AUTOMATING VOLTAGE CONTROL OF THE ELECTRIC TRANSMISSION GRID WITH POWERWORLD AND DEEP REINFORCEMENT LEARNING

A Thesis

by

BRANDON L. THAYER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Thomas Overbye
Co-Chair of Committee,	Katherine Davis
Committee Members,	Jean-Francois Chamberland-Tremblay
	Timothy Davis
Head of Department,	Miroslav Begovic

May 2020

Major Subject: Electrical Engineering

ABSTRACT

This thesis applies a subset of machine learning known as deep reinforcement learning (DRL) to the problem of steady state voltage control in the electric power transmission system. In support of this work, both a new Python package that interfaces with PowerWorld Simulator and a set of modular DRL environments were developed. A state-of-the-art open-source DRL algorithm which leverages "deep Q networks" (DQN) was used along with the developed DRL environments in order to apply DRL to voltage control.

An experiment from a recent publication which applies DRL to voltage control was reproduced and its shortcomings were discussed. This led to experimentation with algorithm and environment modifications. It was found that a novel change to the DQN algorithm in which agents are not allowed to take the same action twice in any given training or testing episode leads to significant improvements. Additionally, it was found that using min-max scaled voltages in observations provided to the DRL agent rather than per unit voltages leads to marked improvements in successfully solving the voltage control problem.

After initial exploration using the IEEE 14 bus test system, use of DRL for voltage control was tested on synthetic 200 and 500 bus systems developed at Texas A&M University. Results for these systems were mixed, and instabilities were observed during training. For the 200 bus system with single line contingencies present in all training and testing episodes, DRL agents were able to achieve success near to that of a heuristically-driven graph-based agent that was developed for comparison. By contrast, with the 500 bus system the DRL agents' success rates were approximately half that of the graph-based agents.

This thesis demonstrates that there is promise in the application of deep reinforcement learning to power system voltage control, but more research is needed in order to enable DRL-based techniques to consistently outperform conventional methods.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professors Thomas Overbye, Katherine Davis, and Jean-Francois Chamberland-Tremblay of the Department of Electrical and Computer Engineering, as well as Professor Timothy Davis of the Department of Computer Science and Engineering.

The "ESA" package introduced in Chapter 2 was developed with assistance from Zeyu Mao and Yijing Liu who are both graduate students in the Department of Electrical and Computer Engineering working underneath Professor Thomas Overbye.

All other work conducted for the thesis was completed by the student (Brandon Thayer) independently.

Funding Sources

This research was supported by funding from the Texas A&M Engineering Experiment Station (TEES) Smart Grid Center (SGC).

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Current Trends in the Electric Power System and Industry

The electric power grid is critical to the functioning of our modern society, and is undergoing a period of major change. Large portions of the U.S. electric power system have undergone deregulation since the 1990's, distributing the responsibilities of electric power generation, transmission, and distribution to separate entities and opening up power markets [1]. Additionally, electricity generation from intermittent renewable resources such as wind and solar is rising while traditional generation sources such as coal and nuclear are declining [2]. Due to numerous factors including increasing extreme weather events, the reliability of the electric grid has been declining in recent years [3]. Further complicating these challenges are current and upcoming labor shortages in the electric power industry [4, 5].

Fortunately, the digital computing revolution of the last several decades has been a boon to the electric power system. Phasor measurement units (PMUs) have been increasing visibility into the electric power system [6], "smart" meter installations are on the rise [7], and energy management systems (EMS) are continually growing in sophistication. Despite these recent advances, many simple grid control actions are still taken by humans [8–11]. As electric grid operation becomes increasingly complex due to increased data flow, changing resource mix, decreasing system inertia, more control options, etc., the automation of grid control is more important than ever.

1.2 Electric Grid Reactive Power and Voltage Control

In order to ensure secure electric grid operation, all buses in the transmission system are kept within a prescribed voltage band. Bus voltages are maintained by a variety of devices through a mixture of human and automatic control. The most prevalent voltage control devices in the transmission system are generators, switched shunts (e.g. capacitors), and on-load tap changing (OLTC) transformers [12].

Electric generators are required to follow a pre-determined voltage schedule created by the

transmission operator or independent system operator, and control of the high side voltage of their step-up transformers is achieved by adjusting reactive power output. Capacitors and voltage regulators may operate automatically based on local measurements, be directly controlled by human operators, and/or be controlled by a centralized optimization program [8, 12–15].

Recent advances in system-wide transmission voltage control leverage hierarchical control schemes and the use of "pilot" (bellwether) buses in different pre-determined voltage control areas. These schemes have been proven effective in Europe, China, and the United States [14–16]. While effective, the aforementioned voltage control schemes must make major modeling and control simplifications due to their reliance on conventional optimization techniques and the large scale of the electric transmission system. Additionally, the optimization can be time consuming, which can make these techniques impractical when the need for rapid control decisions arises.

1.3 Reinforcement Learning

Reinforcement learning is a form of learning in which an agent receives rewards or penalties for performing actions which affect its environment. Based on these rewards or penalties, an agent can potentially learn how to maximize its rewards [17].

In a popular model-free reinforcement learning algorithm, Q-learning, agents attempt to learn the action-utility function (Q-function) which gives the expected utility of performing a particular action (Q-value) given the state of the environment. By learning the Q-function, the agent can then make decisions so as to optimize its cumulative future rewards. The learning of the Q-function often takes the form of a table indexed by state and action. Each table entry represents the value of an action given the state of the environment [17].

Unfortunately, traditional Q-learning algorithms don't scale well due to their use of a tabular Q-function representation. Recent advances use neural networks as an estimator of the Q-function: states are passed to the input layer of the neural network and an estimate of the expected utility of each possible action is emitted from the network's output layer. Reinforcement learning algorithms which use neural networks as Q-function estimators are collectively known as "deep reinforcement learning" (DRL) algorithms and have been proven successful in domains with large state and action

spaces such as Atari games and the board game Go [18–20].

1.4 Reinforcement Learning for Power System Control

Historically, the use of reinforcement learning in the power system domain has been stymied by the large scale of states and control in the power system [21]. As reinforcement learning techniques have grown more successful and proven capable of operating in environments with larger state and action spaces, interest in reinforcement learning's potential for power system control has increased. Reference [22] (published in 2017) provides a survey of the literature involving reinforcement learning for control of the electric power system. Researchers have begun investigating reinforcement learning for a variety of power system control problems including transient generator angle stability, congestion management, economic dispatch, and voltage control [22]. The authors of [22] suggest that the recent success of DRL may warrant a revisiting of previous grid control work where reinforcement learning was applied before recent breakthroughs.

One example of successfully revisiting grid control problems with new DRL algorithms is [23]. The authors leveraged the open-source reinforcement learning environment known as Gym [24] as well as recently published open-source DRL algorithms [25], both created by OpenAI, in order to perform grid emergency control. OpenAI's Gym and DRL algorithms are discussed in more detail later on. The authors of [23] investigated two grid control problems: dynamic generator braking and under-voltage load shedding. It was shown that reinforcement learning agents could be successfully trained both to apply a resistive generator brake in order to prevent the loss of generator synchronism and to shed the minimal amount of load required while maintaining a particular voltage recovery envelope. While the work presented in [23] is quite promising, the authors performed their studies using small test systems (10 and 14 buses).

In [26], traditional, tabular Q-learning was applied to reactive power/voltage control. The authors simplified system states by using a binary representation: if a component is within its operating limits (generator reactive power limits, transformer power flows, and bus voltages), the corresponding component of the state vector is 0. If a component is outside its operating limits, the corresponding state vector component is -1. The discrete action space consisted of transformer

tap positions, shunt switch positions, and generator voltage set points. The authors show that Q-learning can be used to successfully determine control settings to reduce violations in 14 and 136 bus test cases. Unfortunately, the authors of [26] did not specify the tools they utilized for either reinforcement learning or solving the power flow problem.

Similarly to [26], [27] presents the application of reinforcement learning to reactive power/voltage control. However, the authors of [27] use newer DRL algorithms instead of tabular Q-learning. The states considered are the results from solving the power flow problem, e.g. bus voltages, bus angles, active power flows, and reactive power flows. The only control actions considered are discretized generator voltage set points. The authors used GridPACK™ as a power flow solver [28], Gym as a reinforcement learning environment [24], and wrote their own DRL algorithm. A simple reward structure was used in which a positive reward is given for each bus within the acceptable voltage bound, and negative rewards are given for each bus outside of the acceptable voltage bound. The authors show that their DRL agent does in fact learn how to correct voltage issues while minimizing the number of control actions taken.

1.5 Overview of Thesis

This thesis applies deep reinforcement learning to the power system voltage control problem. Chapter 2 presents the software developed and used in order to solve the voltage control problem. Contributions include the development of a Python package which interfaces with PowerWorld [29,30] and the development of a new Gym environment which uses PowerWorld as the underlying power systems simulator [24, 31]. In Chapter 3, a recent experiment from the literature ([27]) is reproduced and its shortcomings are discussed in detail. The primary contribution from Chapter 3 is a novel modification to a deep reinforcement learning algorithm that does not allow the agent to take the same action multiple times within each training and testing episode. Chapter 4 leverages the knowledge gained in Chapter 3 to explore several improvements to the environment. It was found that min-max scaling of voltage observations along with the aforementioned algorithmic improvement lead to good results when performing voltage control. Additionally, two novel reward schemes based on voltage movement are presented. Chapter 5 takes the findings of Chapter 4 and

applies them to larger power systems. Specifically, 200 and 500 bus systems developed at Texas A&M University were used [32, 33]. Results with the larger systems were mixed, but there's clear potential for the application of DRL for voltage control on realistically sized electric grids. Finally, Chapter 6 presents conclusions and future work.

2. SOFTWARE DEVELOPED AND USED FOR DEEP REINFORCEMENT LEARNING

This chapter presents both the software used to perform the research contained within the thesis, as well as the software that was developed in order to perform deep reinforcement learning for steady-state voltage control. Broadly, the software that was developed is broken up into three components: development of an open-source Python package for interfacing with PowerWorld Simulator [29], development of an open-source reinforcement learning environment leveraging Gym [24] and PowerWorld for steady state voltage/reactive power control, and finally using deep reinforcement learning (DRL) algorithms to perform grid voltage/reactive power control. Section 2.1 introduces the pre-existing tools which were leveraged in this research, Section 2.2 discusses the Python/PowerWorld connector, ESA, that has been developed in support of this research, Section 2.3 covers the development of a reinforcement learning environment for voltage control, and Section 2.4 discusses the reinforcement learning experiments I propose to carry out.

2.1 Software Tools

In Chapter 1, power system trends, power system voltage control, reinforcement learning, and reinforcement learning for power system control were all discussed. For the most part, that discussion was agnostic of tools. In this section, background related to the tools used to perform the research proposed here is discussed.

2.1.1 Python

Python is a high=level, open-source, cross-platform, interpreted language that has been rapidly gaining popularity in recent years [34,35]. Python has many freely available and powerful scientific computing packages, such as Numpy and Pandas, which are both part of the SciPy ecosystem [36]. Additionally, many machine learning and artificial intelligence related packages have been released in recent years, including Tensorflow and Keras [37,38]. The combination of ease of programming, available packages, and open-source nature make Python a very attractive tool to use for machine learning/artificial intelligence work.

2.1.2 PowerWorld

PowerWorld Simulator is a power system simulation package with a wide variety of capabilities ranging from power flow analysis to dynamic simulation and beyond [29]. PowerWorld was founded in 1996 by Professor Thomas Overbye, and is under continuous development by PowerWorld Corporation. Note that in this thesis, "PowerWorld Simulator" will often be shortened to just "PowerWorld."

PowerWorld is commercial grade software: it has been heavily optimized for performance, provides a suite excellent power system visualization tools, and has an application programming interface (API) allowing for the use of external tools to configure and control simulations. As is discussed in detail later, training a reinforcement learning agent to perform voltage control involves repeatedly solving the power flow, which is a computationally expensive problem to solve. Fortunately, PowerWorld's sophisticated algorithms and utilization of sparse matrix and vector methods [39, 40] make for rapid power flow solutions.

2.1.3 OpenAI's Gym and Baselines

OpenAI has released both a reinforcement learning platform, Gym [24], as well as a set of recent reinforcement learning algorithm implementations, known as Baselines [25]. Both Gym and Baselines are extendable and open-source, allowing practitioners and researchers to take full advantage of the available tools.

Gym seeks to provide a standardized platform for the development of reinforcement learning environments. In this way, different reinforcement learning algorithms can be tested against a consistent environment. Additionally, using open-source standardized environments aids in making published works more reproducible.

Baselines is a set of high-quality implementations of reinforcement learning algorithms. This allows researchers to apply an established algorithm to a problem in their research domain, e.g. voltage control of the electric transmission system. Of particular interest for the work presented in this thesis is the so called "deep-Q" learning algorithm, introduced in [18, 19]. The deep-Q

algorithm uses a neural network as a Q-function estimator, and is used for problems with discrete control actions. Baselines also contains algorithms for continuous actions spaces, which could be used for a continuous formulation of the voltage control problem.

The work from Baselines has been improved upon by others, and re-released under the name "Stable Baselines" [41]. The Stable Baselines repository is a fork of the original Baselines repository with improved testing coverage, a complete documentation site, and refactored code base. All work presented in this thesis uses the algorithms provided by Stable Baselines rather than Baselines, though it's important to note that Stable Baselines would not exist were it not for Baselines.

2.2 PowerWorld/Python Connector - Easy SimAuto (ESA)

A major portion of the initial work done to enable the deep reinforcement learning research presented in this thesis was the development of Easy SimAuto (ESA) [30]. ESA provides a an easy to use Python class for interfacing with PowerWorld.

PowerWorld contains an API to allow external tools to configure and run PowerWorld simulations. This API is known as the Simulator Automation Server (SimAuto). SimAuto provides a wide range of functions for performing tasks in PowerWorld, such as fetching and changing power system component properties (e.g. generator voltage set points or shunt states), solving the power flow, and more. SimAuto facilitates the connection between PowerWorld and external software via Microsoft Component Object Model (COM) objects.

The use of SimAuto with Python can become quite cumbersome very quickly, as a lot of "boiler plate" code is required to perform simple tasks. Additionally, the required data types that are passed into PowerWorld are not commonly used in Python programming, and data returned by PowerWorld come back as strings or lists of strings. Thus, in order to automate the use of PowerWorld with Python, the programmer must write a significant amount of code just to get data converted, which is an error-prone process. Clearly, there's a need for re-usable code for managing interactions between PowerWorld and Python.

As a preliminary step in conducting the reinforcement learning research proposed here, the Easy SimAuto (ESA) Python package has been developed with assistance from my fellow grad-

uate students (Zeyu Mao and Yijing Liu) here at Texas A&M University [30]. ESA dramatically simplifies controlling PowerWorld from Python programs, and has the following desirable properties:

- Open-source and installable via Python's package manager, Pip
- Uses common scientific computing data types such as Numpy arrays and Pandas DataFrames
- Well tested - all methods have unit tests to ensure they are functioning correctly
- Easy to use - boiler plate code and type conversions occur automatically behind the scenes
- Fully documented API and code including type hinting to ease development
- Object oriented and easily extendable
- And much more...

The creation of ESA lays important ground work for the research proposed here, and is the first step in creating a Gym environment for reinforcement learning with PowerWorld. In addition to its usefulness for the specific research discussed in this document, ESA is a very practical tool which we hope will be adopted by the wider PowerWorld community which has interest in automating PowerWorld tasks.

The following two code samples illustrate the benefit of using ESA. The first sample is directly from PowerWorld [29], and the second sample accomplishes the same tasks by using ESA. More examples illustrating the usefulness of ESA can be found at [30].

```
"""PowerWorld example from powerworld.com, with slight modifications."""
# The following example uses Python 3.x syntax
# Python with COM requires the pyWin32 extensions
import win32com.client
from win32com.client import VARIANT
# This will import VT_VARIANT
import pythoncom
```

```
8
9  # This will establish the connection
10 object = win32com.client.Dispatch("pwrworld.SimulatorAuto")
11
12 # The following function will determine if any errors are returned and
13 # print an appropriate message.
14 def CheckResultForError(SimAutoOutput, Message):
15     if SimAutoOutput[0] != '':
16         print('Error: ' + SimAutoOutput[0])
17     else:
18         print(Message)
19
20 CheckResultForError(object.OpenCase(r"C:\TestCode\B7FLAT.pwb"), 'Case open')
21
22 # VARIANT is needed if passing in array of arrays. BOTH the field list
23 # and the value list must use this syntax. If not passing in arrays of
24 # arrays, the standard list format can be used. Passing out arrays of
25 # arrays from SimAuto in the output parameter seems to work OK with
26 # Python.
27 FieldArray = VARIANT(pythoncom.VT_VARIANT | pythoncom.VT_ARRAY,
28                      ["BusNum", "GenID", "GenMW", "GenAGCAble"])
29 AllValueArray = [None] * 2
30 AllValueArray[0] = VARIANT(pythoncom.VT_VARIANT | pythoncom.VT_ARRAY,
31                            [1, "1", 300, "NO"])
32 AllValueArray[1] = VARIANT(pythoncom.VT_VARIANT | pythoncom.VT_ARRAY,
33                            [2, "1", 1400, "NO"])
34 CheckResultForError(
35     object.ChangeParametersMultipleElement("GEN", FieldArray, AllValueArray)
    ,
36     'Do change')
37
38 CheckResultForError(object.SaveCase(r"C:\TestCode\B7FLAT_changed.pwb",
39                                      "PWB", True), 'Save case')
```

```
40
41 # This will close the connection
42 del object
43 object = None
```

The following example uses ESA to siginificantly simplify the script.

```
1  """Example of using ESA."""
2  # Import SAW class.
3  from esa import SAW
4
5  # Instantiate SAW object.
6  saw = SAW(FileName=r"C:\TestCode\B7FLAT.pwb")
7
8  # Change generator parameters.
9  saw.ChangeParametersMultipleElement(
10     ObjectType='gen', ParamList=['BusNum', 'GenID', 'GenMW', 'GenAGCAble'],
11     ValueList=[[1, "1", 300, "NO"], [2, "1", 1400, "NO"]])
12
13 # Save case.
14 saw.SaveCase(r"C:\TestCode\B7FLAT_changed.pwb")
15
16 # Clean up.
17 saw.exit()
```

2.3 PowerWorld+ESA Gym Environment

2.3.1 Overview

As discussed in section 2.1.3, Gym is an open-source Python package for the creation of standardized reinforcement learning environments [24]. In order to leverage the recent breakthrough algorithms provided in Stable Baselines [41] and to enable future research, Gym environments for voltage control with PowerWorld are necessary. As a part of the research presented in this thesis, several Gym environments have been created and can be found in a repository known as

Gym-PowerWorld [31].

The creation of a Gym environment requires the following class methods to be programmed:

- **__init__**: Initialize environment based on user inputs, initialize action space, initialize observation space
- **reset**: Reset environment for next learning episode, return an initial observation
- **step**: Accept action as input, perform action, return new observation
- **close**: Shut down environment

In the context of using PowerWorld for simulation of a power system, each of the aforementioned methods above perform the following tasks:

- **__init__**: Initialize an ESA class instance given a particular PowerWorld case; randomly initialize a set of training episodes with different generator set points (e.g. active power output, voltage set point), loading conditions, line states (open/closed), etc.; initialize observation space based on case data (e.g. number of buses, generators, lines, etc.); and initialize the action space based on the available voltage control devices in the given case.
- **reset**: Change generator set points and loading conditions for a new episode, solve the power flow for the given conditions, provide an initial observation containing power system data as configured for the particular environment instance (e.g. bus voltage magnitudes, generator states, line states, etc.).
- **step**: Change voltage/var control settings according to given action from reinforcement learning agent, solve the power flow, return an observation (same as described for "reset" above), and a reward/penalty based on how the given action impacted the power system.
- **close**: Properly close ESA and SimAuto instances.

The following subsections will elaborate on some of the more important tasks listed above.

2.3.2 Environment Initialization

In order for the Gym-PowerWorld environment to be useful for a wide range of power system models, it must be capable of properly handling a wide range of possible PowerWorld case configurations. For example, during episode initialization a generator's active power output may be randomly drawn between its minimum and maximum active power output according to the PowerWorld case. However, this is problematic if, for instance, the minimum power output is a negative value (as is the case in the IEEE 14 bus case from [33]). Hence, Gym-PowerWorld must perform several checks during initialization including generator limits, active load models, etc.

In addition to performing case checks and fixes, environment initialization must set several key parameters based on both user input and case data. Examples include minimum/maximum total system loading, minimum load power factor allowed, how generator voltage set points are discretized, etc. The initialization methods for Gym-PowerWorld classes take many different inputs which allow for environment customization. For instance, with a simple Boolean flag passed to the initializer, the environment can be changed to report voltage observations in per unit or in a transformed space.

2.3.3 Episode Initialization

The design of how episodes (scenarios) are created can have a major impact both on a reinforcement learning agent's ability to learn and on the ultimate usefulness of a trained agent. For instance, training under a narrow range of load and generator conditions may enable an agent to learn the best control actions quickly, but the agent's decision making may not generalize well to other conditions not encountered in training. The following sections describe the methodology used for determining load and generation for each episode.

2.3.3.1 System Loading

To compute system loading, first active power is determined for all individual loads for each scenario, and then reactive power is computed based on an allowable range of power factors (e.g. $[0.8, 1.0]$).

Active Power (P) - Given the total number of desired scenarios to generate (M) as well as user-provided minimum (P_{min}) and maximum (P_{max}) total system load bounds, a random number generator (RNG) will be used to randomly draw a vector $\mathbf{V}_{M\times 1}$ with all values on the interval $[P_{min}, P_{max})$. This vector represents total system loading for each episode.

After total system loading has been determined, the load must be apportioned among the individual loads. To do so, first the user provides a probability (p_{on}) that any individual load is on/active. Then, the RNG is used to draw from the uniform distribution on the interval $[0, 1)$ and fill a matrix, $\mathbf{A}_{M\times N}$, where N is the number of individual loads present in the system. Each value in $\mathbf{A}$ is compared with p_{on}, and any values greater than p_{on} indicate that particular load (indexed by column) will be off for that particular scenario (indexed by row).

At this point, total loading for each scenario is known, as well as which loads are active. To compute individual load levels the RNG is used to fill another $M \times N$ matrix, $\mathbf{B}$, with values on the interval $[0, 1)$ from the uniform distribution. Elements in $\mathbf{B}$ are set to zero if the load is off (see previous paragraph). For each row (index m), a scaling factor z_m is computed such that $z_m = \frac{v_m}{\sum_{n=1}^{N} b_{m,n}}$ where $b_{m,n}$ is an entry in $\mathbf{B}$ and v_m is the m^{th} entry in the total scenario loading vector, $\mathbf{V}$. Finally, each row in $\mathbf{B}$ is element-wise multiplied with the corresponding z_m to create the final active power loading levels for each load within each scenario.

Reactive Power (Q) - With active power levels for each scenario and load in hand, reactive power can be computed. The user provides both a minimum load power factor, pf_{min}, and the probability that a load's power factor is leading, p_{lead}. First, the RNG is used to draw power factors for each load and scenario from the uniform distribution on the interval $[pf_{min}, 1]$, allowing reactive power levels to be computed by $Q = P \cdot \tan(\arccos(pf))$. Finally, the RNG is used to draw from the uniform distribution on the interval $[0, 1)$ for each load and scenario. These random values are compared with p_{lead}, and the corresponding reactive power value is multiplied by -1 if the random value is less than p_{lead}.

Computational Considerations - The technique for drawing system loading described above is designed to leverage the vectorized computing capabilities of the Python package Numpy [36],

and is quite fast. However, the approach is memory intensive. Load conditions for each scenario could instead be computed on the fly during learning instead of up front during initialization. However, during the course of this research memory constraints have not been an issue, so the speedier memory-intensive method is leveraged.

2.3.3.2 Generators - Active Power

After computing system loading for each scenario, the generator commitment can be determined and active power levels can be dispatched to meet demand. Since generators have active power output minimum and maximum allowable values, the procedure for determining generation levels differs somewhat from the procedure for determining individual loads. The following description is functionally equivalent to what is done in the Gym-PowerWorld code, but is explained in a simplified manner. The actual code is completely vectorized for efficiency.

For each scenario, a random ordering of all generators in the case is drawn. Then, the generators are looped over in the given random order, and an active power output is drawn from the uniform distribution between the particular generator's minimum and maximum bounds. This process continues until the total active power output of the generators meets or exceeds the total loading for the given scenario, plus assumed losses of 3%. In this way, different generators may be active for each scenario, effectively building in generator contingencies and creating unique system conditions for each scenario. Assuming some losses are present ensures the slack generator must not cover all active power losses, which can be a significant amount of power depending on the size of the system and resistance of the lines.

Similarly to the technique described in Section 2.3.3.1, this technique is fast, but memory intensive. If memory becomes an issue, modifications could be made to compute each episode's generator commitment and dispatch as needed, rather than computing it all up front.

2.3.3.3 Generators - Voltage Set Point

During environment initialization, random voltage set points are drawn for each episode and each generator. These voltage set points are drawn randomly and uniformly with replacement from

the set $\{0.95, 0.975, 1.00, 1.025, 1.05\}$ (per unit). Note that this voltage set point set is configurable, but the aforementioned set is used throughout this work.

2.3.3.4 Lines and Shunts

For cases which contain shunts, initial shunt states (open/closed) are simply randomly drawn from the uniform distribution for all shunts and all scenarios. For environments which contain line contingencies, a single line is randomly selected to be opened for each episode/scenario.

2.3.4 Observation Design

Engineering the observations given to the reinforcement learning agent is critical to successful learning, and requires significant experimentation. It's important to ensure the appropriate amount of information is given to the agent - too little information and the agent may fail to learn due to a lack of observability, while too much information can significantly scale up the size of the neural network required for deep reinforcement learning and cause a failure to learn due to the increased difficulty in finding relationships between variables.

There are many different options available for configuring the observation space in Gym-PowerWorld. The following observation combinations are used throughout this thesis:

- Bus voltage magnitudes only
- Bus voltage magnitudes and generator states (on/off)
- Bus voltage magnitudes and branch states (open/closed)
- Bus voltage magnitudes, generator states, and branch states
- Bus voltage magnitudes, generator states, branch states, and shunt states (open/closed)

All of the observation combinations above can also be used with transformed bus voltage magnitudes. In these cases, voltage are transformed via "min-max" scaling. The Gym-PowerWorld environments consider a power flow to be "failed" if any single bus voltage dips below 0.7 per unit or if any single bus voltage exceeds 1.2 per unit, even if PowerWorld is able to come up with a

power flow solution. In this way, the absolute lower and upper voltage bounds are known for all scenarios/episodes. Also, these voltage limits ensure that training and testing is only carried out with realistic scenarios: in steady state it is *extremely* rare to see voltages below 0.7 per unit or above 1.2 per unit. After the raw voltage values have been min-max scaled, the new minimum voltage observation value is 0.0 (no longer in per unit), and the maximum voltage observation value is 1.0 (also no longer in per unit). Note that all experiments presented in this thesis limit voltages to the range $[0.7, 1.2]$ per unit.

2.3.5 Reward Design

Careful design of the reward given to the reinforcement learning agent can make a large impact on the quality of the final trained agent. The goal is to have the agent learn to bring all bus voltages within the allowed range (if possible) while maintaining available var reserves and minimizing the total number of control actions taken. Therefore, the agent should receive a negative reward (penalty) for each control action taken. Actions which improve out of bounds voltages should be rewarded, and future work may consider rewarding actions which increase var reserves.

In some of the recent RL for power systems literature, agents were primarily rewarded based on the post-action state of the system, rather than the *change* that the given action *induced* in the system [23, 27]. Gym-PowerWorld comes with two reward schemes based on induced voltage changes, rather than the simple post-action system state. For example, if the objective is to bring all bus voltages within the range $[0.95, 1.05]$ per unit, an action that moves a bus voltage from 0.93 to 0.945 per unit would be rewarded, while an action that moves a bus voltage from 0.96 to 0.94 per unit would be penalized. This reward scheme more closely matches the stated goal of bringing bus voltages in bounds - the agent shouldn't be rewarded if an in-bound bus voltage changes but remains in bounds.

The two reward schemes designed for this research are described in detail in Section 4.4.

2.4 Applying Reinforcement Learning to the Voltage Control Problem

In Section 2.3, the learning environment which brings together Python, Gym, and PowerWorld was discussed. Within that section, several important research areas were discussed, such as episode initialization (loads, generation, etc.), observation design, and reward design. This section will instead focus on the portion of the research involving DRL algorithm selection, test case selection, and results evaluation.

2.4.1 Algorithm and Neural Network Selection

2.4.1.1 Algorithm and Hyper Parameters

The work presented in this thesis uses an improved version of the deep Q algorithm with so-called "deep Q networks" (DQN) originally presented in [18]. Several improvements to the algorithm have been published over the years including "dueling DQN" [42], "double-Q learning" [43], and prioritized experience replay [44]. The work here uses all the aforementioned DQN algorithm improvements. As mentioned previously, the algorithm implementation is provided by Stable Baselines [41]. The DQN algorithm requires a discrete action space, so continuous control elements such as generator voltage set points must be discretized as mentioned in Section 2.3.3.3. The DQN algorithm with all improvements has many different hyper parameters which can be tuned. In this work, the majority of hyper parameters were left at their default. The default values for hyper parameters can be found at [41], and the actual hyper parameters used in this research can be found at [45] and are depicted in Table 2.1. The parameter descriptions in Table 2.1 are from [41].

Table 2.1: DQN algorithm hyper parameters

Parameter	**Description**	**Value**
γ	Discount factor	0.99
Learning rate	Learning rate for Adam optimizer	0.0005
Buffer size	Size of the replay buffer	50,000
Exploration fraction	Fraction of entire training period over which exploration rate is annealed	1.0
Exploration final ϵ	Final value of random action probability	0.01
Exploration initial ϵ	Initial value of random action probability	1.0
Train frequency	Model is updated every "train frequency" steps	1.0
Batch size	Size of a batch sampled from replay buffer for training	32
Double-Q	Whether to enable Double-Q learning	True
Learning starts	How many steps of the model to collect transitions for before learning starts	1,000
Target network update frequency	Update the target network every "target network update frequency" steps	500
Prioritized replay	If True, prioritized replay buffer will be used	True
Prioritized replay α	Determines how much prioritization is used in the replay buffer	0.6
Prioritized replay β_0	Initial value of β for replay buffer	0.4
Prioritized replay β iterations	Number of iterations over which β_0 is annealed to 1.0. If None, equals total time steps	None
Prioritized replay ϵ	ϵ to add to the TD errors when updating priorities	0.000001
Parameter noise	Whether or not to apply noise to the parameters of the policy	False
Seed	Random seed	0

The only hyper parameters which were changed from their default are the "exploration fraction" and the "exploration final ϵ." In the DQN algorithm, over the course of training the "exploration ϵ" changes, typically so that the agent explores (takes a random action) more early in training, and exploits (takes an action according to the learned Q-function representation) more later in training. The exploration fraction refers to the percentage of training in which the exploration ϵ changes. In this work, the exploration fraction is 1.0, meaning the exploration ϵ is annealed over the entire duration of training. The exploration final ϵ used in this work is 0.01, meaning that by the end of training the agent takes a random (exploratory) action 1% of the time.

2.4.1.2 Neural Networks

The neural networks used in this work are fully connected multi-layer perceptrons (MLP). The rectifier linear unit (ReLU) activation function is used for all neurons in the networks. All networks in this work have an input layer, two hidden layers, and an output layer. The input layer is the same dimension as the observation space, and the output layer is the same dimension as the action space. Throughout this work, the networks will be referenced by the number of neurons in their hidden layers. For simplicity, the number of neurons provided here will be the number which is passed to the DQN algorithm. However, it's important to note that the use of the "dueling DQN" improvement actually doubles the number of neurons in the hidden layers [41, 42]. So, if a network is denoted here as having hidden layers with $[64, 64]$ neurons, in reality, the hidden layers have $[128, 128]$ neurons.

Future work may use a convolutional neural network (CNN) with observations mapped into grid-like structures representing the geographically located nature of the electric grid.

2.4.2 Power System Cases

Despite recent advances in deep reinforcement learning's ability to handle large state and action spaces, recent research in the literature still presents results for small test systems which are significantly smaller than the real power system [23,27]. It is still valuable to use small test systems as a stepping stone, but ultimately the scalability of these algorithms for power system control is

of major interest.

Chapters 3 and 4 use the IEEE 14 bus system. A oneline diagram depicting the system is shown in Figure 2.1.

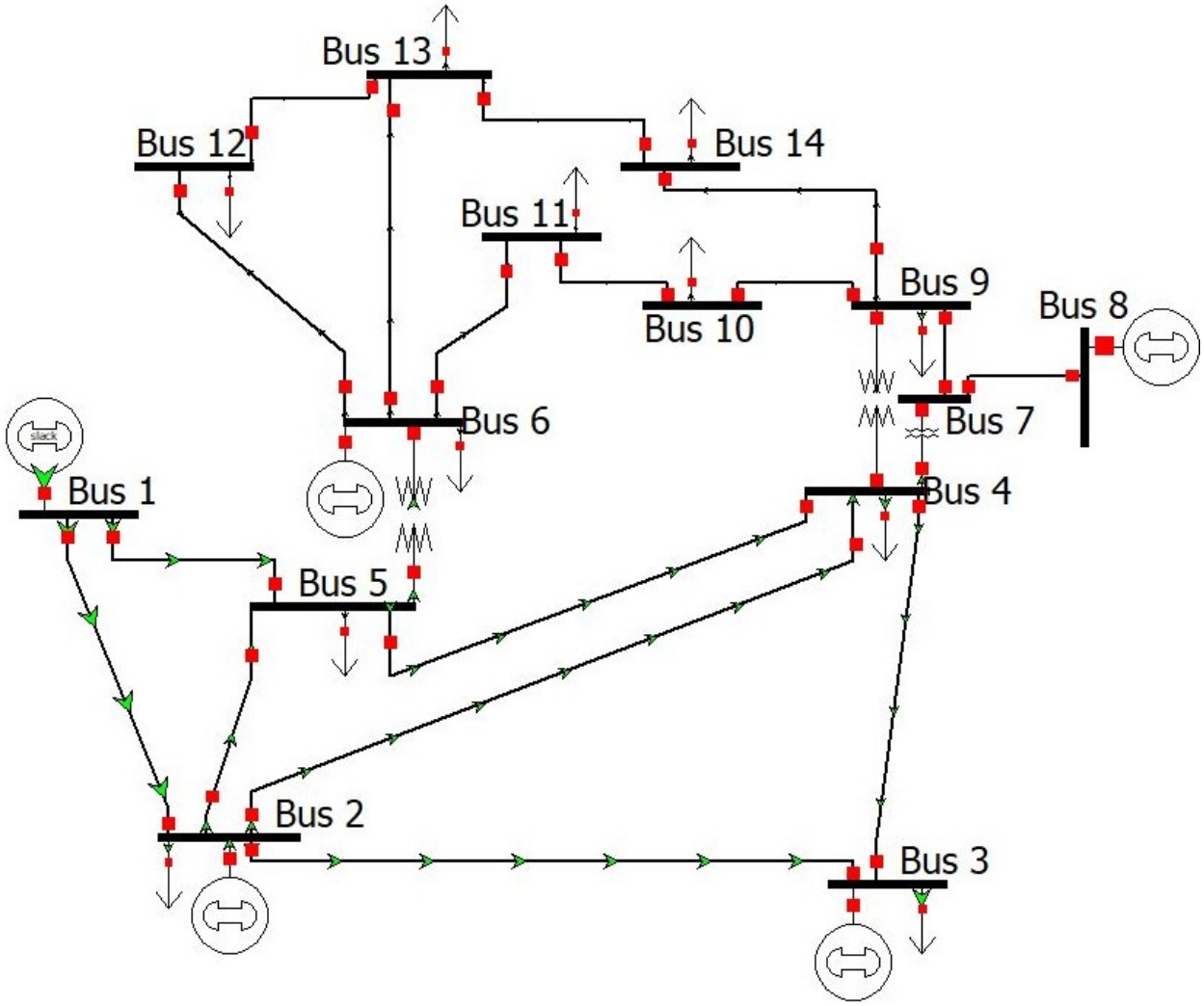


Figure 2.1: IEEE 14 bus test system

Chapter 5 tests the scalability of the DQN algorithm for power system control, and uses test cases with 200 and 500 buses from [32, 33].

3. REPRODUCING THE "GRIDMIND" EXPERIMENT

This chapter discusses the reproduction of the experiment and results given in [27]. The authors of [27] refer to their reinforcement learning environment and agent collectively as "GridMind."

