Pain-Inspired Intrinsic Reward For Deep Reinforcement Learning

by

Trevor Woods Richardson

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved June 2018 by the
Graduate Supervisory Committee:

Heni Ben Amor, Chair
Yezhou Yang
Siddharth Srivastava

ARIZONA STATE UNIVERSITY

August 2018

ABSTRACT

Reinforcement learning (RL) is a powerful methodology for teaching autonomous agents complex behaviors and skills. A critical component in most RL algorithms is the reward function – a mathematical function that provides numerical estimates for desirable and undesirable states. Typically, the reward function must be hand-designed by a human expert and, as a result, the scope of a robot's autonomy and ability to safely explore and learn in new and unforeseen environments is constrained by the specifics of the designed reward function. In this thesis, I design and implement a stateful collision anticipation model with powerful predictive capability based upon my research of sequential data modeling and modern recurrent neural networks. I also develop deep reinforcement learning methods whose rewards are generated by self-supervised training and intrinsic signals. The main objective is to work towards the development of resilient robots that can learn to anticipate and avoid damaging interactions by combining visual and proprioceptive cues from internal sensors. The introduced solutions are inspired by pain pathways in humans and animals, because such pathways are known to guide decision-making processes and promote self-preservation. A new "robot dodge ball' benchmark is introduced in order to test the validity of the developed algorithms in dynamic environments.

ACKNOWLEDGEMENTS

I would like to thank everyone who has helped and supported me during my year and a half doing research at ASU. I would like to thank my committee, my lab and Dr. Ben Amor for being substantial resources in my first experience doing academic research in the field of artificial intelligence, robotics and machine learning. I would like to thank Dr. Ben Amor for the countless nights he spent molding me into a better researcher. I would like to give special thanks to give special thanks to Dr. Linda Chattin and Dr. Debra and Dr. Frank Calliss for being ASU professors that took special interest in me. From my lab, I would like to thank Kevin Luck, Nambi Srivastav, Joe Campbell, Geoffrey Clark Clark, Mark Strickland, and Indranil Sur; these individuals were significant sources of help throughout the research process. I would like to give special thanks to Kevin Luck for taking a special interest in me as a researcher. I would like to give special thanks Trevor Barron for continually working with me on varying problems and inspiring me to chase the research problems I feel passionate about. I would like to give special thanks to Simon Stepputtis for being an invaluable resource on all things robotics and a person who was always willing to help me with any task I struggled with solving. I would like to give special thanks Rudra Saha for all of the coffee talks we had regarding research, and for all of the research and ideas he exposed me too. I would like to give special thanks to Nathan Kelley as he added so much value to the final research product I am presenting today. Nathan Kelley added invaluable insights and assistance towards the completion of this thesis. I would like to give special thanks to Richard Dunkle and Walter Johnson for motivating me to pursue a career in computer science.

While this list of friends deserve special thanks it is in no way an exhaustive list of all the friends I need to thank for their continued support through this experience. These people are people that have gone above and beyond for me. I would

like to give special thanks to Gaizka Urreiztieta, Jason Walker, Kelle Dhein, Kyle Testerman, Jake Smith, Natalie Andros, Uncle Alex, Thad Botham, Katrina Eory, Chelsea Cummings, Jared Tevis, Robert Drier, Vicente Terran, Leilani Gilpin, Willie Wilson, Will Cranmer, Alex Young, David Ponessa, Fatima Naveed, Ryan Schachte, Jessi Lammers, and Jon Reasoner. These individuals have helped and supported me in countless ways and I'm blessed to have them in my life.

I would like to give special thanks to Tom and Andrea for being two of the most important connections I had during this experience. I would like to thank my Nana, Tata, Grandma and Grandpa for all they've given me and taught me. More than anything I would like to give special thanks to my Mom and Dad; no one gave me more than them.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

PAIN

1.1 Introduction

Pain plays a central role in our lives and is of paramount importance to many brain and body mechanisms such as cognition, social interaction, motor control, memory, learning, autonomy and, most importantly, self-preservation. It acts as a critical signal that guides our decision-making processes and shapes the choices we make. A long-standing theory, articulated by Descartes [3], describes pain as bodily perturbations that are detected by nerve fibers and communicated to the brain. This theory limits the role of pain to the sensation of bodily harm, failing to acknowledge the many other functions involved in complex biological pain systems. Contemporary scientific evidence indicates that pain is generated through a complex interplay of a variety of signals and predictions involving multiple areas of the brain [24, 15, 30].

Despite its central role in many functions of the human brain, to date, pain has attracted relatively little interest in the robotics community. Pain and its relationship to robotics, however, has not been completely overlooked [16, 33]. Researchers have attempted to formalize pain for robotics, which will be referred to in this paper as robotic-pain. One recent result has been the development of an "artificial Robot Nervous System" that can react to multi-modal stimuli much like a biological organism's pain-reflex [20]. This robotic-pain system is similar to Descartes' view of pain and does not encompass various other roles and interactions of complex biological pain systems. In contrast to the reflex-only approach, Sur and Ben Amor have shown that perturbations can be learned and anticipated [33]. Harmful interactions are often

included in reinforcement learning (RL) algorithms as negative rewards [34]. RL, however, requires a human expert to specify how this negative reward is calculated, typically resulting in extremely task-specific algorithms.

