: end while
```

where G_t is the cumulative reward of the episode, and $N(S_t)$ and $V(S_t)$ are the increment counter and value function respectively.

The key features of Monte-Carlo learning is as follows:

- High variance

- Low Bias
- Good convergence properties
- Not sensitive to initial value
- Simple to use
- Does not exploit the Markov property

4.3.2 Temporal-Difference

Temporal-difference learning is another method that ”learns” the true value function of any policy. However, unlike Monte-Carlo learning, Temporal-difference learning does not have to finish an episode to update the value function, more specifically, temporal-difference learning learns from incomplete episodes [60]. The one-step temporal-difference algorithm is shown in Algorithm 3.

Algorithm 3 One-Step Temporal-Difference

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
Input: Update step size: $\alpha$
1: Initialize value function: $V(S) = 0$ for all states
2: while $V(S) \neq v_\pi(S)$ do
3:     Perform single step following policy $\pi$
4:     $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
5: end while
```

The algorithm attempts to update the current state value function with its immediate reward R_{t+1} plus the discounted value of the next state $\gamma V(S_{t+1})$, which is called the target. The difference between the target and the value of the current state, which is called the error, is multiplied by a step size α. The error value is then added onto the previously computed state value, to update the previous state value.

The key features of Temporal-Difference learning is as follows:

- Low variance
- Medium Bias
- Very efficient
- Sensitive to initial value
- Exploits the Markov property

4.3.3 Temporal-Difference Lambda

Monte-Carlo learning updates the state value function after the entire episode ends, and Temporal-Difference updates the state value after one step. The Temporal-Difference Lambda combines the two methods, and updates the value function every step until the end of an episode [61].

Let n be the number of steps taken in an episode. A weight of $(1-\lambda)\lambda^{n-1}$ is used to assign the weight of each individual step. The higher the λ, the more importance is given to later steps in an episode.

The total return of an episode using the Temporal-Difference Lambda method is as follows:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{n} \lambda^{n-1} G_t^n \tag{4.30}$$

The update algorithm for Temporal-Difference Lambda is shown in Algorithm 4.

Algorithm 4 Temporal-Difference Lamda

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
Input: Update step size: $\alpha$
1: Initialize value function: $V(S) = 0$ for all states
2: while $V(S) \neq v_\pi(S)$ do
3:     Begin episode following policy $\pi$
4:     $V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda + \gamma V(S_{t+1}) - V(S_t))$
5: end while
```

This method will retain the advantages of both Monte-Carlo learning and Temporal-Difference learning, and has a good balance between variance and bias.

These methods can only attempt to learn the value function of any given policy, when the MDP is unknown. They cannot, however, learn an optimal or near-optimal policy. The following subsection will discuss how a policy can be learned without knowledge of the MDP.

4.4 Model-Free Control

There are many methods of model-free control reinforcement learning, a few will be discussed in the following sections.

4.4.1 On-Policy Learning

An on-policy learning method attempts to learn the best policy, while following the policy. Precisely, the RL agent will follow a policy, and update the same policy throughout the episode.

4.4.1.1 Monte-Carlo Control

The most common policy to follow in RL is the ϵ-greedy policy. According to the policy, the action a, will be chosen as $a = argmax_{a \epsilon A} q(s, a)$ with probability $1 - \epsilon$. And with probability ϵ, the action will be chosen at random. This policy gives a chance to explore to unseen state, but at the same time takes greedy policy to ensure maximized state-action value [62].

For policy improvement using the ϵ-greedy policy, the update is as follows:

$$q_\pi(S, \pi'(S)) = \epsilon/m \sum_{a \epsilon A} q_\pi(S, a) + (1 - \epsilon) max_{a \epsilon A} q_\pi(S, a), \tag{4.31}$$

where π is the current policy, π' is the updated and improved policy, $q_\pi(S, \pi'(S))$ is the state-action value following the improved policy, ϵ is the exploration-exploitation variable, m is the total number of available actions in state S, and $q_\pi(S, a)$ is the state-action value of taking action a in state S.

It can be shown that by taking the ϵ-greedy policy π, the improved ϵ-greedy policy π' is will have either equal or greater state-action value, as in:

$$\epsilon/m \sum_{a \epsilon A} q_\pi(S, a) + (1-\epsilon) max_{a \epsilon A} q_\pi(S, a) \geq \epsilon/m \sum_{a \epsilon A} q_\pi(S, a) + (1-\epsilon) \sum_{a \epsilon A} \frac{\pi(a|S) - \epsilon/m}{1-\epsilon} q_\pi(S, a) \quad (4.32)$$

It then follows that:

$$v_{\pi'}(S) \geq v_\pi(S) \quad (4.33)$$

By the law of large numbers, if enough state-action pairs are explored, or explored infinite number of times, the policy will converge on a greedy policy, as in:

$$\lim_{t \to \infty} \pi_t(a|S) = \pi^*, \quad (4.34)$$

where π^* is the optimal policy.

4.4.1.2 SARSA

On-policy SARSA is similar to temporal difference learning, except that it is applied to a control problem. Instead of updating the state-action value at the end of an episode, the immediate reward is added to the state-action value of the next state [63].

The update is as follows:

$$q(S, a) \leftarrow Q(S, a) + \alpha(R + \gamma q(S', a') - Q(S, a)), \quad (4.35)$$

where $Q(S, a)$ is the current state-action value, α is the update step size, R is the immediate reward, γ is the discount factor, and $q(S', a')$ is the state-action value of the state-action pair arrived at.

The advantage of using SARSA over Monte-Carlo control is that it updates online, has lower variance, and can learn from incomplete sequences.

The complete On-Policy SARSA algorithm is shown in Algorithm 5.

Algorithm 5 SARSA

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
Input: Update step size: $\alpha$
1: Initialize state-action value $Q(S, a)$
2: while $Q(S) \neq q_\pi(S)$ do
3:     Begin episode following policy $\pi$
4:     Take action $a$ from $S$ following policy derived from $Q(S, a)$
5:     $Q(S, a) \leftarrow Q(S, a) + \alpha[R + \gamma Q(S', a') - Q(S, a)]$
6:     $S \leftarrow S'$
7:     $A \leftarrow A'$
8: end while
```

The On-Policy SARSA algorithm will converge when number of experiences approaches infinity. In some cases, a sub-optimal solution can be obtained using a large number of experiences.

A variation of SARSA algorithm named SARSA-$lambda$ is used to reduce bias. Similar to the Temporal-Difference learning algorithm, SARSA-λ updates the state-action value with the $q(S, a)$ value of every state and action pair taken in an episode. The weight value is the same as the one shown in equation 4.30.

Using weight $(1 - \lambda)\lambda^{t-1}$ for every step t, the state-action value $q(S, a)$ for SARSA-λ is as follows:

$$q_t^\lambda = (1 - \lambda) \sum_{t=1}^{\infty} \lambda^{t-1} q_t \tag{4.36}$$

The update for the state-action value $q(S, a)$ is shown as:

$$q(S, a) \leftarrow q(S, a) + \alpha(q_t^\lambda - q(S, a)), \tag{4.37}$$

where α is the update step size.

A backward view SARSA-λ algorithm using eligibility trace is shown in Algorithm 6 [64].