3.1 Overview of GridMind

In [27], the authors demonstrate the use of deep reinforcement learning to perform voltage control in the electric transmission grid by controlling generator voltage set points. This paper won a best paper award at the IEEE Power and Energy Society General Meeting in 2019. In order to obtain baseline results and a baseline environment, the GridMind environment was recreated as closely as possible given the information in the paper and with some additional information obtained via email correspondence with one of the authors. The following subsections will provide an overview of the GridMind environment, and the results the authors presented.

3.1.1 Environment and Algorithm

The authors of [27] implemented their own custom reinforcement learning environment using GridPACK [28] as a power flow solver. The authors are intentionally vague about the details of the reinforcement learning algorithm used as well as neural network design, as they desire to maintain a competitive advantage in this research area. However, they do mention that the "DQN" algorithm is used. Training was performed on a relatively powerful Linux server with an Intel Xeon processor and 528 GB of RAM.

All testing and results presented in [27] use the IEEE 14 bus test system (which can be found at [33]).

3.1.2 States, Rewards, and Actions

3.1.2.1 States

In [27], the authors mention that the states they use consist of bus voltage magnitudes and angles as well as line power flows (active and reactive power). However, after communicating

with one of the authors (Jiajun Duan), it was determined that for the results presented in the paper only bus voltage magnitudes were used as input states.

3.1.2.2 Rewards

The authors of [27] used a very simple reward structure. After the agent takes an action, it is given a reward of +100 if all bus voltage magnitudes are within the "normal" range ($[0.95, 1.05]$ per unit). If any bus voltage magnitudes are in the so called "diverged" range (< 0.8 or > 1.25 per unit), a reward (penalty) of -100 is given. The third option is if no bus voltage magnitudes are in the "diverged" zone, but not all are in the "normal" range, a reward (penalty) of -50 is given.

At the end of each episode, a final end-of-episode reward is given. This reward is equal to the total cumulative episode rewards divided by the number of actions taken. The authors postulate this encourages the agent to take fewer actions.

3.1.2.3 Actions

In [27], a single action is considered to be the update of all generator voltage set points. This action may not change all generator set points (*i.e.*, the command may contain preexisting set points). All generator voltage set points were discretized into the set $0.95, 0.975, 1.0, 1.025, 1.5$. In general, this technique leads to an action space of dimension $n_v^{n_g}$, where n_v is the number of discrete voltage set points and n_g is the number of available generators. Since the IEEE 14 bus test case has five generators and five voltage set points are considered, the resulting action space is of dimension $5^5 = 3125$.

3.1.3 Training

3.1.3.1 Environment Initialization

For each training episode described in [27], all loads are randomly scaled between 80% and 120% of their nominal value (both active and reactive power levels). A random scaling factor is used for each load individually. The generators at buses 1 and 2 are considered to be available for active power dispatch, and participation-factor-based automatic generation control is used to meet demand for each episode. the remaining three generators (at buses 3, 6, and 8) are not considered

available for active power dispatch, and only provide voltage support.

In a second variation of environment initialization, the authors of [27] additionally introduce a small set of contingencies. Specifically, in each episode a random line is taken out of service. The lines considered are (from bus - to bus) 1-5, 2-3, 4-5, and 7-9.

3.1.3.2 Episode Termination

An episode is considered to be complete if one of the following three conditions are met:

- All bus voltages are within the "normal" range ($[0.95, 1.05]$ per unit).
- The power flow simulation failed to converge.
- The agent has exceeded its per-episode action cap.

In [27], the number of per episode allowed actions is not specified directly. Upon contacting an author, they specified "10-20." To reproduce the work in [27], an action cap of 15 is used.

3.1.3.3 Training Duration

All training discussed in [27] involves training for 10,000 episodes.

3.1.4 Results

The primary results presented in [27] are episode rewards as training progresses. The authors show that at first, the agent does not earn very high rewards, but by the end of training, the agent can consistently earn high rewards and take very few actions. Unfortunately the authors do not delve into specifics such as what actions the agent takes or what initial voltage conditions are like in the various episodes.

3.2 Reproduction of GridMind

In order to establish a baseline for the work presented in this thesis, the GridMind experiments presented in [27] have been reproduced as accurately as possible given the details the authors of [27] were willing to divulge. The experiment was recreated using PowerWorld Simulator as the power flow solver [29], a new Gym environment [24, 31], and a DQN algorithm provided by [41].

3.2.1 DQN Algorithm

The DQN algorithm is as discussed in Section 2.4 (DQN with all improvements), and the hyper parameters used are reported in Table 2.1. All experiments are carried out with a neural network architecture containing two hidden layers, each with 64 neurons.

3.2.1.1 Training Procedures

Since the required training duration will vary drastically based on neural network architecture, specific DQN algorithm, hyper parameters, and more, training was not simply performed for 10,000 episodes as presented in [27]. Instead, training is allowed to proceed for 500,000 simulation steps, with each episode capped at 15 steps (actions) per episode. As discussed in Section 3.1.3.2, an episode will also terminate if all bus voltages are in the "normal" range or the power flow fails to converge. Training was halted early if the agent successfully fixed voltage issues with a single action in 99 of the most recent 100 episodes. Recall that in the case of GridMind, all generator voltages are set simultaneously in a single action.

3.2.1.2 Testing Procedures

After training is complete, the agent is tested on 5,000 episodes which were not present in training. No learning is performed during this 5,000 episode testing session.

3.2.2 GridMind Reproduction Results and Discussion

This subsection will detail the results of reproducing the environment and results presented in [27] without any alterations of the setup or parameters.

3.2.2.1 Episode Initialization Discussion

Through the course of reproducing the work in [27], it was found that in the absence of contingencies, the loading conditions described in [27] never lead to low voltage conditions. In fact, if all load power levels are set to the maximum (120% of nominal), the lowest voltage in the system is approximately 1.01 per unit. Therefore, in the course of training the reinforcement learning agent will never actually see any low voltage conditions. Conversely, the IEEE 14 bus base case has

three generators set above the maximum acceptable voltage of 1.05 per unit. Thus, every episode the agent sees begins with bus over-voltages. These issues will be addressed in Section 3.3.

3.2.2.2 Results - No Contingencies

Figure 3.1 illustrates training and testing results for the simple case where there are no contingencies. Training rewards and number of training actions are presented in Figures 3.1a and 3.1b, respectively. Note that the results in Figures 3.1a and 3.1b are average per 100 episodes with a sliding window. The normalized frequencies of testing rewards and episode action counts are given in Figures 3.1c and 3.1d, respectively. The bar values should be read as a percentage.

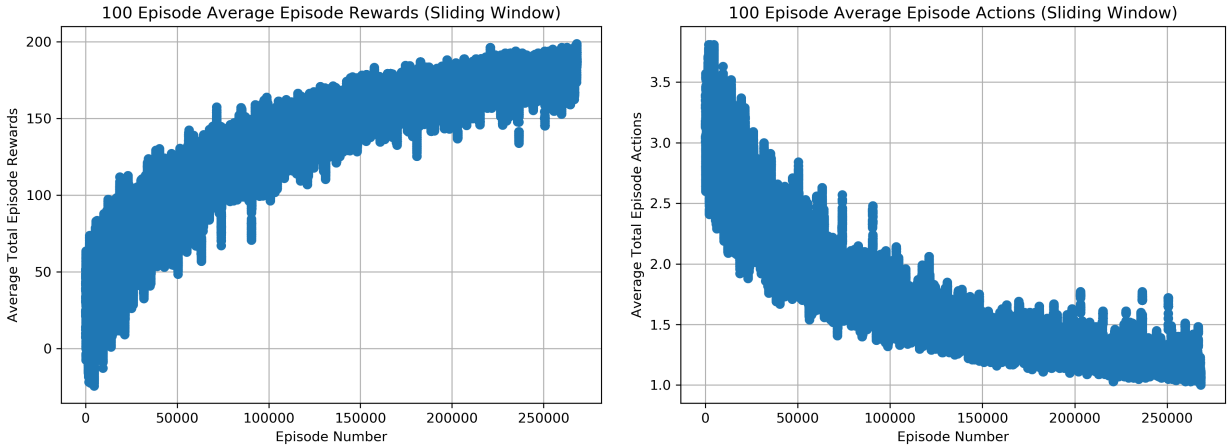
Keeping in mind the caveats discussed in Section 3.2.2.1, it can be seen in Figure 3.1a that as training progresses, the agent becomes increasingly adept at fixing voltage issues. Similarly, Figure 3.1b shows that the agent requires less actions per episode as training progresses. Figure 3.1c illustrates that during testing the agent earned the maximum possible reward of 200 (which can only be achieved by fixing voltage issues with one action) in 100% of the 5,000 testing episodes. Similarly, Figure 3.1d shows the agent fixed voltage issues with a single action in all testing episodes.

Upon investigating the actions which the agent took in testing, it turns out that only two of the available 3,125 actions were utilized. Similarly, in a repeated run of this exact experiment but with a different random number generator seed, the agent took the exact same single action in 100% of the testing episodes. While it's clear that the agent is correctly learning, the environment initialization is too simple to show that the GridMind agent can generalize to more complex power systems with more diverse system loading and generator conditions (e.g. starting voltage set point, active power dispatch, etc.).

3.2.2.3 Results - With Single Line Contingencies

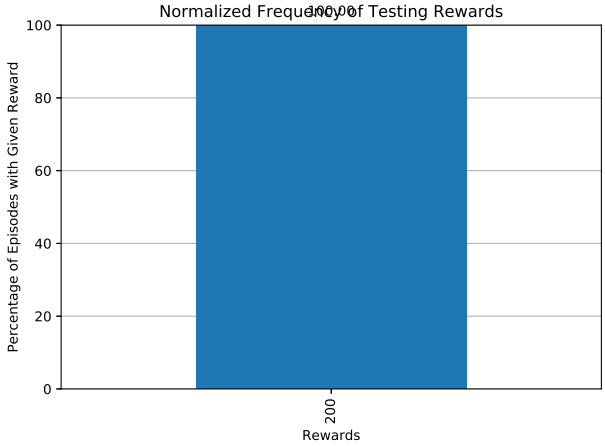
As discussed in Section 3.1.3.1, the authors of [27] considered a variant of environment initialization where each episode contains a random single line contingency.

Figure 3.2 shows training and testing results when a neural network with two hidden layers each with 64 neurons is leveraged. Figure 3.2 uses the same conventions as Figure 3.1, which was

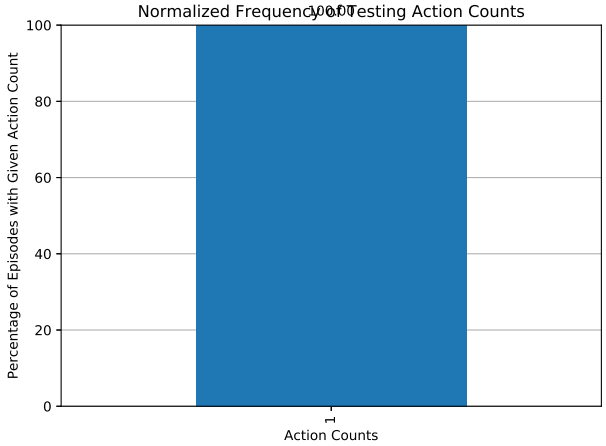


(a) Training: average rewards

(b) Training: number of actions taken



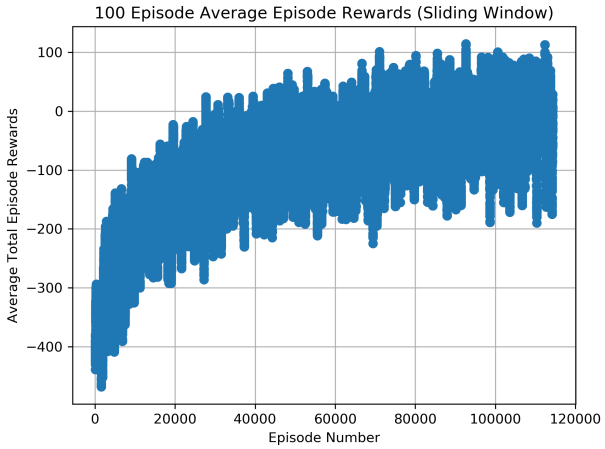
(c) Testing: reward distribution



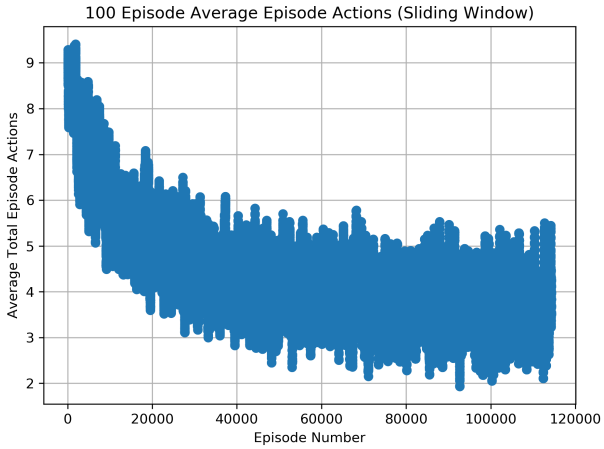
(d) Testing: action count distribution

Figure 3.1: Training and testing rewards and actions: GridMind without contingencies, neural network hidden layers: $[64, 64]$.

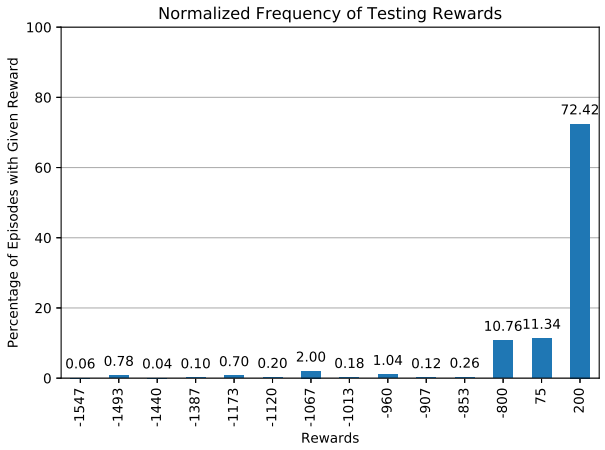
discussed in Section 3.2.2.2. It can be seen in Figure 3.2 that in contrast to the training rewards presented in Figure 3.1a, the presence of single line contingencies results in a noisier and slower "learning curve." Additionally, it can be seen in Figure 3.2c that the agent only achieved the maximum reward in approximately 74.2% of the testing episodes. As illustrated in Figure 3.2d, the agent hit the action cap of 15 actions per episode in 16.2% of testing episodes.



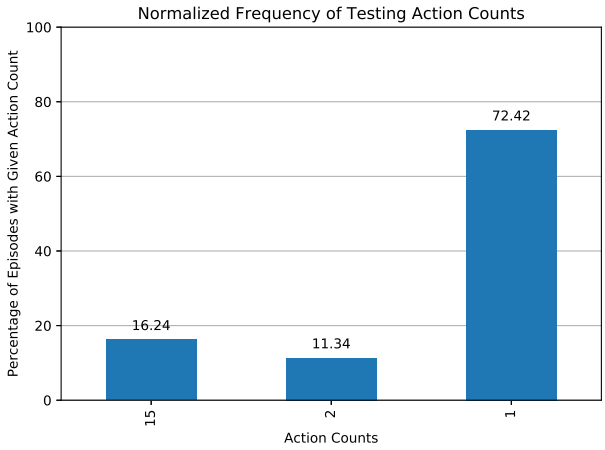
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward distribution



(d) Testing: action count distribution

Figure 3.2: Training and testing rewards and actions: GridMind with contingencies, neural network hidden layers: $[64, 64]$.

Upon investigating the specific sequences of actions taken during testing, it was found that the agent can exhibit cyclic or repetitive behavior if it does not succeed in its first action. Table 3.1 illustrates cyclic behavior in 3.1a and repetitive behavior in 3.1b. Note that the reward of -100 at the end of each table is due to the agent hitting the action cap of 15, and being additionally assessed the end of episode reward (penalty). This cyclic and repetitive behavior is clearly not desirable, and has inspired two modifications that will be discussed in Sections 3.4 and 4.3. The first modification restricts training and testing actions to be unique on a per-episode basis, and the second modification gives the agent a "no-op" action which does not instantiate any change in the environment.

Table 3.1: Cyclic and repetitive action-taking behavior in GridMind testing with contingencies

(a) Cyclic behavior

Action Count	Action Taken	Reward
1	2467	-50
2	2900	-50
3	3099	-50
4	2467	-50
5	2900	-50
6	3099	-50
7	2467	-50
8	2900	-50
9	3099	-50
10	2467	-50
11	2900	-50
12	3099	-50
13	2467	-50
14	2900	-50
15	3099	-100

(b) Repetitive behavior

Action Count	Action Taken	Reward
1	3023	-50
2	1673	-50
3	1673	-50
4	1673	-50
5	1673	-50
6	1673	-50
7	1673	-50
8	1673	-50
9	1673	-50
10	1673	-50
11	1673	-50
12	1673	-50
13	1673	-50
14	1673	-50
15	1673	-100

3.3 GridMind with Modified Episode Initialization

Sections 3.2.2.3 and 3.2.2.2 show that the GridMind agent performs relatively well both with and without single line contingencies, obtaining the maximum per-episode reward in 74.2% and

100% of testing episodes, respectively. However, as mentioned in Section 3.2.2.1, in the absence of contingencies there are never any undervoltages in the system, simplifying what the agent must learn. Additionally, as discussed in Section 3.1.3.1, the GridMind environment does not cover a large portion of the possible grid state space since. In addition to the narrow loading band (80%-120%), the generators use simple participation factor control, always start with the same voltage set points, and the same two generators (at buses 1 and 2) are used to meet active power demand.

In order to test whether the GridMind agent/architecture is generalizable to a more diverse set of grid conditions, the episode initialization procedure discussed in Section 2.3.3 is leveraged. Table 3.2 illustrates the parameters used for environment and episode initialization. Note that single line contingencies are applied for each episode as described in Sections 3.1.3.1 and 3.2.2.3.

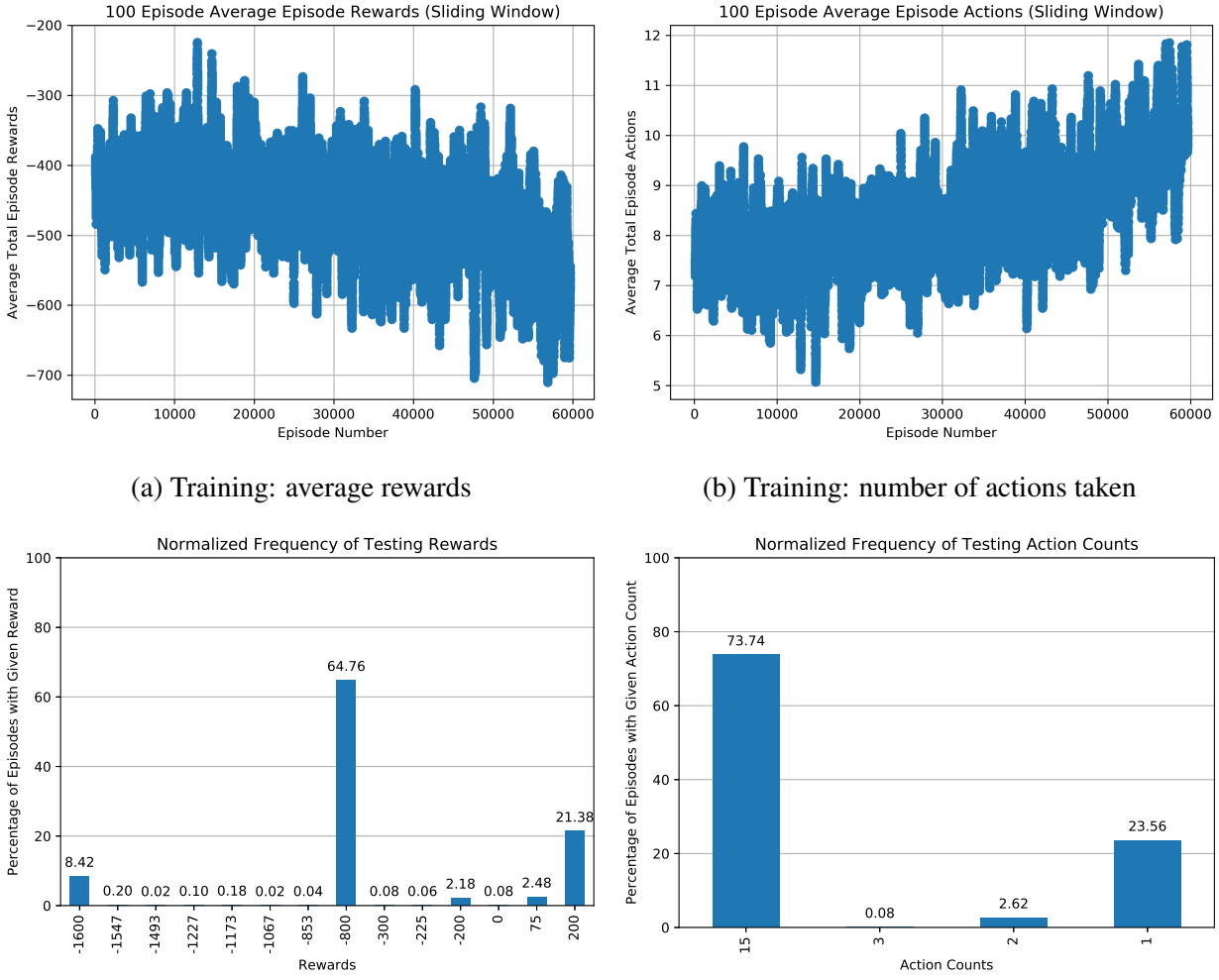
Table 3.2: Episode/environment initialization parameters

Variable	**Variable Value**
Maximum possible system loading (% of nominal)	140%
Minimum possible system loading (% of nominal)	60%
Percent chance an individual load is off	10%
Percent chance loads have leading power factor	10%
Minimum possible load power factor	0.8
Possible generator voltage set points	0.95, 0.975, 1.0, 1.025, 1.05
Lines which may be opened (one open per episode)	1-5, 2-3, 4-5, 7-9
Generators available for active power dispatch (by bus number)	1, 2, 3, 6, and 8

Note that in addition to the lower minimum and higher maximum system loading, the process by which load active and reactive power levels are determined leads to much more diverse loading conditions than the GridMind process. Additionally, the generator commitment and dispatch procedure leads to different generators being online for different scenarios, and dispatched in different proportions.

Training and testing results for GridMind in this more challenging environment are shown in Figure 3.3. It's clear from Figure 3.3a that the agent failed to learn a useful policy, as there is

no clear reward increase over training episodes. Additionally, Figure 3.3b shows an upward trend in the average action required per episode, further strengthening the argument that the agent had failed to learn. It can be seen in Figure 3.3c that the agent earned the maximum per-episode reward (200) in 21.4% of the testing episodes, and earned the minimum possible reward (-1600) in 8.42% of the episodes. Furthermore, the agent was successful in bringing all voltages in-band for 19.2% of testing episodes which started with out-of-band voltages.



(a) Training: average rewards

(b) Training: number of actions taken

(c) Testing: reward distribution

(d) Testing: action count distribution

Figure 3.3: Training and testing rewards and actions: GridMind with modified episode initialization, neural network hidden layers: $[64, 64]$.

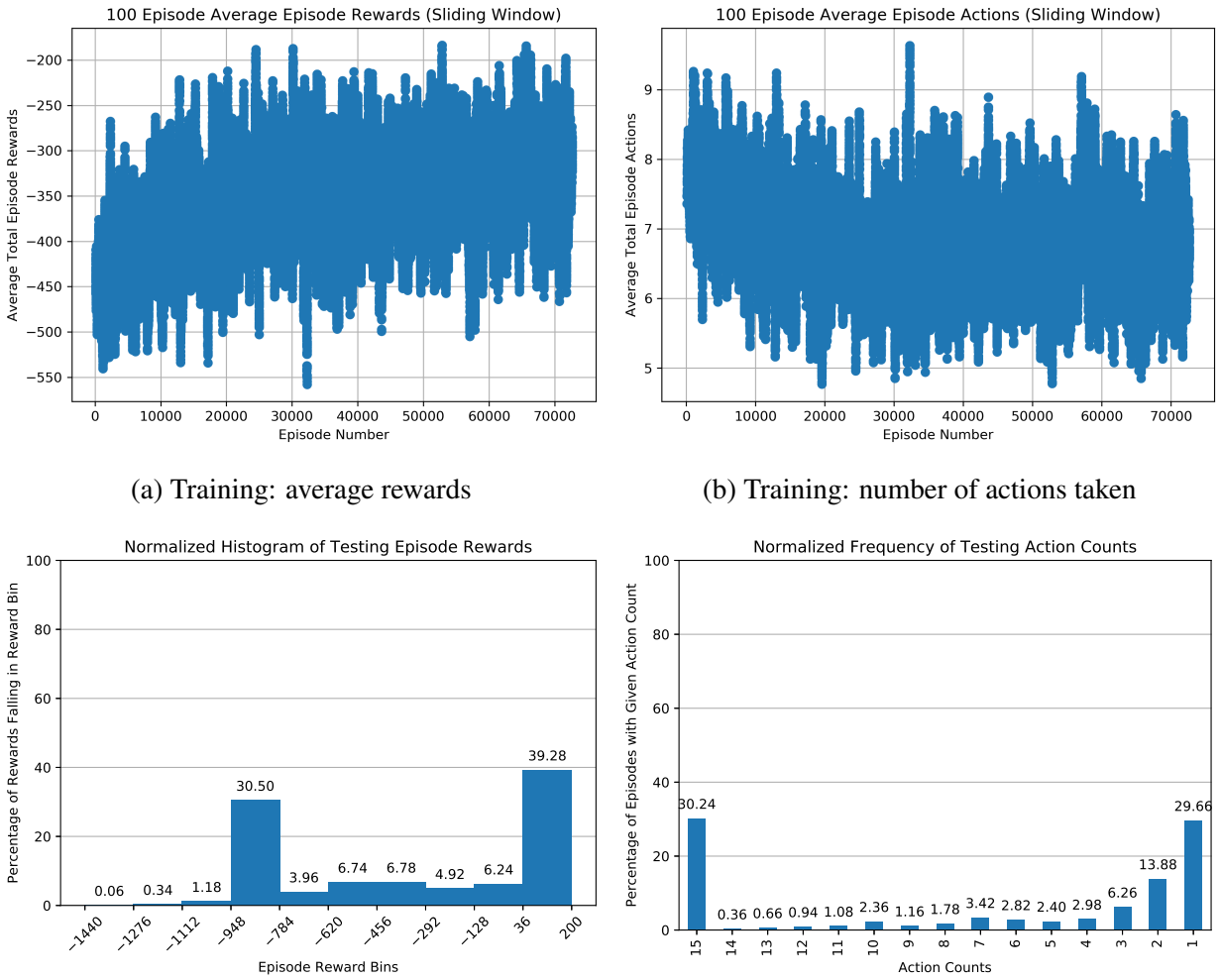
3.4 GridMind with Modified Episode Initialization and Unique Actions per Episode

As shown in Section 3.3, the GridMind agent performs poorly in the presence of a more challenging environment with widely varying grid conditions. Section 3.2.2.3 and Table 3.1 show that when using the "out-of-the-box" DQN algorithm [18, 20] with improvements [42–44] provided by [41], the agent tends to take the same action more than once per episode. While this is certainly desirable behavior in the context of Atari video games, it is not desirable for power system voltage control. To address this issue, the algorithm provided by [41] was modified to ensure that during both testing and training, specific actions are only allowed to be taken once per episode. If an action has already been taken in the episode, the action with the next highest Q-value estimate is selected instead. This is a novel contribution to the application of deep reinforcement learning to the field of power system voltage control. The implementation of the DQN algorithm modification can be found in [45].

The results for using the GridMind architecture with the modified DQN algorithm are shown in Figure 3.4. Note that Figure 3.4c is now a histogram, as opposed to previous figures which showed the raw reward distribution. The modified algorithm leads to significantly more unique rewards than the stock algorithm, since the agent is forced to take a different action if the top action has already been taken in the given training episode.

It can be seen in Figure 3.4a and 3.4b that the agent struggled to learn a useful policy, similarly to the GridMind agent with the unmodified DQN algorithm. While not shown explicitly in Figure 3.4c, the agent earned the maximum reward in 29.7% of testing episodes. Figure 3.4c does show the agent earned a reward between 36 and 200 in 39.28% of testing episodes. However, Figure 3.4d illustrates that the agent took 15 actions (the maximum allowed) in 30.2% of testing episodes, indicating the agent was not particularly successful.

While the modified DQN algorithm does effectively prevent repetitive and cyclic behavior, it does not prevent the agent from getting stuck learning a single best voltage setting (recall GridMind sets all generator voltage set points simultaneously in a single action). It was found in testing that the agent took the exact same first action in all 5,000 testing episodes. However, it's important to



(a) Training: average rewards

(b) Training: number of actions taken

(c) Testing: reward histogram

(d) Testing: action count distribution

Figure 3.4: Training and testing rewards and actions: GridMind with modified episode initialization and unique actions per episode, neural network hidden layers: $[64, 64]$.

note this is more likely due a combination of the the relatively simple observation space and reward definition, action space in which all generators are set simultaneously, and the noted failure to learn illustrated in Figure 3.4. As mentioned in Section 3.2.2.2, the unmodified algorithm sometimes learns a single action as well.

3.4.1 Comparison of Modified and Unmodified DQN Algorithm

Despite the lack of a clear "learning curve" in training (see Figure 3.4a), the modified DQN algorithm did lead to more successful voltage control when compared to the unmodified DQN

algorithm. As learning and success can vary across different training and testing runs due to the stochastic nature of both neural network initialization and episode creation (grid case initialization), Table 3.3 presents summary testing metrics for three runs with different random seeds. A testing episode is considered "successful" if the agent brings all voltages within the acceptable range ($[0.95, 1.05]$ per unit) before the episode is terminated for other reasons (hit action cap, power flow diverged). While the agent that learned with the unmodified algorithm had a composite average success rate across the three runs of 20.4%, the modified algorithm's average success rate was 45.8%. Unsurprisingly, the modified algorithm takes more unique actions in testing than the unmodified algorithm, which tends to take repetitive or cyclic actions as discussed in Section 3.2.2.3. While this algorithm modification does not cause GridMind to succeed in the majority of episodes in this more challenging environment, it clearly helps the agent be more successful in fixing voltage issues.

Table 3.3: GridMind testing results with and without DQN algorithm modification

(a) Unmodified DQN Algorithm

Random Seed	**Success Percentage**	**Mean Reward**	**Number of Unique Test Actions**
0	23.9	-620.2	7
1	16.2	-628.98	20
2	21.0	-630.05	1

(b) Modified DQN Algorithm

Random Seed	**Success Percentage**	**Mean Reward**	**Number of Unique Test Actions**
0	48.5	-304.10	39
1	46.4	-332.70	85
2	42.5	-478.18	18

3.5 GridMind Conclusions and Discussion

This chapter provided an overview of the "GridMind" DQN environment, provided a good-faith reproduction of GridMind, tested GridMind's generalizability in a more challenging environ-

ment with more diverse grid conditions, and tested a novel DQN algorithm modification with the GridMind environment.

In Section 3.2.2, the experiments from [27] were reproduced. While the results were not identical to those presented in [27], the trends were similar. Without single line contingencies, the agent was able to learn a policy which successfully brought voltages in bounds in 100% of testing episodes. When contingencies were included, the agent's success rate dropped to 83.8%. While these success rates are impressive, the environment consisted of a very narrow range of loading conditions and very simple generator dispatch, as discussed in Section 3.2.2.1.

In order to test the generalizability of the GridMind architecture, it was tested with more challenging grid conditions in Section 3.3. There it was shown that the GridMind agent effectively fails to learn a useful policy and was successful at fixing voltage issues in a small percentage of cases. It's worth noting that in this more challenging environment, there is no guarantee that it is even possible to bring all bus voltages in bounds.

It was shown in Section 3.4 that modifying the DQN algorithm to allow each unique action to be taken only once per episode in both training and testing leads to an improvement in agent success. However, the algorithm modification alone does not bring the agent's success anywhere near to the rate seen in the simple environment with narrow loading band and simple generation dispatch.

In light of the issues mentioned in this section, several potential architectural improvements are proposed:

1. **Reward design.** The GridMind reward system is very simple, and does not reward incremental improvements. An improved reward design could account for per-bus voltage movement after an action is taken.

2. **Action space.** The GridMind action space is not scalable to large systems as $n_v^{n_g}$ actions are available, where n_v is the number of discrete voltage set points and n_g is the number of generators. Rather than using a combinatorial action space, a better action space might have a single action per control setting. In the case of generator voltage control alone, this would

lead to an action space with dimension $n_v \times n_g$, which is significantly more scalable.

3. **Observations.** The GridMind architecture relies on bus voltages alone. It was shown that the agent's success decreased in the presence of single line contingencies, as well as in the challenging environment with different generator conditions. Adding additional observations such as line state and/or generator state (on/off) may help in learning.

Experimentation with these improvements will be presented in the following chapter.

4. TESTING ARCHITECTURAL IMPROVEMENTS ON THE IEEE 14 BUS SYSTEM

The focus of this chapter is on experimenting with improvements to the deep reinforcement learning environment to facilitate better learning and scalability. Chapter 3 presented the so-called "GridMind" environment, reproducing as faithfully as possible the work in [27]. This chapter presents environment modifications designed to address GridMind's shortcomings, which are summarized in Section 3.5. Broadly speaking, these improvements are related to reward design, action space design, and observation space design.

4.1 Overview

All experiments in this chapter will use identical environment initialization parameters (e.g. minimum load factor, random seed, etc.). These parameters were presented and discussed in Section 3.3 and are summarized in Table 3.2. In this chapter, all experiments use the IEEE 14 bus system obtained from [33]. In general, all experiments will be performed both with and without the algorithm modification which forces each action to be unique within each episode, as discussed in Section 3.4. Additionally, a random agent and a graph-based agent were developed in order to provide a basis for comparison with the deep reinforcement learning agents. The remaining sections in this chapter will provide details related to the observations, actions, and rewards used for the experiments; a description of the random and graph-based agents along with their testing results; training and testing results for the DRL agents with different observation and reward designs; and finally a discussion of all the results contained in this chapter.

4.2 Observations

Experiments in this chapter examine several different observation combinations and variants:

- Voltage observations only
- Voltage observations and generator state observations (on/off, also known as open/closed)
- Voltage observations and line state observations (open/closed)

- Voltage observations, generator state observations, and line state observations

In addition to the observation variants listed above, each variant is also tried with the use of so-called "min-max" scaling for voltage observations. All experiments consider a power flow to be "failed" if any single bus voltage dips below 0.7 per unit or if any single bus voltage exceeds 1.2 per unit. In this way, the absolute lower and upper voltage bounds are known for all scenarios/episodes. After min-max scaling, the new minimum voltage observation value is 0.0 (no longer in per unit), and the maximum voltage observation value is 1.0 (also no longer in per unit).

The choice to implement min-max scaling originates from the fact that in general, ensuring neural network inputs are consistently scaled can aid in learning stability and speed. During typical (non-emergency) power system operations, most voltages are close to nominal (one per unit). If most voltages are numerically similar, a neural network may have difficulties learning the importance of voltage magnitude differences (e.g. 0.94 per unit is unacceptable, while 0.95 per unit is acceptable). The linear "stretching" the min-max scaling provides can help ease this differentiation.

4.3 Action Space

Since the 14 bus system does not contain any shunts or on-load tap-changing voltage regulators, only the setting of generator voltage set points are considered as actions. However, in contrast to the GridMind architecture, each action represents a single voltage set point for a single generator. Thus, the dimension of the action space is $n_v \times n_g$, where n_v is the number of discrete voltage set points and n_g is the number of generators. With the 14 bus system's five generators and the use of five voltage set points, the action space has dimension $[25 \times 1]$. Recall that by contrast the GridMind architecture utilizes an action space where each action is a unique combination of all generator set points, resulting in an action space dimension of $n_v^{n_g}$ (3125 for the 14 bus system with five generator voltage set points).

When analyzing results, it is important to keep in mind that the new $[25 \times 1]$ action space definition may result in more actions to fix voltage issues, since each action only changes the voltage at a single generator at a time, rather than each action setting all generator voltage set

points all at once as is done with GridMind.