This chapter explores the evolutionary basis for biological pain and the potential to relate various beneficial aspects of biological pain to robotic systems. The goal is to develop resilient machines and systems that can learn to anticipate and avoid harmful sensations, with a concomitant increase in longevity and autonomy. This chapter will first discuss the evolutionary origin of biological pain, as well as the complex web of underlying mechanisms and functions of biological pain systems. Next, biological pain will be distinguished from robotic-pain systems. After that, the opportunities and challenges that arise from studying computational frameworks that mimic nociceptive pathways will be addressed. Finally, two benchmark tasks will be described that can be leveraged to accelerate research in this area. The primary objectives of this chapter are to highlight a critical knowledge gap in our understanding of intelligent, physical systems and to identify a new, promising avenue for further research by the robotics community.

1.2 The Evolutionary Role of Pain

Pain is a sensation that many species experience. It is not unique to humans and has been observed in vertebrates as well as invertebrates such as cephalopods. Pain is a dominant neurobiological process that is essential to the survival of our species; its influence is felt in almost all functional areas of the human brain [2]. The central nervous system (CNS) generates pain signals that influence our behavior and guide our learning within the contexts of self-preservation and reproduction.

The CNS has proven to be evolutionarily advantageous, having arisen as a result of natural selection. Accordingly, biological pain pathways are heritable traits

that promote the fitness of individuals within a given population. Individuals with congenital insensitivity to pain frequently die at a relatively young age due to tissue damage, infections, or both [26]. Dawkins [11] offers a persuasive thought experiment to illustrate the importance of pain's role in evolution. He asks his audience to consider the potential fitness of gazelles with genes that cause analgesic states when fleeing predators. He concludes that this gene pool of gazelles would not be favored by natural selection unless tranquilizing a gazelle that is attempting to evade predation improves that gazelle's probability of reproducing [11]. He asserts that one must infer that gazelles experience extreme agony before death, because this system promotes self-preservation and the likelihood of reproduction. His thought experiment reinforces the principle that pain is a product of natural selection.

In addition to its evolutionary advantage, pain functions as a guiding signal by which a biological organism learns to navigate its environment safely, interact with living beings and inanimate objects, and promote its own well-being. It is therefore central to the behaviors learned and exhibited by a species within a lifetime. According to Craig [9], pain is not only a sensation, but also a motivation that is rooted in an emotional drive that results in homeostatic behavior. Not only does pain impose evolutionarily significant influences on behavior, it also serves as an educational feedback signal. The use and maintenance of a CNS of sufficient complexity to experience pain requires the expenditure of considerable energy. Such a CNS would be wasteful unless it served an evolutionarily advantageous purpose. Current research indicates that most insects do not experience pain [31]. A prevalent scientific theory suggests that this is evidence that pain is more advantageous to organisms with longer life spans because learning complex relationships is more advantageous to organisms that need to live longer to reproduce. Pain is beneficial to complex learning within the scope of self-preservation. Even more important may be the relationship between pain and

emotional learning. Apkarian found that the representation of acute pain is related to the areas of the brain primarily responsible for emotional learning, memory and reward/addictive behavior [1]. Scientific evidence indicates that as we navigate our lives, pain consistently influences our behavior; it plays a central role in our ability to safely learn complex relationships while engaging with our environment [24].

1.3 Learning, Empathy, Memory and Fear

The neurobiological processes that produce pain significantly influence other human and animal functions. In particular, our ability to learn, empathize, remember and fear are all mechanisms that are affected by pain pathways in the central nervous system [4, 5, 6, 15, 18, 21, 30]. With regard to learning, this impact is realized in two distinct ways. First, fear-based conditioning leads to associative and avoidance learning [5, 29, 35, 36, 38]. Second, pain impacts learning through the bidirectional relationship between the formation of an individual's motivations and the pain that is experienced when those motivational goals are pursued [5, 29, 35, 36, 38]. Contemporary research has shown that personal experiences of pain are altered based on an individual's motivations and conditioned fear-based associations, as well as social factors that are unique to that individual [1, 36, 38]. Because the impact of experiencing pain is bidirectional, it functions as a fear-based conditioner. Fear-based conditioning that results from pain directs an organism's motivations and, as a result, affects how that organism learns [29, 35, 36] and how quickly it learns [15].

Another function that bears a close relationship to pain is empathy. There is a considerable overlap of brain activation between individuals experiencing pain and those experiencing empathy [6, 21, 30]. Research has shown that the brain's signature for empathy overlaps specifically with the brain's signature for pain in areas that are associated with pain's affective as opposed to sensory qualities [30]. Some results

suggest that empathy is not exclusively a human emotive state, but instead is one that also exists in other living organisms such as rats [6].

The attentional resource needs of pain systems are considerable and parts of the brain outside the pain matrix can be altered by CNS pain networks. For example, an individual's memories can be altered by a painful experience [4, 10, 18, 27, 28]. In many cases, experiencing intense pain results in an enhanced ability to accurately recall a memory or to recall the emotions experienced during the painful event [4, 18, 28]. In contrast, due to pain's attentional requirements, painful experiences can also limit an individual's capability to remember information about his or her environment, especially when the information is not directly related to the cause of the painful experience [4]. In one study, subjects were found to remember their emotions with high accuracy after experiencing intense, acute pain. The same subjects, however, were much less accurate in recalling an unrelated stimulus present during the painful event [4].