Algorithm 6 Backward SARSA

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
Input: Update step size: $\alpha$
1: Initialize state-action value $Q(S, a)$
2: Initialize eligibility trace $E(S, a) = 0$ for all states and action
3: while $Q(S) \neq q_\pi(S)$ do
4:     Begin episode following policy $\pi$
5:     Take action $a$ from $S$ following policy derived from $Q(S, a)$
6:     Observe $R$ and $S'$
7:     Choose $a'$ from $S'$ using policy derived from $Q(S', a')$
8:     $\delta \leftarrow R + \gamma q(S', A') - q(S, A)$
9:     $E(S, a) \leftarrow E(S, a) + 1$
10:    For all $S$ and $a$:
11:    $Q(S, a) \leftarrow Q(S, a) + \alpha\delta E(S, a)$
12:    $E(S, a) \leftarrow \gamma\lambda E(S, a)$
13:    $S \leftarrow S'$
14:    $A \leftarrow A'$
15: end while
```

The reason a backward view is used is to be able to memorize everything that happened in an episode, and then go ”backwards” and update all the state-action values that occurred in the episode.

4.4.2 Off-Policy Learning

An off-policy learning method is when the RL agent learns an optimal policy while following another policy. In many real-world scenarios, experiences are limited and some episodes must be learned from historical data. For instance, learning by observing humans, learning from experiences obtained from other policies, or learning an optimal policy while following an exploratory policy.

These examples all have very practical real-life uses, which makes off-policy learning very practical.

4.4.2.1 Q-Learning

The update algorithm, Q-learning, updates the policy using the maximum state-action value of the next possible state-action values [65]. The update algorithm is as follows:

$$q(S,a) \leftarrow q(S,a) + \alpha(R + \gamma \max_{a'} q(S',a') - q(S,a) \tag{4.38}$$

The full Q-Learning algorithm for off-policy control is shown in Algorithm 7. The Q-Learning algorithm

Algorithm 7 Q-Learning

```
Input: State: $S$
Input: Action: $a$
Input: Reward: $R$
Input: Update step size: $\alpha$
1: Initialize state-action value $Q(S,a)$
2: Initialize eligibility trace $E(S,a) = 0$ for all states and action
3: while $Q(S) \neq q_\pi(S)$ do
4:     Begin episode following policy $\pi$
5:     Take action $a$ from $S$ following policy derived from $Q(S,a)$
6:     Observe $R$ and $S'$
7:     $q(S,a) \leftarrow q(S,a) + \alpha(R + \gamma max_{a'} q(S',a') - q(S,a)$
8:     $S \leftarrow S'$
9: end while
```

outlined above only looks at the next state-action value, and chooses the max state-action value over all possible values.

These model-free control reinforcement learning methods are widely used in many of today's research and industry problems, famous applications include: AlphaGo, Resource management in computer clusters, traffic light control, and robotics.

For PEV charging scheduling, many research paper using the above method had been published and outlined in previous sections.

There are a few downsides to this approach, the most important ones are listed below:

- Slow convergence: Using a model-free method on a PEV user without any historical data will reach convergence far too slowly, the user will be expected to drive for a very long time for the RL agent to learn an optimal policy.

- Not applicable for large state-action space: If the electricity price is continuous, there would be a near infinite number of state action space, and a method which can generalize would be needed.

Due to the above limitations, a state-action value estimation method is proposed in the following subsection.

4.5 Fitted Q-Iteration

One method of learning a function approximation is an off-line method called fitted Q-iteration. The algorithm learns the optimal policy from a set of transition samples D. The transition samples are in the form of (S, a, R, S'), where S is a state, a is the action taken in that state, R is the immediate reward received, and S' is the state arrived at. The samples can and often is obtained from historical samples [66].

Any supervised learning algorithm can be used to generalize the state-action value $q(S, a)$, and the loss function to be optimized will be dependent on the supervised learning algorithm chosen.

This is especially useful when new experiences are difficult or costly to obtain. The fitted Q-iteration algorithm is shown in 8.

Algorithm 8 Fitted Q-Iteration

```
Input: Transition samples $D$
1: Initialize supervised learning algorithm parameters
2: while $q(S) \neq q_\pi(S)$ do
3:     Initialize state-action value $Q(S, a : \theta)$
4:     Sample historical experience $(S, a, R, S')$
5:     Update $q(S, a)$ towards target value $R_t + \gamma max_{a'} q(S', a')$
6: end while
```

This algorithm is useful when state-action space are vast or continuous, such as the PEV charging scheduling problem that we are considering. However, this algorithm needs historical user behaviour data and cannot update on-line, which will not be able to adapt to any user behaviour changes.

4.6 Model Based Reinforcement Learning

Model based reinforcement learning attempts to construct a model which can generate or simulate experiences. This is useful when real experiences are scarce or hard to obtain. Model based reinforcement learning can plan or learn a value function or policy directly from a model without any real life experiences [67].

The advantages of a model based reinforcement learning is a model can be learned efficiently using a supervised learning method. However, there are now two sources of approximation error, one from learning the model, and another from constructing a value function.

A model M is a representation of a MDP $< S, a, P, R >$. A model M is therefore a representation of state transitions and rewards. The model consists of $< P_n, R_n >$. More specifically, a next state S_{t+1} can be created with $P_n(S_{t+1}|S_t, a_t)$, which gives the probability of transitioning into the next state S_{t+1} when given current state S_t and action taken a_t. The reward should also be represented by the model, that is, $R_{t+1} = R_n(R_{t+1}|S_t, a_t)$. As seen, the reward R_{t+1} is obtained when action a_t is taken in state S_t. In summary:

$$P[S_{t+1}, R_{t+1}|S_t, a_t] = P[S_{t+1}|S_t, a_t]P[R_{t+1}|S_t, a_t] \tag{4.39}$$

This assumes that state transition and rewards are independent, which is typically the case. The model should fully represent the transition probabilities and rewards of every state-action pair.

Learning the model can be framed as a supervised learning problem or a distribution problem. The goal is to estimate model M from experiences $S_1, a_1, R_2, ..., S_T$. The supervised learning algorithm will attempt to predict the state S_2 and reward R_2 from state S_1 and action a_1. More specifically, the supervised learning algorithm attempts to predict the following:

$$S_t, a_t \rightarrow R_{t+1}, S_{t+1}, \tag{4.40}$$

where S_t is the previous state, a_t is the action taken in the previous state, R_{t+1} is the reward obtained, and S_{t+1} is the state transitioned into.

Two separate supervised learning algorithms can be used to approximate S_{t+1} and R_{t+1} separately. With enough training examples, the model should sufficiently represent reality, and can generate simulated experiences to train the reinforcement learning model.

In the following section, a method that combines model-free and model-based reinforcement learning named Dyna-Q algorithm is presented and utilized to solve the PEV charging scheduling problem. Alongside Dyna-Q algorithm, a value approximation method named deep-Q network, which has shown to be an improvement over the existing fitted Q-iteration, is also used to solve the PEV charging problem. The Adam optimizer is used in back-propagation to train the neural network in deep-Q network.

4.7 Deep-Q Network and Dyna-Q Algorithm

A traditional Q-table update method, in which the Q value of every state-action pair is stored in a table, updates according to the Q-learning algorithm [68]:

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha(R_t + \gamma \max_{a_{t+1}} Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t)). \tag{4.41}$$

The update requires the current state S_t, the action taken a_t, the immediate reward R_t, the next state S_{t+1}, and the action a_{t+1} to be taken in S_{t+1} that yields the highest state-action pair value $Q(S_{t+1}, a_{t+1})$. However, the Q-table method is impractical for the PEV charging problem as the state-action space is too vast. Therefore we use a neural network to estimate the Q value instead, as this allows the approximation of Q value of unvisited state-action pairs. This method is called the Deep-Q Network (DQN) and is used to approximate the $Q(S_t, a_t)$ values of all existing state-action space. The DQN will update from both real and model generated experience as this allows quicker learning time. For a DQN, instead of updating the table value of $Q(S_t, a_t)$, the weights θ, of the neural network are updated instead. Traditionally, the stochastic gradient descent algorithm is used to update the neural network weights. But recent studies have found that the adaptive moment estimation (ADAM) optimizer, shown in Algorithm 9, can reach convergence faster when the user behaviour is stationary[69]. The ADAM update performed seeks to minimize the following objective function with respect to θ:

$$f(\theta) = (R_t + \gamma \max_{a_{t+1}} Q(S_{t+1}, a_{t+1}; \theta) - Q(S_t, a_t; \theta))^2 \tag{4.42}$$

Algorithm 9 Adam Optimizer