In addition to the simplified action space, a "no-op" (no operation) action is introduced, which as the name implies, makes no change in the power system. This action is introduced in the hope that if the agent has performed all sensible actions yet voltage issues remain, it may choose to take no action. With the inclusion of the no-op action, the final dimension of the action space is $[26 \times 1]$.

4.4 Reward Design

Since there is no guarantee that a given episode has a viable solution which brings all voltages into the acceptable band ($[0.95, 1.05]$ per unit), it is important that rewards not simply only consider whether or not the objective (bring all voltages within limits) has been met. This chapter presents two similar reward schemes, both of which emphasize the movement of voltages toward the acceptable band.

Equations (4.1) through (4.9) present definitions for sets which will be leveraged when presenting the two reward schemes. Time step $t-1$ corresponds to the discrete time step before the most recent action was taken, and time step t corresponds to the time step after the most recent action was taken. Bus numbers are denoted with a superscript i, while time steps are denoted with a subscript. The quantity n_{buses} corresponds to the number of buses in the system, and v is used to denote per unit voltage.

Bus voltage (V) set:

$$V = \{v^i \quad | \quad i \in [1, 2, \ldots, n_{\text{buses}}]\} \tag{4.1}$$

Voltages out-of-band (O) at time step $t-1$:

$$O_{t-1} = \{v^i \in V \quad | \quad (v^i_{t-1} < 0.95) \vee (v^i_{t-1} > 1.05)\} \tag{4.2}$$

Voltages out-of-band (O) at time step t:

$$O_t = \{v^i \in V \quad | \quad (v^i_t < 0.95) \vee (v^i_t > 1.05)\} \tag{4.3}$$

Voltages in-band (I) at time step $t-1$:

$$I_{t-1} = \{v^i \in V \quad | \quad 0.95 \leq v^i_{t-1} \leq 1.05\} \tag{4.4}$$

Voltages in-band (I) at time step t:

$$I_t = \{v^i \in V \quad | \quad 0.95 \leq v^i_t \leq 1.05\} \tag{4.5}$$

Voltages in-band at time step $t-1$, but moved below (M_b) band at time step t:

$$M_b = \{v^i \in V \quad | \quad (v^i \in I_b) \wedge (v^i_t < 0.95)\} \tag{4.6}$$

Voltages in-band at time step $t-1$, but moved above (M_a) band at time step t:

$$M_a = \{v^i \in V \quad | \quad (v^i \in I_b) \wedge (v^i_t > 1.05)\} \tag{4.7}$$

Voltages out-of-band at time step $t-1$, moved in the right direction (rd):

$$P_{rd} = \left\{ v^i \in V \quad | \quad \Big(\big((v^i_{t-1} < 0.95) \wedge (v^i_t > v^i_{t-1}) \big) \vee \big((v^i_{t-1} > 1.05) \wedge (v^i_t < v^i_{t-1}) \big) \Big) \right\} \tag{4.8}$$

Voltages out-of-band at time step $t-1$, moved in the wrong direction (wd):

$$P_{wd} = \left\{ v^i \in V \quad | \quad \Big(\big((v^i_{t-1} < 0.95) \wedge (v^i_t < v^i_{t-1}) \big) \vee \big((v^i_{t-1} > 1.05) \wedge (v^i_t > v^i_{t-1}) \big) \Big) \right\} \tag{4.9}$$

4.4.1 Reward Scheme 1: Per-Bus Movement Magnitude Scheme

This reward scheme provides rewards for moving voltages in the right direction (toward the acceptable band), while providing penalties for voltages that move in the wrong direction (away from the acceptable band). The movement rewards and penalties are scaled by the movement magnitude so that a larger voltage movement obtains a larger reward or penalty. Additionally, a

penalty is given for taking an action in order to help incentivize the agent to minimize the number of actions taken. Equations (4.10) through (4.16) illustrate several components of the reward scheme, and Figure 4.1 presents the complete flow necessary to compute the reward after an action has been taken. In several of the equations below, some terms are multiplied by 100. This is to make the corresponding reward/penalty apply per 0.01 per unit movement ($\frac{1}{0.01} = 100$). Note in Figure 4.1 that the corresponding equations for different components of the reward computation are indicated to the right of the relevant blocks. Also note that the $\rightarrow$ symbol used within a block is used as shorthand for "moved." The code for the reward scheme can be found at [31].

Absolute change in distance from nominal voltage times 100:

$$\Delta n_i = ||v_{i,t-1} - 1.0| - |v_{i,t} - 1.0|| \cdot 100, \quad \forall v_i \in V \tag{4.10}$$

Reward (r) for moving in the right direction (δ is a constant scalar value. For all experiments presented in this chapter, $\delta = 1$):

$$r = r + \Delta n_i \cdot \delta, \quad \forall v^i \in P_{rd} \tag{4.11}$$

Penalty for moving in the wrong direction:

$$r = r - \Delta n_i \cdot \delta, \quad \forall v^i \in P_{wd} \tag{4.12}$$

Penalty for starting in-band, but moving below the band:

$$r = r + (v_t^i - 0.95) \cdot 100 \cdot \delta, \quad \forall v^i \in M_b \tag{4.13}$$

Penalty for starting in-band, but moving above the band:

$$r = r + (1.05 - v_t^i) \cdot 100 \cdot \delta, \quad \forall v^i \in M_a \tag{4.14}$$

Extra penalty for any buses that moved out of the band (ρ is a constant scalar value. For all experiments presented in this chapter, $\rho = 10$):

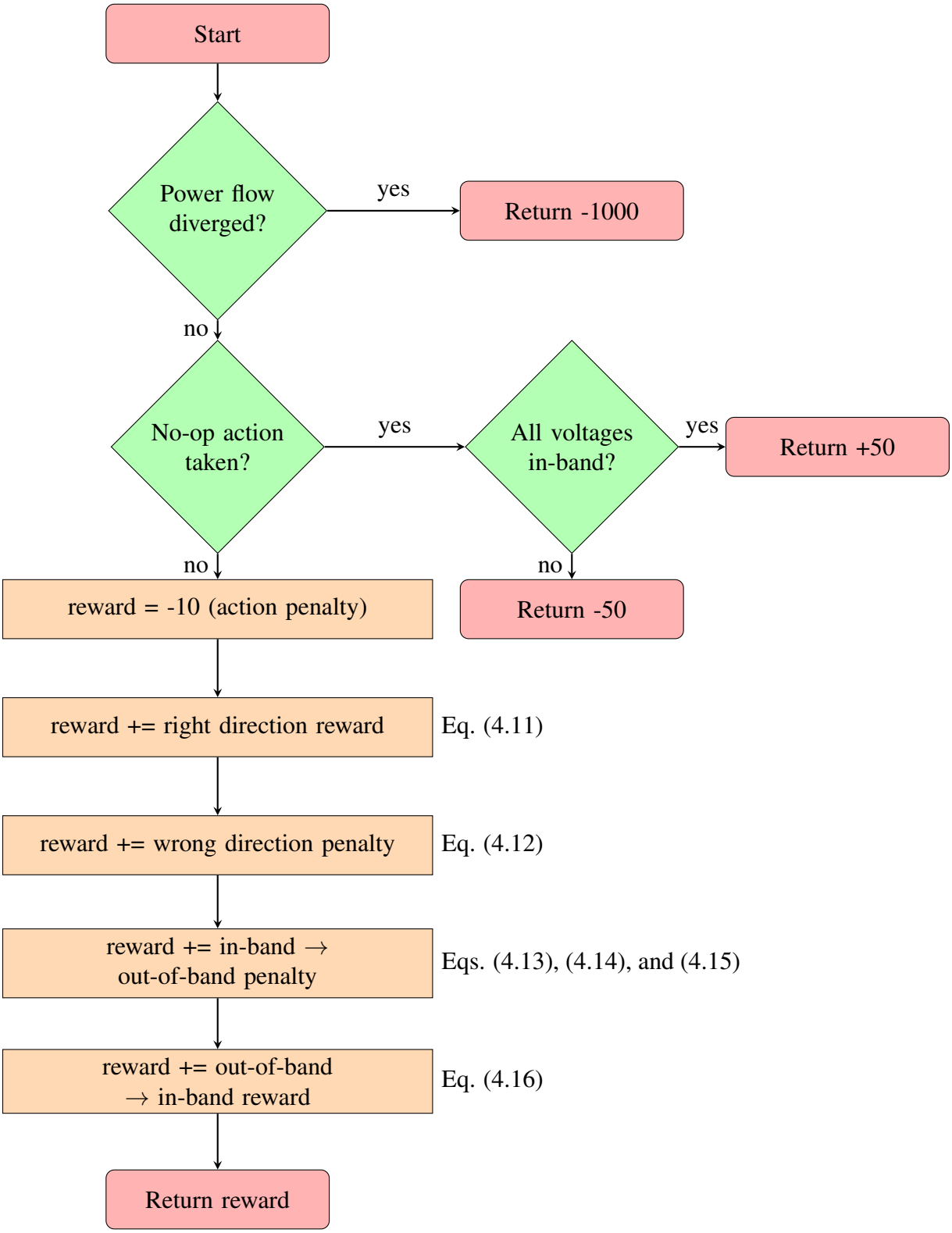
$$r = r - \rho, \quad \forall v^i \in (M_b \cup M_a) \tag{4.15}$$

Extra reward for any buses that moved in to the band:

$$r = r + \rho, \quad \forall v^i \in (O_{t-1} \cap I_t) \tag{4.16}$$

Note that the parameter ρ is set equal to 10 so that if just one bus voltage moves from out-of-band to in-band, the action penalty of -10 is counter-acted. A very large penalty (-1000) is assessed if the agent causes the power flow to diverge. The no-op reward ($+50$) is in place so that if a given episode starts with all voltages in bounds, the agent can choose to not take a control action and still be rewarded. Conversely, if the agent takes no action but not all voltages are in bounds, it is penalized (-50). This -50 no-op penalty has the potential to cause issues if the particular episode/scenario is unsolvable (i.e., no sequence of control actions can possibly bring all voltages in-band). For example, if all generators are already at their maximum voltage set point yet low voltage conditions still exist, the agent may choose to lower a voltage set point rather than take no action. The second reward scheme presented in the following section addresses this shortcoming.

Figure 4.1: Reward scheme 1 flow chart

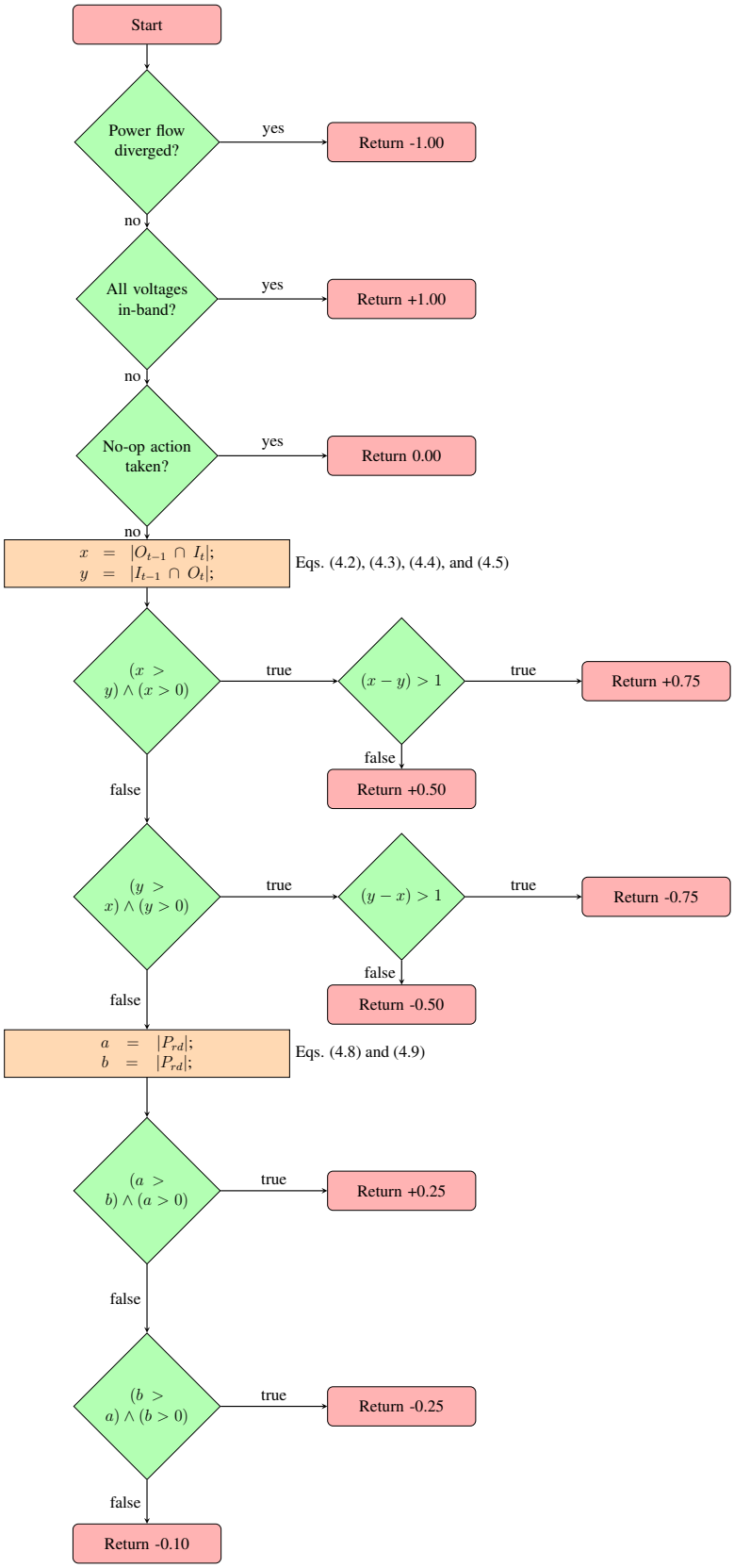


4.4.2 Reward Scheme 2: Clipped and Simplified Movement Scheme

This scheme keeps all rewards within the range $[-1.0, 1.0]$, inspired by the reward "clipping" done in [18, 19]. The authors of [18, 19] note that keeping rewards in a fixed band "limits the scale of the error derivatives" and thus makes the reward function useful across multiple games. In the case of the work presented in this thesis, the same notion applies, but instead helps the reward generalize across power system cases. The clipped scheme presented here has the following discrete reward possibilities: $\{-1.00, -0.75, -0.50, -0.25, -0.10, 0.00, +0.25, +0.50, +0.75, +1.00\}$, and the flow for determining the reward is presented in Figure 4.2.

It can be seen from Figure 4.2 that the agent is given the minimum reward (-1.00) if its action caused the power flow to diverge, and is given the maximum reward ($+1.00$) if its action brought all voltages in band. The no-op action is always given a reward of 0.00, unless all voltages are in bounds. A greater reward (penalty) is given if multiple bus voltages are brought in-band (out-of-band). If voltages move in the right (wrong) direction, the smallest reward (penalty) is assessed to the agent. If the agent's action is not the no-op action, but does not move any out-of-band voltages in the right or wrong direction, a "useless action" penalty of -0.10 is given. One potential shortcoming of this reward scheme is that moving 20 out-of-band voltages in-band receives the same reward as moving 2 out-of-band voltages in-band.

Figure 4.2: Reward scheme 2 flow chart



4.5 Random and Graph-Based Agents for Comparison

In order to provide a basis for comparison, both a random agent and a graph-based agent were created and tested against the same testing scenarios that the DRL agents were tested against. The random agent behaves exactly as one might expect: during each time step it randomly chooses and takes an action from the environment's action space. As per usual, each episode proceeds either until all voltage issues have been fixed, the power flow diverges, any bus voltage goes below 0.7 per unit or above 1.2 per unit, or the agent hits the per-episode action cap. The random agent is run both with and without the unique-actions-per-episode requirement.

4.5.1 Graph-Based Agent

The graph-based agent has been created in an attempt to provide a reasonable benchmark for comparison with the DRL agents without writing a full-blown optimization program. As such, it is heuristically driven, leveraging the idea that voltage issues are typically "local" and are often remedied by dispatching reactive power resources near to the buses with voltage issues. In short, the agent constructs a graph of the power system network and changes the voltage set point at the generator which is "nearest" to the bus with minimum/maximum voltage. A more detailed description of the graph-based agent's algorithm for each testing episode is as follows:

1. Using PowerWorld [29] and ESA [30], obtain the Y-bus matrix for the system's current topology (recall single line contingencies are included).

2. Using the Python package NetworkX [46], create a graph representing the transmission system which uses reactance between buses as edge weights.

3. If all bus voltages are in band ($[0.95, 1.05]$ per unit), take the "no-op" action. Otherwise, proceed.

4. Determine the buses with the highest and lowest voltages in the system.

   (a) If the bus with the lowest voltage is below 0.95 per unit voltage, the "nearest" generator's voltage set point will be set to 1.05 per unit, and the bus with the lowest voltage

will be considered when computing distances to generators in Step 5.

(b) Otherwise, if the bus with the highest voltage is above 1.05 per unit voltage, the "nearest" generator's voltage set point will be set to 1.025 per unit, and the bus with the highest voltage will be considered when computing distances to generators in Step 5.

5. Using NetworkX's implementation of Dijkstra's algorithm [47], determine the "nearest" generator to the bus with the lowest or highest voltage (determined in Step 4), considering line reactance as a measure of electrical distance. Only generators which are on/active are considered.

   (a) If the "nearest" generator already has its voltage set point set to the desired voltage (determined in Step 4), find the next-nearest generator instead. Continue until a valid generator is found. If no valid generator is found, take the "no-op" action. Otherwise, proceed.

   (b) Set the determined "nearest" (or next-nearest or next-next-nearest, etc.) generator's voltage set point to the desired voltage.

There are several noteworthy items with respect to the graph-based agent. First, if an overvoltage condition is not fixed by reducing generator voltage set points to 1.025 per unit, it will never be fixed. In the absence of shunts (recall the 14 bus system does not have shunts) or off-nominal transformer tap positions, this should not generally be an issue. Second, the agent receives more system knowledge than the DRL agents do (e.g. full topology and line reactances). Third, the agent's "nearest" heuristic does not fully account for the meshed nature of the electric transmission system. A possibly more robust heuristic might instead perform some network equivalencing (e.g. Ward equivalents) before computing the shortest path in order to better determine which generator may have the highest impact.

4.5.2 Results for Random and Graph-Based Agents with the 14 Bus System

The random and graph-based agents were tested on the same three sets of 5,000 testing scenarios that the DRL agents were tested on (see Sections 3.2.1.2, 3.4.1, and 4.6). Table 4.1 presents

results for the graph-based and random agents. The values presented in Table 4.1 represent the mean across the three testing scenario sets. Note that "O.O.B." stands for "out of band," so the fifth column in Table 4.1 presents the mean success rate for episodes which started with voltages outside of the acceptable band. Since the random and graph-based agents do not consider rewards in their decision making, the success rates are identical across tests with different reward schemes. This also indicates that the random seeding is working as intended, since separate environment initializations result in identical results.

Table 4.1: Mean testing results for graph-based and random agents

Agent	**Reward Scheme**	**Unique Actions per Episode?**	**Percent Success**	**Percent Success, Episode Starts O.O.B.**	**Mean Reward**
Graph-based	1	Yes	46.97	40.79	-118.95
Graph-based	2	Yes	46.97	40.79	0.83
Random	1	No	22.46	14.47	-206.21
Random	1	Yes	24.34	16.52	-214.20
Random	2	No	22.46	14.47	-0.43
Random	2	Yes	24.34	16.52	-0.39

Due to the simplicity of the 14 bus system, it is believed that the graph-based agent's success rate is very near to the highest possible success rate that is physically achievable. As previously mentioned, not all scenarios are "solvable," i.e., there is no guarantee that a generator voltage set point configuration exists that brings all voltages in band. It's interesting to note that the GridMind environment with the modified DQN algorithm comes very close to the performance of the graph-based agent (see Section 3.4.1) with a mean O.O.B. success rate of approximately 39.90% compared to the graph-based agent's rate of 40.79%. However, when the DQN algorithm is unmodified (repeated actions are allowed), GridMind's performance is close to the random agent's performance (16.22% and 14.47% O.O.B. success rate, respectively). The random and graph-based agents will be used as a basis for comparison for the results presented in the following section.

4.6 Results

This section will group results in subsections by observations given to the agent. For each observation subsection, results will be presented for:

- Per unit voltage observations with reward scheme 1 (see Section 4.4.1)
- Per unit voltage observations with reward scheme 1 and the unique-action-per-episode algorithm modification (see Section 3.4)
- Min-max scaled voltage observations (see Section 4.2) with reward scheme 1
- Min-max scaled voltage observations with reward scheme 1 and the unique-action-per-episode algorithm modification
- Min-max scaled voltage observations with reward scheme 2 (see Section 4.4.2) and unique-actions-per-episode algorithm modification

It's important to note that all agents were trained and tested with identical environments with identical scenarios. However, since the training duration is based on the number of time steps rather than the number of episodes, some agents see more training episodes than others. In testing, a consistent set of episodes are used for all agents, so that testing results are directly comparable across the different agents and architectures.

4.6.1 Abbreviations, Terminology, and Methodology

Figures and tables within this results section contain several abbreviations for space. Per unit will be abbreviated as "P.U.," min-max scaled will be abbreviated as "M.M.,", out-of-band will be abbreviated as "O.O.B.", modified will be abbreviated as "mod.," and reward will be abbreviated as "rew." Some success rates will be described as "O.O.B." success rates. This refers to an agent's success rate only accounting for episodes which began with out-of-band voltages.

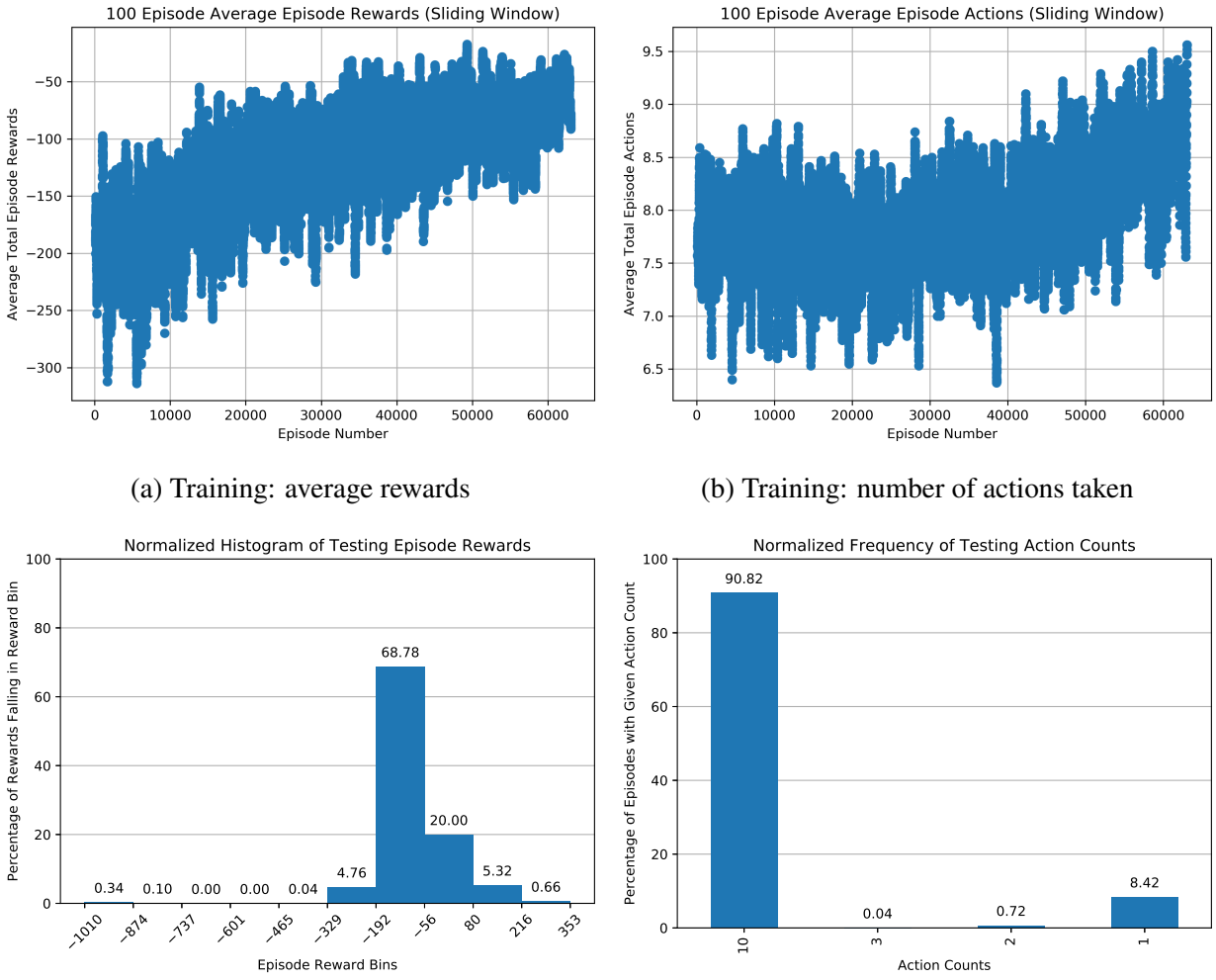
Each of the experiments in the following subsections were carried out three times with different random seeds. These independent experiments entail both training and testing. Figures

which present training rewards over time, actions taken per episode over training, testing reward histograms, and testing action count distributions all come from a single experimental run where the random seed equals zero. Unless otherwise specified, figures and tables which present mean success rates refer to the overall mean across all three experimental runs. A single experimental run involves 5,000 testing episodes which were never seen by the agent during training.

4.6.2 Bus Voltage Observations Only

This subsection presents results for agents which only received bus voltage observations in Figures 4.3, 4.4, 4.5, 4.6, and 4.7.

4.6.2.1 *Per Unit Voltage Observations with Reward Scheme 1*



(a) Training: average rewards

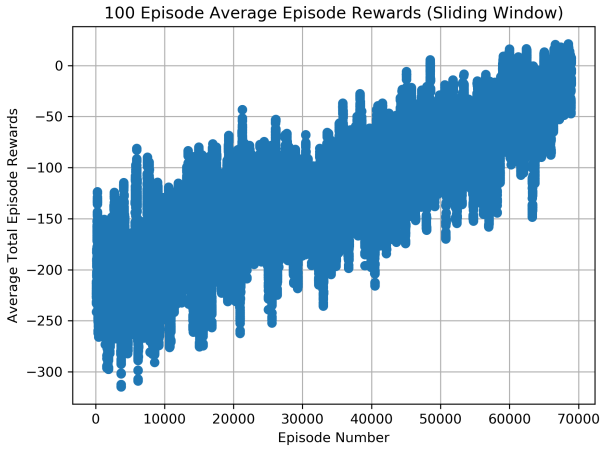
(b) Training: number of actions taken

(c) Testing: reward histogram

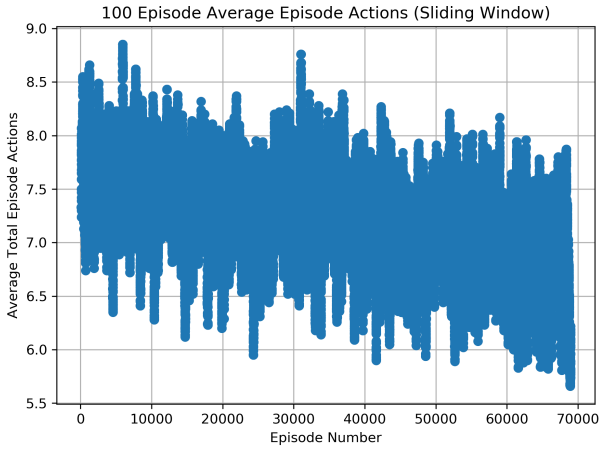
(d) Testing: action count distribution

Figure 4.3: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

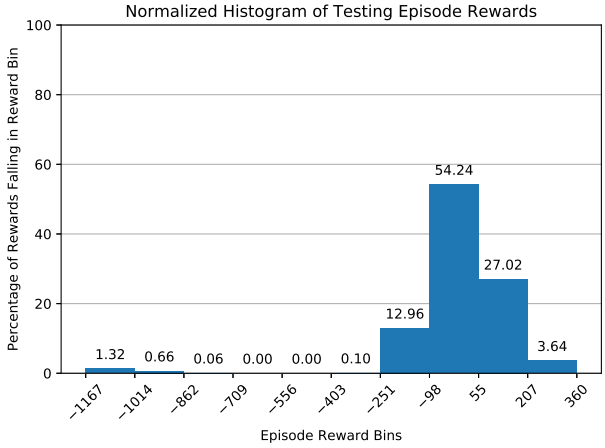
4.6.2.2 *Per Unit Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



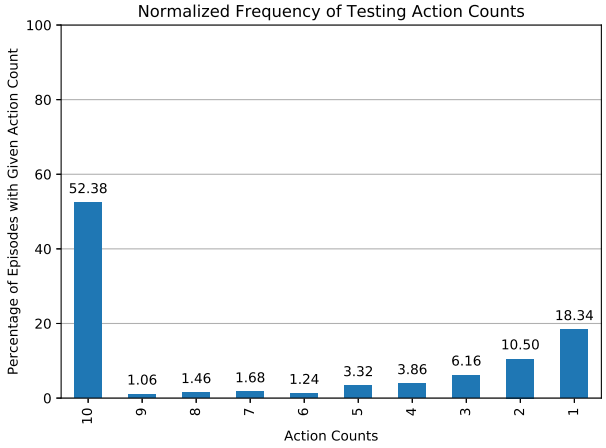
(a) Training: average rewards



(b) Training: number of actions taken



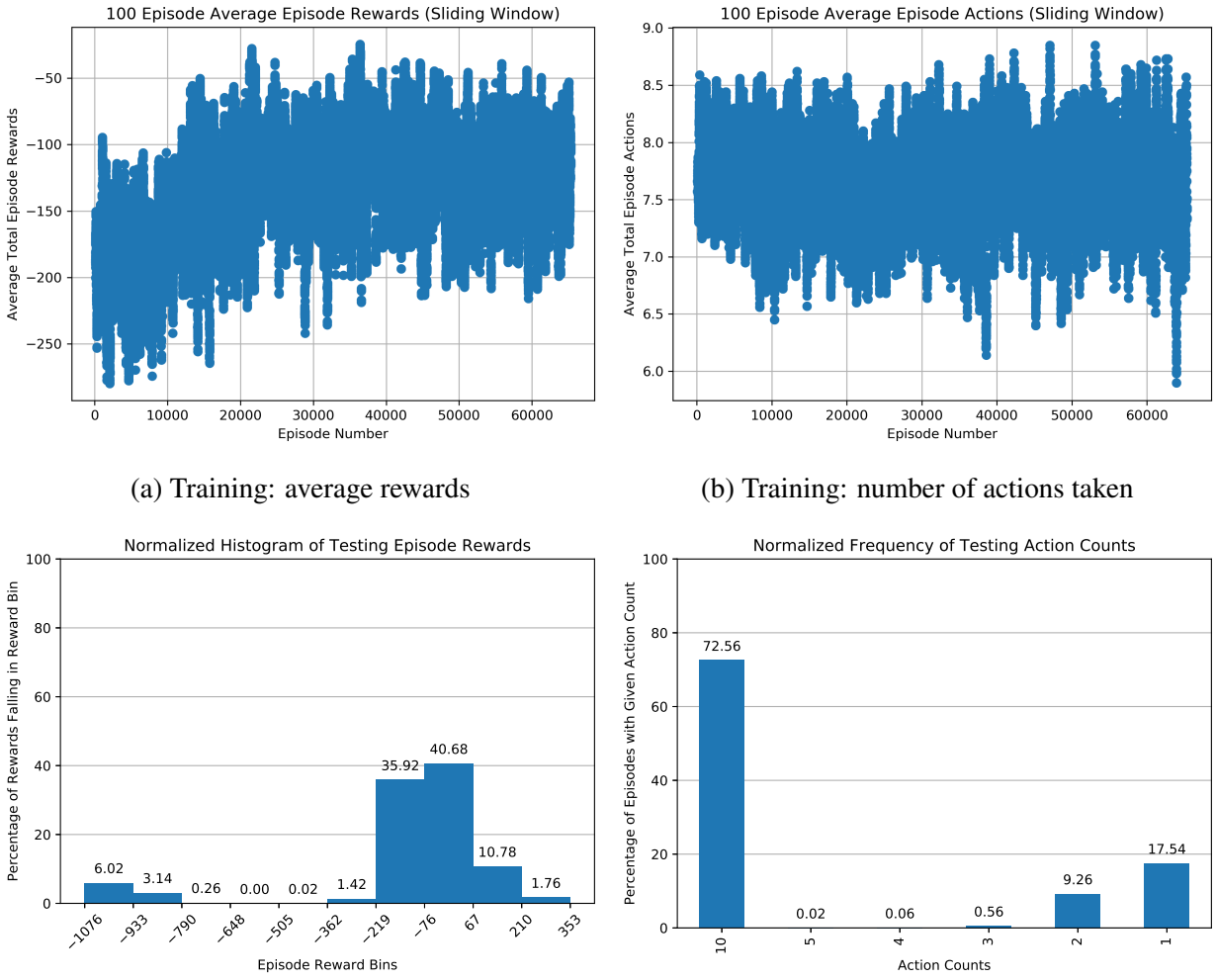
(c) Testing: reward histogram



(d) Testing: action count distribution

Figure 4.4: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.2.3 *Min-Max Scaled Voltage Observations with Reward Scheme 1*



(a) Training: average rewards

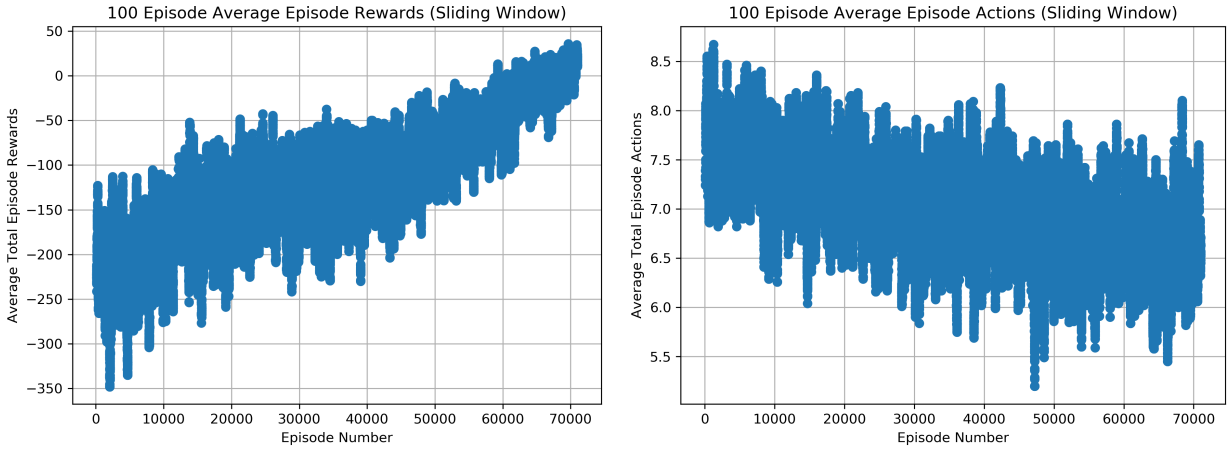
(b) Training: number of actions taken

(c) Testing: reward histogram

(d) Testing: action count distribution

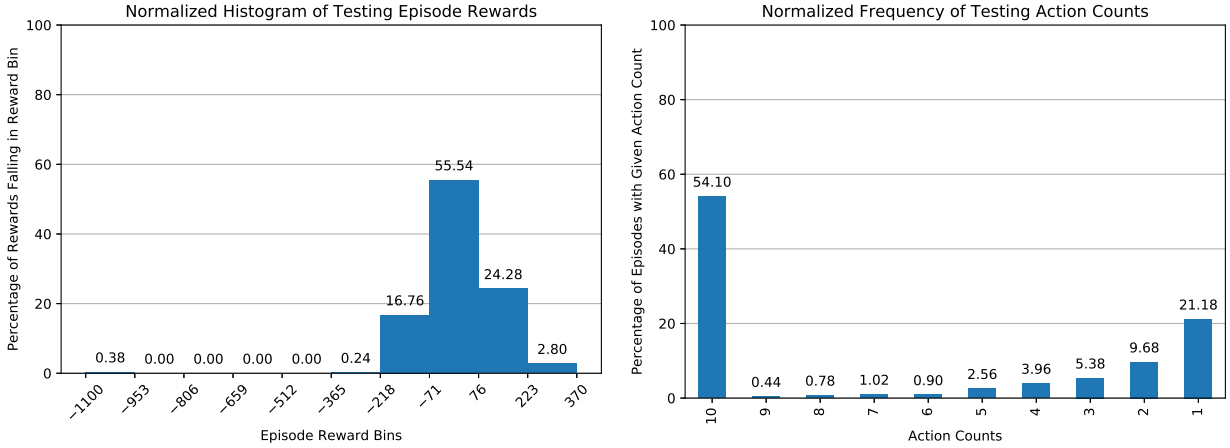
Figure 4.5: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

4.6.2.4 *Min-Max Scaled Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken

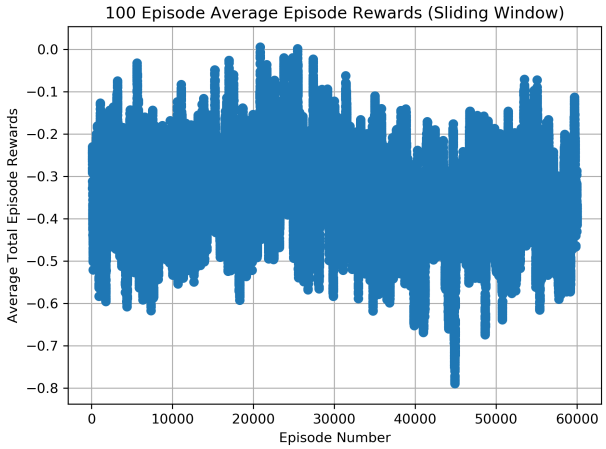


(c) Testing: reward histogram

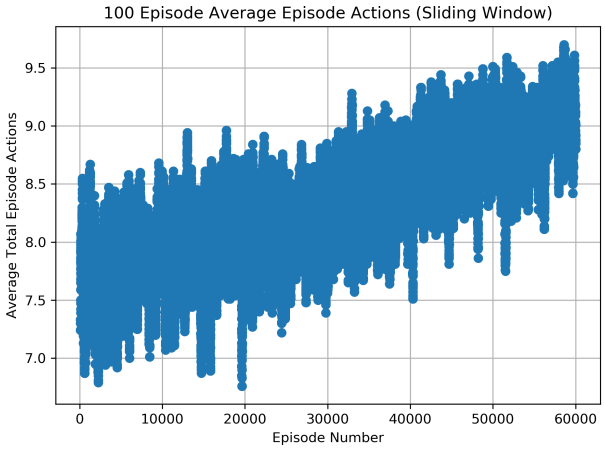
(d) Testing: action count distribution

Figure 4.6: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

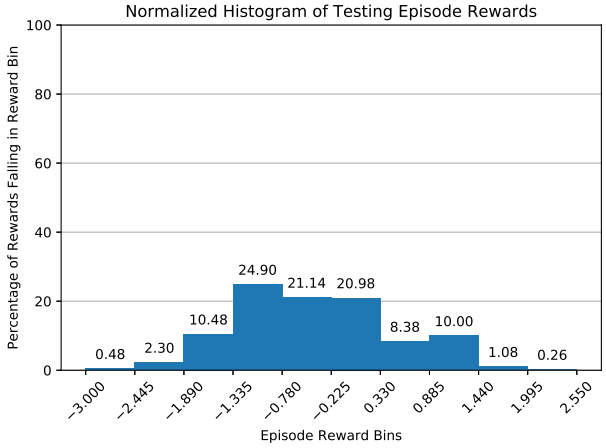
4.6.2.5 *Min-Max Scaled Voltage Observations with Reward Scheme 2 and Unique-Actions-Per-Episode Algorithm Modification*



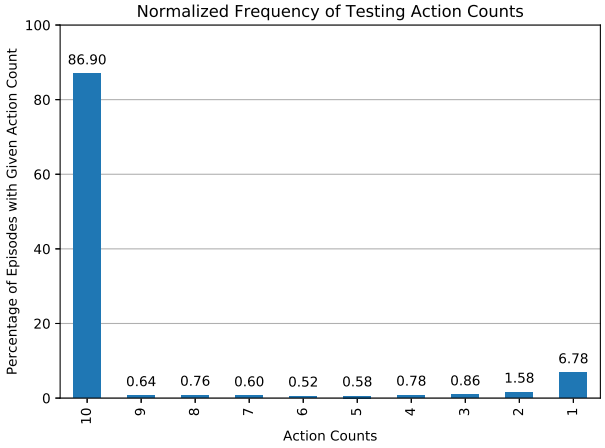
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count distribution

Figure 4.7: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 2, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.2.6 *Comparison and Discussion*

The "learning curves" in Figures 4.3a, 4.4a, and 4.6a show marked reward improvement over time. By contrast, Figures 4.5a and 4.7a do not show a clear trend over training. Figure 4.8 presents mean out-of-band success rates for all five experiments with different observation, reward, and algorithm combinations. Note that only two of the five experiments exceeded the success rate of the random agent, and none met or exceeded the performance of the graph-based agent. The two best performing experiments both used the unique-actions-per-episode algorithm modification and reward scheme 1.