1.4 Defining Robotic-Pain and Distinguishing Biological Pain From Robotic-Pain

In this paper, the use of the word robotic-pain does not equate to biological pain. Biological pain and robotic-pain are distinctly different. Robotic-pain does not cause the robot to experience anguish, suffering, or unpleasantness. Robotic-pain does not have the same morphological structures as biological pain. The goal of robotic-pain systems is to emulate the benefits resulting from biological pain systems–avoid harmful interactions with environment. Throughout this paper, the term robotic-pain refers to any robotic and/or algorithmic system that attempts emulate these benefits.

1.5 Pain Maps to Robotics and the Benefit of Robotic Pain Systems

The development of robotic systems with the capacity to perceive robotic-pain would further the dream of creating a fully autonomous robot that can explore dangerous environments in as safe a manner as reasonably possible. With advances in hardware and software, many of the beneficial aspects of biological pain described in sections 1.2 and 1.3 have the potential to be realized in robotic systems. Such developments would provide immense benefits to functioning robots.

Fig. 1.1 displays specific interrelated and essential properties of pain-based systems. Hardware and software research has the potential to map these properties to the robotic domain. The construction of systems that improve a robot's self-awareness and promote self-preservation have far reaching applications. Such systems would benefit robotics in general by enhancing various learning capabilities, reducing the financial cost of replacing robotic parts and ensuring the longevity of robotic systems as a whole. A robotic system that learns to associate certain scenes with negative internal states has the potential to understand the behaviors of other robots and organisms in the same environment. Empathy involves the comprehension of another organism's internal states. If a robot can predict a potentially precarious situation facing another robot, not only can that robot learn from the other robot's situation, but it also has the potential to assist the endangered robot. A robot that algorithmically encodes negative internal harm and events that may cause harm has the potential to protect not only itself and other robots, but also human beings and other organisms. It is computationally intractable to preprogram all possible noxious states. In many experiments, robots only learn how to interact with their environment over a small period of time within fixed boundaries. In the future, robots will need to learn how to accomplish significantly more complex tasks that require them to reason about and

Figure 1.1: Critical functions that are interconnected to pain.

interact with their environment over much longer periods of time and in multiple and varying locations. One could imagine giving a robot the tasks of shopping at a grocery store that it has never been to before and then cooking a meal for you and your family. These tasks present many perilous possibilities for the robot as it interacts with new situations. Learning how to accomplish these tasks and overcome the obstacles it will face (such as navigating new doorways in high traffic areas, transporting the required ingredients and overcoming food preparation dangers) can best be addressed through a system that promotes learning that involves concerns about internal safety and sustainability. Such a system would need to be able to reevaluate its own goals and motivations based on new data coming from nociceptive robotic-pain systems. Any learned information about noxious experiences with the environment would need to be stored, reused and, most importantly, generalized to new contexts.

It is crucial that intelligent, autonomous robots with finite capacity for storing

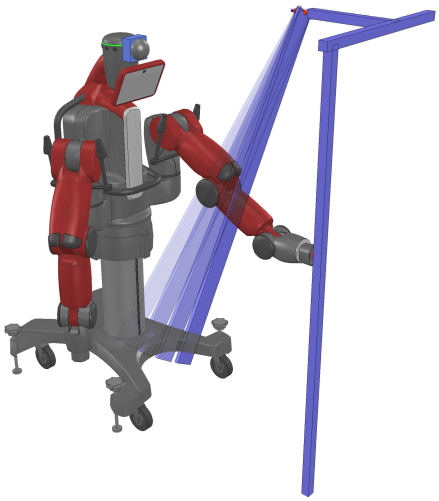

Figure 1.2: Rotating Bar Benchmark.

memories retain the most relevant and important experiences. A fully integrated robotic-pain system can assist in grading the relative importance of specific memories. For a fully autonomous robot to be realized, these types of Bayesian conditioning associations must be learned in real time. A robotic-pain system would provide important feedback that would enable probabilistic conditioning. Feedback signals about harmful interactions would provide valuable insight regarding whether a given action or state would be beneficial or harmful to the robot. In general, robotic pain systems will promote robotic autonomy, system lifespan and robotic altruism – a robot's ability to assist other robots, humans and other living organisms. The creation of robotic-pain systems requires experimental platforms that enable scientists to collaborate in this promising new area of research. The remaining portions of this chapter describe two scenarios – the Rotating Bar task and the Dodge Ball task. These scenarios allow the exploration of fundamental questions regarding computational theories of robotic-pain.

Rotating Bar: In the Rotating Bar task displayed in Fig. 2, a fixed-position

robot must learn how to extend its arms and end effectors, while at the same time avoiding damage from a rotating bar. In this scenario, the robot uses an RGB vision camera to observe its environment. The robot can query about its own internal states such as the position and orientation of its end effectors and arms. Parameters that can be varied include the angular velocity of the bar, angular acceleration of the bar, size of the bar, length of the robotic arms, and complexity of the task assigned. The robot's position, however, is fixed. A proper robotic-pain solution requires the robot to rapidly learn to avoid negative harmful states and to balance these nociceptive stimuli against its desire to complete its assigned task. The robot must learn to balance its goals and motivations with its own well-being. Note that any solution should not include any hard-coded reward function, e.g., `if(arm_torque > thresh) pain = 1`. This scenario tests the bi-directionality of competing interests and avoidance learning and provides the ability to analyze various nociceptive software models. The robot must learn to understand where negative feedback is occurring and how to respond to possible harm such as a damaged arm or end effector.