```
Input: Step-size: $\alpha$
Input: Exponential decay rates for the moment estimates: $\beta_1, \beta_2 \epsilon [0, 1)$
Input: Stochastic objective function with parameters $\theta$: $f(\theta)$
Input: $\theta_0$: Initial parameter vector
 1: Initialize first moment vector $m_0 \leftarrow 0$
 2: Initialize second moment vector $v_0 \leftarrow 0$
 3: Initialize time step $t \leftarrow 0$
 4: $t \leftarrow t + 1$
 5: Get gradients w.r.t stochastic objective at time step $t$: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
 6: Update biased first moment estimate:
 7: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
 8: Update biased second raw moment estimate:
 9: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
10: Compute bias-corrected first moment estimate: $m'_t \leftarrow m_t/(1 - \beta_1^t)$
11: Compute bias-corrected second raw moment estimate: $v'_t \leftarrow v_t/(1 - \beta_2^t)$
12: Update parameters: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot m'_t/(\sqrt{v'_t} + \epsilon)$
```

The update performed is therefore on the squared difference of the predicted value and the one step reward plus the predicted value of the next step with respect to the weights θ. This squared difference is used to update the neural network through back-propagation.

When choosing actions, the RL agent cannot simply choose the action that has the highest $Q(s, a)$ value, in other words, follow a greedy policy. The reason is that the agent using a greedy policy will not "explore" to discover unseen states. Therefore an ϵ-greedy policy is taken instead, where the agent will have an ϵ chance of taking a random action, and $1 - \epsilon$ chance of taking the greedy action. To allow more exploration in the early episodes, and more greedy action taking in later episodes, an ϵ with exponential decay is used [70]. The variable ϵ is chosen and shown in equation (4.43):

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \cdot e^{-\lambda \cdot N}, \tag{4.43}$$

where ϵ_{min} and ϵ_{max} are the minimum and maximum ϵ respectively; λ is the decaying factor; and N is the number of episodes.

Taking an ϵ-greedy policy is not ideal in the real world, because there is a probability of taking an unwanted random action. Therefore the RL agent will only follow the ϵ-greedy policy in model-based aspect of the RL scheme, where experience is artificially generated. The RL agent will follow a pure greedy policy when taking actions in the real world.

There is a problem when updating the deep neural network using the ADAM optimizer. The problem lies in that the neural network used to predict $Q(S, a; \theta)$ is the same neural network that is currently being updated, and this will create instability as the neural network is chasing an ever moving target. To combat this issue, a separate target network is created, in which we hold the current weights θ' constant in a separate memory for T time steps, and update the actual weights θ of the neural network using the state-action value predicted with the memorized weights θ'. More precisely, instead of using $\text{argmax}_{a_{t+1}} Q(S_{t+1}, a_{t+1}; \theta)$ to perform Q-learning update, we use $\text{argmax}_{a_{t+1}} Q'(S_{t+1}, a_{t+1}\theta')$, where Q' is predicted using memorized and constant weights θ' from T time steps ago. After U time steps, we set the memory weights equal to the actual weights of the neural network, as in $\theta = \theta'$ [71].

A user behaviour model is used to generate "imagined" experience to allow the DQN train faster with less real experience. The model creation and update process are discussed in depth in section V, and this approach is called the Dyna-Q algorithm. Every time a real experience occurs, the model is updated alongside the DQN, the DQN then samples from a model generated mini batch of experience and updates its weights. There is an endless loop in the algorithm because it updates on-line at every time step t, and the RL agent will use the most recently updated policy. Refer to Figure 4.4 for a representation of the Dyna-Q algorithm. The full DQN update algorithm, with the Dyna-Q algorithm and separate target network, is shown in Algorithm 10.

Algorithm 10 Deep Q-Network