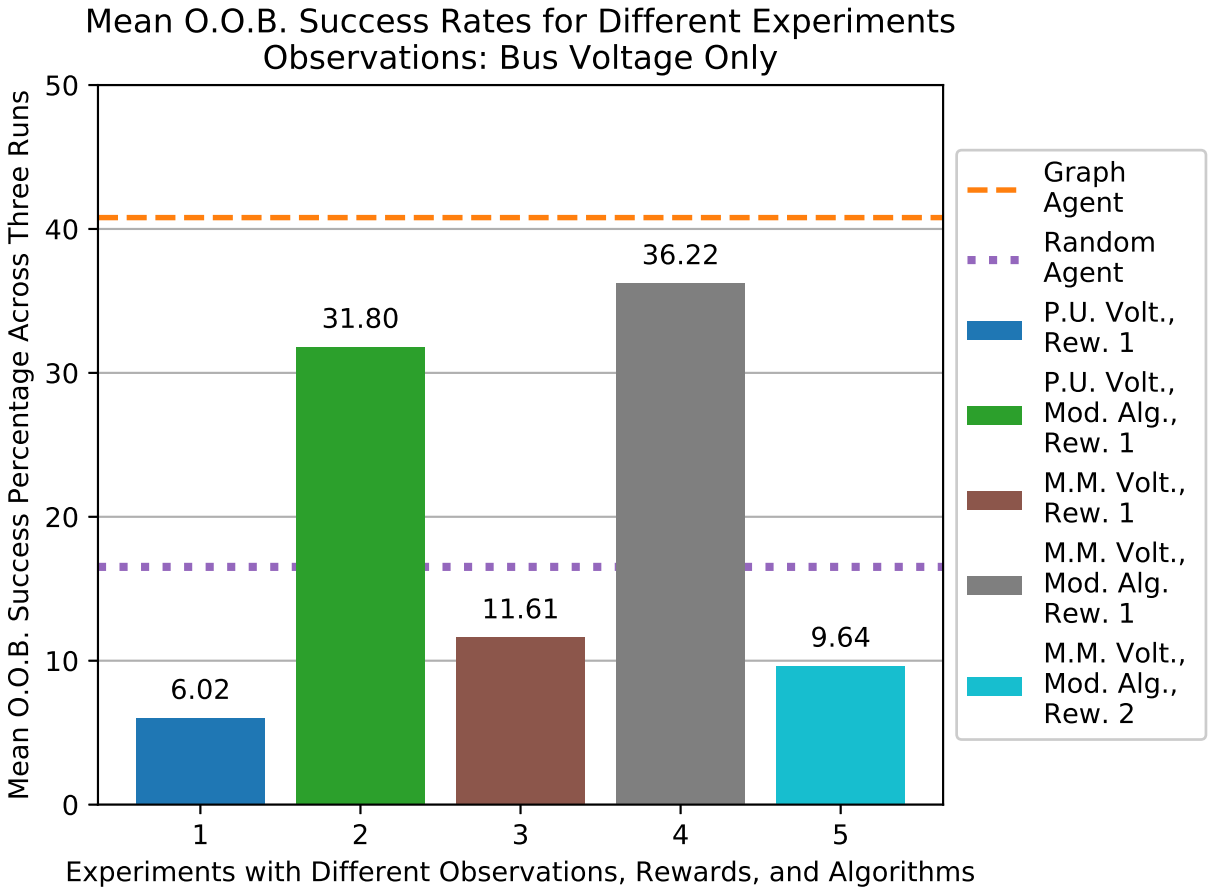
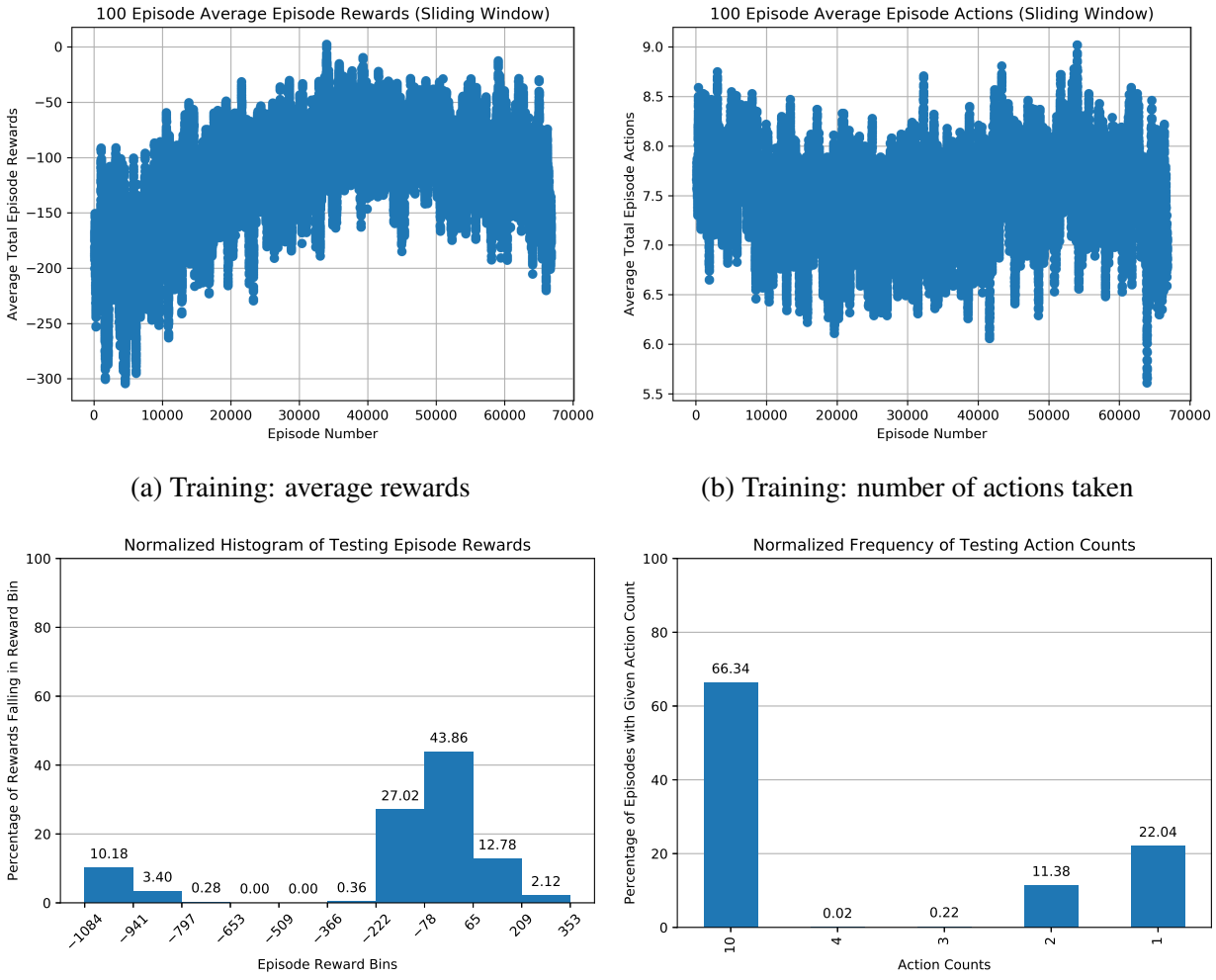


Figure 4.8: Mean out-of-band (O.O.B.) success rates for different experiments: bus voltage observations only

4.6.3 Bus Voltage Observations and Generator State Observations

The figures presented in this section illustrate training and testing results with the agent's observations contain both bus per unit voltage observations and generator state (on/off) observations. Training and testing results can be found in in Figures 4.9, 4.10, 4.11, 4.12, and 4.13.

4.6.3.1 Per Unit Voltage Observations with Reward Scheme 1



(a) Training: average rewards

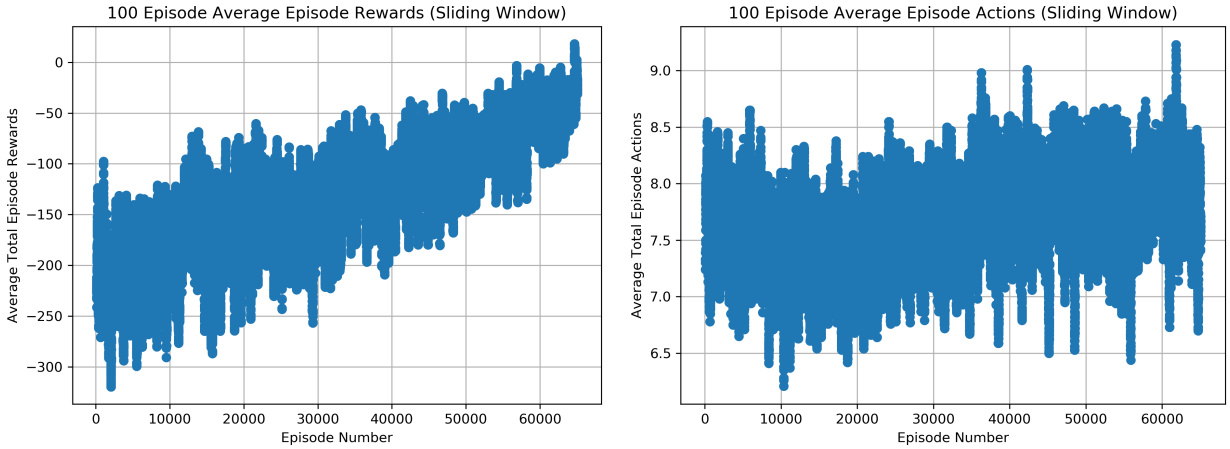
(b) Training: number of actions taken

(c) Testing: reward histogram

(d) Testing: action count distribution

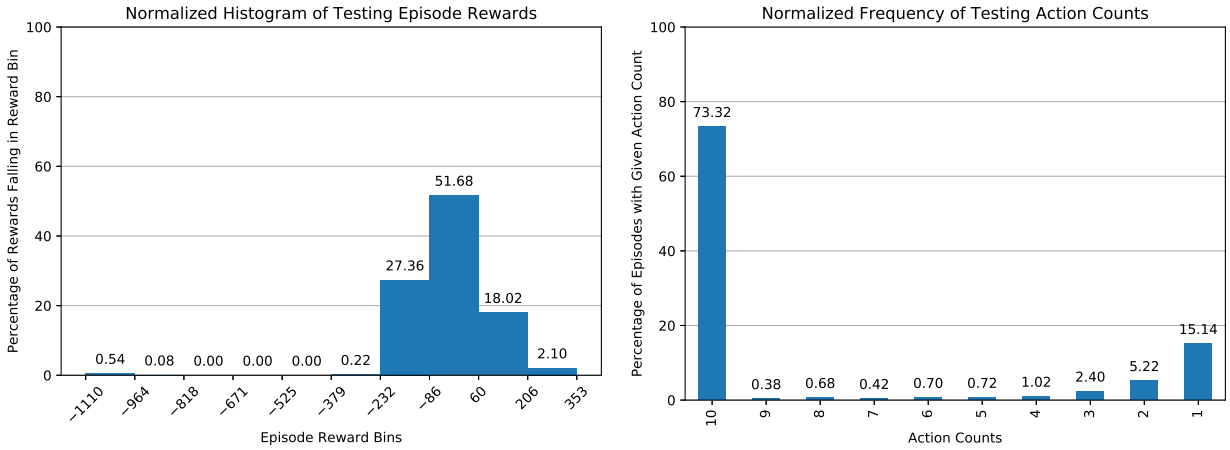
Figure 4.9: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

4.6.3.2 *Per Unit Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken

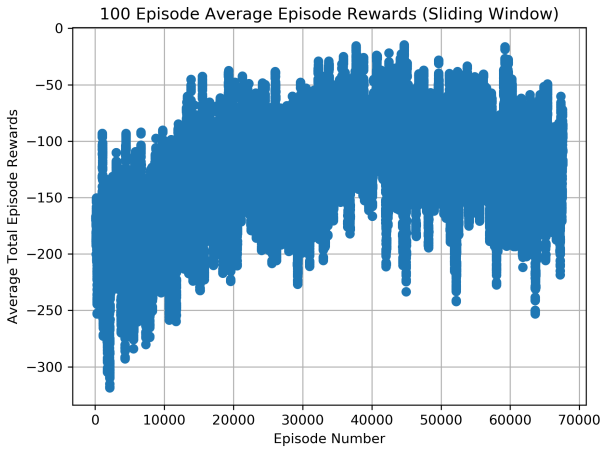


(c) Testing: reward histogram

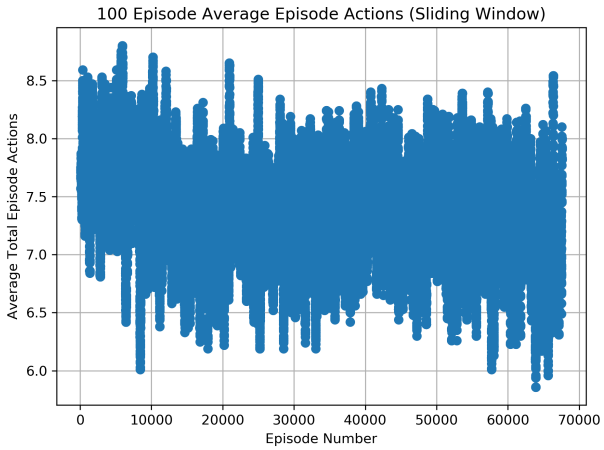
(d) Testing: action count distribution

Figure 4.10: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

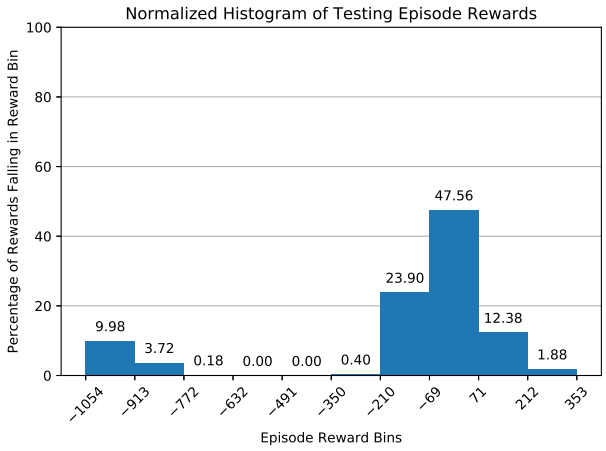
4.6.3.3 *Min-Max Scaled Voltage Observations with Reward Scheme 1*



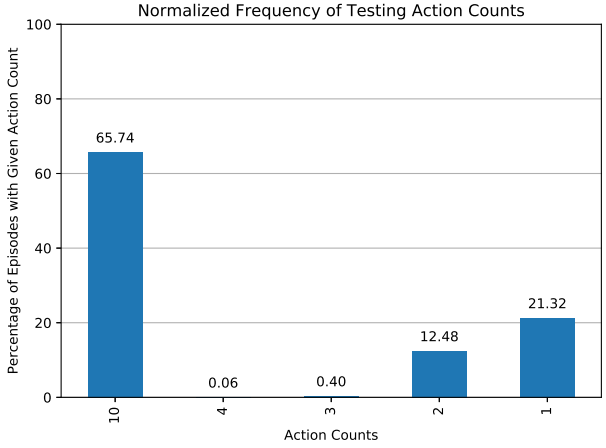
(a) Training: average rewards



(b) Training: number of actions taken



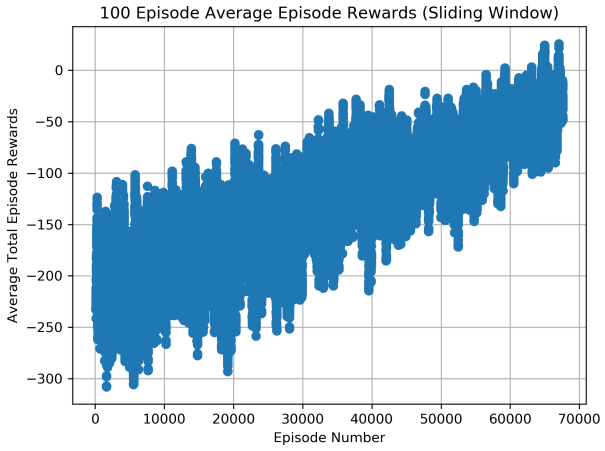
(c) Testing: reward histogram



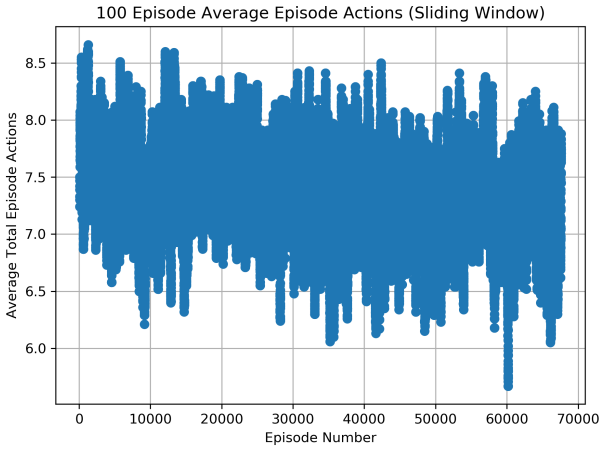
(d) Testing: action count distribution

Figure 4.11: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

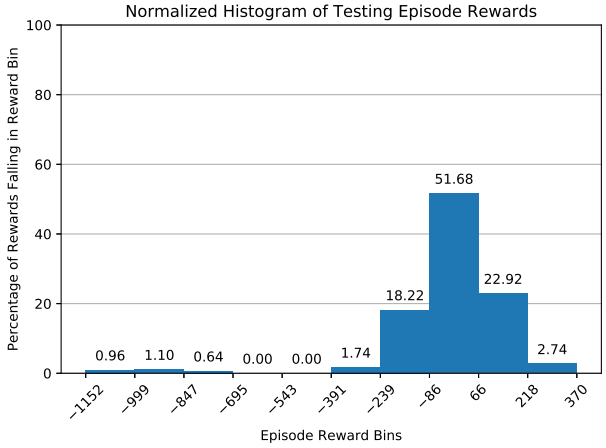
4.6.3.4 *Min-Max Scaled Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



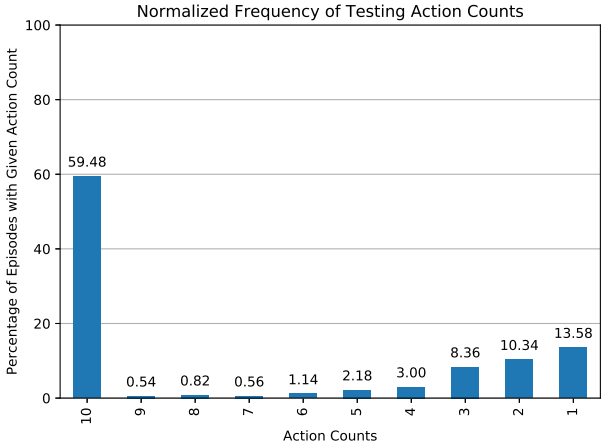
(a) Training: average rewards



(b) Training: number of actions taken



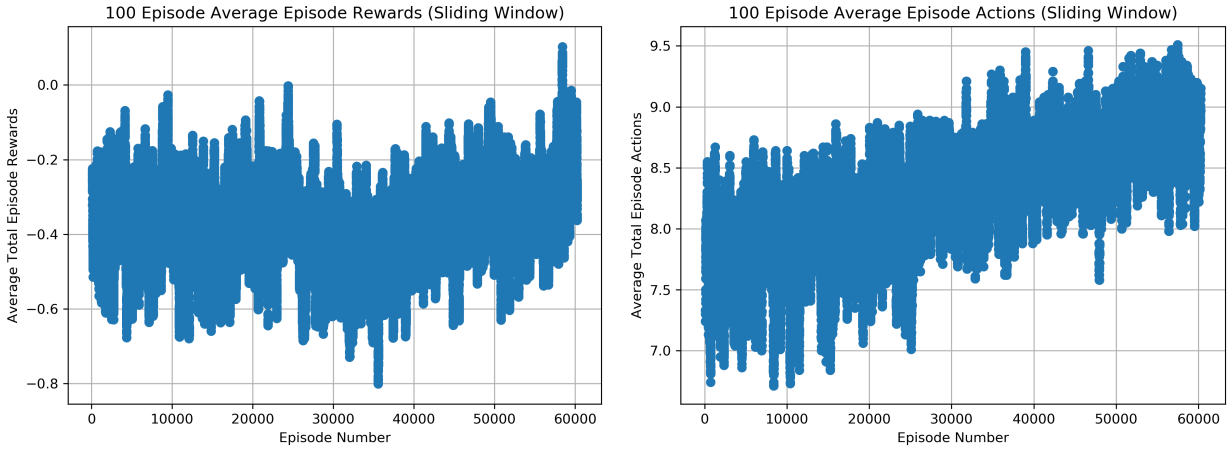
(c) Testing: reward histogram



(d) Testing: action count distribution

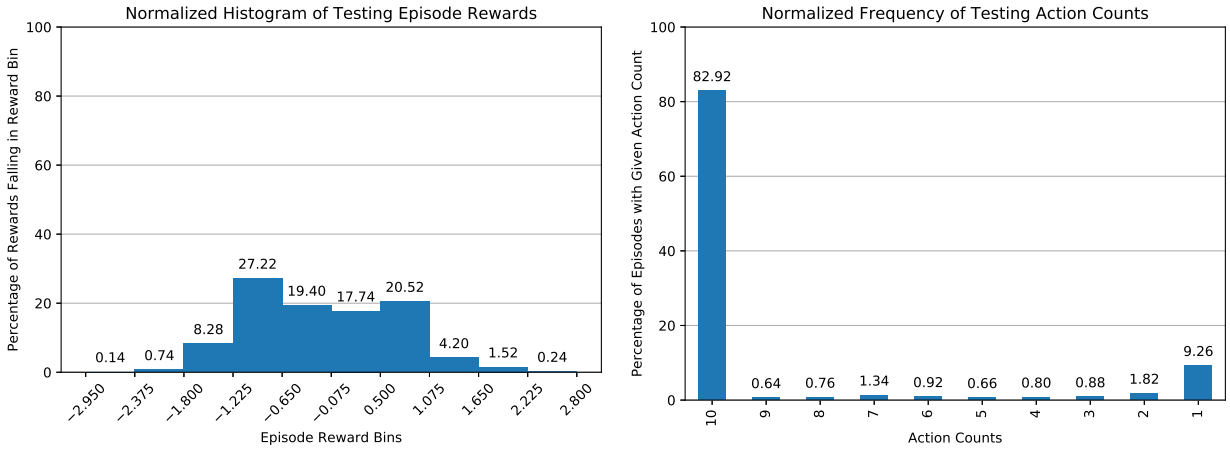
Figure 4.12: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.3.5 *Min-Max Scaled Voltage Observations with Reward Scheme 2 and Unique-Actions-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken



(c) Testing: reward histogram

(d) Testing: action count distribution

Figure 4.13: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 2, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.3.6 Comparison and Discussion

Of the five experiments, only two show consistently improving rewards over time, as illustrated in Figures 4.10a and 4.12a. Figure 4.14 illustrates mean out-of-band success rates for the various experiments. Only two experiments outperform the random agent, and both of these experiments use the modified DQN algorithm and reward scheme 1. Interestingly, the addition of generator state observations does not lead to improvements in the agent's success for the the two experiments which beat the random agent when compared to agents which received only voltage observations (presented in Section 4.6.2). However, the success for the other three experiments with success rates less than the random agent increases when compared to agents which only received voltage observations.

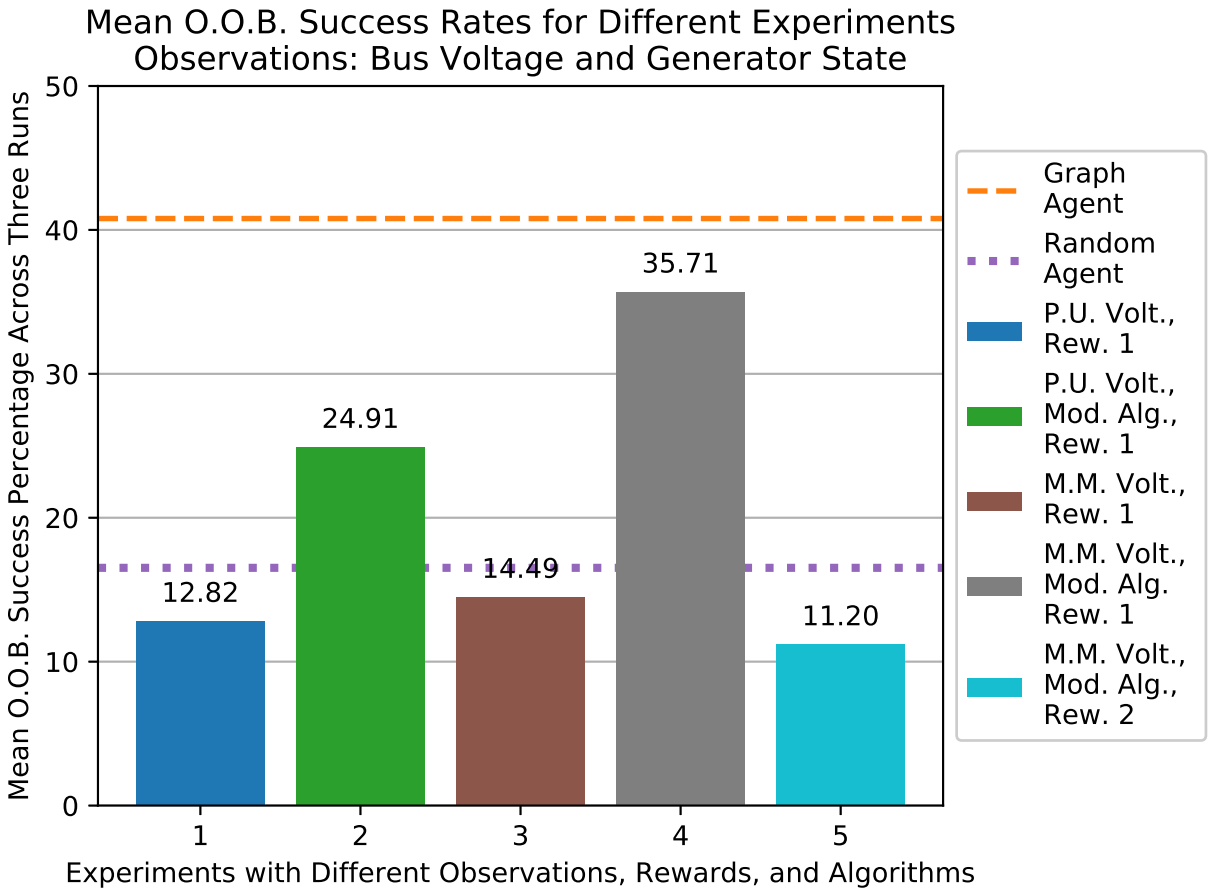


Figure 4.14: Mean out-of-band (O.O.B.) success rates for different experiments: bus voltage and generator state observations

4.6.4 Bus Voltage Observations and Branch State Observations

The figures presented in this section illustrate training and testing results with the agent's observations contain both bus per unit voltage observations and branch state (open/closed) observations. Training and testing results can be found in in Figures 4.15, 4.16, 4.17, 4.18, and 4.19.