Dodge Ball: In the Dodge Ball scenario, see Fig. 3, N balls with Gaussian distributed initial positions are sent into projectile motion with randomly distributed initial velocities in the x, y, z planes toward a robot that can move along one dimension only. The robot is confined to a limited space. It accesses information about its environment using an RGB camera. It also can obtain information about its own internal state such as its velocity, orientation, and position. If the robot chooses not to move, or moves randomly, it will eventually be hit by some of the projectiles. Movement guided by intelligent anticipation is necessary to minimize the number of collisions –often referred to in this paper as perturbations– with incoming projectiles. The robot needs to learn from experience what visual information is predictive of impending harm. This platform provides multiple parameters of interest, such as the

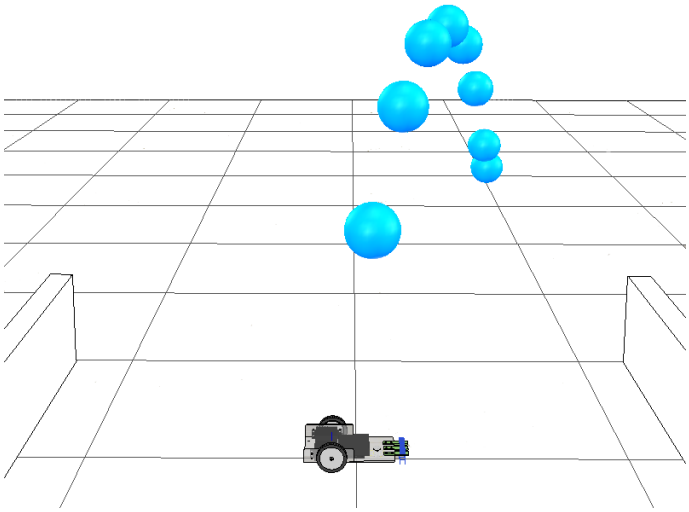

Figure 1.3: Dodge Ball Scenario

rate and speed at which balls are fired, the distance from the robot to the balls, and the damage incurred to the robot from each collision. The collisions provided by this simulation offer a way to test and build software that learns to predict negative future internal states. The robot must prioritize some noxious states over others. In these situations, the robot will need to endure potentially damaging stimuli in order to develop an effective long term self-preservation strategy. The robot needs to reason about future states in order to maximize its longevity by incurring minimal damage over time.

In order to approach the development of a fully integrated robotic-pain system that promotes robotic well-being and autonomy, scenarios such as the two described above are required for comparison and testing purposes.

1.6 Conclusions

Biological pain is an adaptive trait produced by natural selection that promotes homeostatic behavior by influencing the way organisms learn, empathize, remember, and fear. Feedback that results from pain is essential to the biological fitness of many organisms. Robotic-pain systems that emulate the benefits of biological pain systems have the potential to minimize negative interactions between a robot and its environment.

Chapter 2

TEMPORAL FEATURE LEARNING

2.1 Introduction

Biological pain provides a system that allows animals to learn causal relationships between current environmental states and future potentially harmful environmental states. Animals actively avoid most painful experiences. Inspired by the benefits that result from the ability of animals to predict and avoid painful experiences, the goal is to develop technologies that provide this same benefit to robots. The development of these technologies likely require a data driven approach to learning temporal relationships in a given environment. In this work, the machine learning technique chosen to learn temporal correlations is sequential modeling by the use of Recurrent Neural Networks (RNNs). RNNs learn arbitrarily distant correlations in sequential data. They are used to regress, predict, classify and generate sequential data in almost all machine learning domains. This section discusses many of the widely used, as well as cutting-edge, RNN architectures. This section also analyzes the performance of these architectures on five baseline datasets commonly used to test the capability of RNN models in research. Each model was built from scratch in order to hold certain aspects in the comparison constant. This section ends with a novel model created as part of this research that tested the benefits of disentangling the cell state size from the hidden state size and using linear projection to extract the important features from the cell state or long term memory vector/matrix.

2.2 General Theory

The original recurrent neural network has the form depicted in equation 2.1.

$$h_t^l = \phi(\mathbf{W}^l h_t^{l-1} + \mathbf{V}^l h_{t-1}^l + b^l) \tag{2.1}$$

The input weight matrix $\mathbf{W}^l$ is described by $\mathbf{W}^l \in \mathbb{R}^{N_x \times N_h}$ where N_x is the size of the vector input to layer l and N_h represents the number of neurons in matrices $\mathbf{W}^l$ and $\mathbf{V}^l$. The recurrent weight matrix, bias term and activation functions for layer l described, respectively, by: $\mathbf{V}^l \in \mathbb{R}^{N_h \times N_h}$, $b^l \in \mathbb{R}^{N_h}$, and ϕ. The hidden output of layer l at time-step t is $h_t^l \in \mathbb{R}^{N_h}$. One can view equation 2.1 as the unrolled directed graph below where the dataset $D = \{((x_0^n, y_0^n), ..., (x_{T-1}^n, y_{T-1}^n))\}_{n=0}^{N-1}$.