```
 1: Initialize action-value function $Q$ with random weights $\theta$
 2: Initialize target action-value function $Q'$ with weights $\theta' = \theta$
 3: Initialize counter $c = 0$ and time step $U$
 4: loop
 5:     Obtain current state $S_t$ from real experience
 6:     With probability $\varepsilon$ select a random charging action $a_t$
 7:     Otherwise predict all state-action value in state $S_t$ with neural network and select $a_t = argmax_a Q(S_t, a; \theta)$
 8:     Execute action $a_t$ and observe reward $R_t$ and next state $S_{t+1}$
 9:     Perform Adam Optimization on $(R_t + \gamma \max_{a+1} Q'(S_{t+1}, a_{t+1}; \theta') - Q(S_t, a_t; \theta))^2$ with respect to the weights $\theta$
10:     Update model with the obtained $S_t, a_t, R_t, S_{t+1}$
11:     Generate a mini batch of transitions $S_t, a_t, R_t, S_{t+1}$ from model
12:     Perform Adam Optimization on mini batch generated from model with respect to weight $\theta$
13:     Increment counter $c$
14:     if $c = U$ then
15:         Set weights $\theta' = \theta$, Reset counter $c = 0$
16:     else
17:         Do nothing
18:     end if
19: end loop
```

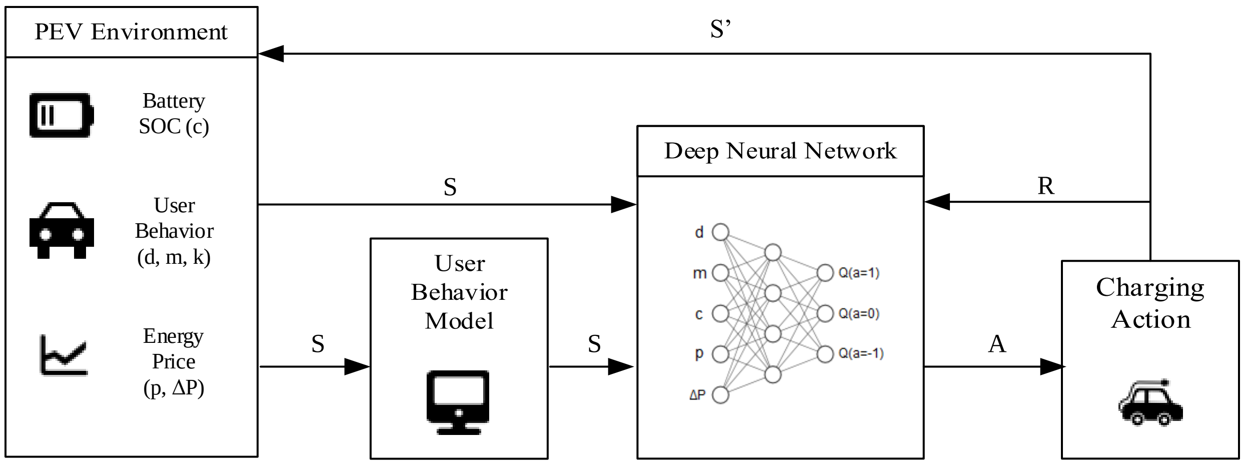

Figure 4.4: Dyna-Q Environment

Chapter 5

User Experience Model Generation

A model is used to generate experience that is used to train the DQN alongside real experience. The model of the PEV owner's experience should accurately reflect the driving routine and trip behaviour of the user. There are three aspects of a PEV owner that requires modelling:

- Probability of PEV being plugged in during each time step: this models the probability that the PEV is plugged in at home for every minute of a day.
- Trip duration: this models the trip duration of trips, when the departure time is known.
- Charge used: this models the charge used for trips, when the departure time and the trip duration are known.

These three aspects should accurately represent any user's driving pattern, as it reflects when the PEV owner takes a trip, how long each trip is, and how much charge is used per trip. The model is updated every time a real experience occurs. The model can be initialized with values if there are some knowledge of the user's behaviours, the details will be discussed in the following subsections.

5.1 PEV at Home Probability

The probability that the PEV is plugged in with respect to the time and day is stored in a look-up table. This table is continuously updated as data is gathered on the PEV owner's driving routine. The value stored for each time slot is the mean of all the times the PEV is plugged-in or away during that time. During each update, we assign a "1" for when the PEV is at home, and a "0" for when the PEV is away. Table 1 shows an example of the look-up table. For instance, according to Table 1, the probability that the PEV is home

Table 5.1: PEV plug-in probability

Day / Min	Sun	M	T	W	Th	F	Sat
0	0.875	0.995	0.995	0.993	0.992	0.984	0.943
1	0.867	0.994	0.994	0.993	0.998	0.976	0.934
2	0.834	0.995	0.993	0.995	0.995	0.994	0.965
3	0.894	0.997	0.994	0.992	0.993	0.996	0.943 .. ..
1439	0.893	0.987	0.974	0.982	0.994	0.994	0.941

and plugged-in at midnight on Sunday is 0.875.

If nothing is known about the user's driving pattern initially, a constant can be assigned to each value in the table.

Assigning a lower percentage to the PEV being plugged-in implies that the probability of the user taking a trip is high, and the RL agent will try to avoid the battery SOC running out during a trip by keeping battery SOC high at all times.

Since updating the table requires computing the mean of all previous data, memory and computation requirement will grow larger as more user experience is acquired. A more practical on-line update algorithm, in which we can reduce the computing and memory requirements, is shown below,

$$N[d,m] \leftarrow N[d,m] + 1 \tag{5.1}$$

$$P[d,m] \leftarrow P[d,m] + \frac{1}{N[d,m]}(H - P[d,m]), \tag{5.2}$$

where d and m represent day and minute of a day respectively. The variable $N[d,m]$ is a counter that represents the total number of times a real experience was gathered during that specific time slot and is incremented every time a real experience occurs. $P[d,m]$ is the estimated probability that the user is plugged in during that time; and H is a binary number that is obtained from real experience of the plug-in status of the PEV, with "1" being the PEV plugged-in and "0" being the PEV plugged-out.

Every time a real experience during a time slot occurs, the counter increments, and the probability that the PEV is home or away is updated towards the real experience. At every time slot of every day of the week, the plug-in status of the PEV is obtained, and the probability table is updated. Over time, the PEV owner's plug-in behaviour can be fully described by the probability table.

This accurately represents the plug-in and plug-out behaviour of the PEV owner, and with this it can be assumed that when the PEV is plugged-out, the user has taken a trip.

5.2 Trip Duration

The duration of trips is based on the time and day of departure. We assume the user has routines for each specific day of the week. Specifically, if the user has work on Mondays at 9am, he will leave home at 8am, and come back home from work around 5:50pm to plug in his vehicle. Therefore we assume the duration of his trip will be around 9 hours and 50 minutes given any Monday with a departure time around 8am.

Some variances can be expected for each trip time, but due to the routinely nature of the user's driving behaviour, each trip time will be centred around a mean. A truncated normal distribution for every departure time interval is used to model the probability of trip duration, with the variable bounded above 0, as trip duration cannot be less or equal to 0. The departure time interval is set to every 15 minutes, ie. 8:00am - 8:15am, 8:15am-8:30am, and so on. The truncated normal distribution would have a mean of $\mu = \frac{1}{n}\Sigma_{i=1}^{n} x_i$, and variance $\sigma^2 = \frac{1}{n}\Sigma_{i=1}^{n}(x_i - \mu)^2$, where x_i is the length of current trip i.

This function will be updated every time the PEV owner takes a trip. Updating the PDF requires the mean and variance of the trip duration, which needs to be computed using all previous trip samples.

This requires ever growing memory and computation power. Therefore a more efficient on-line updating algorithm called Welford Online algorithm is used [72], shown as,

$$N[d,t] \leftarrow N[d,t] + 1 \tag{5.3}$$