4.6.4.1 Per Unit Voltage Observations with Reward Scheme 1

(a) Training: average rewards

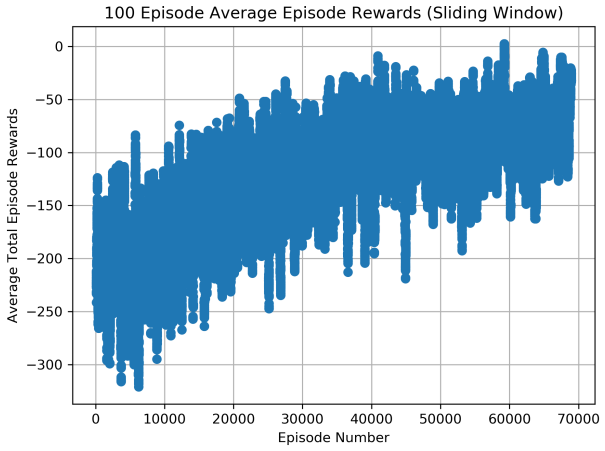
(b) Training: number of actions taken

(c) Testing: reward histogram

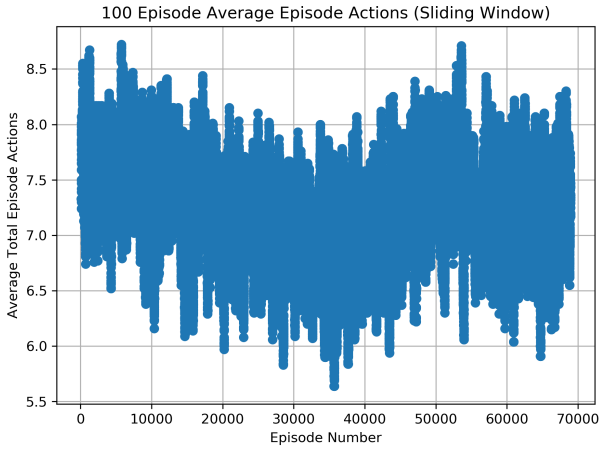
(d) Testing: action count distribution

Figure 4.15: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

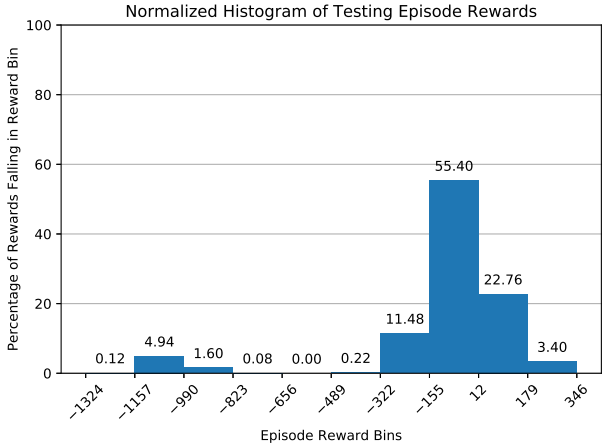
4.6.4.2 *Per Unit Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



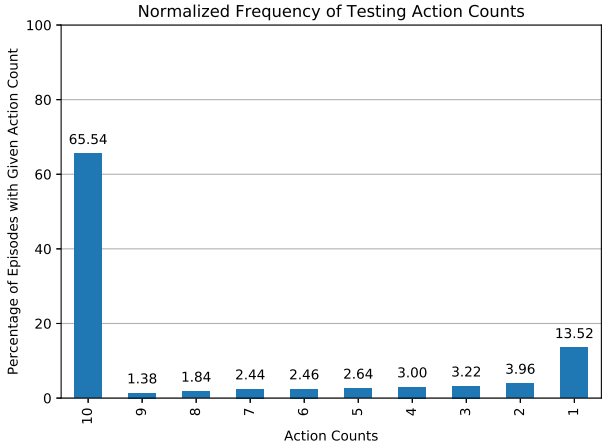
(a) Training: average rewards



(b) Training: number of actions taken



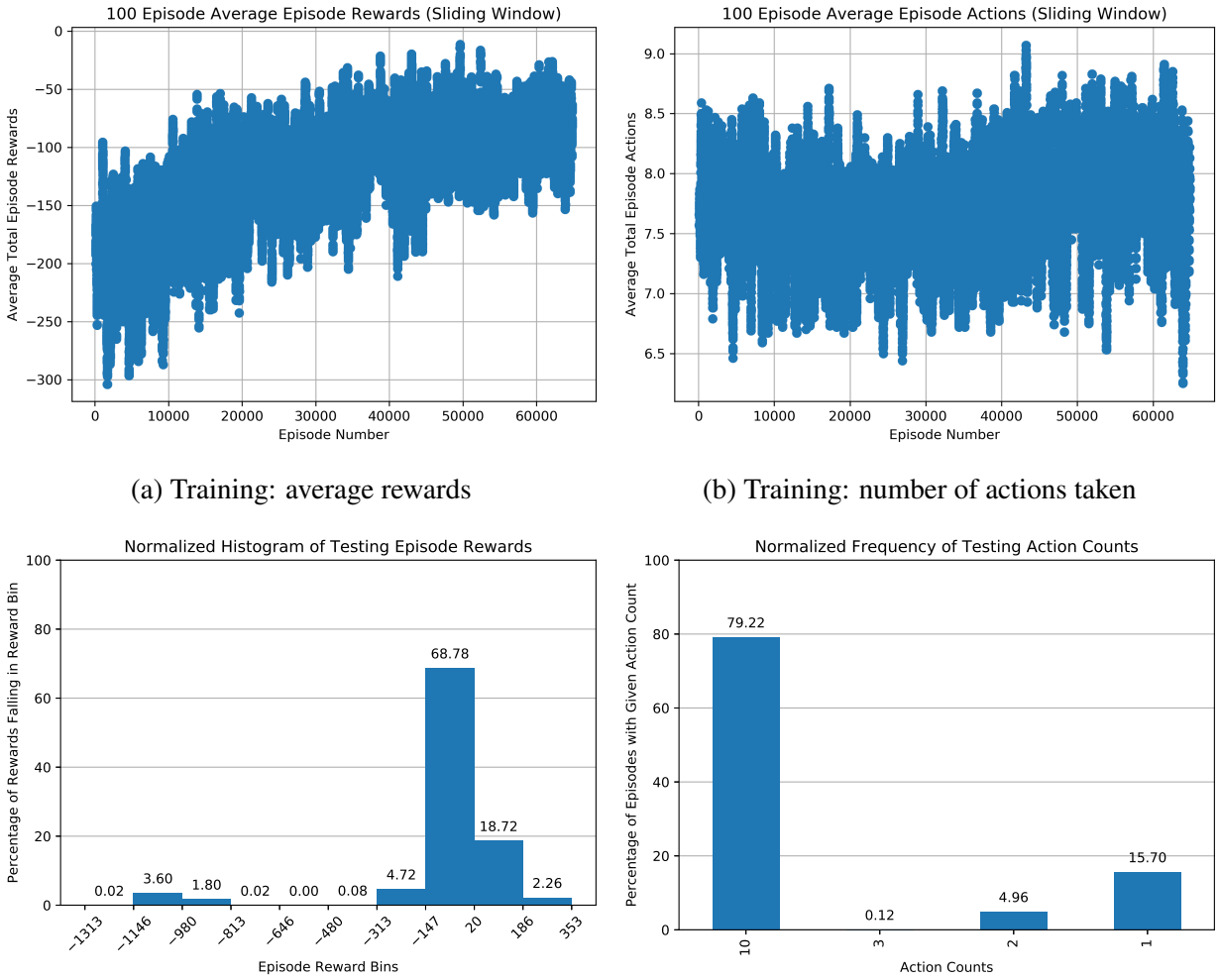
(c) Testing: reward histogram



(d) Testing: action count distribution

Figure 4.16: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.4.3 *Min-Max Scaled Voltage Observations with Reward Scheme 1*



(a) Training: average rewards

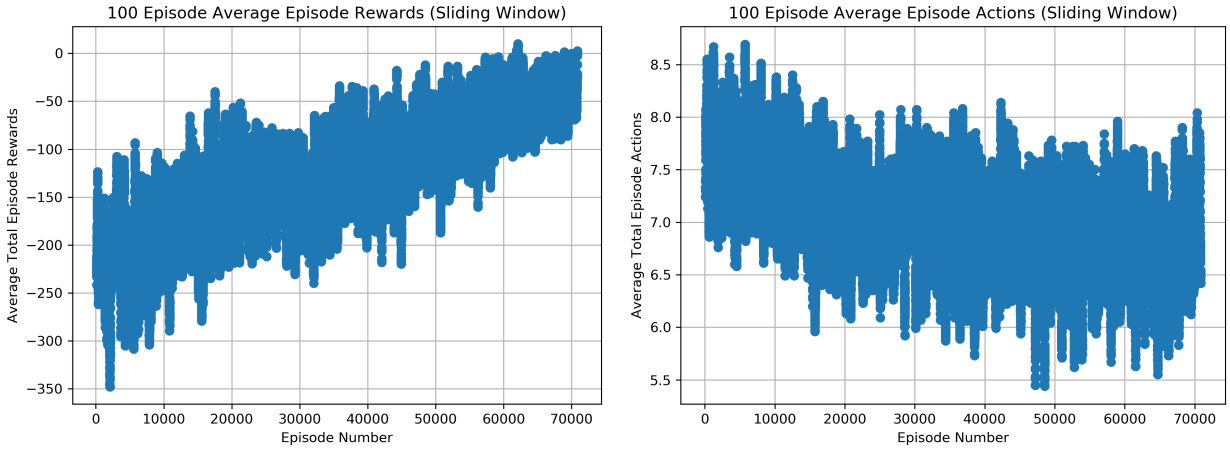
(b) Training: number of actions taken

(c) Testing: reward histogram

(d) Testing: action count distribution

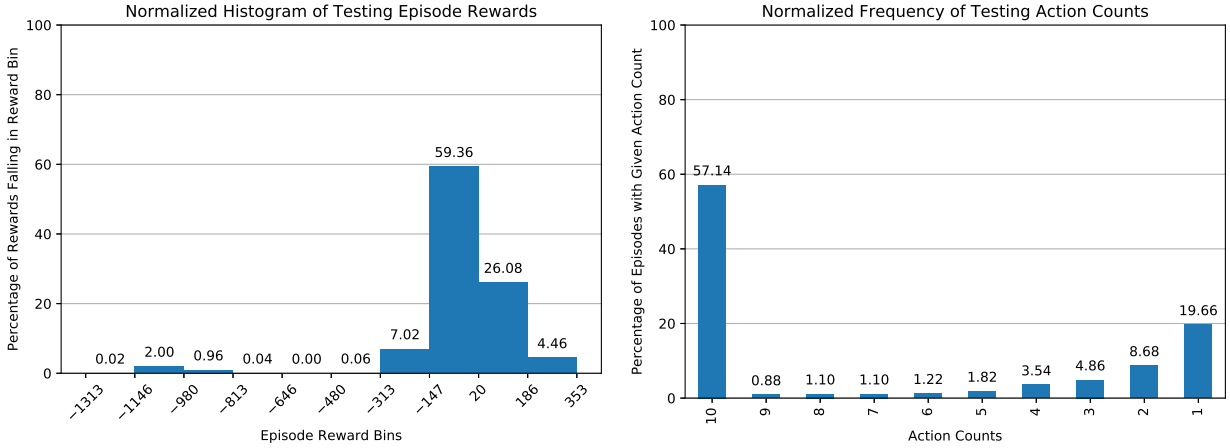
Figure 4.17: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

4.6.4.4 *Min-Max Scaled Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken

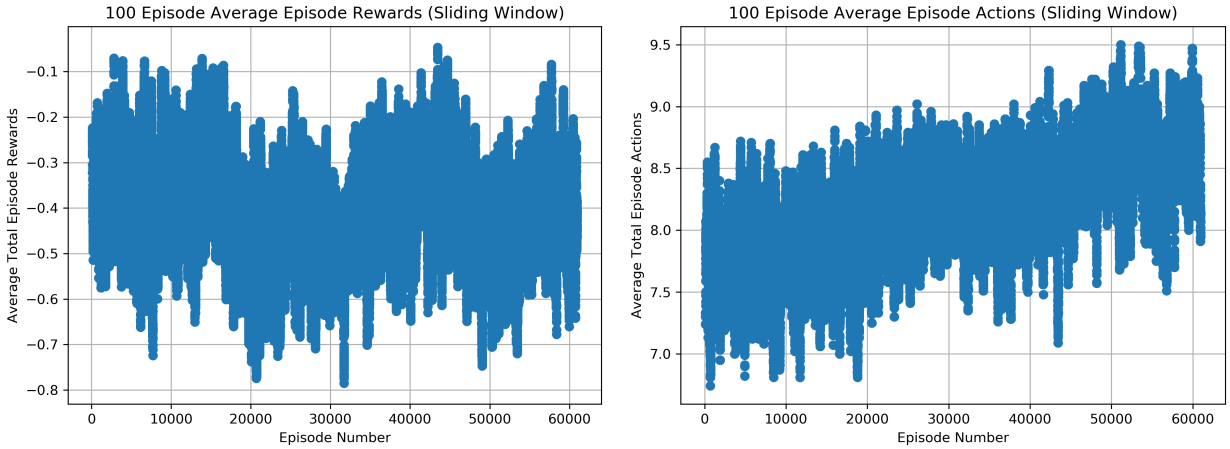


(c) Testing: reward histogram

(d) Testing: action count distribution

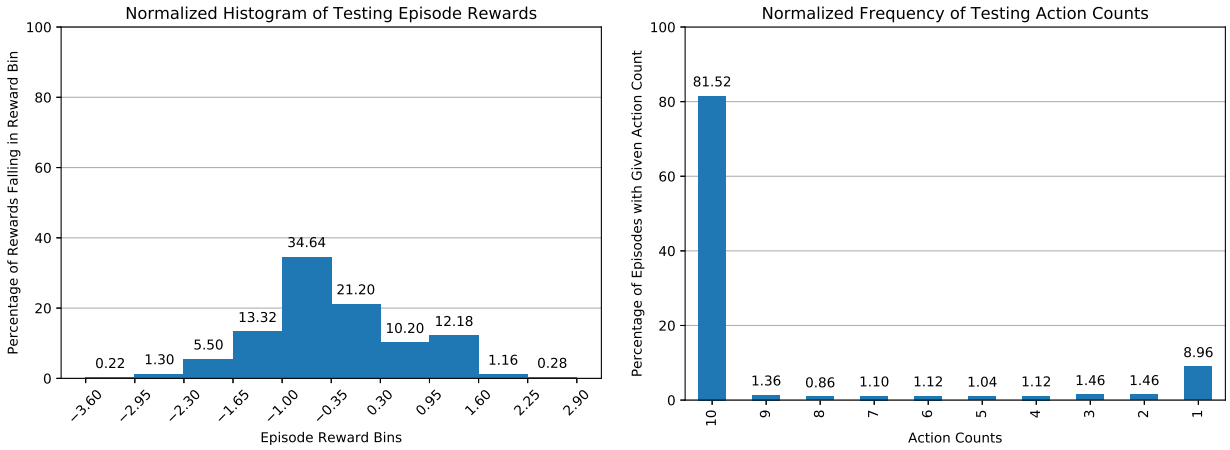
Figure 4.18: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.4.5 *Min-Max Scaled Voltage Observations with Reward Scheme 2 and Unique-Actions-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken



(c) Testing: reward histogram

(d) Testing: action count distribution

Figure 4.19: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 2, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.4.6 *Comparison and Discussion*

When branch states are included, four of the five experiments show an upward sloped "learning curve" as can be seen in Figures 4.15a, 4.16a, 4.17a, and 4.18. The only experiment with a learning curve without a clear upward slope used reward scheme 2, and can be seen in Figure 4.19. Figure 4.20 illustrates mean out-of-band success rates for the various experiments. As was seen in Sections 4.6.2 and 4.6.3, only two experiments exceed the success rate of the random agent, and both of these experiments use the modified DQN algorithm and reward scheme 1.

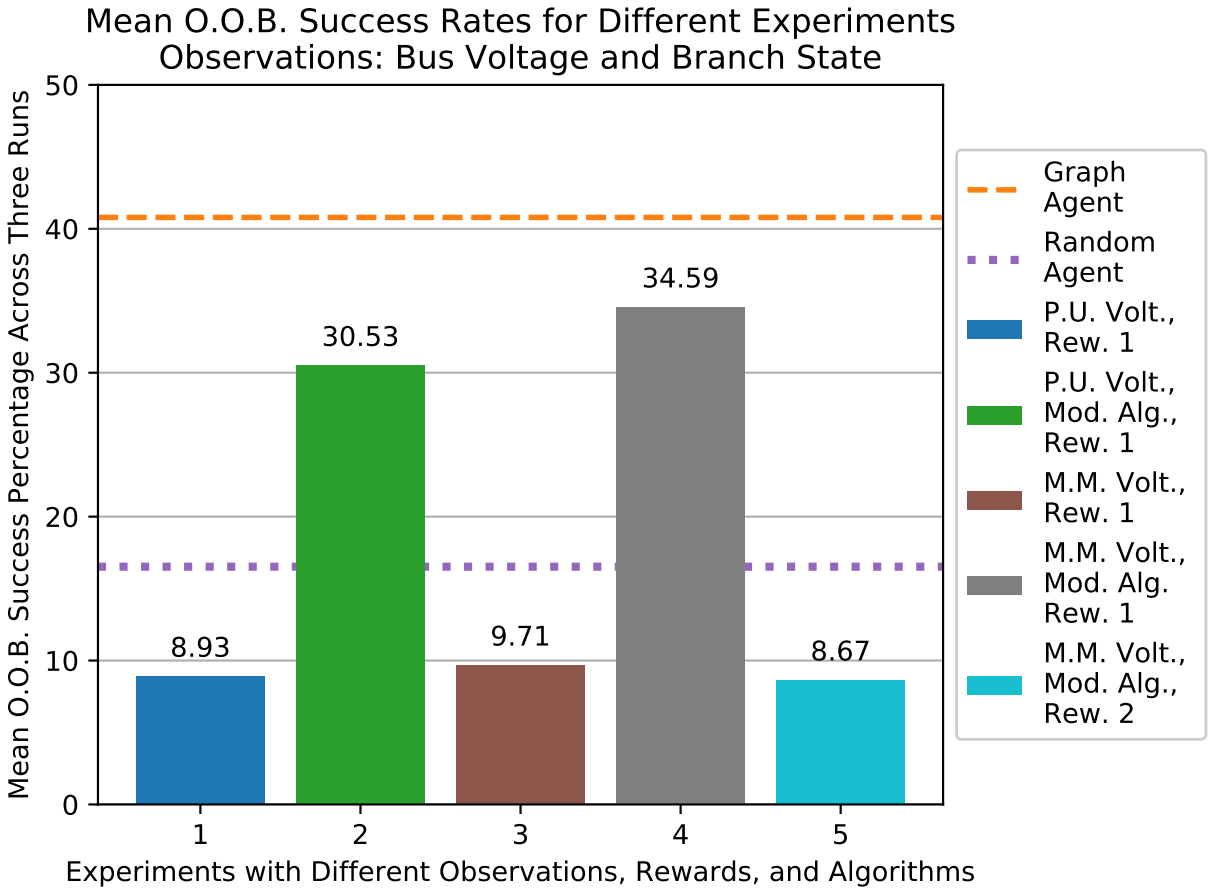
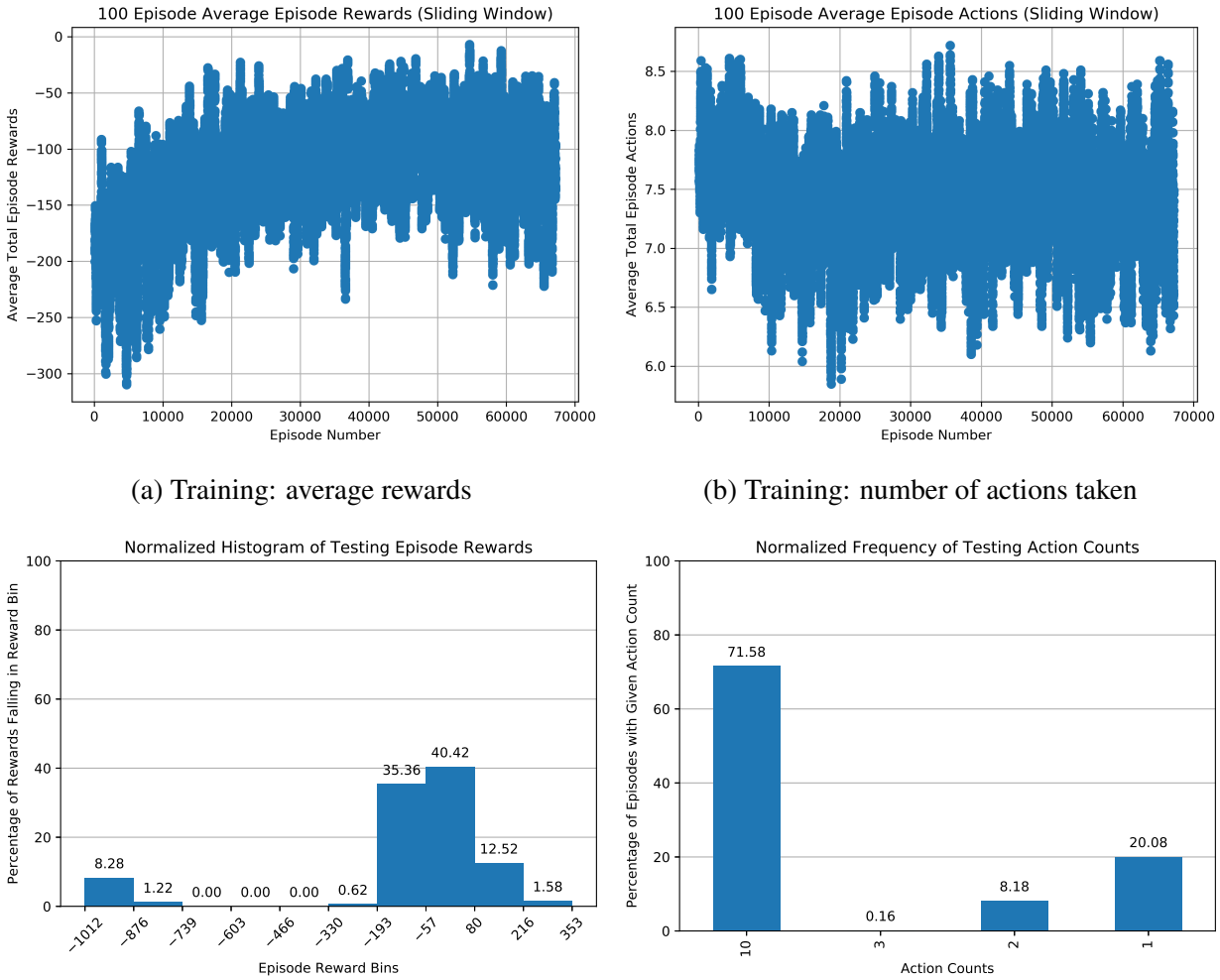


Figure 4.20: Mean out-of-band (O.O.B.) success rates for different experiments: bus voltage and branch state observations

4.6.5 Bus Voltage, Generator State, and Branch State Observations

The figures presented in this section illustrate training and testing results with the agent's observations contain bus per unit voltage observations, branch state (open/closed) and generator state (on/off) observations. Training and testing results can be found in in Figures 4.21, 4.22, 4.23, 4.24, and 4.25.

4.6.5.1 Per Unit Voltage Observations with Reward Scheme 1



(a) Training: average rewards

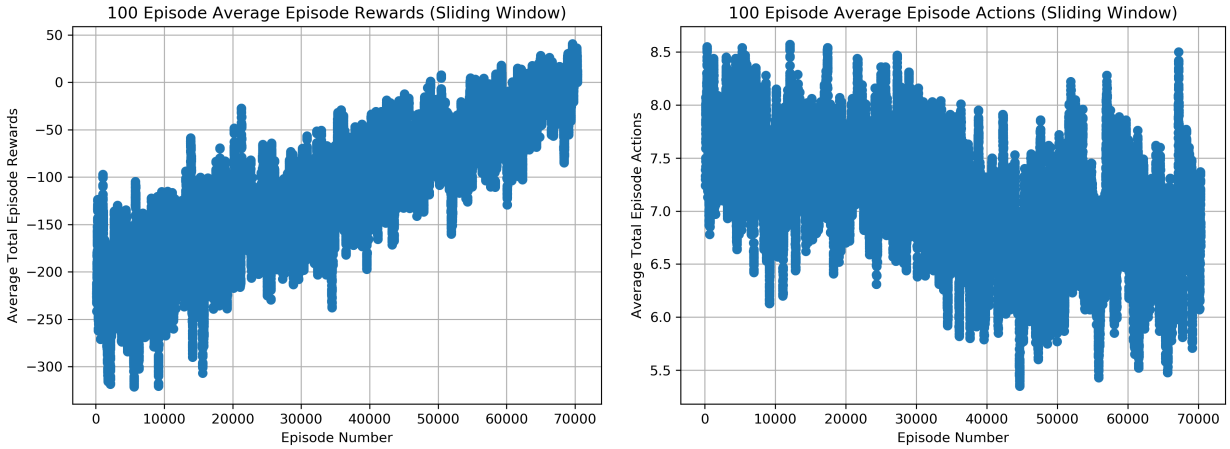
(b) Training: number of actions taken

(c) Testing: reward histogram

(d) Testing: action count distribution

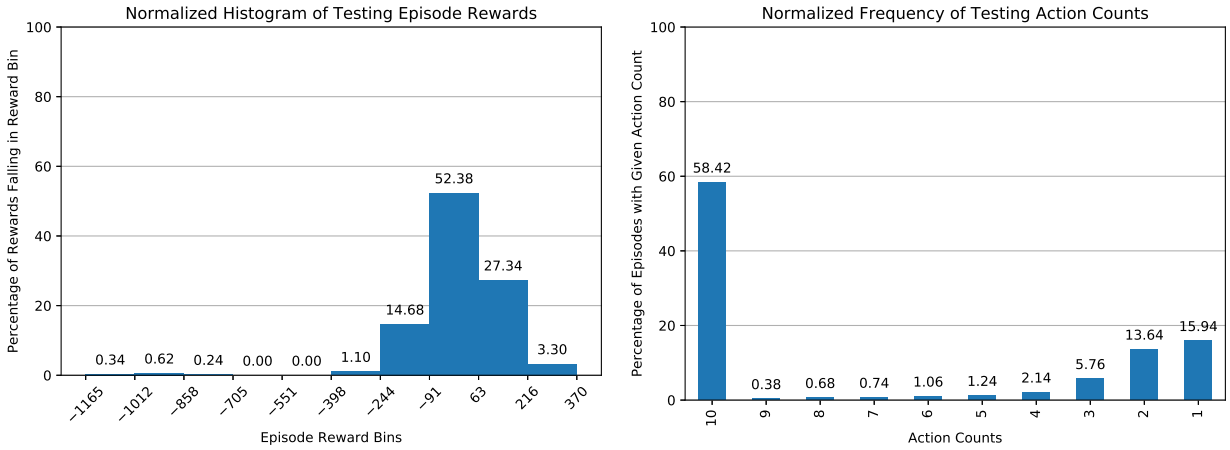
Figure 4.21: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

4.6.5.2 *Per Unit Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken

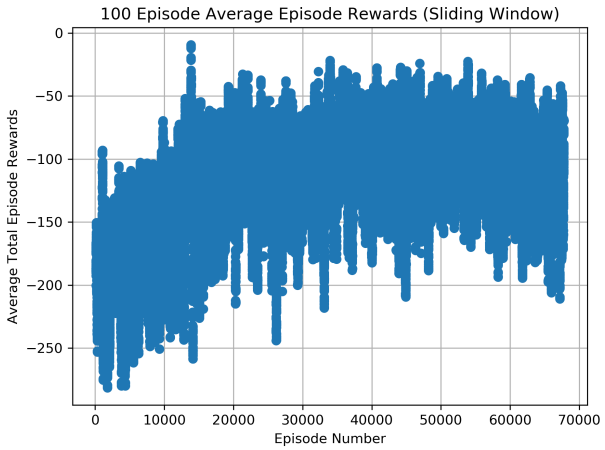


(c) Testing: reward histogram

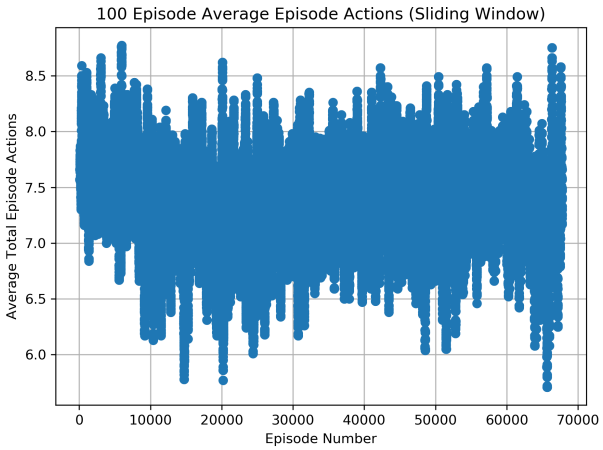
(d) Testing: action count distribution

Figure 4.22: Training and testing rewards and actions: Per unit voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

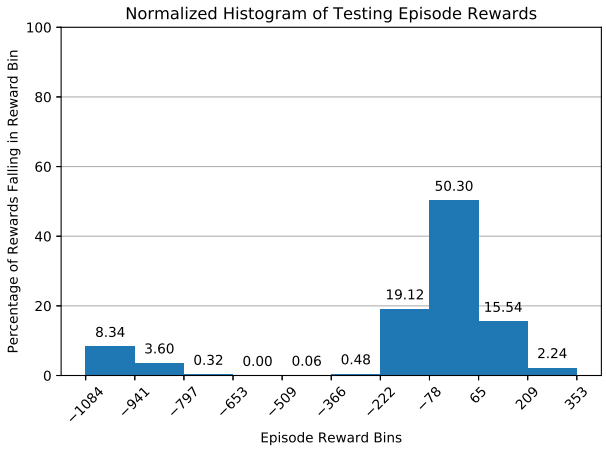
4.6.5.3 *Min-Max Scaled Voltage Observations with Reward Scheme 1*



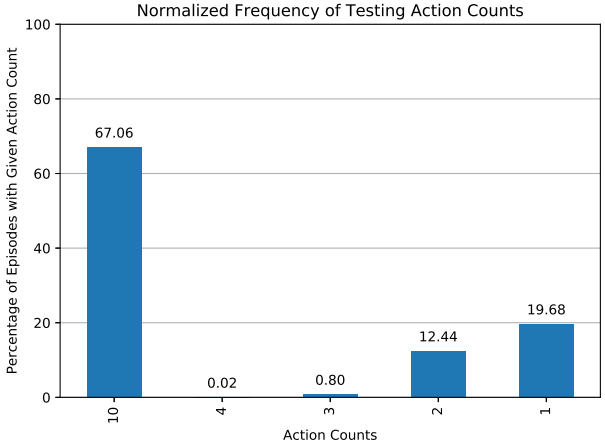
(a) Training: average rewards



(b) Training: number of actions taken



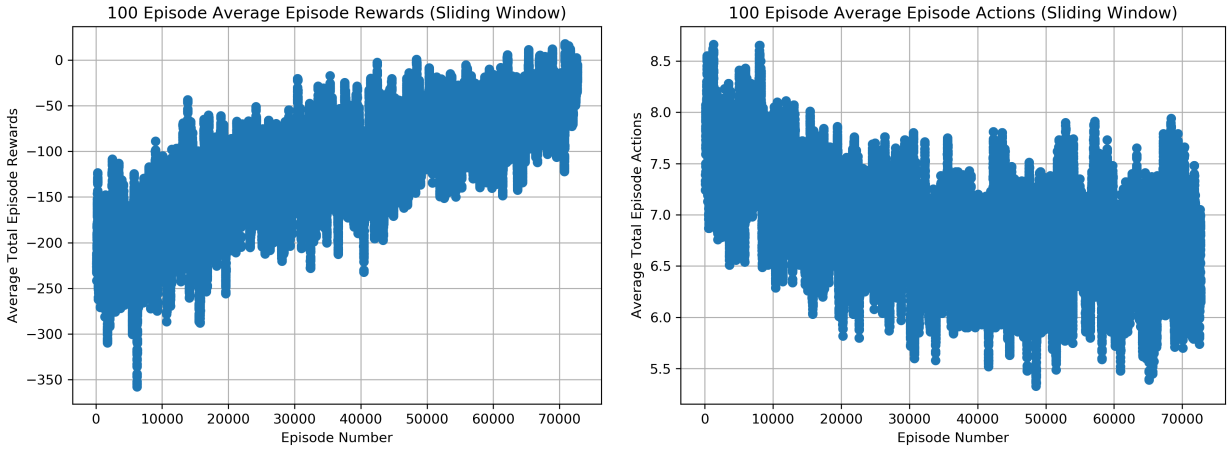
(c) Testing: reward histogram



(d) Testing: action count distribution

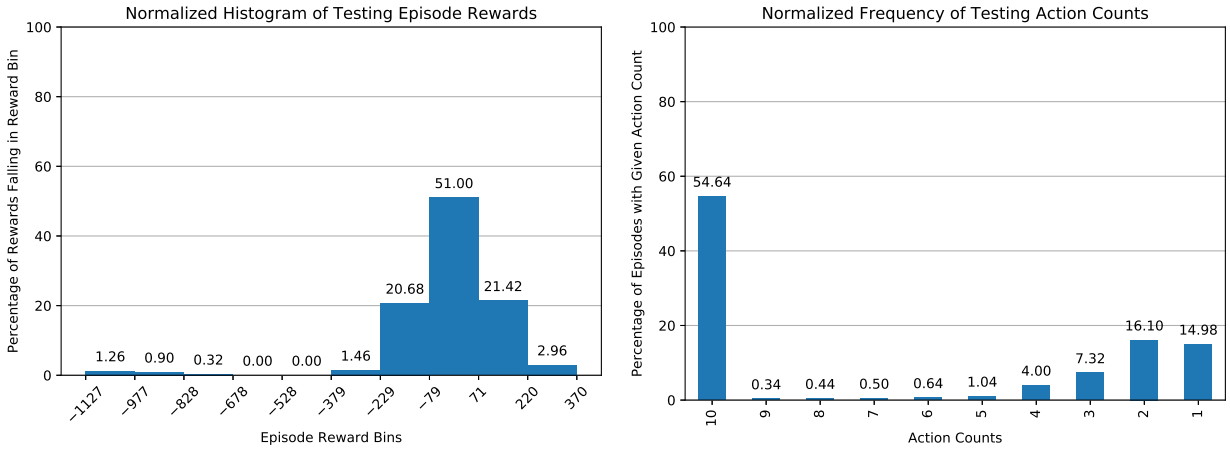
Figure 4.23: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, neural network hidden layers: $[64, 64]$.

4.6.5.4 *Min-Max Scaled Voltage Observations with Reward Scheme 1 and the Unique-Action-Per-Episode Algorithm Modification*



(a) Training: average rewards

(b) Training: number of actions taken

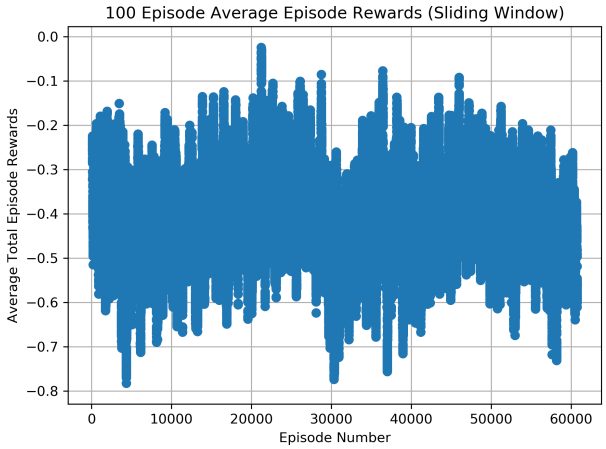


(c) Testing: reward histogram

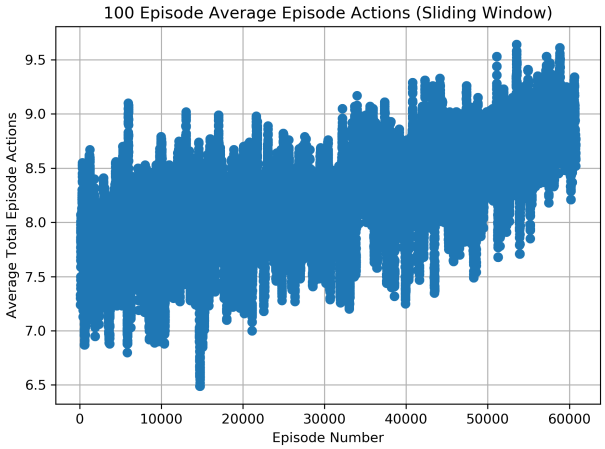
(d) Testing: action count distribution

Figure 4.24: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

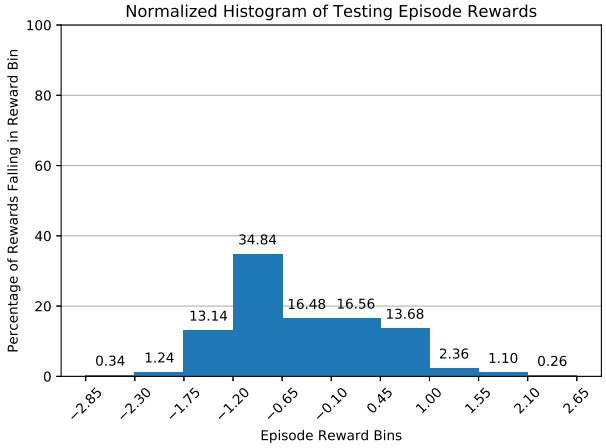
4.6.5.5 *Min-Max Scaled Voltage Observations with Reward Scheme 2 and Unique-Actions-Per-Episode Algorithm Modification*



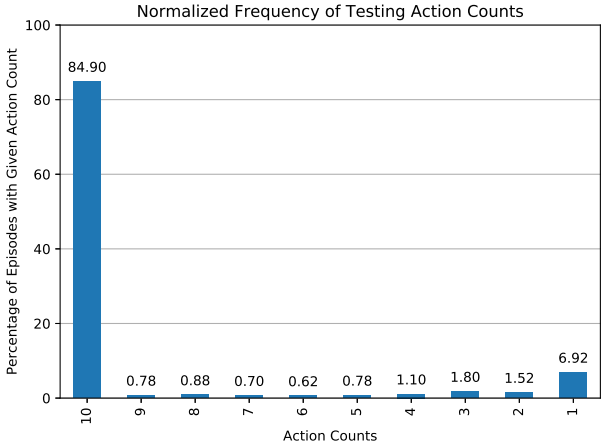
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count distribution

Figure 4.25: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 2, unique-actions-per-episode, neural network hidden layers: $[64, 64]$.

4.6.5.6 Comparison and Discussion

The observation combination in this subsection provides the agent with the most information of all the experiments. Despite this fact, only two experiments show learning curves with consistent improvement, as can be seen in Figures 4.22a and 4.24a. Figure 4.26 illustrates mean out-of-band success rates for the various experiments. This is the only observation combination in which the per unit voltage observations with the modified algorithm and reward scheme 1 was more successful than the min-max scaled voltage observations with the modified algorithm and reward scheme 1. However, the difference is less than 0.5%. As has been seen in the previous sections, the only two experiments which beat the random agent are those with the modified algorithm and reward scheme 1.