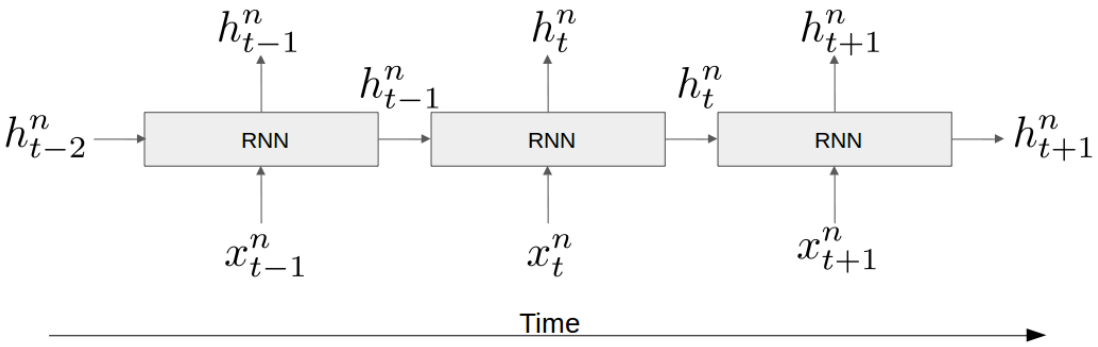

Figure 2.1: Depiction of an RNN unrolled in time

For the tasks presented in section 2.3, the sequences were never truncated in time. With a fully unrolled RNN, the machine learning has the capability to correlate any two inputs in this sequence. Unfortunately, however, a fully unrolled RNN presents a uniquely difficult problem – commonly termed the vanishing and exploding gradients problem – due to the size of the neural network structure. The vanishing and exploding gradient problem is the most well studied problem for recurrent neural networks and it continues to pose the most significant barriers to the performance RNNs.

Equation 2.2 below represents the relationship between the loss function $\mathcal{L}$ and parameters θ of an unrolled network.

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=0}^{T-1} \frac{\partial \mathcal{L}_t}{\partial \theta} \tag{2.2}$$

Equation 2.3 expands the partial derivative using the chain rule and produces the term $\frac{\partial h_t}{\partial h_k}$ which models the temporal gradient from t to k. The vanishing and exploding gradient problem occurs as k << t.

$$\frac{\partial \mathcal{L}_t}{\partial \theta} = \sum_{k=0}^{t} (\frac{\partial \mathcal{L}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k^+}{\partial \theta}) \tag{2.3}$$

In equation 2.4, $\frac{\partial h_t}{\partial h_k}$ can become unreasonably large or small. The two terms that have the most significant effect on the degradation or explosion of $\frac{\partial h_t}{\partial h_k}$, are the derivative of the activation function and the repeated matrix multiplication of $\mathbf{W}_{rec}$.

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k}^{t} \mathbf{W}_{rec} diag\left(\phi'(h_{i-1})\right) \tag{2.4}$$

The models presented in equations 2.5 through 2.11 were developed as means to address the vanishing and exploding gradients problem. Equation 2.5 depicts the most commonly used RNN variant called the Long Short-Term Memory (LSTM). Originally proposed by Hochreiter and Schmidhuber, the LSTM model is very popular because of the fact that it can remember and forget information [17, 13]. It uses complex gating equations in order to make it more easily trainable than the classic RNN model. Two key insights in this model are the creation of a second memory vector called the long term memory $c_t^l \in \mathbb{N}_{\sim}$, and the creation of the forget gate $f_t^l \in \mathbb{N}_{\sim}$, which scales the

previous cell state by a number between 0 and 1, essentially choosing what to forget.

$$
\begin{aligned}
i_t^l &= \sigma(\mathbf{W}_i^l h_t^{l-1} + \mathbf{V}_i^l h_{t-1}^l + b_i^l) \\
f_t^l &= \sigma(\mathbf{W}_f^l h_t^{l-1} + \mathbf{V}_f^l h_{t-1}^l + b_f^l) \\
o_t^l &= \sigma(\mathbf{W}_o^l h_t^{l-1} + \mathbf{V}_o^l h_{t-1}^l + b_o^l) \\
c_t^l &= f_t^l \odot c_{t-1}^l + \sigma(\mathbf{W}_c^l h_t^{l-1} + \mathbf{V}_c^l h_{t-1}^l + b_c^l) \\
h_t^l &= c_t^l \odot tanh(o_t^l)
\end{aligned}
\tag{2.5}
$$

The Peephole LSTM, as shown in equation 2.6, was developed by Gers et al. [14]. This architecture uses the same gating equations as the LSTM architecture; however, h_{t-1} is replaced with c_{t-1} for the recurrent matrix multiplications for calculating the current cell state as well as the input, forget and output gates.