$$\bar{l}[d,t]_n \leftarrow \bar{l}[d,t]_n + \frac{1}{N[d,t]}(L - \bar{l}[d,t]_{n-1}) \tag{5.4}$$

$$\sigma^2[d,t]_n \leftarrow \sigma^2[d,t]_{n-1} + \frac{1}{N[d,t]}[(L - \bar{l}[d,t]_{n-1})(L - \bar{l}_n) - \sigma^2_{n-1}], \tag{5.5}$$

where d and t are the day and time intervals respectively, and N is a counter for the amount of times this particular day and time interval has been sampled for a trip.

The variables $\bar{l}$ and σ are the mean and standard deviation of the trip duration respectively. The length of the most recent trip obtained from real experience is represented by L.

5.3 Charge Used Per Trip

The charge used per trip is dependent on the nature of the trip. The user's time and day of departure, and the length of the trip is a good indicator of the expected charge used on that trip.

The time of departure is separated into morning, afternoon, evening, and night. The length of the trip is separated into short, medium, and long for trips under 3 hours, from 3 hours to 7 hours, and longer than 7 hours respectively.

A truncated normal distribution for each day and time of departure, and trip length is used to model the probability of the charge that will be used on a particular trip, the distribution is bounded above 0, as no trip can use less or equal to 0 charge. More specifically, if a user drives his PEV every Monday to work from 8am to 5:50pm, he will use a similar amount of charge for that trip, and the normal distribution for departure on Monday mornings, with a long trip length, will be centered around that amount. Similar to trip length modelling, Welford's Online algorithm is used, and is shown below,

$$N[d,t,l] \leftarrow N[d,t,l] + 1 \tag{5.6}$$

$$\bar{c}[d,t,l]_n \leftarrow \bar{c}[d,t,l] + \frac{1}{N[d,t,l]}(M - \bar{c}[d,t,l])_{n-1} \tag{5.7}$$

$$\varphi^2[d,t,l]_n \leftarrow \varphi^2[d,t,l]_{n-1} + \frac{1}{N[d,t,l]}[(M - \bar{c}[d,t,l]_{n-1})(M - \bar{c}_n) - \varphi^2_{n-1}], \tag{5.8}$$

where d, t, and l are day, time slot of departure, and the length of the trip respectively; and N is a counter for the amount of times this particular day, time interval, and length of trip has been sampled. The variables $\bar{c}$ and φ are the mean and standard deviation of the charge used during trip respectively. The charge used of the most recent trip obtained from real experience is represented by M.

5.4 Model State Generation

To generate a PEV charging/discharging experience from model, independent state transitions are created. Specifically, the state at time t: $S_t = [d_t, m_t, c_t, p_t, \Delta P_t]$ and the next state at time $t+1$: $S_{t+1} = [d_{t+1}, m_{t+1}, c_{t+1}, p_{t+1}, \Delta P_{t+1}]$ needs to be generated. The day, minute, and SOC of the initial state S_t, are generated with a uniform distribution, with $d\,\epsilon(1, .., 7)$, $m\,\epsilon(0, .., 1439)$, and $c\,\epsilon(0, .., 100)$.

The reason for uniformly generating the starting state S is to ensure the probability of exploring all states are equal. The price p is sampled from historical price that has the same day and time, and ΔP is also sampled from historical price averaged over its next 6 hours. The action a is chosen based on the ϵ-greedy policy using values predicted by the neural network.

The next state S_t is generated by firstly checking if the PEV is plugged in, this is done by generating its plug-in status based on the plug-in probability table shown in table 5.1. If the PEV is plugged in at time t then the next state's day, minute, and SOC are deterministic and can be calculated using equations (3.4) - (3.6).

If the PEV is not plugged in, then a trip duration is sampled from the trip duration model based on the departure day and time. The charge used on that trip will be sampled according to the departure day and time, and the length of the trip. The next state's day and time is when the PEV is plugged-in again after the trip, which is deterministic when the trip length is known. The state SOC is the previous state's SOC minus the charge used during the trip. The electricity price of all the states is looked up from historical price.

This process of generating state transitions is repeated to generate a small number of transitions to be used for the training of the DQN.

Chapter 6

Simulations

In this section, we discuss the experimental set-up, and evaluate the performance of the proposed approach compared to some existing approaches of single PEV charging.

6.1 Experimental Set-up

6.1.1 PEV environment

To evaluate the performance of the proposed method, we modelled the BYD e6, which has a battery capacity of 82 kWh. Battery capacities are growing ever-larger, and the BYD e6 has one of the largest battery capacities, which reflects the size of PEV batteries in the near future. The charge/discharge rate of the PEV is 19.2 kW, which is the current level 2 charging rate and most residential homes have the capability to install level 2 charging.

The minutely electricity price is taken from Energy Exchange Austria. Three months of electricity price data, from August 2018 to November 2018, is taken to find the average historical electricity price. Price taken from November 2018 to December 2018 is used as the real minutely price in the actual simulation.

Three types of PEV owner behaviour are simulated:

- Normal user behavior: the PEV owner leaves his home from 8:00 am to 9:00 am on weekdays, and uses a charge between 25% to 30% with a trip duration of 7.5 to 8.5 hours. On weekends the user leaves his home from 5:00 am to 5:00 pm and uses a charge between 15% to 50%, with a trip duration of 6 to 12 hours. The user's driving routine changes within that range from week to week.

- Stationary user behavior: the PEV owner leaves his home at 8:00 am on weekdays, and from 10:00 am on weekends. The trip duration on weekdays is 8 hours, and 4 hours on weekends. The charge used is based on the length of trip, and is between 25% to 28% on weekdays and between 25% to 28% on weekends. The owner driving behavior will remain exactly the same from week to week.

- Stochastic user behavior: the PEV owner leaves his home from 7:30 am to 9:00 am on weekdays, and from 5:00 am to 5:00 pm on weekends. The trip duration are between 7.5 hours to 9 hours on weekdays, and 3 hours to 15 hours on weekends. The charge used is based on the length of the trip, and is between 5% to 50% on weekdays, and between 3% to 40% on weekends. The driving behaviour of the PEV also changes every week, which makes it highly unpredictable.