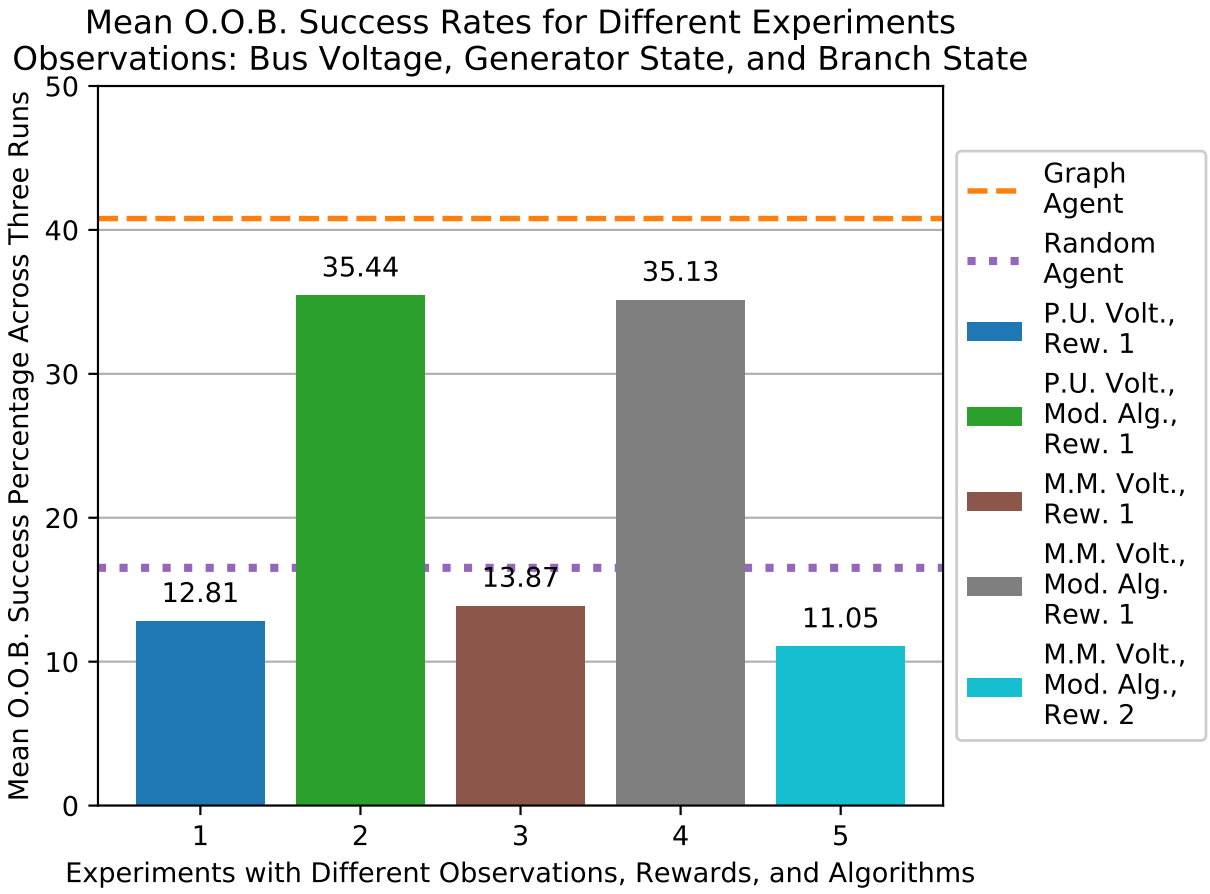


Figure 4.26: Mean out-of-band (O.O.B.) success rates for different experiments: bus voltage, generator state, and branch state observations

4.6.6 Additional Training

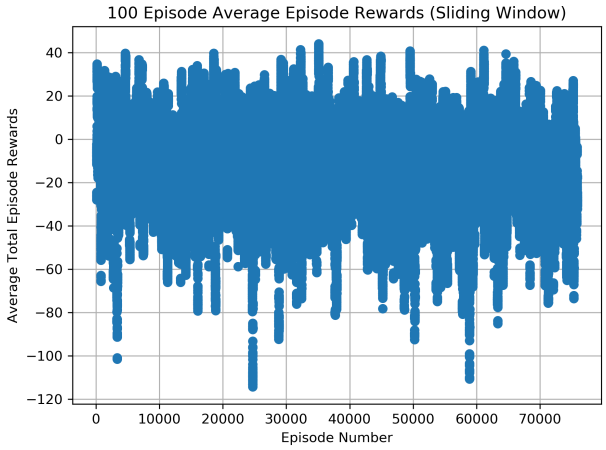
As can be seen in the the previous sections, some agents show a linear learning curve which does not have a "plateau" at the end, indicating additional training may lead to a continued increase in agent success. Some examples of these curves can be seen in Figures 4.6 and 4.24. To test if more training is useful, two agents received additional training. The first agent received only voltage observations with min-max scaling, used reward scheme 1, and used the unique-action-per-episode algorithm modification. This agent's initial results can be found in Section 4.6.2.4. The second agent received min-max voltage observations, generator state observations, and line state observations. The second agent also used reward scheme 1 and the unique-action-per-episode DQN algorithm modification. The second agent's initial results can be found in Section 4.6.5.4.

To perform additional training, the agent's state (including neural network parameters) at the end of initial training was loaded from file using the save feature provided by [41]. Training was performed for another 500,000 time steps. In order to make this additional training simply look like continued training, the following hyper parameter changes were made:

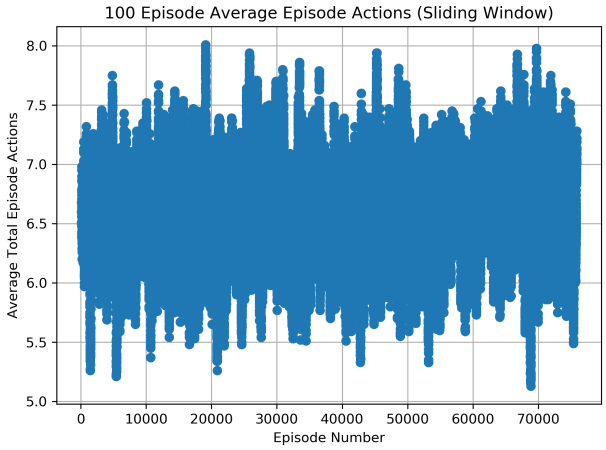
- Exploration fraction fixed at 10%, as done in [18, 19].
- Prioritized replay β fixed at 1.0, as during initial training it annealed from 0.4 to 1.0.
- Learning does not begin until the replay buffer is full (after 50,000 time steps).

Additional training results for these two agents can be found in Figures 4.27 and 4.28. The agent that only received voltage observations had an out-of-band success rate of 37.92% after additional training, which is slightly better than the mean out-of-band-success rate reported in Figure 4.8. However, this success rate is slightly worse than the individual agent's out-of-band success rate after initial training, which was 40.19%. The agent which received voltage observations as well as generator and branch state observations had an out-of-band success rate of 29.43%, which is markedly worse than the mean success rate of 35.13% reported in Figure 4.26. This new success rate is also worse than the individual agent's initial O.O.B. success rate of 37.19%.

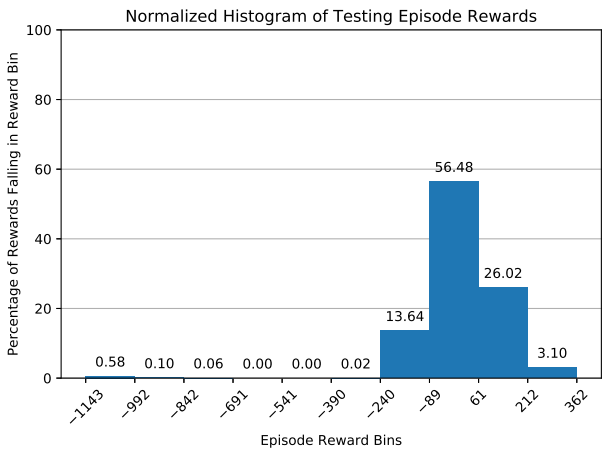
In these two cases, additional training did not seem to improve performance.



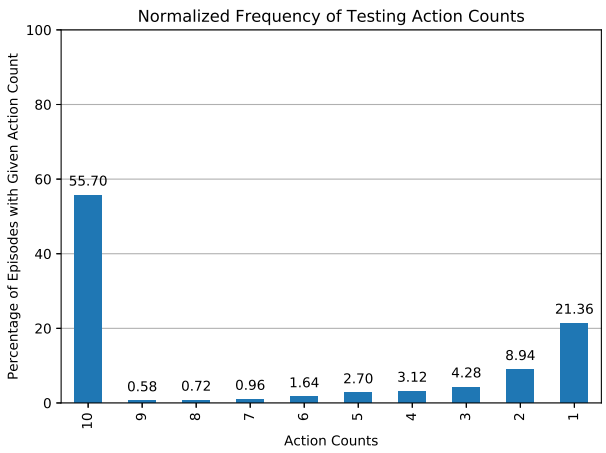
(a) Training: average rewards



(b) Training: number of actions taken

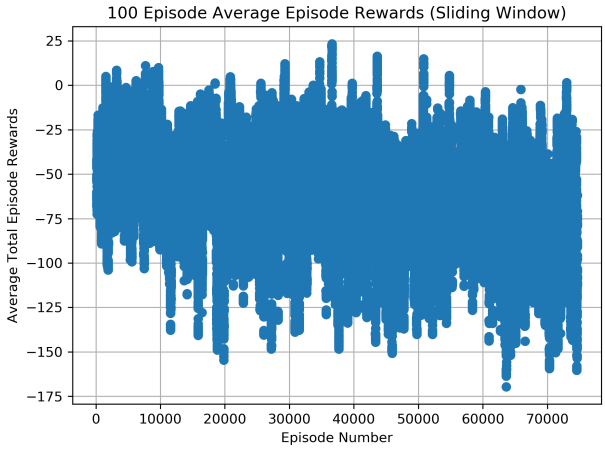


(c) Testing: reward histogram

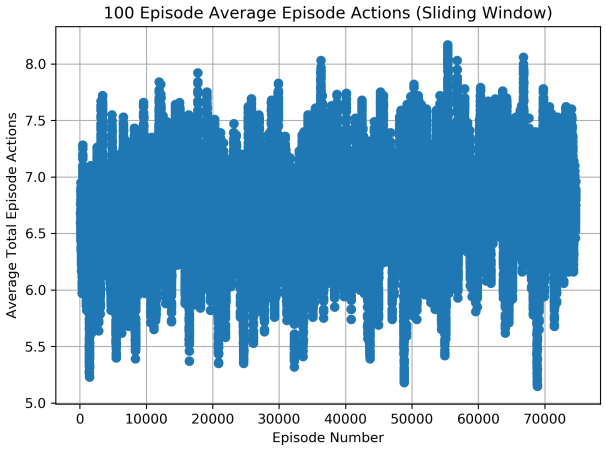


(d) Testing: action count distribution

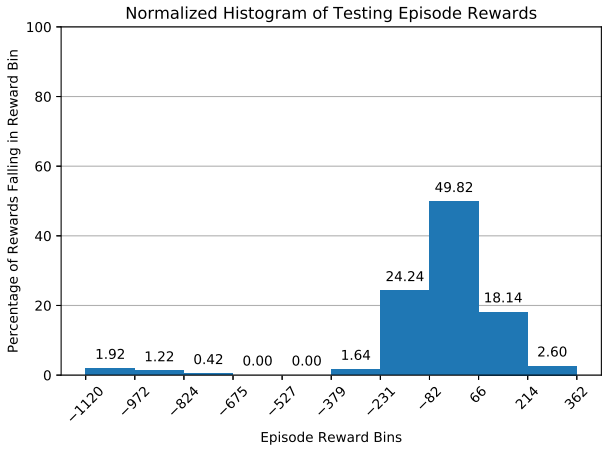
Figure 4.27: Training and testing rewards and actions: Min-max scaled voltage observations only, reward scheme 1, unique-actions-per-episode, additional training, neural network hidden layers: $[64, 64]$.



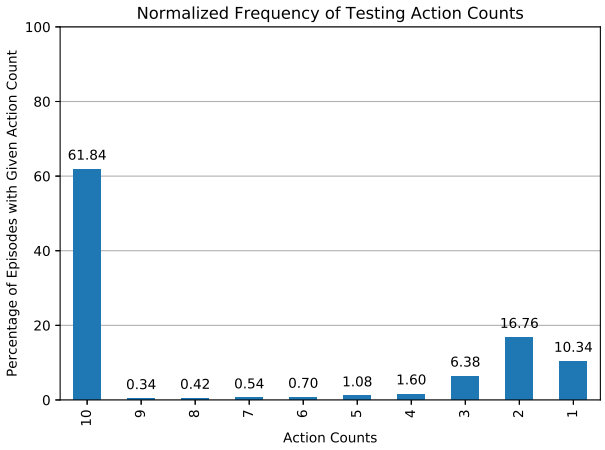
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count distribution

Figure 4.28: Training and testing rewards and actions: Min-max scaled voltage observations, generator and branch state observations, reward scheme 1, unique-actions-per-episode, additional training, neural network hidden layers: $[64, 64]$.

4.7 Discussion and Conclusions

Several key takeaways can be gathered from the results presented in Section 4.6. First, the min-max scaling of voltage observations leads to a higher success rate than raw per unit voltage observations in all but one case. In that one case, the difference was negligible (less than 0.5% difference in success rate). Second, reward scheme 1 consistently outperforms reward scheme 2. However, it's worth noting that reward scheme 2 was only tested with min-max scaled voltages and the modified DQN algorithm which enforces unique actions per-episode. This combination had the highest success rate when reward scheme 1 was used. Third, agents which used the unmodified DQN algorithm never once outperformed the random agent. By contrast, agents which used the modified DQN algorithm and reward scheme 1 always outperformed the random agent, and often approached the success rate of the graph-based agent.

With the exception of the agent which received bus voltage, generator state, and branch state observations, the combination of min-max scaled voltages, modified DQN algorithm, and reward scheme 1 consistently led to the highest out-of-band success rate. As this combination is the most successful, it will be used in experiments with larger electric grids as will be presented in Chapter 5.

As can be seen in several figures in Section 4.6, such as Figure 4.6a, it's quite possible that training was terminated early. One might expect the slope of the rewards over episodes curve to begin to decay once the agent has been fully trained. However, none of the "learning curves" in Section 4.6 exhibit this decaying slope at the end of training. This is likely in part due to the fact that training halted once the exploration rate was annealed to 1%. It's possible that more training could lead to higher success rates and average rewards.

Table 4.2 presents tabulated results for all experiments presented in Section 4.6. Interestingly, the highest out-of-band success rate (36.22%) comes from an agent which only received min-max voltage observations. This agent received min-max scaled voltages and used the unique-actions-per-episode algorithm modification. However, agents which received generator state, branch state, and both generator and branch states were not far behind with respective out-of-band success rates

of 35.71%, 34.59%, and 35.13%. While in the case of the 14 bus system the agent with the least observations performed the best, it is hypothesized that this is a direct result of the simplicity of the 14 bus system. Since the success rates are not wildly different for the agents which received more observations, the full set of observations (bus voltage, generator states, and branch states) will be used when testing with larger systems in Chapter 5.

Table 4.2: Percent success and mean rewards for all 14 bus experiments

Observations	**Unique Actions per Episode?**	**Min-Max Voltages?**	**Reward Scheme**	**Pct. Success**	**Pct. Success, O.O.B.**	**Mean Reward**
Voltage Only	No	No	1	13.44	6.02	-52.67
	Yes	No	1	38.48	31.8	-50.94
	No	Yes	1	19.42	11.61	-84.7
	Yes	Yes	1	42.75	36.22	-7.98
	Yes	Yes	2	14.76	9.64	-0.37
Voltage and Gen. State	No	No	1	19.39	12.82	-94.51
	Yes	No	1	31.69	24.91	-12.59
	No	Yes	1	21.03	14.49	-111.1
	Yes	Yes	1	41.61	35.71	-12.39
	Yes	Yes	2	16.39	11.2	-0.26
Voltage and Branch State	No	No	1	16.93	8.93	-46.04
	Yes	No	1	37.37	30.53	-41.53
	No	Yes	1	17.29	9.71	-86.49
	Yes	Yes	1	41.29	34.59	-29.77
	Yes	Yes	2	14.61	8.67	-0.39
Voltage, Gen. State, and Branch State	No	No	1	19.45	12.81	-80.99
	Yes	No	1	40.83	35.44	6.8
	No	Yes	1	20.08	13.87	-116.79
	Yes	Yes	1	40.9	35.13	-18.16
	Yes	Yes	2	16.01	11.05	-0.33

5. SCALING TO LARGER ELECTRIC GRIDS

In this chapter, the lessons learned from Chapter 4 are leveraged to test the application of deep reinforcement learning to larger power systems.

5.1 Overview

Synthetic grid test cases with 200 and 500 buses presented in [32] and available at [33] are leveraged in order to test the scalability of DRL for power system voltage control. These synthetic grids have been constructed to be representative of the real electric grid, but are intentionally different in order to avoid data sensitivity issues.

Some figures and tables use shorthand terminology which is described in Section 4.6.1.

The following subsections will provide a brief overview of the synthetic test cases, environment initialization, observation spaces, action spaces, reward design, modifications to the graph-based agent, and training and testing procedures.

5.1.1 200 Bus Case

A one line diagram for the 200 bus test case is illustrated in Figure 5.1. This synthetic grid is geographically located in Central Illinois. It contains 200 buses, 49 generators, four switched shunts, and 180 transmission lines.

5.1.2 500 Bus Case

A one line diagram for the 500 bus test case is illustrated in Figure 5.2. This synthetic grid is geographically located in a portion of South Carolina. It contains 500 buses, 90 generators, 17 switched shunts, and 468 transmission lines.

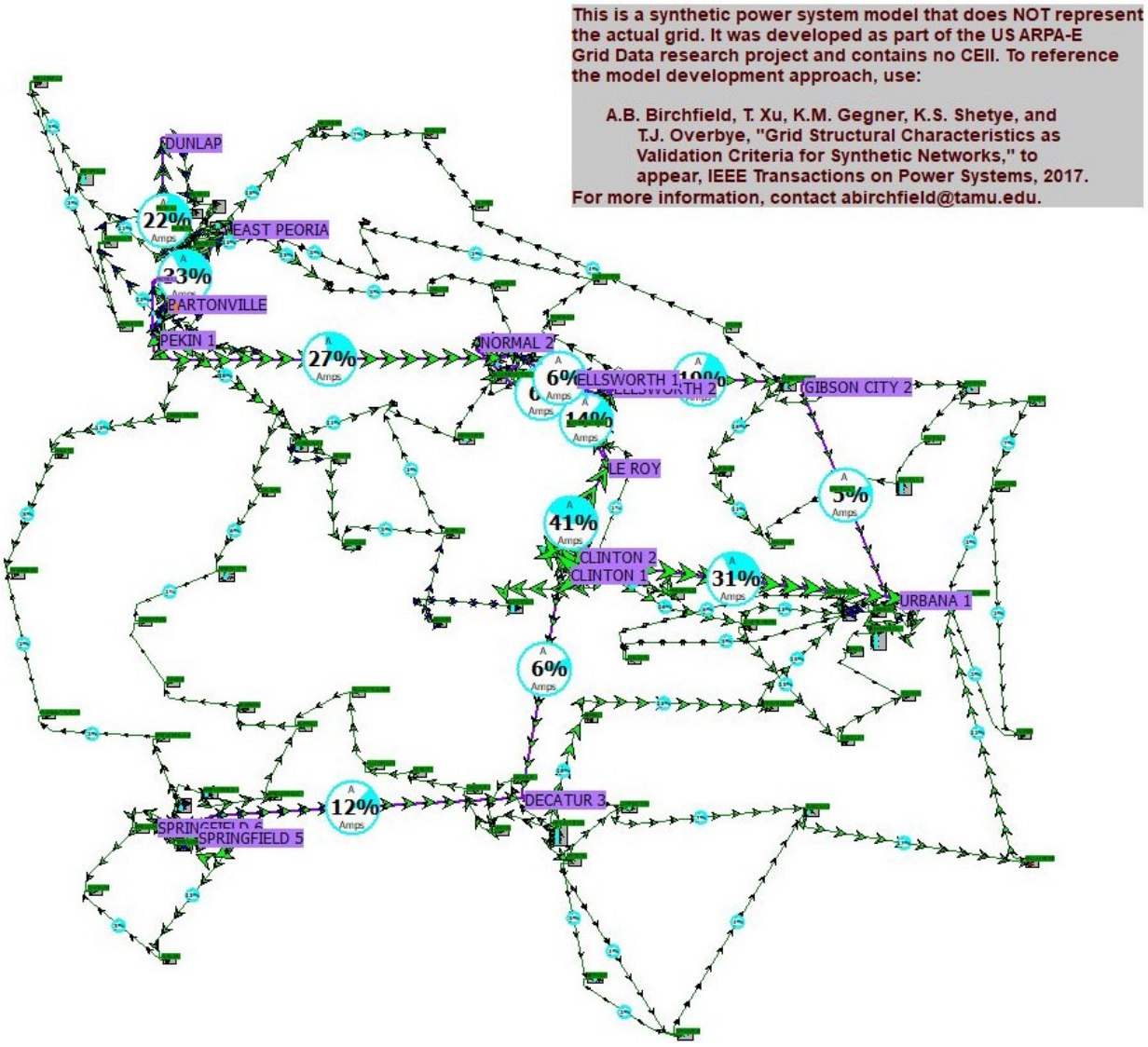


Figure 5.1: Synthetic 200 bus test case

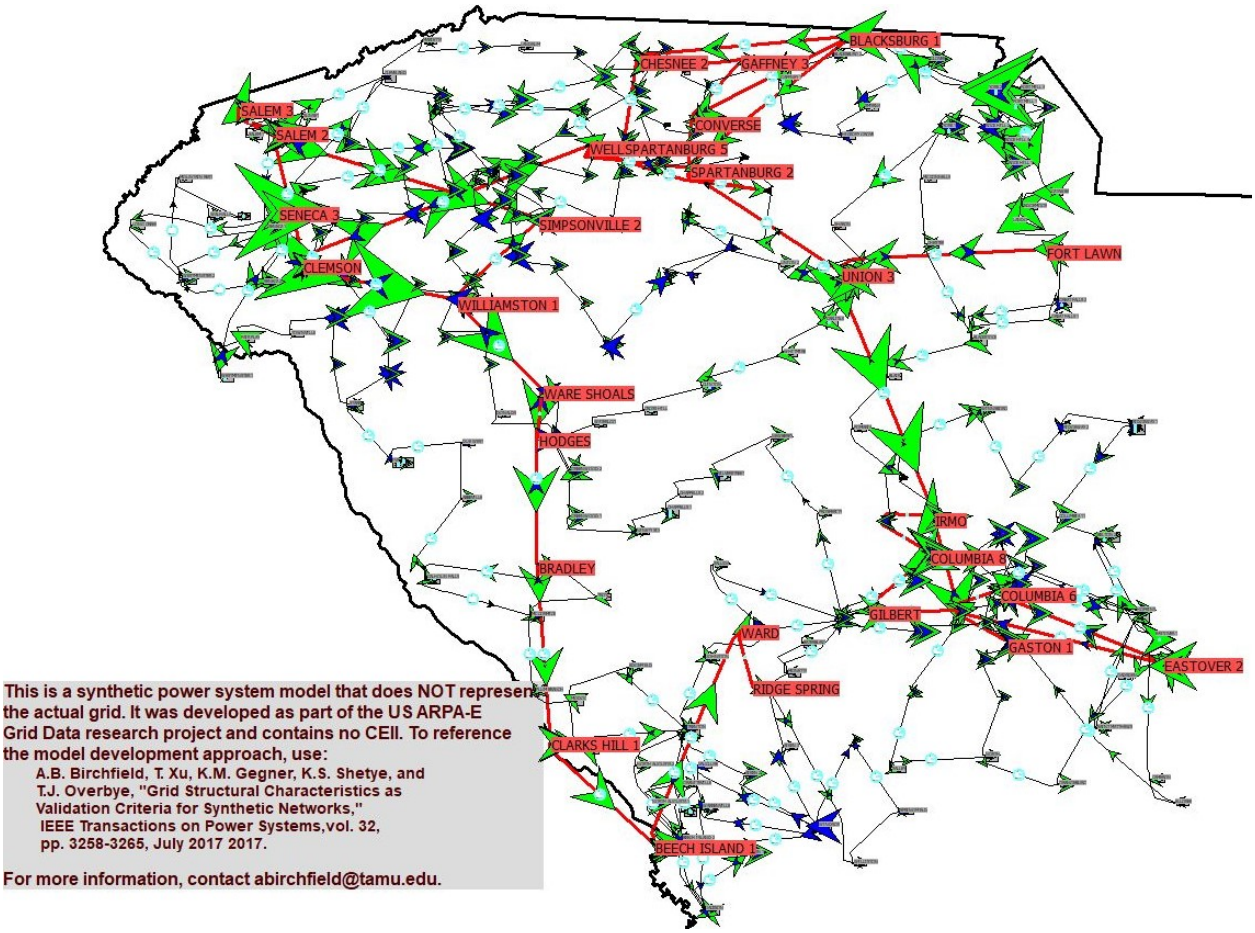


Figure 5.2: Synthetic 500 bus test case

5.1.3 Environment Initialization

The training and testing episodes for all experiments are initialized using the procedure described in Sections 2.3.3 and 3.3. Initialization parameters are the same as provided in Table 3.2 except that when contingencies are applied any line in the system may be opened and all generators are available for active power dispatch.

5.1.4 Observation Spaces

In all experiments, min-max scaled voltages (see Section 4.2), generator states (open/closed), and shunt states (open/closed) are included. Each system is tested both with and without single line contingencies. If contingencies are included, the agent is additionally given all line states (open/closed) as observations.

5.1.5 Action Spaces

As done in Chapters 3 and 4, generator voltage set points are discretized into the set $\{0.95, 0.975, 1.00, 1.025, 1.05\}$ per unit, and an action is available for each generator for each available voltage set point. As introduced in Section 4.3, a "no-op" action is included which does nothing. Since both the 200 and 500 bus systems contain switched shunts, a toggle action is included in the action space for each shunt in the system. This action switches the shunt from open to closed or from closed to open depending on the previous state of the shunt.

5.1.6 Reward Design

All experiments in this chapter use reward scheme 1 as presented in Section 4.4.1. Reward scheme 2 (see Section 4.4.2) is not used since it was found in Section 4.6 that reward scheme 2 never outperformed reward scheme 1 for the given experiments.

5.1.7 Graph-Based Agent Modification

The graph-based agent introduced in Section 4.5 only changed generator voltage set points since the IEEE 14 bus system does not contain any shunts. To support the 200 and 500 bus cases which do contain shunts, the graph-based agent has been modified. In short, the graph-agent looks

for capacitors at buses which neighbor the bus with the lowest or highest voltage and toggles them if appropriate. If there are no capacitors for which it is appropriate to toggle, the agent controls generators instead (as described in Section 4.5.1). The following steps are performed immediately before Step 4 in Section 4.5.1, and then the algorithm proceeds as normal.

1. Determine the buses with the highest and lowest voltages in the system.
   (a) If the bus with the lowest voltage is below 0.95 per unit, capacitors at neighboring buses (including this bus) will be closed if they are not already.
   (b) Otherwise, if the bus with the highest voltage is above 1.05 per unit, capacitors at neighboring buses (including this bus) will be opened if they are not already.
2. Get a list of all buses which neighbor the bus determined in Step 1 (including itself).
3. Iterate over the buses from Step 2:
   (a) If this bus has a capacitor, it is appropriate to toggle it (see Steps 1a and 1b), and this capacitor has not yet been toggled in this episode, toggle it and exit the graph-agent algorithm.

The full algorithm is performed for each time step, so in the event that capacitor switching does not solve the voltage issues, generator voltage set points will then be considered.

5.1.8 Training and Testing Procedures

As was done in Chapter 4, all experiments are performed in triplicate with different random seeds. The results sections for the different power system cases will specify how many training time steps were carried out. After training, testing is performed on 5,000 episodes which were not present during training.

Any results reported as "mean" are mean results for all 15,000 (5,000 $\times$ 3) testing episodes unless otherwise specified. Results from the random and graph-based agents will be used as a basis for comparison. In all results for the random agent, the random agent is not allowed to

take the same action more than once per episode. The graph-based agent controls both switched shunts as well as generator voltage set points. Note that mean rewards will not be presented for the graph-based agent since episodes were terminated prematurely if the agent selected the no-op action.

5.2 Results: 200 Bus Case

Results for the 200 bus test system are described in this section. In Chapters 3 and 4 all agents were trained over 500,000 time steps and used neural networks with two hidden layers of 64 neurons each. Since the 200 bus system is an order of magnitude larger in all relevant aspects (number of buses, number of generators, number of transmission lines) than the 14 bus system, training is performed over 2,000,000 time steps instead of 500,000, and the two neural network hidden layers contain 1,024 neurons each (not including the doubling that the "dueling DQN" algorithm improvement performs. See Section 2.4.1.2 for more details). Training would have been carried out for even more time steps had time allowed.

The following subsections present results for the random and graph-based agents (see Section 4.5) and DRL agents with and without single line contingencies.

5.2.1 Results, Random and Graph-Based Agents

The following subsections present testing results for the random and graph-based agents. Results for the three individual testing runs as well as the overall mean are reported.

5.2.1.1 No Contingencies

The results presented in this subsection were created by using an environment which does not apply any line contingencies during episode initialization. Tables 5.1 and 5.2 present results for the random and graph-based agents, respectively. It can be seen from these tables that even in the absence of contingencies approximately 99.13% of testing episodes start with out-of-band voltages. The random agent is successful in fixing voltage issues in 2.71% of testing cases that start with out-of-band voltages, while the graph-based agent is successful in 32.95% of cases.

Table 5.1: Success percentages and rewards, 200 bus system without contingencies, random agent

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	3.62	2.82	99.18	-1031.80
1	3.68	2.83	99.10	-1031.88
2	3.32	2.48	99.10	-1054.00
Overall Mean	3.54	2.71	99.13	-1039.23

Table 5.2: Success percentages, 200 bus system without contingencies, graph-based agent

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**
0	33.52	32.97	99.18
1	33.08	32.47	99.10
2	34.02	33.42	99.10
Overall Mean	33.54	32.95	99.13

5.2.1.2 Single Line Contingencies

The results presented in this subsection were created by using an environment which applies single line contingencies during episode initialization. Tables 5.3 and 5.4 present results for the random and graph-based agents, respectively. It can be seen from these tables that in the presence of contingencies approximately 99.31% of testing episodes start with out-of-band voltages. The random agent is successful in fixing voltage issues in 2.58% of testing cases that start with out-of-band voltages, while the graph-based agent is successful in 31.56% of cases.

Table 5.3: Success percentages and rewards, 200 bus system with contingencies, random agent

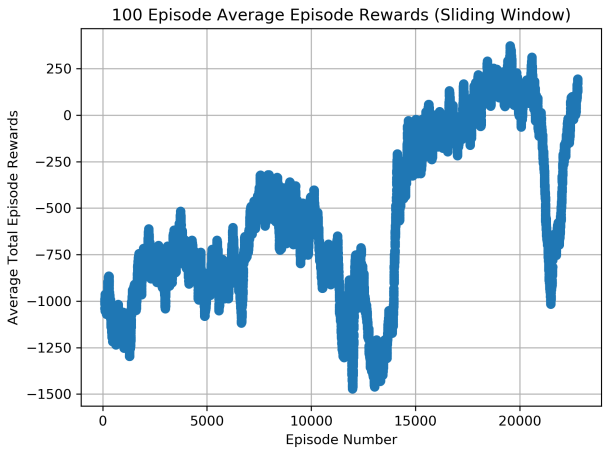
Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	3.76	3.04	99.26	-1033.62
1	2.92	2.31	99.38	-1043.64
2	3.02	2.38	99.30	-1062.13
Overall Mean	3.23	2.58	99.31	-1046.46

Table 5.4: Success percentages, 200 bus system with contingencies, graph-based agent

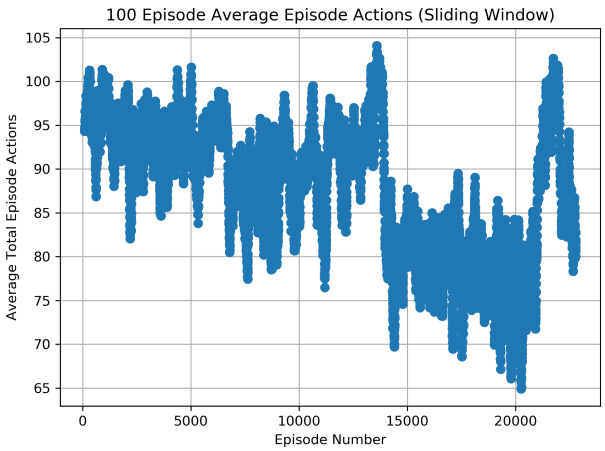
Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**
0	31.84	31.33	99.26
1	31.82	31.39	99.38
2	32.44	31.96	99.30
Overall Mean	32.03	31.56	99.31

5.2.2 Results, DRL Agent, No Contingencies

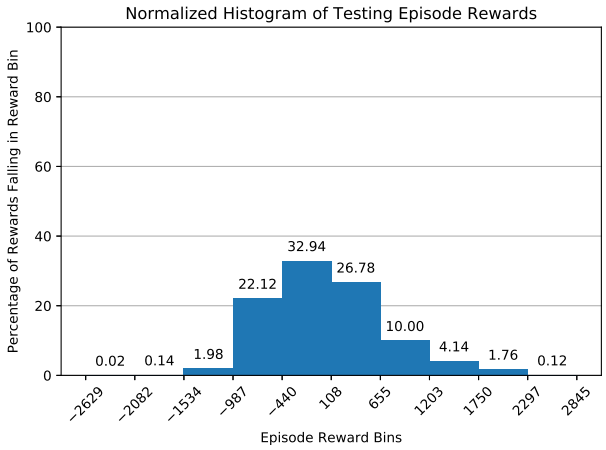
Figures 5.3, 5.4, and 5.5 present training and testing results for the DRL agent when the environment does not include single line contingencies. Since there are no contingencies, the agent does not receive line states as observations as mentioned in Section 5.1.4. Note that unlike figures in previous chapters, the testing action count sub-figure is a histogram rather than the raw distribution. Table 5.5 presents testing success rates and mean rewards for the experiments depicted in Figures 5.3, 5.4, and 5.5. Figure 5.6 presents a comparison of the random, DRL, and graph-based agents.



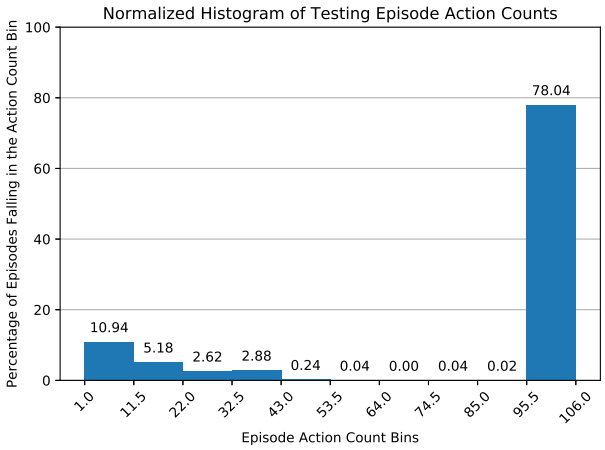
(a) Training: average rewards



(b) Training: number of actions taken

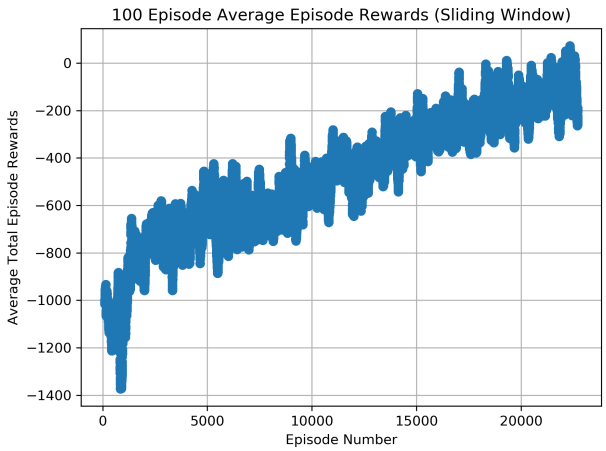


(c) Testing: reward histogram

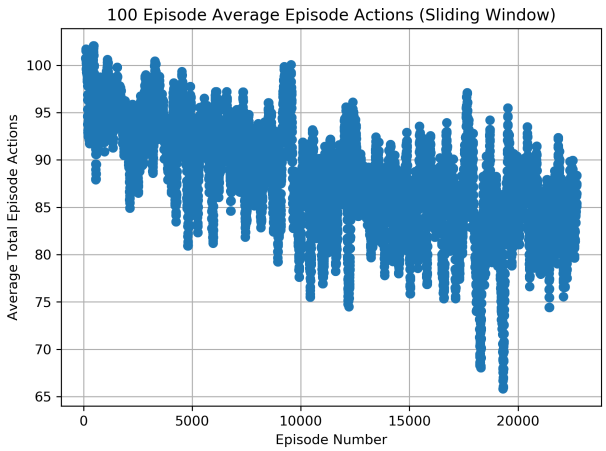


(d) Testing: action count histogram

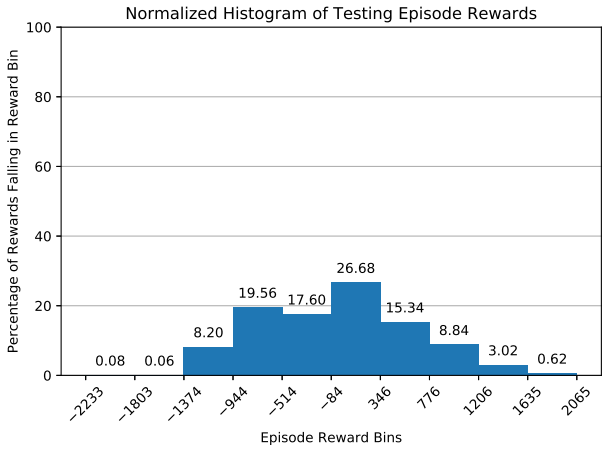
Figure 5.3: Training and testing rewards and actions: 200 bus system without contingencies, random seed: 0, neural network hidden layers: $[1024, 1024]$.