$$
\begin{aligned}
i_t^l &= \sigma(\mathbf{W}_i^l h_t^{l-1} + \mathbf{V}_i^l c_{t-1}^l + b_i^l) \\
f_t^l &= \sigma(\mathbf{W}_f^l h_t^{l-1} + \mathbf{V}_f^l c_{t-1}^l + b_f^l) \\
o_t^l &= \sigma(\mathbf{W}_o^l h_t^{l-1} + \mathbf{V}_o^l c_{t-1}^l + b_o^l) \\
c_t^l &= f_t^l \odot c_{t-1}^l + \sigma(\mathbf{W}_c^l h_t^{l-1} + \mathbf{V}_c^l c_{t-1}^l + b_c^l) \\
h_t^l &= c_t^l \odot tanh(o_t^l)
\end{aligned}
\tag{2.6}
$$

The Gated Recurrent Unit (GRU), as shown in equation 2.7, was developed by Cho et al. [7]. The GRU model is arguably the second most widely used RNN variant. It uses gating equations very similar to that of the LSTM; however, it combines the input and forget gates in the LSTM into a single update gate z_t^l.

$$
\begin{aligned}
z_t^l &= \sigma(\mathbf{W}_z^l h_t^{l-1} + \mathbf{V}_z^l h_{t-1}^l + b_z^l) \\
r_t^l &= \sigma(\mathbf{W}_r^l h_t^{l-1} + \mathbf{V}_r^l h_{t-1}^l + b_r^l) \\
h_t^l &= (1 - z_t^l) \odot h_{t-1}^l + z_t^l \odot tanh\left(\mathbf{W}_h^l h_t^{l-1} + \mathbf{V}_h^l (r_t^l \odot h_{t-1}^l) + b_h^l\right)
\end{aligned}
\tag{2.7}
$$

Equations 2.8 and 2.9 are recent models proposed by Collins et al. at Google Brain [8]. These models were developed by studying capacity and trainability of RNN models. Equation 2.9 was shown to work better for deeper architectures and equation 2.8 was shown to work better for shallower architectures.

$$
\begin{aligned}
c_t^l &= tanh(\mathbf{W}_c^l h_t^{l-1} + \mathbf{V}_c^l h_{t-1}^l + b_c^l) \\
g_t^l &= \sigma(\mathbf{W}_g^l h_t^{l-1} + \mathbf{V}_g^l h_{t-1}^l + b_g^l) \\
h_t^l &= g_t^l h_{t-1}^l + (1 - g_t^l) c_t^l
\end{aligned}
\tag{2.8}
$$

$$
\begin{aligned}
y_{in}^l &= ReLu(\mathbf{W}_y^l h_t^{l-1} + \mathbf{V}_y^l h_{t-1}^l + b_y^l) \\
h_{in}^l &= tanh(\mathbf{W}_h^l h_t^{l-1} + \mathbf{V}_h^l h_{t-1}^l + b_h^l) \\
g_y^l &= \sigma(\mathbf{W}_{gy}^l h_t^{l-1} + \mathbf{V}_{gy}^l h_{t-1}^l + b_{gy}^l) \\
g_h^l &= \sigma(\mathbf{W}_{gh}^l h_t^{l-1} + \mathbf{V}_{gh}^l h_{t-1}^l + b_{gh}^l) \\
y_t^l &= g_y^l h_t^{l-1} + (1 - g_y^l) y_{in}^l \\
h_t^l &= g_h^l h_{t-1}^l + (1 - g_h^l) h_{in}^l
\end{aligned}
\tag{2.9}
$$

The IRNN model, proposed by Le et al. and shown in equation 2.10, addresses the vanishing and exploding gradient problem from a weight initialization and activation function standpoint [22]. It was previously noted in equation 2.4, the two primary contributing factors to the vanishing and exploding gradients problem are the repeated product of both the recurrent weight matrix and the derivative of the activation function. The IRNN model uses rectified linear unit (ReLu) activation in order to force the derivative of the activation function to equal one (only, however, for values above zero). The IRNN model initializes the recurrent weight matrix to be

the identity matrix.

$$h_t^l = ReLu(\mathbf{W}^l h_t^{l-1} + \mathbf{V}^l h_{t-1}^l + b^l) \tag{2.10}$$

The IndRNN model, proposed by Li et al. and shown in equation 2.11, addresses the vanishing and exploding gradient in a very similar way to the IRNN model [25]. It uses ReLu for the activation function to prevent neuron saturation and gradient decay. This model also forces the recurrent weight matrix to be a recurrent weight vector. The IndRNN model replaces the matrix multiplication between the previous hidden output and the recurrent weight vector with the Hadamard product. This analogous to using an infinitely strong prior on certain locations of the recurrent weight matrix or as a scaling function of past features.