All PEV simulation starts with an initial SOC of 100%, and the time starts on Monday at 12:00 am.

The reward received by the RL agent upon taking any charging/discharging action is the cost of electricity at that minute multiplied by the PEV's rate of charge. It is positive when discharging and negative when charging. The reward for idling, charging the PEV at full battery SOC, and discharging the PEV at 0 battery SOC is 0. Having the PEV battery running out of energy during trips will cause an inconvenience. In the best case scenario the PEV owner has to find a charging station to charge his vehicle, and in the worst case scenario the PEV owner must tow his vehicle. Therefore a reward of -28 plus the cost of charging is obtained as a reward when the PEV battery is depleted during trips. The -28 is a rough estimate on the inconvenience experienced by the owner when the PEV runs out of battery during trips, including the rare times when he needs to get his vehicle towed. If the RL agent's previous action before the trip was to charge the vehicle, then a reward of -24 plus the cost of charge used is given. This incentivizes charging when the battery SOC is near 0 and lets the RL agent learn to keep sufficient battery charge faster.

Figure 6.1 shows the configuration of the neural network used in the DQN for $Q(s, a)$ prediction. The neural network used for the DQN is configured as follows:

- Input layer has 5 nodes: time of day, day of week, current battery SOC, current electricity price, and average historical electricity price of the next 6 hours
- Output layer has 3 nodes: the state-action quality, $Q(s, a)$, of charging the PEV, remaining idle, and discharging the PEV
- Three fully connected hidden layers, first layer has 4 nodes, second layer has 3 nodes, and third layer has 2 nodes. This configuration seems to perform better than other tried configurations (50-50-50, 10-7-5, and 50-40-30)
- Rectified Linear activation function at each hidden layers, and no activation function at the output layer for regression

6.1.2 Neural Network

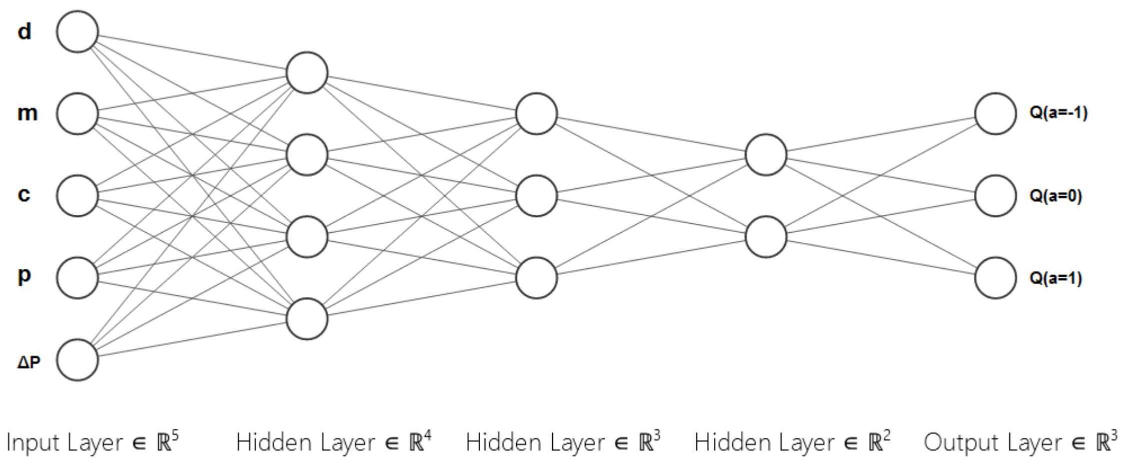

Figure 6.1: Neural Network Configuration.

6.1.3 Training Process

The RL agent is trained over 40000 decision epochs. A decision epoch occurs when a charging/discharging action needs to be taken. When the PEV is plugged in, a charging/discharging action occurs every minute.

However, when the PEV owner is on a trip, there are no actions to be taken. The next decision epoch occurs when the PEV is plugged in at home. This set-up means that the 40000 decision epochs span over a time frame of roughly 50000 real life minutes, as the additional 10000 minutes are spent on trips, where there are no decisions to be made.

At every decision epoch, the RL agent also samples from model generated experience. The model generates 32 experience per real experience.

6.2 Experimental Results

The proposed scheme is compared to five other charging schemes:

- Model-free DQN proposed in [35].
- Cheap Scheme: The PEV charges/discharges when the current electricity price is lower/higher than the historical average price for the same month and date in the previous year.
- Always Charge Scheme: The PEV always charges when it is plugged in.
- Low SOC and Cheap Scheme: The PEV will always charge when it has less than 20% battery SOC, and will charge/discharge when the electricity price is lower/higher than the historical average price otherwise.

The cumulative reward of the RL agent when the PEV owner's driving nature is regular, stationary, and stochastic is shown in Figures 6.2, 6.3, 6.4 respectively. The proposed scheme achieves the highest reward overall, and drastically outperforms a purely model-free DQN strategy. This is due to the fact that only 30 days are simulated, and this is far too little experience for a pure model-free DQN to discover a policy that ensures the PEV does not deplete its battery during trips. The biggest penalty that the model-free DQN suffers from is running out of battery during trips, as seen in Figure 6.5. This scheme has ran out of battery during trips a total of 22 times in 30 days. Due to computation and time constraints, the model-free DQN

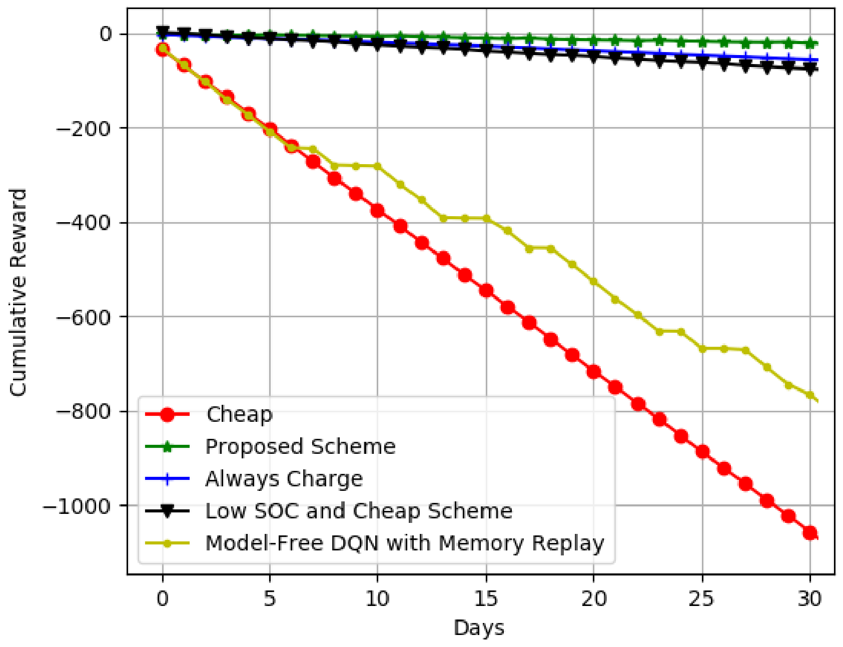

Figure 6.2: RL Agent Cumulative Reward for Normal Owner Behavior.

cannot be sufficiently simulated until convergence. Because of the nature of the always charge scheme and low SOC and cheap scheme, the PEV owner will never deplete his battery during trips. The proposed scheme performs just as well in this regard, and never have depleted its battery during trips and is shown in Figure 6.5, which shows the battery depletion during trips amount for a normal user. The above mentioned three schemes is overlapped in Figure 6.5, due to all three schemes run out of battery exactly 0 times during the 30 simulated days.

The always charge scheme was able to outperform the low SOC and cheap scheme because the historical electricity prices is not always a good predictor of future prices, and some charging decisions made by the low SOC and cheap scheme are likely not optimal.

Although it may appear that the proposed scheme only outperforms the always charge scheme by a little, in actuality, if we observe Figure 6.6, which shows the cumulative reward for a stochastic user, the

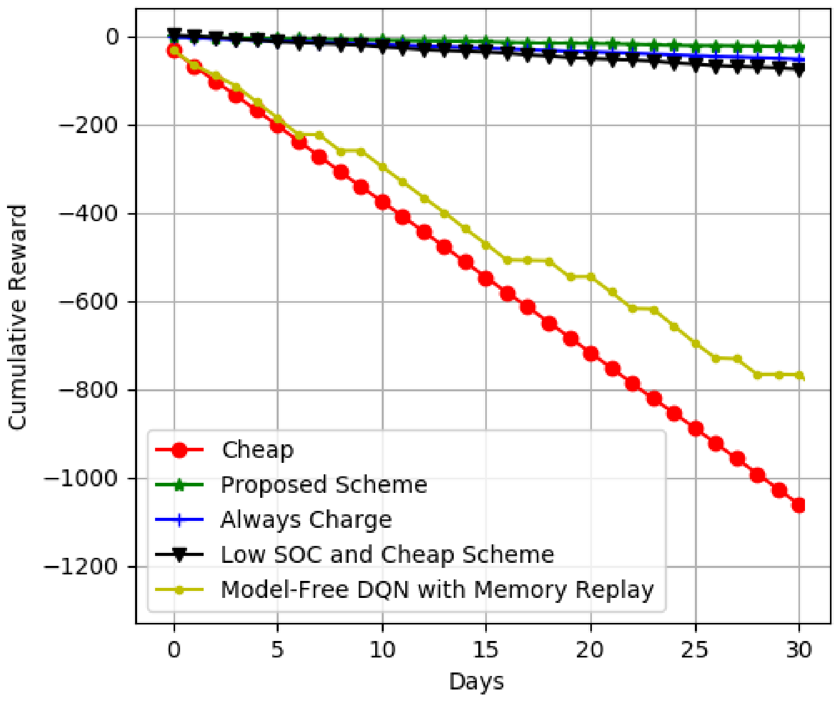

Figure 6.3: RL Agent Cumulative Reward for Stationary Owner Behavior.

cumulative reward for the proposed scheme was -22, while the always charge scheme has a cumulative reward of -43. Therefore the proposed scheme actually reduces the cost of charging by 51% when compared with the always charge scheme.

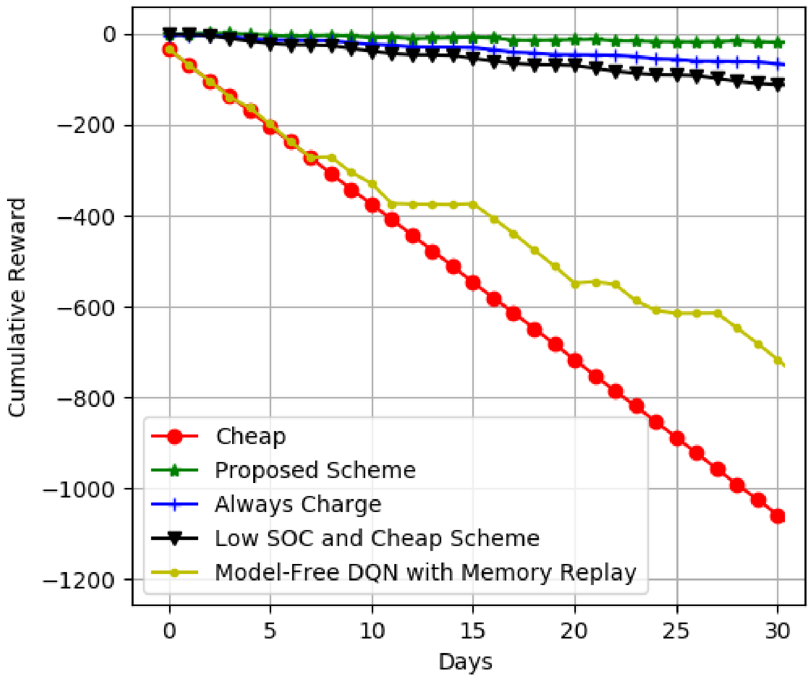

Figure 6.4: RL Agent Cumulative Reward for Stochastic Owner Behavior.

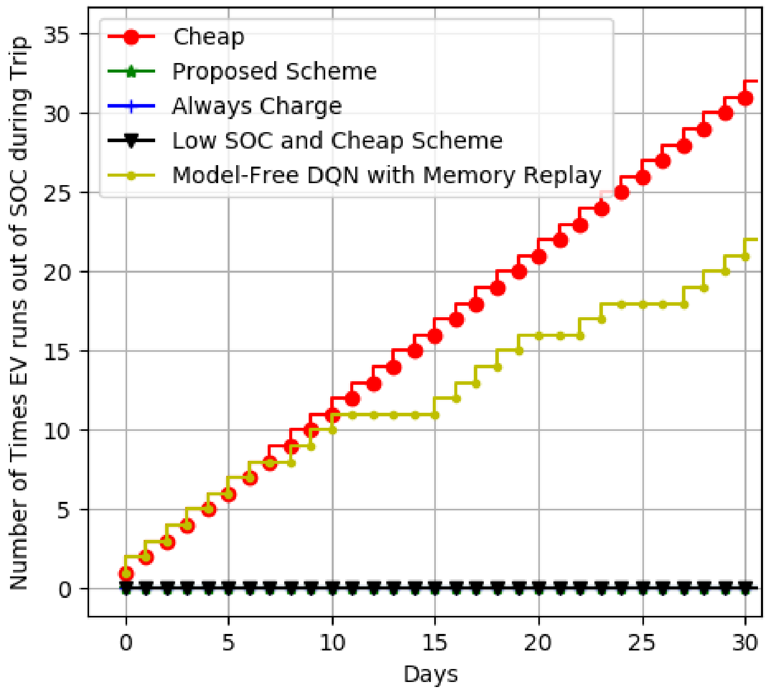

Figure 6.5: Number of Times PEV runs out of Battery During Trips

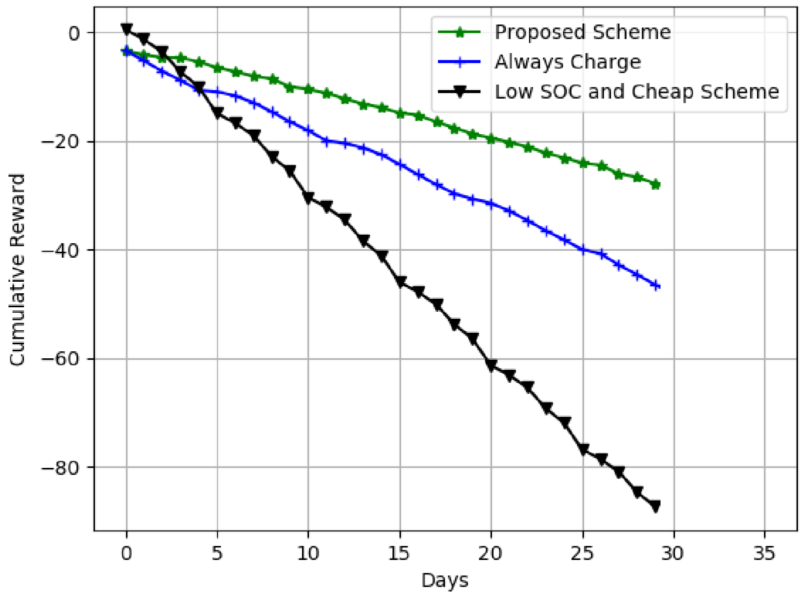

Figure 6.6: Comparison of Three Schemes

Chapter 7

Conclusions

A method that combines both model-free and model-based reinforcement learning for PEV charging was proposed in this thesis. The proposed method implicitly takes into consideration the user's driving behavior, traffic conditions, electricity price, and PEV energy usage. This method tries to ensure the owner has sufficient charge for trips, all the while charging the PEV cheaply according to the user's needs. The effectiveness of this method is demonstrated in simulations, where it is shown to run out of battery zero times during trips, and costs less than other methods.