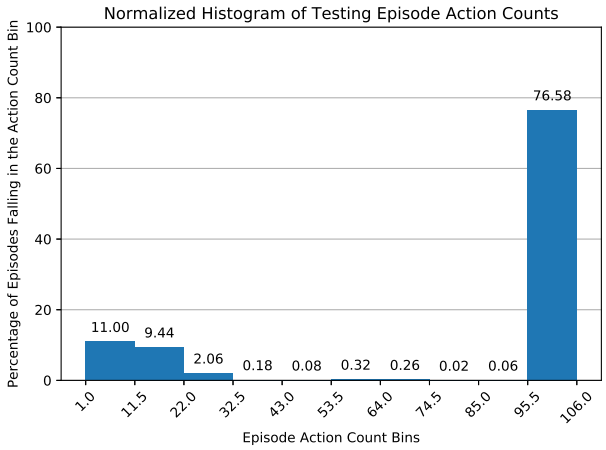
(a) Training: average rewards



(b) Training: number of actions taken

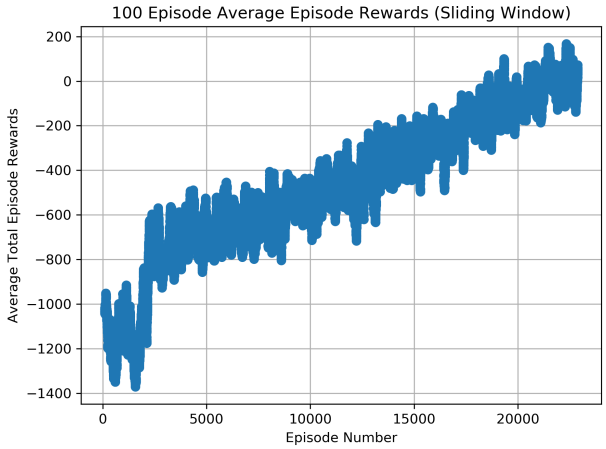


(c) Testing: reward histogram

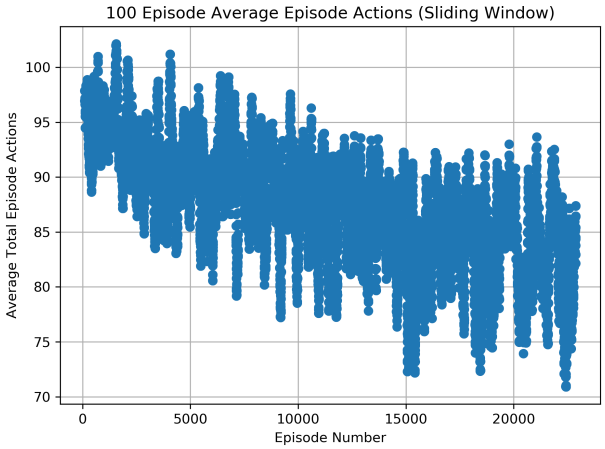


(d) Testing: action count histogram

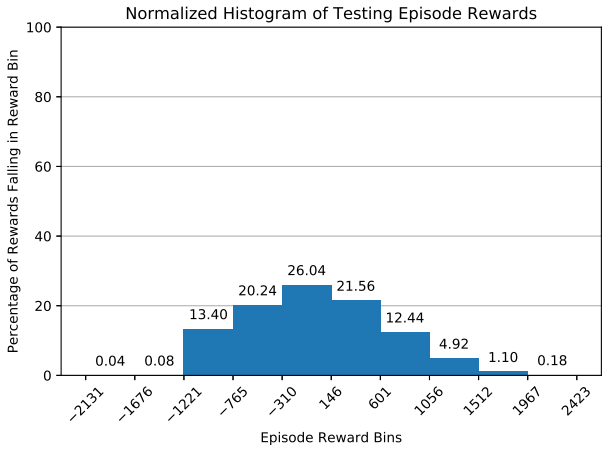
Figure 5.4: Training and testing rewards and actions: 200 bus system without contingencies, random seed: 1, neural network hidden layers: $[1024, 1024]$.



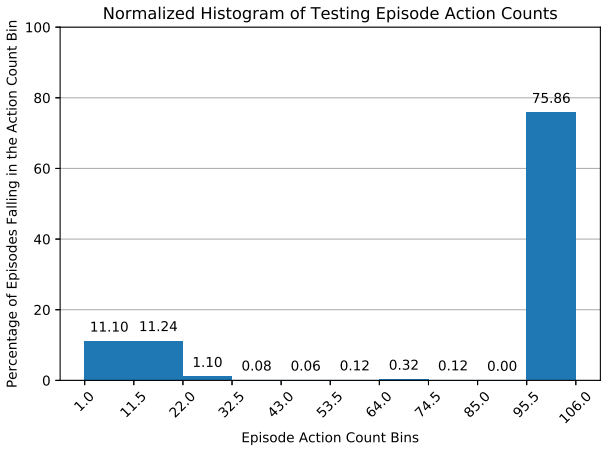
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count histogram

Figure 5.5: Training and testing rewards and actions: 200 bus system without contingencies, random seed: 2, neural network hidden layers: $[1024, 1024]$.

Table 5.5: Success percentages and rewards, 200 bus system without contingencies, DRL agent

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	20.40	19.74	99.18	56.8
1	23.32	22.62	99.10	-29.01
2	24.16	23.47	99.10	10.55
Overall Mean	22.63	21.95	99.13	12.78

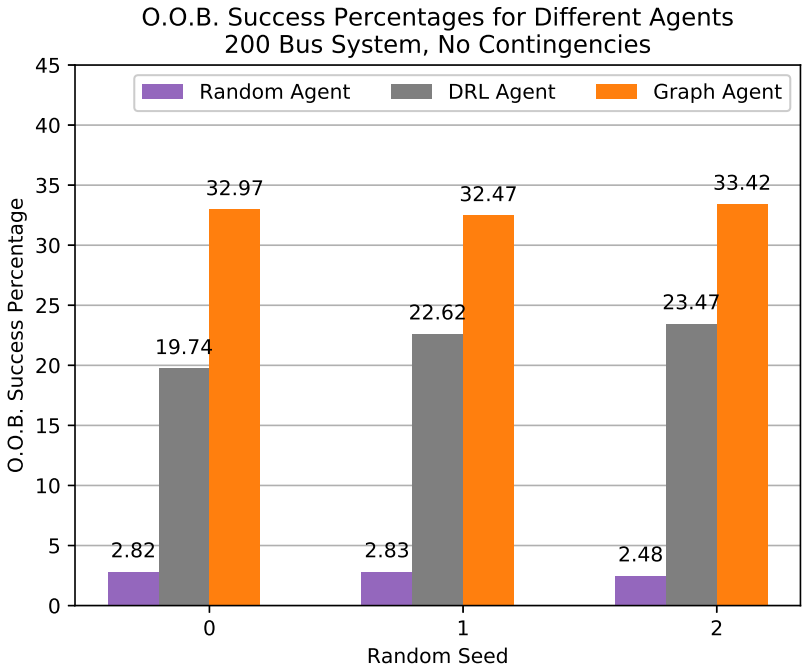


Figure 5.6: Comparison of random, DRL, and graph-based agents, 200 bus system, no contingencies

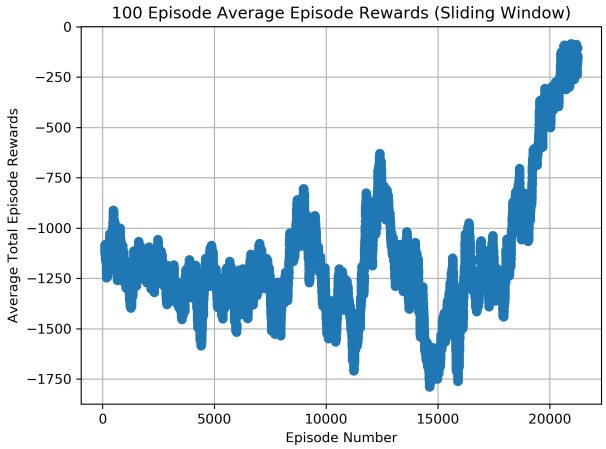
Even in the absence of contingencies, Table 5.5 shows that more than 99% of testing episodes begin with out-of-band voltages. It's interesting to note that the "learning curve" in Figure 5.3a does not exhibit a monotonic learning trend, while Figures 5.4a and 5.5a show learning trends

which appear more linear in nature. As one might expect based on these results, Table 5.5 shows that the success rates for the experiments with random seeds 1 and 2 were more successful in fixing voltage problems than the experiment with a random seed of 0.

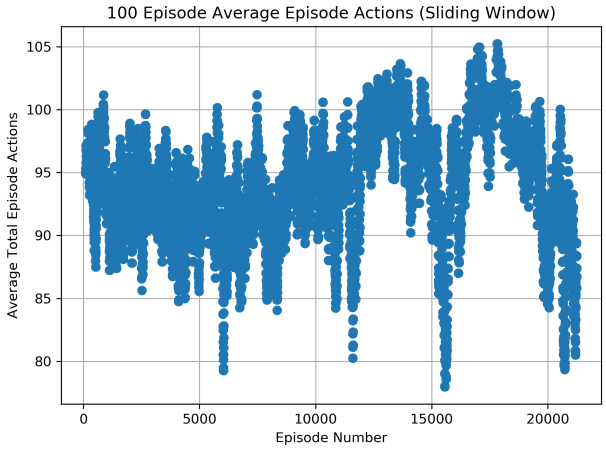
When comparing Table 5.5 with Tables 5.1 and 5.2, it's clear the DRL agent performs significantly better than the random agent. However, the DRL agent's best out-of-band success rate of 23.47% (random seed $= 2$) is significantly worse than the graph-based agent's corresponding success rate of 33.42%.

5.2.3 Results, DRL Agent, With Single Line Contingencies

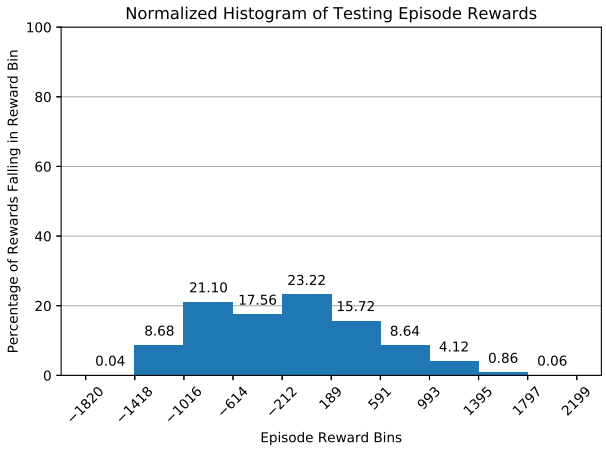
Figures 5.7, 5.8, and 5.9 present training and testing results for the 200 bus system when the environment includes single line contingencies for every episode. As noted in Section 5.1.4, since contingencies are included branch states are included in the observations given to the agent. Table 5.6 presents testing success rates and mean rewards for all three experiments with different random seeds and Figure 5.10 presents a comparison of the random, DRL, and graph-based agents.



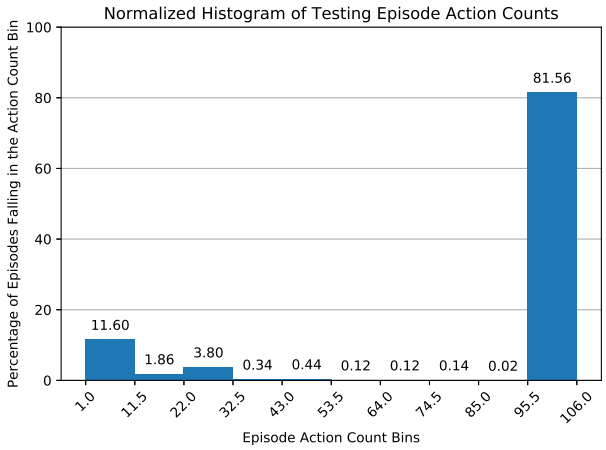
(a) Training: average rewards



(b) Training: number of actions taken

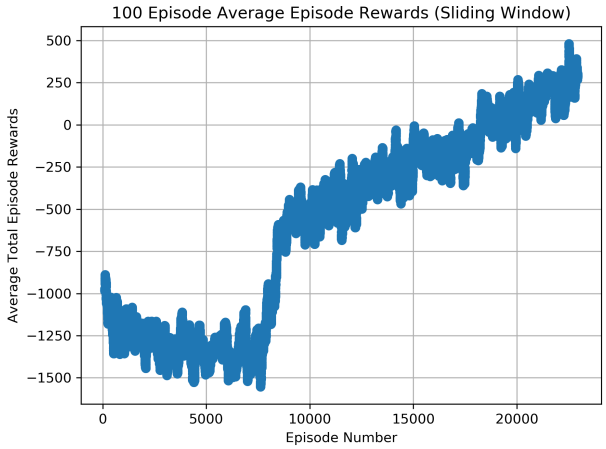


(c) Testing: reward histogram

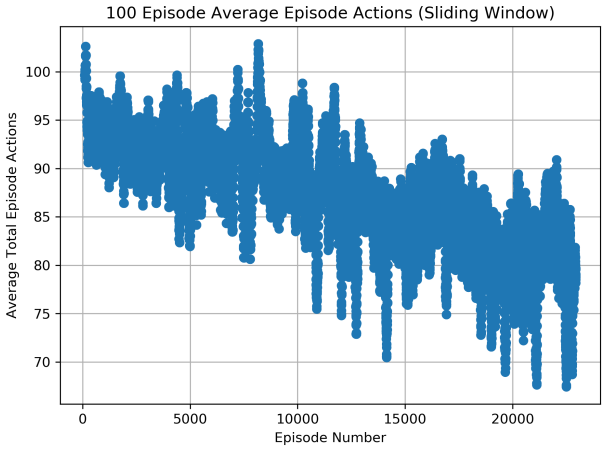


(d) Testing: action count histogram

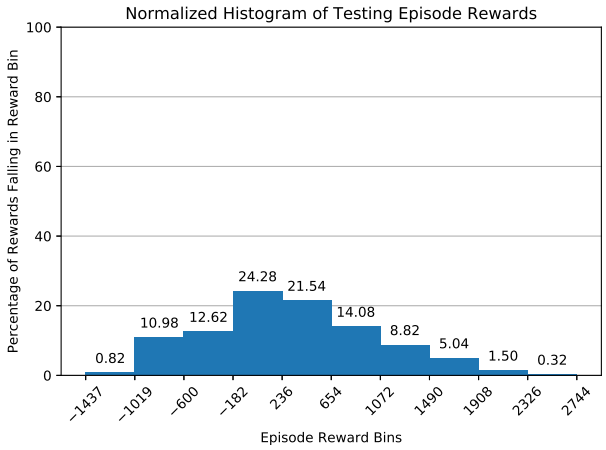
Figure 5.7: Training and testing rewards and actions: 200 bus system with contingencies, random seed: 0, neural network hidden layers: $[1024, 1024]$.



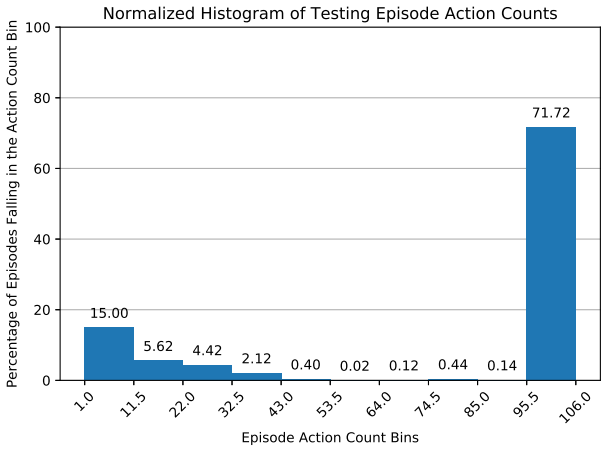
(a) Training: average rewards



(b) Training: number of actions taken

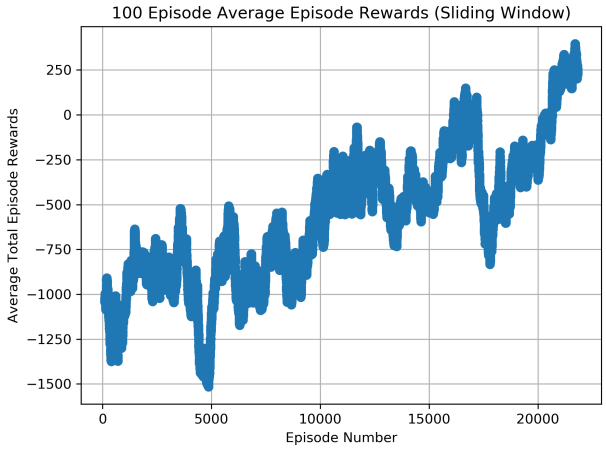


(c) Testing: reward histogram

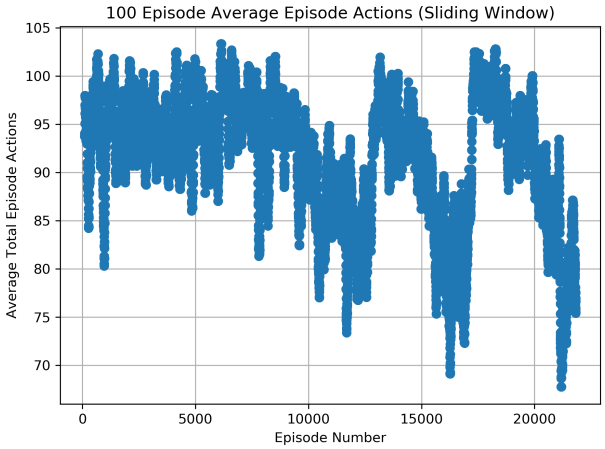


(d) Testing: action count histogram

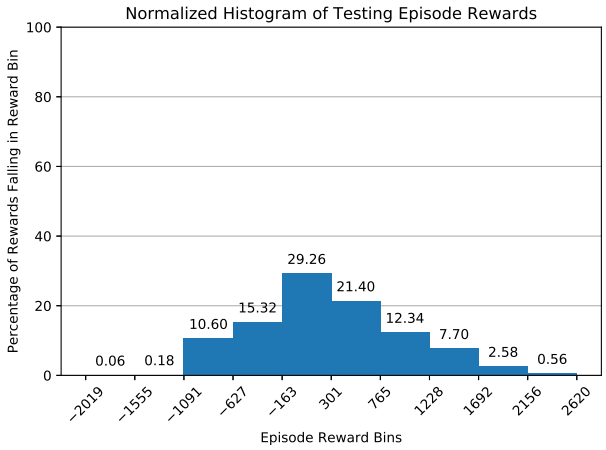
Figure 5.8: Training and testing rewards and actions: 200 bus system with contingencies, random seed: 1, neural network hidden layers: $[1024, 1024]$.



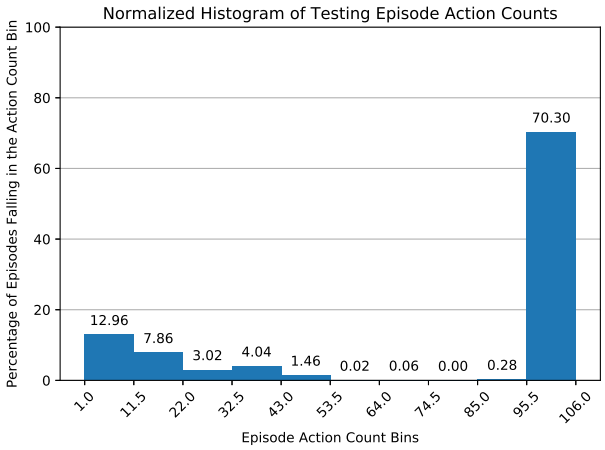
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count histogram

Figure 5.9: Training and testing rewards and actions: 200 bus system with contingencies, random seed: 2, neural network hidden layers: $[1024, 1024]$.

Table 5.6: Success percentages and rewards, 200 bus system with contingencies

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	18.32	17.71	99.26	-157.30
1	28.4	27.97	99.38	315.61
2	29.98	29.49	99.30	279.00
Overall Mean	25.57	25.06	99.31	145.77

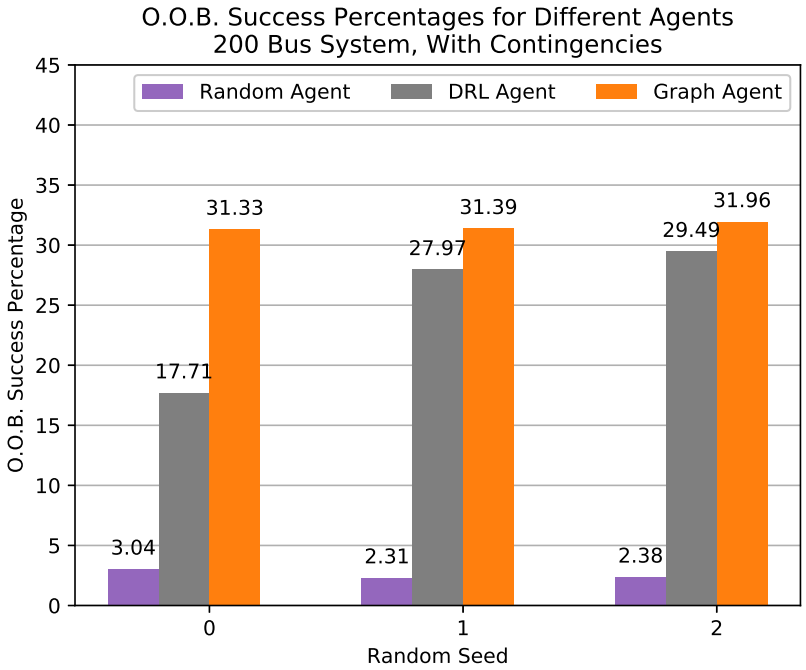


Figure 5.10: Comparison of random, DRL, and graph-based agents, 200 bus system, with contingencies

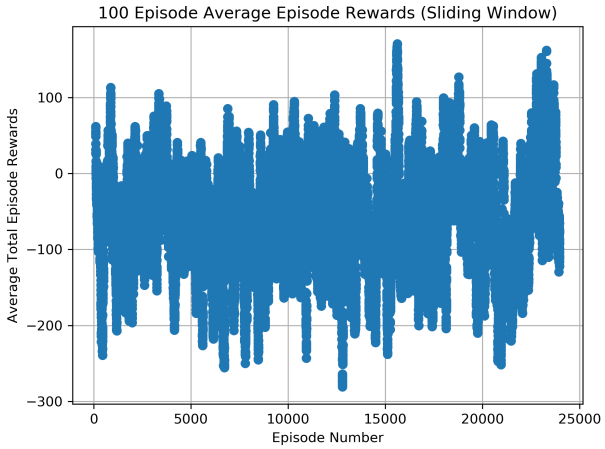
In this case, both Figures 5.7a and 5.9a show learning curves that lack monotonistic behavior, while Figure 5.8a has a relatively linear section beginning before training episode 10,000. Despite

the lack of a linear learning curve for the experiment with a random seed of 2, this experiment obtained the highest success rates as can be seen in Table 5.6.

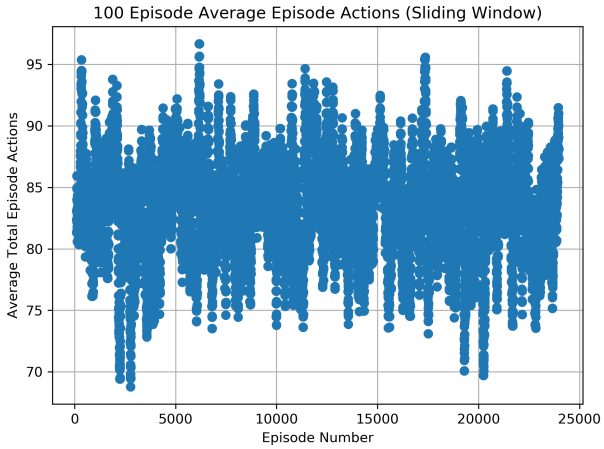
As can be seen from Tables 5.6 and 5.3, the DRL agent consistently significantly outperforms the random agent. Interestingly, in the presence of contingencies the DRL agent's best out-of-band success rate (29.49%, random seed $= 2$) is quite close to the graph-based agent's corresponding success rate (31.96%).

5.2.4 Results for Agents with Additional Training

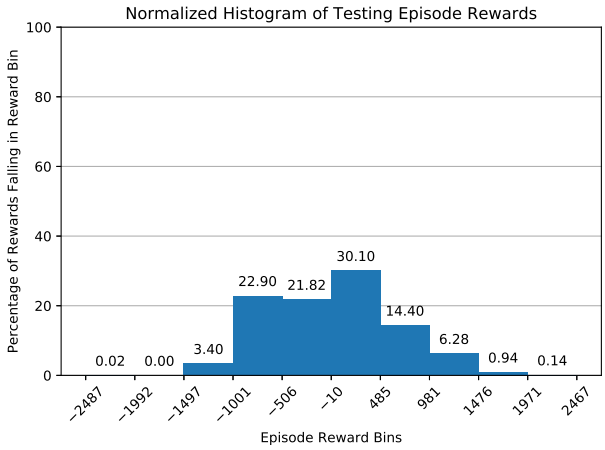
This subsection builds upon the results presented in Sections 5.2.2 and 5.2.3. The agents from the experiments with the highest out-of-band success rate from these two sections were trained for an additional 2,000,000 time steps. The additional training is carried out as described in Section 4.6.6. Additional training results without and with contingencies are presented in Figures 5.11 and 5.12, respectively. The agent acting in the environment without contingencies saw a slight uptick in out-of-band success rate, increasing from 23.47% to 24.12% while the agent which experienced contingencies saw a large decline in success, decreasing from 29.49% to 15.07%. It's clear from Figure 5.12a that toward the end of the additional training with contingencies the agent's average rewards saw a major decline.



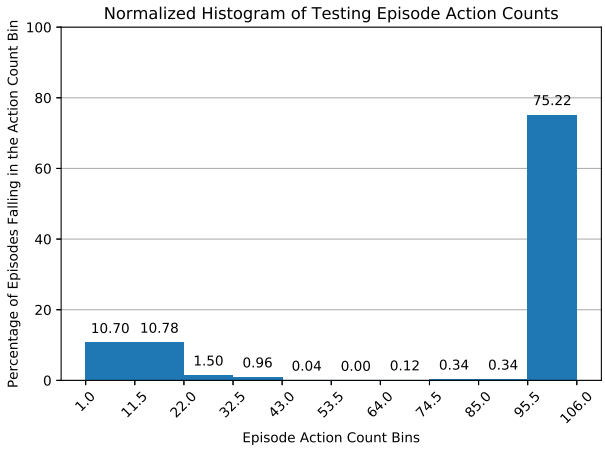
(a) Training: average rewards



(b) Training: number of actions taken

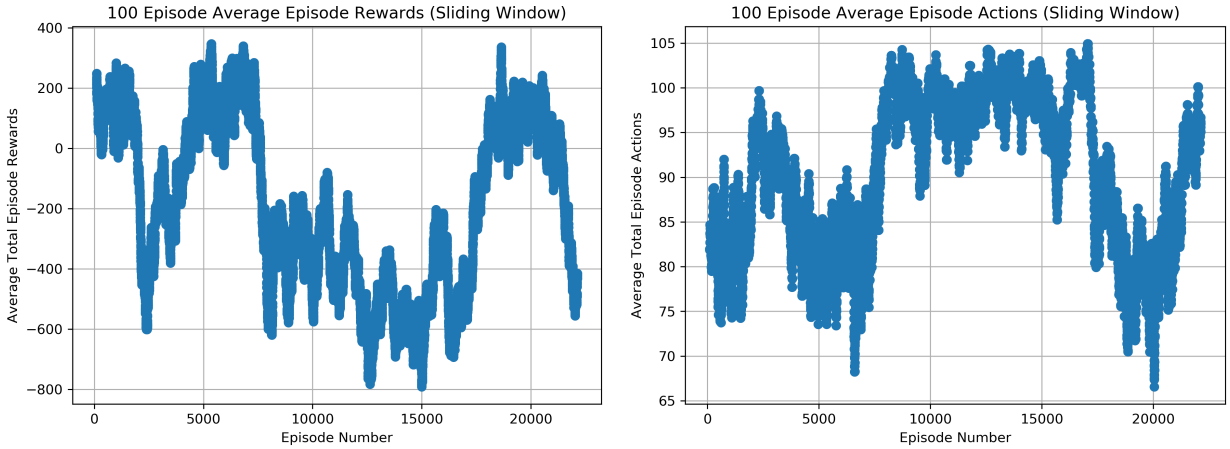


(c) Testing: reward histogram



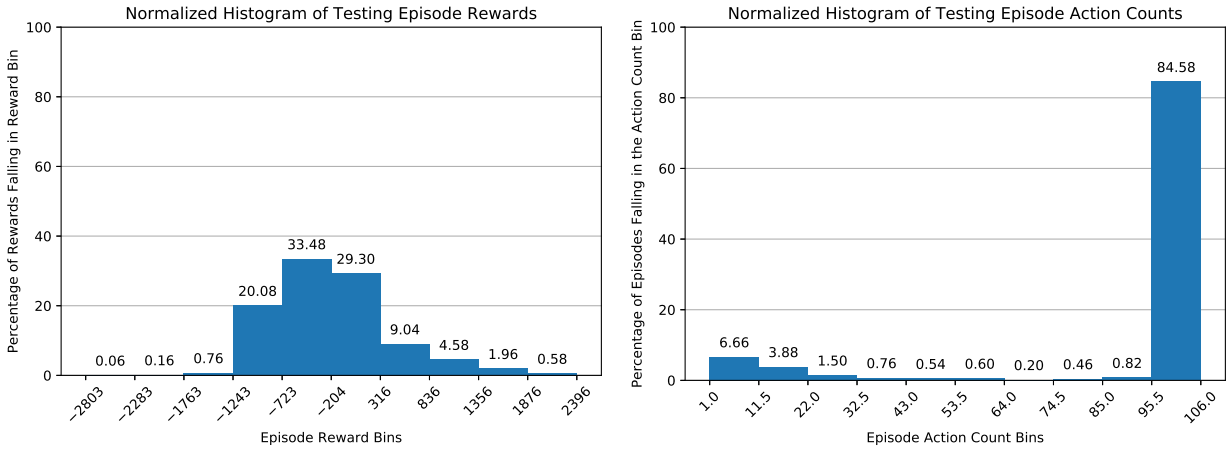
(d) Testing: action count histogram

Figure 5.11: Training and testing rewards and actions: 200 bus system without contingencies, additional training, random seed: 0, neural network hidden layers: $[1024, 1024]$.



(a) Training: average rewards

(b) Training: number of actions taken



(c) Testing: reward histogram

(d) Testing: action count histogram

Figure 5.12: Training and testing rewards and actions: 200 bus system with contingencies, additional training, random seed: 0, neural network hidden layers: $[1024, 1024]$.

5.3 Results: 500 Bus Case

Results for the 500 bus test system are described in this section. Since the 500 bus system is roughly double the size of the 200 bus system (see Sections 5.1.1 and 5.1.2 for object counts), the neural network hidden layers used for the 500 bus system have 2,048 neurons each (as opposed to 1,024 used for the 200 bus case). Training was carried out for 3,000,000 time steps. More training would have been performed had time permitted.

5.3.1 Results, Random and Graph-Based Agents

The following subsections present testing results for the random and graph-based agents. Results for the three individual testing runs as well as the overall mean are reported.

5.3.1.1 No Contingencies

Tables 5.7 and 5.8 depict results for the random and graph-based agents when no single line contingencies are applied. It can be seen that approximately 92.31% of all testing episodes start with out-of-band voltages. The random agent is successful in fixing voltage issues in 17.43% of episodes which begin O.O.B., while the graph-based agent is successful in 53.02% of O.O.B. episodes.

Table 5.7: Success percentages and rewards, 500 bus system without contingencies, random agent

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	23.68	17.66	92.66	-1554.15
1	23.66	17.09	91.98	-1528.12
2	23.88	17.55	92.30	-1547.21
Overall Mean	23.74	17.43	92.31	-1543.16

Table 5.8: Success percentages, 500 bus system without contingencies, graph-based agent

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**
0	56.44	52.99	92.66
1	57.40	53.69	91.98
2	56.04	52.37	92.30
Overall Mean	56.63	53.02	92.31

5.3.1.2 Single Line Contingencies

Tables 5.9 and 5.10 depict results for the random and graph-based agents when single line contingencies are applied. It can be seen that approximately 93.17% of all testing episodes start with out-of-band voltages. The random agent is successful in fixing voltage issues in 16.56% of episodes which begin O.O.B., while the graph-based agent is successful in 50.87% of O.O.B. episodes.

Table 5.9: Success percentages and rewards, 500 bus system with contingencies, random agent

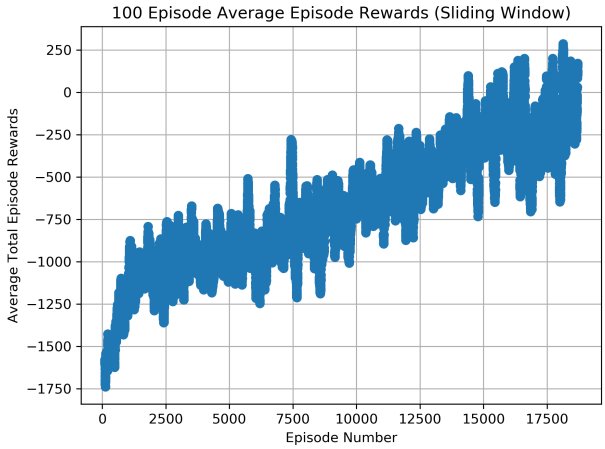
Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	21.88	16.54	93.60	-1582.15
1	21.98	15.86	92.68	-1569.83
2	22.90	17.29	93.22	-1539.96
Overall Mean	22.25	16.56	93.17	-1563.98

Table 5.10: Success percentages, 500 bus system with contingencies, graph-based agent

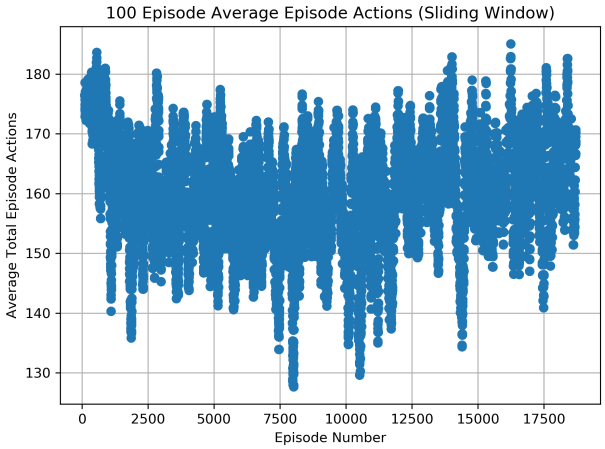
Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**
0	53.52	50.34	93.60
1	54.68	51.10	92.68
2	54.48	51.17	93.22
Overall Mean	54.23	50.87	93.17

5.3.2 Results, DRL Agent, No Contingencies

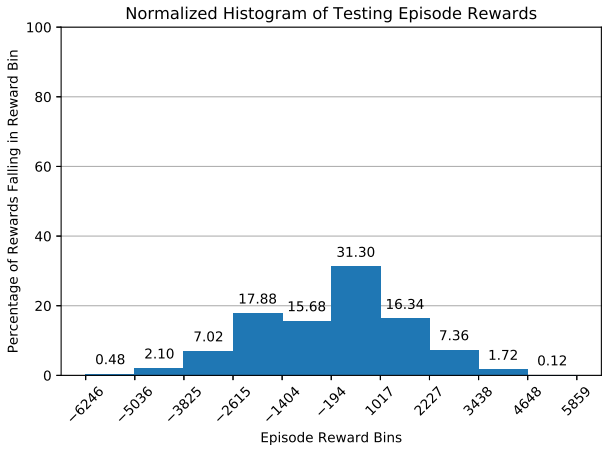
Training and testing results for the DRL agents when single-line contingencies are not applied are shown in Figures 5.13, 5.14, and 5.15. Table 5.11 presents testing success rates and rewards for the DRL agents. Figure 5.16 presents a comparison of the random, DRL, and graph-based agents. Figures 5.13a and 5.15a show increased rewards over time, while Figure 5.14a shows an extreme drop-off in rewards toward the end of training. The best agent was successful in fixing voltage problems in 21.50% of episodes which began with out-of-band voltages. This is slightly better than the corresponding random agent (17.43%) but significantly worse than the corresponding graph-based agent (52.37%).