$$h_t^l = ReLu(\mathbf{W}^l h_t^{l-1} + V^l \odot h_{t-1}^l + b^l) \tag{2.11}$$

2.3 Tasks and Results

The original RNN and the seven variants discussed above are compared and tested on five benchmark datasets that are commonly used in research. Below, a description of all five tasks is presented and the training and validation results for each. All of the models below were built specifically for this research (in house) using the PyTorch library. Ten random seeds of each model were run for ten epochs. The validation error and training error were recorded at each epoch. For the XOR, SeqMNIST and PSeqMNIST, the training and validation accuracies were also recorded. Every model's recurrent weight matrix was initialized according to the model. If no initialization was specified for an input weight matrix, then that weight matrix was initialized to Xavier Normal. If no initialization was specified for a recurrent weight matrix then that matrix was initialized using orthogonal initialization. A learning rate of .0001

and the Adam optimizer were used for training [19]. A three layer neural network was used for all models. The first two layers are identical copies of the chosen RNN variant, each with 50 neurons as its hidden state size. Each model had an output layer whose shape, output activation, and loss function was uniform across every model variant for a specific task, but unique for the task at hand. Visualized in figures 2.2-2.17, are the validation and training comparison results for each task for all variants. Each colored line represents the mean loss or accuracy at the specified epoch for the specific RNN variant. The colored region represents two standard deviations from the mean for each variant at each epoch. The five tasks described below each test an RNN's ability to cope with the vanishing and exploding gradient problem. The loss is computed with the output from the final time-step; there are no auxiliary losses calculated.

Task 0 – the Add Task

The add task is a human generated dataset where two relevant inputs are randomly spaced in a temporal sequence of length T. T was chosen to be 50 for this task. The output at the end of the sequence is the sum of the two randomly chosen locations for each full sequence. The input at every time-step x_t is defined to be a vector of size two with a randomly generated real value between 0 and 1 in position zero and a boolean value in position one. The boolean is set to zero for all inputs except the two randomly chosen inputs to sum. Figures 2.2 and 2.3 below shows the training and validation error on the add task for ten seeds of all of the variants above. The mean and two standard deviations from the mean computed from all ten seeds for each variant at each epoch is depicted.

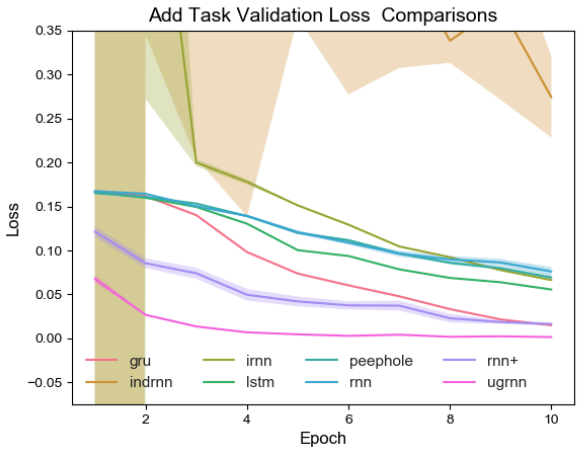

Figure 2.2: Validation Loss

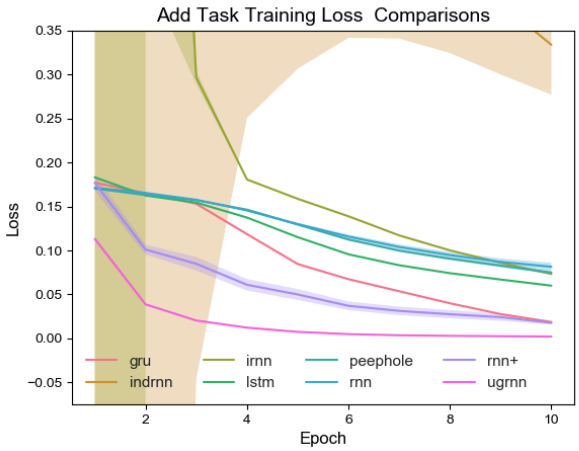

Figure 2.3: Training Loss

Task 1 – the Multiply Task

The multiply task is a human generated dataset where two relevant inputs are randomly spaced in a temporal sequence of length T. The output at the end of the sequence is the product of the two randomly chosen signal locations for each full sequence. The input at every time-step x_t is defined to be a vector with a randomly generated real value in index zero and a boolean value in position one. The boolean is set to zero for all inputs except the two randomly chosen signal inputs. Figures 2.4 and 2.5 below shows the training and validation error on the multiply task for ten seeds of all of the variants above. The mean and two standard deviations from the mean computed from all ten seeds for each variant at each epoch is depicted.