Simulation results have shown that the model-free DQN will run out of SOC during trips quite often, and is impractical for real life charging scenarios. Dynamic programming approaches assumes the probability transition matrix and is not ideal for charging scenarios in which the PEV owner's behaviour is unknown, which is often the case in real life. A realistic scenario for owner's manual charging is shown to be unreliable, and will often times not even outperform always charge schemes. Charging the PEV manually also creates extra work for the PEV owner which is unwanted.

Future works include the use of multi-agent reinforcement learning, which can reduce costs for an entire household, community, or neighborhood. The current reward function can also be modified to include demand response strategies, which can alleviate peak demand while still giving PEV owner incentives.

By combining multi-agent reinforcement learning and modifying the reward function, we will be able to realistically utilize an entire community's PEV batteries as demand response storage, while giving an incentive for PEV owners to participate in these strategies. This should offer advantage over most research in this field, as it is a much more realistic and adaptable strategy due to learning from real life experiences.

Bibliography

[1] P. H. Andersen, J. A. Mathews, and M. Rask, "Integrating private transport into renewable energy policy: The strategy of creating intelligent recharging grids for electric vehicles," *Energy policy*, vol. 37, no. 7, pp. 2481–2486, 2009.

[2] R. Nealer, *Cleaner cars from cradle to grave: How electric cars beat gasoline cars on lifetime global warming emissions*. Union of Concerned Scientists., 2015.

[3] A. Emadi, Y. J. Lee, and K. Rajashekara, "Power electronics and motor drives in electric, hybrid electric, and plug-in hybrid electric vehicles," *IEEE Transactions on industrial electronics*, vol. 55, no. 6, pp. 2237–2245, 2008.

[4] *Plug-In Electric Vehicles: What Role for Washington?* Brookings Institution Press, 2009. [Online]. Available: http://www.jstor.org/stable/10.7864/j.ctt1262t0

[5] T. J. Geiles and S. Islam, "Impact of pev charging and rooftop pv penetration on distribution transformer life," in *2013 IEEE Power & Energy Society General Meeting*. IEEE, 2013, pp. 1–5.

[6] B. K. Sovacool and R. F. Hirsh, "Beyond batteries: An examination of the benefits and barriers to plug-in hybrid electric vehicles (phevs) and a vehicle-to-grid (v2g) transition," *Energy Policy*, vol. 37, no. 3, pp. 1095–1103, 2009.

[7] L. Situ, "Electric vehicle development: the past, present & future," in *2009 3rd International Conference on Power Electronics Systems and Applications (PESA)*. IEEE, 2009, pp. 1–3.

[8] Z. Li, M. Chowdhury, P. Bhavsar, and Y. He, "Optimizing the performance of vehicle-to-grid (v2g) enabled battery electric vehicles through a smart charge scheduling model," *International Journal of Automotive Technology*, vol. 16, no. 5, pp. 827–837, 2015.

[9] R. A. Howard, "Dynamic programming and markov processes." 1960.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[11] M. H. Amini, M. P. Moghaddam, and E. H. Forushani, "Forecasting the PEV owner reaction to the electricity price based on the customer acceptance index," in *2013 Smart Grid Conference (SGC)*, Dec 2013, pp. 264–267.

[12] L. Cai, J. Pan, L. Zhao, and X. Shen, "Networked electric vehicles for green intelligent transportation," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 77–83, 2017.

[13] G. Tal and M. A. Nicholas, "Studying the PEV market in california: Comparing the PEV, PHEV and hybrid markets," in *2013 World Electric Vehicle Symposium and Exhibition (EVS27)*, Nov 2013, pp. 1–10.

[14] S. Shafiee, M. Fotuhi-Firuzabad, and M. Rastegar, "Investigating the impacts of plug-in hybrid electric vehicles on power distribution systems," *IEEE Transactions on Smart Grid*, vol. 4, no. 3, pp. 1351–1360, Sept 2013.

[15] T. Namerikawa, N. Okubo, R. Sato, Y. Okawa, and M. Ono, "Real-time pricing mechanism for electricity market with built-in incentive for participation," *IEEE Transactions on Smart Grid*, vol. 6, no. 6, pp. 2714–2724, Nov 2015.

[16] J. R. Pillai and B. Bak-Jensen, "Integration of vehicle-to-grid in the western danish power system," *IEEE Transactions on Sustainable Energy*, vol. 2, no. 1, pp. 12–19, Jan 2011.

[17] C. L. Floch, F. di Meglio, and S. Moura, "Optimal charging of vehicle-to-grid fleets via pde aggregation techniques," in *2015 American Control Conference (ACC)*, July 2015, pp. 3285–3291.

[18] J. Mohammadi, S. Kar, and G. Hug, "Distributed cooperative charging for plug-in electric vehicles: A consensus+innovations approach," in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec 2016, pp. 896–900.

[19] J. Mohammadi, M. G. Vayá, S. Kar, and G. Hug, "A fully distributed approach for plug-in electric vehicle charging," in *2016 Power Systems Computation Conference (PSCC)*, June 2016, pp. 1–7.

[20] H. Wu, M. Shahidehpour, A. Alabdulwahab, and A. Abusorrah, "A game theoretic approach to risk-based optimal bidding strategies for electric vehicle aggregators in electricity markets with variable wind energy resources," *IEEE Transactions on Sustainable Energy*, vol. 7, no. 1, pp. 374–385, Jan 2016.

[21] M. G. Vayá and G. Andersson, "Optimal bidding strategy of a plug-in electric vehicle aggregator in day-ahead electricity markets under uncertainty," *IEEE Transactions on Power Systems*, vol. 30, no. 5, pp. 2375–2385, Sept 2015.

[22] M. Li and L. Zhao, "A decentralized load balancing approach for neighbouring charging stations via ev fleets," in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Sep. 2017, pp. 1–5.

[23] M. A. Ortega-Vazquez, "Optimal scheduling of electric vehicle charging and vehicle-to-grid services at household level including battery degradation and price uncertainty," *IET Generation, Transmission Distribution*, vol. 8, no. 6, pp. 1007–1016, June 2014.

[24] J. Zhao, C. Wan, Z. Xu, and J. Wang, "Risk-based day-ahead scheduling of electric vehicle aggregator using information gap decision theory," *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 1609–1618, July 2017.

[25] Y. Zhao, C. Feng, Z. Lin, F. Wen, C. He, and Z. Lin, "Development of optimal bidding strategy for an electric vehicle aggregator in a real-time electricity market," in *2018 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, May 2018, pp. 288–293.