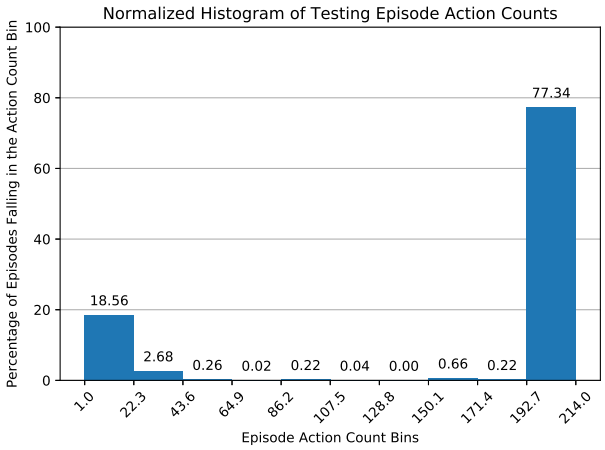
(a) Training: average rewards



(b) Training: number of actions taken

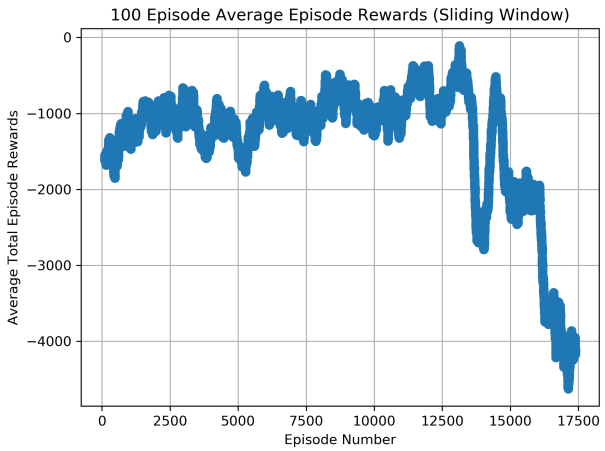


(c) Testing: reward histogram

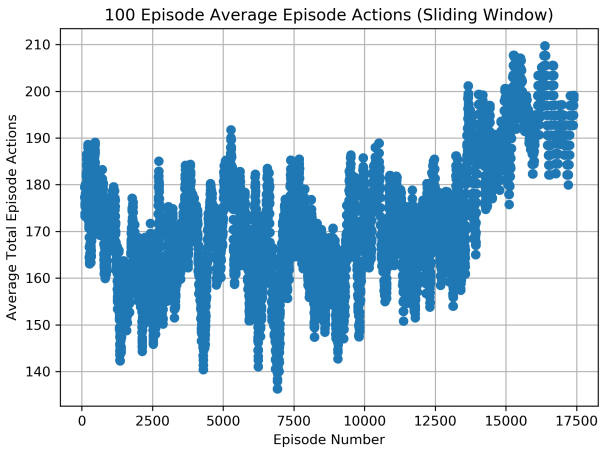


(d) Testing: action count histogram

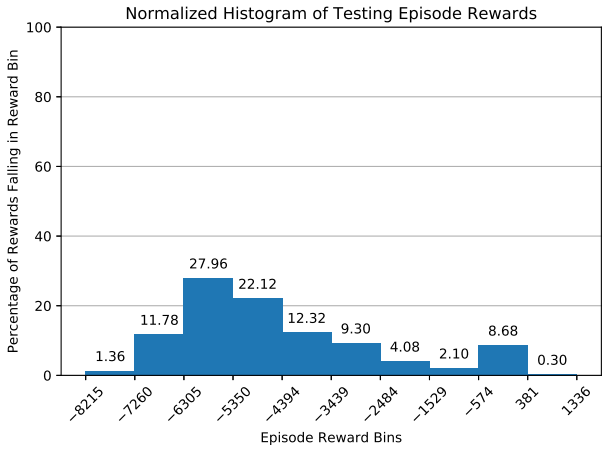
Figure 5.13: Training and testing rewards and actions: 500 bus system without contingencies, random seed: 0, neural network hidden layers: $[2048, 2048]$.



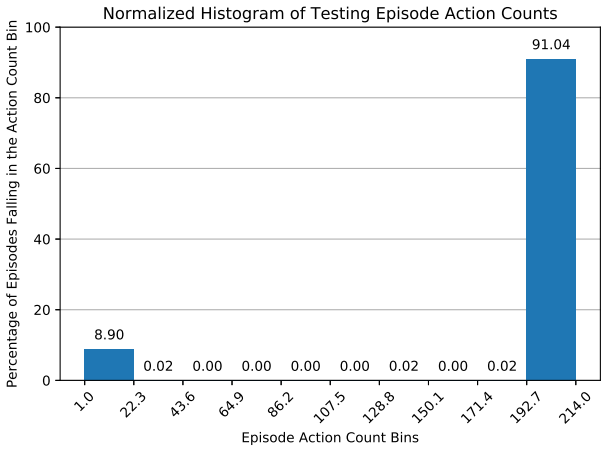
(a) Training: average rewards



(b) Training: number of actions taken

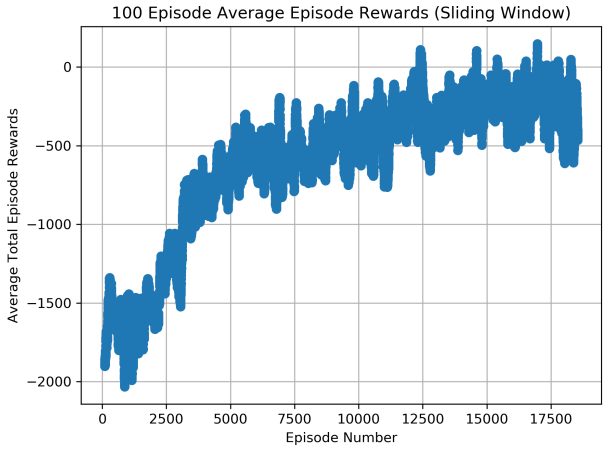


(c) Testing: reward histogram

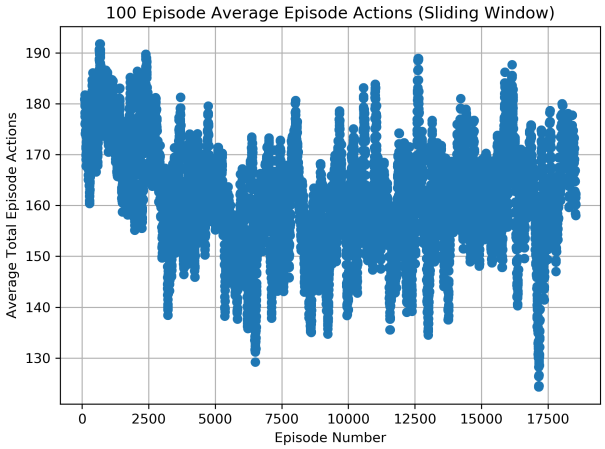


(d) Testing: action count histogram

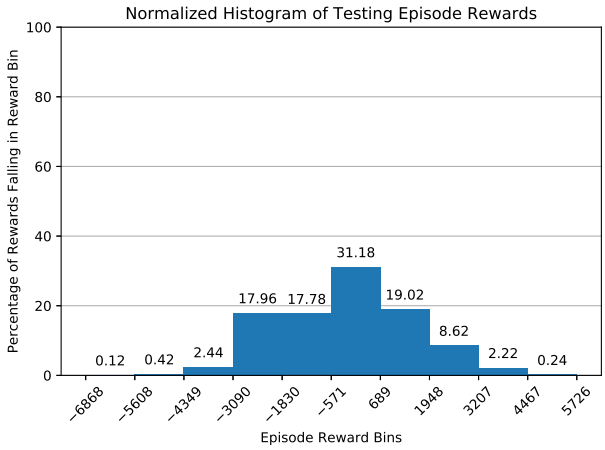
Figure 5.14: Training and testing rewards and actions: 500 bus system without contingencies, random seed: 1, neural network hidden layers: $[2048, 2048]$.



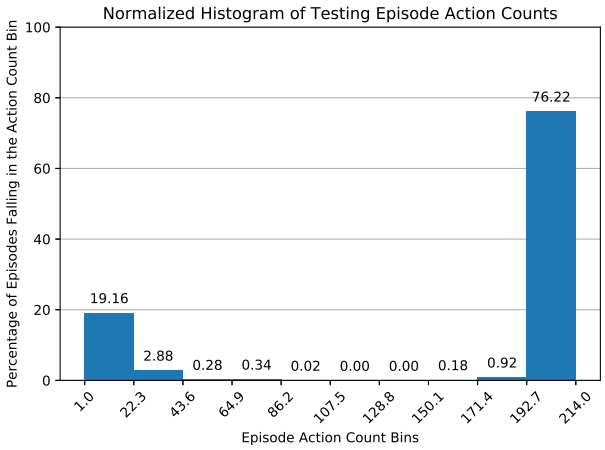
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count histogram

Figure 5.15: Training and testing rewards and actions: 500 bus system without contingencies, random seed: 2, neural network hidden layers: $[2048, 2048]$.

Table 5.11: Success percentages and rewards, 500 bus system without contingencies, DRL agent

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	23.14	20.38	92.66	-179.805
1	8.02	2.46	91.98	-4471.18
2	23.82	21.50	92.30	-168.83
Overall Mean	18.33	14.78	92.31	-1606.60

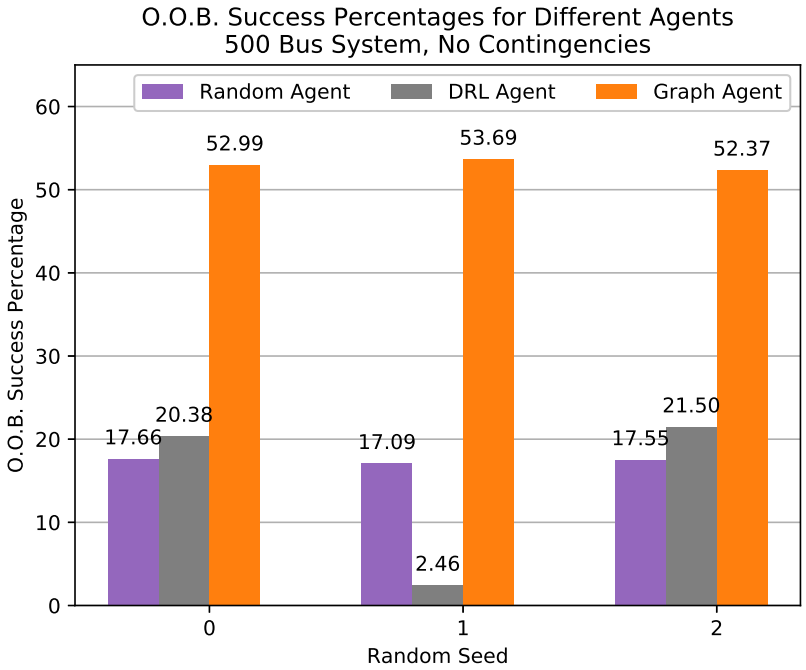
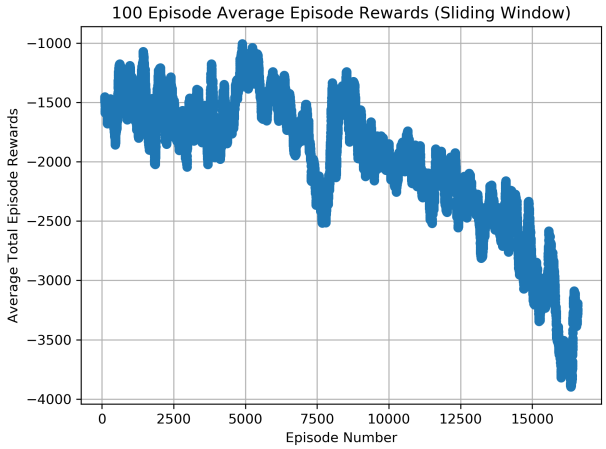


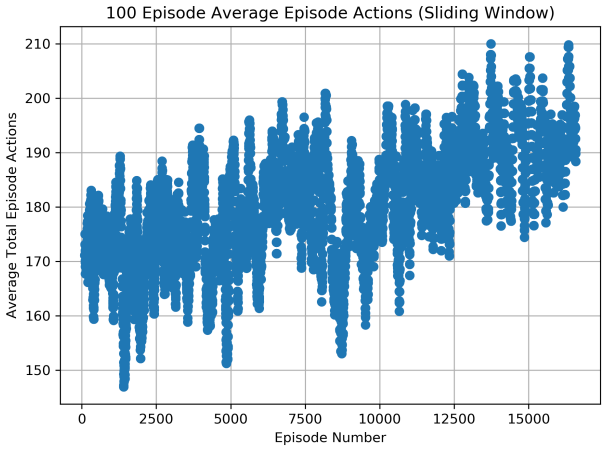
Figure 5.16: Comparison of random, DRL, and graph-based agents, 500 bus system, no contingencies

5.3.3 Results, DRL Agent, With Single Line Contingencies

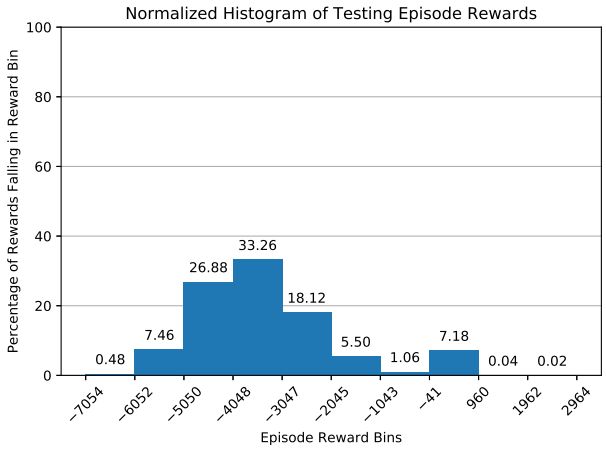
Training and testing results for the DRL agents when single-line contingencies are applied are shown in Figures 5.17, 5.18, and 5.19. None of these figures show a consistent increase in rewards over time, though Figures 5.18a and 5.19a do have upticks toward the end of training. Table 5.12 presents results for the DRL agents and Figure 5.16 presents a comparison of the random, DRL, and graph-based agents. The best agent was successful in fixing voltage problems in 26.28% of episodes which began with out-of-band voltages. This is significantly better than the corresponding random agent (17.29%) but significantly worse than the corresponding graph-based agent (51.17%).



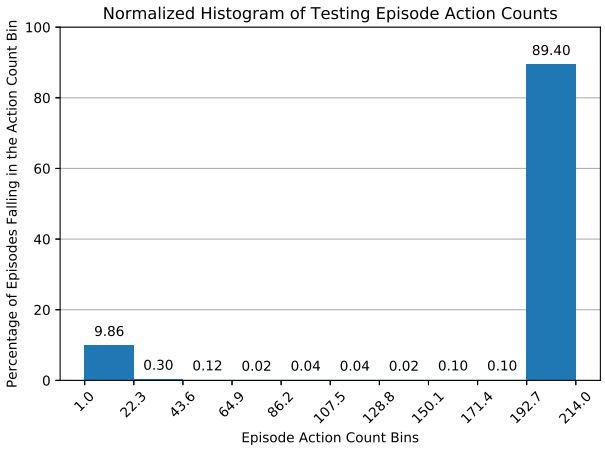
(a) Training: average rewards



(b) Training: number of actions taken

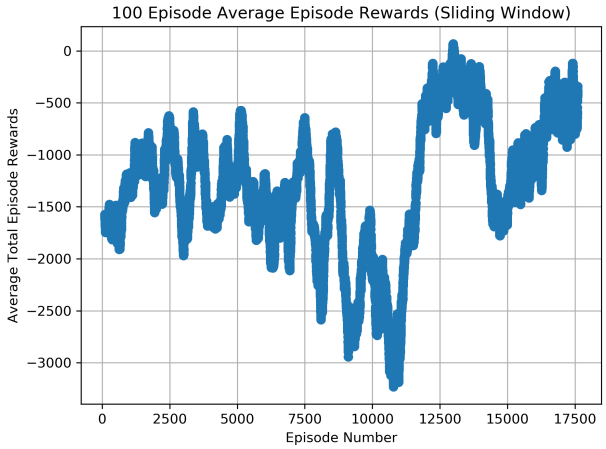


(c) Testing: reward histogram

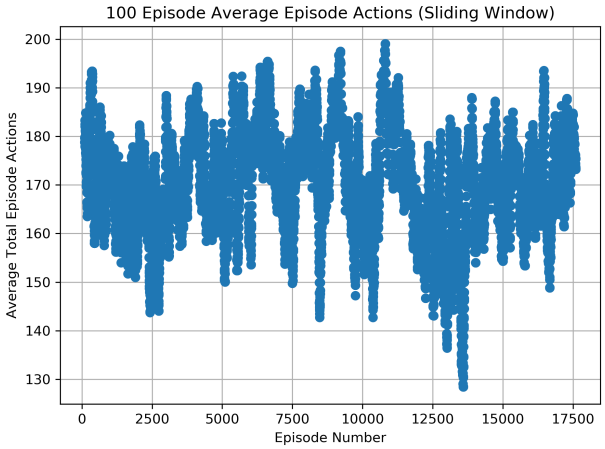


(d) Testing: action count histogram

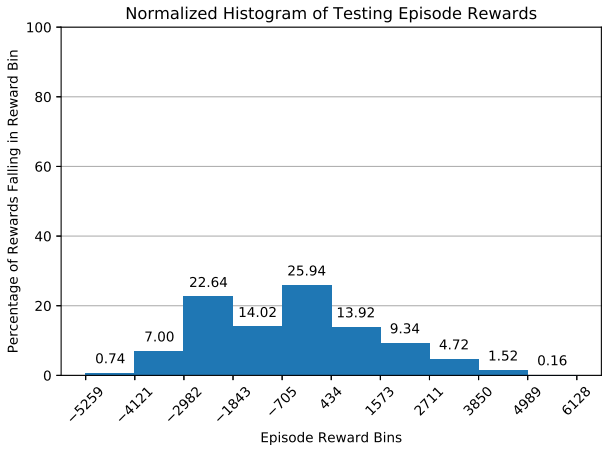
Figure 5.17: Training and testing rewards and actions: 500 bus system with contingencies, random seed: 0, neural network hidden layers: $[2048, 2048]$.



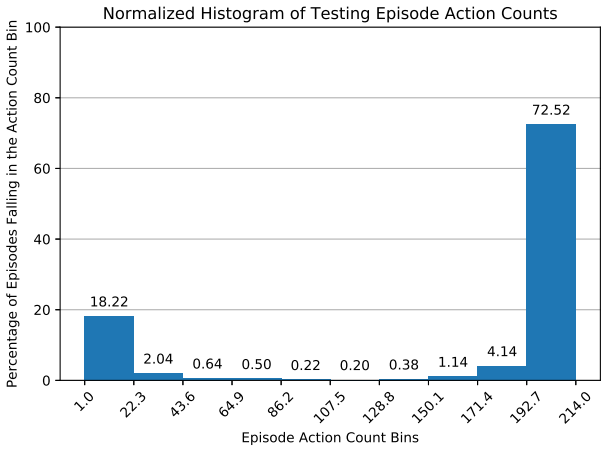
(a) Training: average rewards



(b) Training: number of actions taken

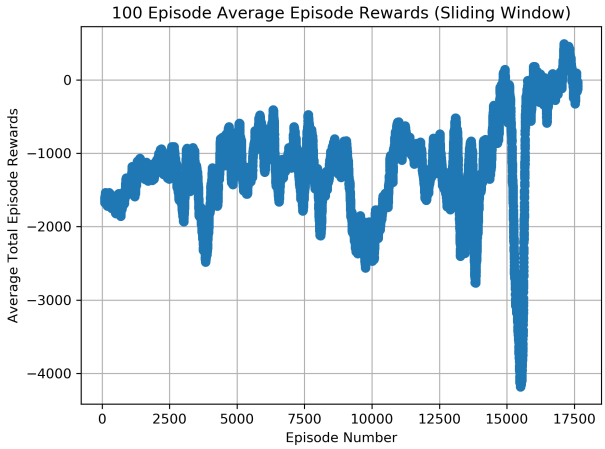


(c) Testing: reward histogram

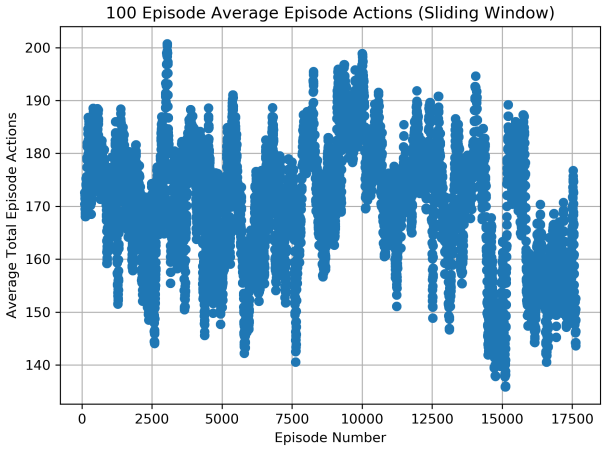


(d) Testing: action count histogram

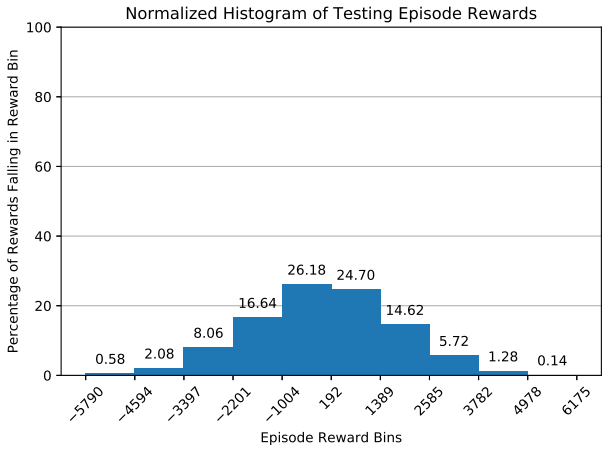
Figure 5.18: Training and testing rewards and actions: 500 bus system with contingencies, random seed: 1, neural network hidden layers: $[2048, 2048]$.



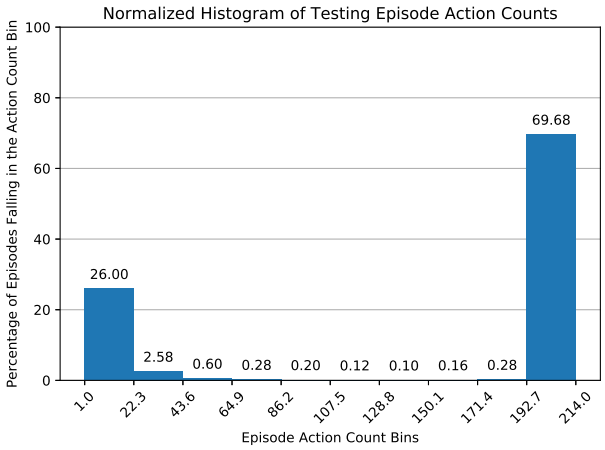
(a) Training: average rewards



(b) Training: number of actions taken



(c) Testing: reward histogram



(d) Testing: action count histogram

Figure 5.19: Training and testing rewards and actions: 500 bus system with contingencies, random seed: 2, neural network hidden layers: $[2048, 2048]$.

Table 5.12: Success percentages and rewards, 500 bus system with contingencies

Random Seed	**Success Percentage**	**Success Percentage, O.O.B.**	**Percent Episodes Start O.O.B.**	**Mean Reward**
0	7.40	3.21	93.60	-3392.79
1	30.24	25.10	92.68	-441.74
2	30.30	26.28	93.22	42.96
Overall Mean	22.65	18.19	93.17	-1263.86

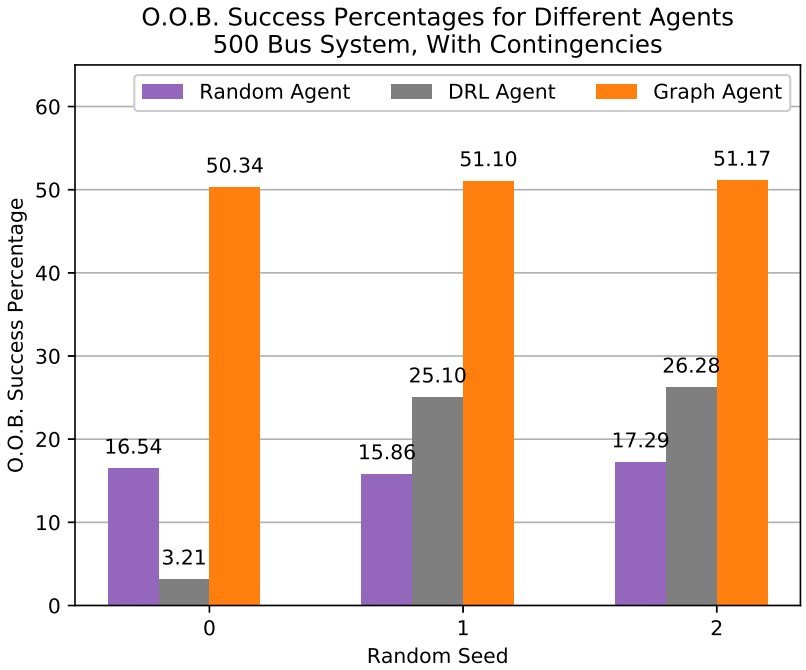


Figure 5.20: Comparison of random, DRL, and graph-based agents, 500 bus system, with contingencies

5.4 Discussion

In both Sections 5.2 and 5.3, no DRL agent ever met or exceeded the success rate of the graph-based agent. In most cases, the DRL agent's performance was better than the random agent's performance. It's interesting to note that the graph-based agent had a significantly higher success rate for the 200 bus system than the 500 bus system. It was also found when pre-screening power flow cases that a higher percentage of cases converged for the 500 bus system than the 200 bus system.

For both the 200 and 500 bus systems, DRL agent success rates in general increased when contingencies were included. The agents operating in the contingency-free environments did not receive line state observations, while the agents operating in environments with contingencies did receive line state observations. The higher success rates in the environments with contingencies may indicate that the neural networks for the contingency-free environments were too large, thus destabilizing training. More investigation is needed to verify this hypothesis.

For the 200 bus system, two agents received additional training beyond the initial 2,000,000 time steps. It was found that this additional training did not significantly increase success rates. Also for the 200 bus system, two of the three agents trained in the environment with contingencies achieved success rates near to graph-based agents.

6. CONCLUSIONS AND FUTURE WORK

This thesis provided an in-depth exploration of the application of deep reinforcement learning to the electric power transmission system voltage control problem. Primary software-related contributions of the work include the development of a Python package which interfaces with PowerWorld Simulator [30] and the development of open-source DRL environments for power system voltage control [31]. The so-called "GridMind" architecture and experiments from [27] were reproduced, and the shortcomings discussed. A novel deep-Q network algorithm modification wherein the agent is not allowed to take the same action multiple times in any given training or testing episode was shown to provide significant DRL performance improvements. Additionally, it was shown that the min-max scaling of bus voltage observations can lead to performance improvements as opposed to simply using per unit voltages. As opposed to the GridMind environment, the environments presented in this work are scalable to larger power systems. Additionally, the environments created for this work are capable of actuating switched shunts, which has not before been demonstrated in the context of DRL for voltage control. DRL agents were trained to control 200 and 500 bus power systems developed at Texas A&M University in order to test the scalability of DRL for voltage control [32, 33]. While no DRL agents were able to exceed the performance of the graph-based agents which were developed for comparison with DRL agents, there were cases with both the 14 and 200 bus systems where DRL agent performance approached graph-based agent performance.

The research presented in this thesis shows clear potential for using DRL to solve the voltage control problem, but more work is needed to ensure DRL techniques can consistently outperform conventional techniques.

Opportunities for future work are numerous. To name a few:

- Map observations onto a 2D geographically-based coordinate plane and leverage convolutional neural networks (CNNs) instead of MLPs

- Tune hyper parameters
- Experiment with different neural network architectures
- Improve neural network training, e.g. use of drop out for pruning neural networks
- Scale or normalize rewards with scheme 1 (see Section 4.4.1) by pre-running episodes with random actions to obtain a representation of the possible reward distribution
- Modify reward scheme 1 to never penalize the no-op action
- Modify reward scheme 2 (see Section 4.4.2) to provide more reward tiers
- Modify environments such that the no-op action terminates the episode
- Include sensitivities from PowerWorld in observations provided to the DRL agent
- Investigate methods for determining when to stop training
- Include whether or not generators are at var limits in observation space
- Train with time series and stacked observations rather than snapshot cases
- Compare DRL performance with that of expert human operators
- Test DRL performance with even larger power system cases, e.g. 2000 buses
- Include on-load tap-changing (OLTC) transformers in action space
- Create environments which only control shunts and taps, not generator voltage set points
- Move beyond the voltage control problem into security-constrained optimal power flow

Of the items mentioned above, it is believed that some of the most immediately useful experiments include using CNNs with geographically arranged observation data, improving neural network training with drop out (or other regularization techniques), changing the no-op action to always result in a reward of 0.0 and terminate the current episode, and to include observations related to generator var limits.

REFERENCES

[1] W. M. Warwick, "A primer on electric utilities, deregulation, and restructuring of u.s. electricity markets," tech. rep., Pacific Northwest National Laboratory, 2002.

[2] U.S. Energy Information Administration, "Eia forecasts renewables will be the fastest growing source of electricity generation." https://www.eia.gov/todayinenergy/detail.php?id=38053#. Accessed: 2019-12-23.

[3] P. H. Larsen, K. H. LaCommare, J. H. Eto, and J. L. Sweeney, "Recent trends in power system reliability and implications for evaluating future investments in resiliency," *Energy*, vol. 117, pp. 29 – 46, 2016.

[4] T&D World, "Trends report: Power companies facing labor shortage and skills gap." https://www.tdworld.com/safety-and-training/article/20972193/trends-report-power-companies-facing-labor-shortage-and-skills-gap. Accessed: 2019-12-23.

[5] Incremental Systems Corporation, "What is a system operator?." http://www.incsys.com/power4vets/what-is-a-system-operator/. Accessed: 2019-12-23.

[6] A. G. PHADKE and T. BI, "Phasor measurement units, wams, and their applications in protection and control of power systems," *Journal of Modern Power Systems and Clean Energy*, vol. 6, pp. 619–629, Jul 2018.

[7] U.S. Energy Information Administration, "Nearly half of all u.s. electricity customers have smart meters." https://www.eia.gov/todayinenergy/detail.php?id=34012. Accessed: 2019-12-23.

[8] C. Taylor, M. Venkatasubramanian, and C. Yonghong, "Wide-area stability and voltage control," *Symposium of Specialists in Electric Operational and Expansion Planning*, vol. 1, 01 2000.

[9] Electric Reliability Council of Texas, "Transmission and security desk operating procedure." http://www.ercot.com/mktrules/guides/procedures/. Accessed: 2019-12-23.

[10] Electric Reliability Council of Texas, "ERCOT control room video." https://youtu.be/TlsfItQc05o. Accessed: 2019-12-23.

[11] PJM, "A day in the life of dispatch." https://youtu.be/hDNouhlJWa4. Accessed: 2019-12-23.

[12] North American Electric Reliability Corporation, "Reliability guideline - reactive power planning," tech. rep., North American Electric Reliability Corporation, Dec. 2016. Available: https://www.nerc.com/comm/PC_Reliability_Guidelines_DL/Reliability%20Guideline%20-%20Reactive%20Power%20Planning.pdf.

[13] R. E. Wilson and C. W. Taylor, "Using dynamic simulations to design the wide-area stability and voltage control system (wacs)," in *IEEE PES Power Systems Conference and Exposition, 2004.*, pp. 100–107 vol.1, Oct 2004.

[14] S. Corsi, M. Pozzi, C. Sabelli, and A. Serrani, "The coordinated automatic voltage control of the italian transmission grid-part i: reasons of the choice and overview of the consolidated hierarchical system," *IEEE Transactions on Power Systems*, vol. 19, pp. 1723–1732, Nov 2004.

[15] J. Tong, D. W. Souder, C. Pilong, M. Zhang, Q. Guo, H. Sun, and B. Zhang, "Voltage control practices and tools used for system voltage control of pjm," in *2011 IEEE Power and Energy Society General Meeting*, pp. 1–5, July 2011.

[16] H. Sun, Q. Guo, B. Zhang, W. Wu, and J. Tong, "Development and applications of system-wide automatic voltage control system in china," in *2009 IEEE Power Energy Society General Meeting*, pp. 1–5, July 2009.

[17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Peasrson, 3 ed., Dec. 2009.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," in *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[20] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of Go without human knowledge," in *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[21] T. P. I. Ahamed, E. A. Jasmin, and E. A. Al-Ammar, "Reinforcement learning in power system scheduling and control: A unified perspective," in *2011 IEEE Symposium on Computers Informatics*, pp. 650–655, March 2011.

[22] M. Glavic, R. Fonteneau, and D. Ernst, "Reinforcement learning for electric power system decision and control: Past considerations and perspectives," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6918 – 6927, 2017. 20th IFAC World Congress.

[23] Q. Huang, R. Huang, W. Hao, J. Tan, R. Fan, and Z. Huang, "Adaptive power system emergency control using deep reinforcement learning," *IEEE Transactions on Smart Grid*, pp. 1–1, 2019.

[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[25] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." https://github.com/openai/baselines, 2017.

[26] J. G. Vlachogiannis and N. D. Hatziargyriou, "Reinforcement learning for reactive power control," *IEEE Transactions on Power Systems*, vol. 19, pp. 1317–1325, Aug 2004.

[27] R. Diao, Z. Wang, D. Shi, Q. Chang, J. Duan, and X. Zhang, "Autonomous voltage control for grid operation using deep reinforcement learning," in *2019 IEEE Power Energy Society General Meeting (PESGM)*, pp. 1–5, Aug 2019.

[28] B. Palmer, W. Perkins, Y. Chen, S. Jin, D. Callahan, K. Glass, R. Diao, M. Rice, S. Elbert, M. Vallem, and Z. Huang, "Gridpack: A framework for developing power grid simulations on high performance computing platforms," in *2014 Fourth International Workshop on Domain-*

Specific Languages and High-Level Frameworks for High Performance Computing, pp. 68–77, Nov 2014.

[29] PowerWorld Corporation, "PowerWorld Simulator." https://www.powerworld.com/. Accessed: 2019-12-24.

[30] Z. Mao, B. Thayer, and Y. Liu, "Easy simauto (esa)." https://github.com/mzy2240/ESA, 2019.

[31] B. Thayer, "gym-powerworld." https://github.com/blthayer/gym-powerworld, 2020.

[32] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Grid structural characteristics as validation criteria for synthetic networks," *IEEE Transactions on Power Systems*, vol. 32, pp. 3258–3265, July 2017.

[33] Texas A&M University, "Electric grid test cases." https://electricgrids.engr.tamu.edu/electric-grid-test-cases/, 2019. Accessed: 2019-12-27.

[34] Python Software Foundation, "Python." https://www.python.org/. Accessed: 2019-12-26.

[35] D. Robinson, "The incredible growth of python." https://stackoverflow.blog/2017/09/06/incredible-growth-python/. Accessed: 2019-12-26.

[36] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *arXiv e-prints*, p. arXiv:1907.10121, Jul 2019.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke,

V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[38] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[39] W. Tinney and W. Meyer, "Solution of large sparse systems by ordered triangular factorization," *IEEE Transactions on Automatic Control*, vol. 18, pp. 333–346, August 1973.

[40] W. F. Tinney, V. Brandwajn, and S. M. Chan, "Sparse vector methods," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-104, pp. 295–301, Feb 1985.

[41] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." https://github.com/hill-a/stable-baselines, 2018.

[42] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015.

[43] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015.

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015.

[45] B. Thayer, "drl-powerworld." https://github.com/blthayer/drl-powerworld, 2020.

[46] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

[47] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, December 1959.