[26] D. Wu, H. Zeng, C. Lu, and B. Boulet, "Two-stage energy management for office buildings with workplace ev charging and renewable energy," *IEEE Transactions on Transportation Electrification*, vol. 3, no. 1, pp. 225–237, March 2017.

[27] L. Yao, W. H. Lim, and T. S. Tsai, "A real-time charging scheme for demand response in electric vehicle parking station," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 52–62, Jan 2017.

[28] M. Li, J. Gao, L. Zhao, and X. Shen, "Task time allocation and reward scheme for pev charging station advertising," in *IEEE ICC'19*, May 2019.

[29] B. Zheng, P. He, L. Zhao, and H. Li, "A hybrid machine learning model for range estimation of electric vehicles," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[30] S. Morsalin, K. Mahmud, and G. Town, "Electric vehicle charge scheduling using an artificial neural network," in *2016 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)*, Nov 2016, pp. 276–280.

[31] A. Chiş, J. Lundén, and V. Koivunen, "Reinforcement learning-based plug-in electric vehicle charging with forecasted price," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 3674–3684, May 2017.

[32] J. Donadee, M. Ilic, and O. Karabasoglu, "Optimal autonomous charging of electric vehicles with stochastic driver behavior," in *2014 IEEE Vehicle Power and Propulsion Conference (VPPC)*, Oct 2014, pp. 1–6.

[33] Z. Wen, D. O'Neill, and H. Maei, "Optimal demand response using device-based reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 6, no. 5, pp. 2312–2324, Sept 2015.

[34] S. Vandael, B. Claessens, D. Ernst, T. Holvoet, and G. Deconinck, "Reinforcement learning of heuristic ev fleet charging in a day-ahead electricity market," *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1795–1805, July 2015.

[35] Z. Wan, H. L. H. He, and D. Prokhorov, "Model-free real-time EV charging scheduling based on deep reinforcement learning," *IEEE Transactions on Smart Grid*, pp. 1–1, 2018.

[36] D. Michie, D. J. Spiegelhalter, C. Taylor *et al.*, "Machine learning," *Neural and Statistical Classification*, vol. 13, 1994.

[37] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[38] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE signal processing magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[39] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in neural information processing systems*, 2018, pp. 8778–8788.

[40] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[41] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[42] H. H. Yang and S.-i. Amari, "Complexity issues in natural gradient descent method for training multilayer perceptrons," *Neural Computation*, vol. 10, no. 8, pp. 2137–2157, 1998.

[43] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[44] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, p. 12, 1991.

[45] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 661–670.

[46] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[47] F. Zou and L. Shen, "On the convergence of adagrad with momentum for training deep neural networks," *arXiv preprint arXiv:1808.03408*, 2018.

[48] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 161–168.

[49] J. M. Zurada, *Introduction to artificial neural systems*. West publishing company St. Paul, 1992, vol. 8.

[50] M. R. Zadeh, S. Amin, D. Khalili, and V. P. Singh, "Daily outflow prediction by multi layer perceptron with logistic sigmoid and tangent sigmoid activation functions," *Water resources management*, vol. 24, no. 11, pp. 2673–2688, 2010.

[51] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[52] R. A. Dunne and N. A. Campbell, "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function," in *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, vol. 181. Citeseer, 1997, p. 185.

[53] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with relu activation," in *Advances in Neural Information Processing Systems*, 2017, pp. 597–607.

[54] S. D. Tran and R. Manmatha, "Activation layers for deep learning networks," Feb. 13 2018, uS Patent 9,892,344.

[55] M. N. Moghadasi, A. T. Haghighat, and S. S. Ghidary, "Evaluating markov decision process as a model for decision making under uncertainty environment," in *2007 International Conference on Machine Learning and Cybernetics*, vol. 5, Aug 2007, pp. 2446–2450.

[56] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[57] A. R. Cassandra, "A survey of pomdp applications," in *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, vol. 1724, 1998.

[58] A. Reibman, R. Smith, and K. Trivedi, "Markov and markov reward model transient analysis: An overview of numerical approaches," *European Journal of Operational Research*, vol. 40, no. 2, pp. 257–267, 1989.

[59] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[60] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[61] P. Cichosz, "Truncating temporal differences: On the efficient implementation of td (lambda) for reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 2, pp. 287–318, 1994.

[62] G. Tesauro and G. R. Galperin, "On-line policy improvement using monte-carlo search," in *Advances in Neural Information Processing Systems*, 1997, pp. 1068–1074.

[63] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine learning*, vol. 38, no. 3, pp. 287–308, 2000.

[64] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, “Backward q-learning: The combination of sarsa algorithm and q-learning,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184–2193, 2013.

[65] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[66] M. Riedmiller, “Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method,” in *European Conference on Machine Learning*. Springer, 2005, pp. 317–328.

[67] K. Doya, K. Samejima, K.-i. Katagiri, and M. Kawato, “Multiple model-based reinforcement learning,” *Neural computation*, vol. 14, no. 6, pp. 1347–1369, 2002.

[68] J. Liang and J. Xu, “A novel contour extraction approach based on q-learning,” in *2006 International Conference on Machine Learning and Cybernetics*, Aug 2006, pp. 3807–3810.

[69] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, 2016.

[70] A. Masadeh, Z. Wang, and A. E. Kamal, “Reinforcement learning exploration algorithms for energy harvesting communications systems,” in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[71] T. P. Lillicrap, J. J. Hunt, A. e. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv e-prints*, p. arXiv:1509.02971, Sep 2015.

[72] B. P. Welford, “Note on a method for calculating corrected sums of squares and products,” *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962. [Online]. Available: http://www.jstor.org/stable/1266577