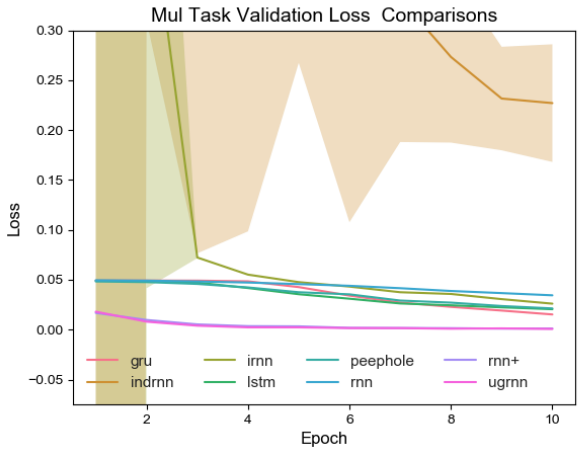

Figure 2.4: Validation Loss

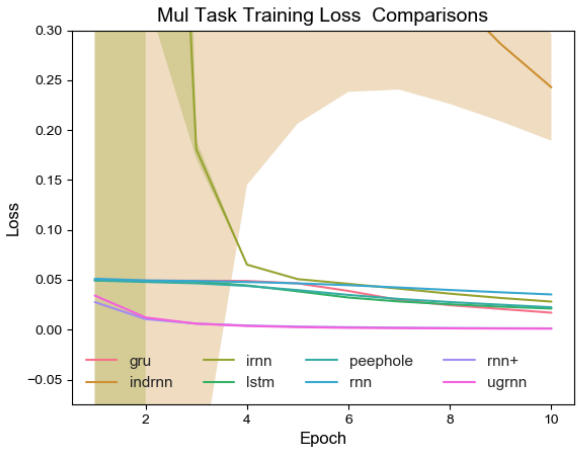

Figure 2.5: Training Loss

Task 2 – the XOR Task

The XOR task is a human generated dataset where two relevant inputs are randomly spaced in a temporal sequence of length T. The output at the end of the sequence is the XOR of the two randomly chosen signal locations for each full sequence. The input at every time-step x_t is defined to be a vector with a randomly generated boolean at position zero and a boolean value in position one. The boolean at position one is the signal that determines if the algorithm should XOR that vector's position zero boolean. The boolean at position one is set to zero for all inputs except the two randomly chosen signal inputs. Figures 2.6-2.9 below shows the training and validation error and accuracy on the XOR task for ten seeds of all of the variants above. The mean and two standard deviations from the mean computed from all ten seeds for each variant at each epoch is depicted.

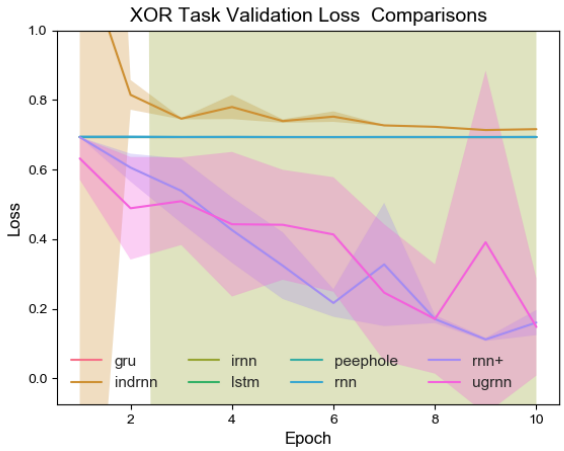

Figure 2.6: Validation Loss

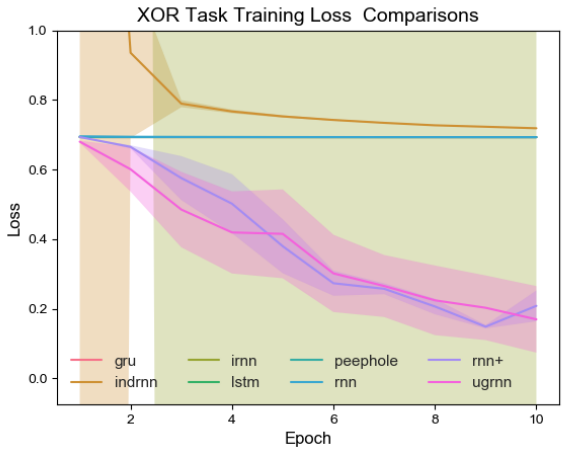

Figure 2.7: Training Loss

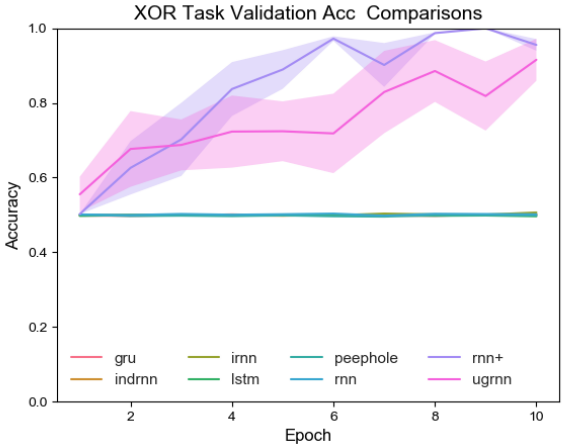

Figure 2.8: Validation Acc

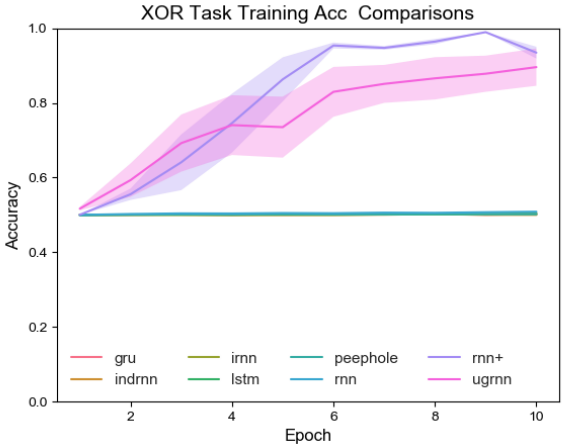

Figure 2.9: Training Acc

Task 3 – the Sequential MNIST Task

The Sequential MNIST task (SeqMNIST) is a variant of the classical MNIST task proposed by Lecun et al. [23]. Sequential MNIST is a dataset where you read one pixel at a time from the top left corner of the one channel MNIST image to the bottom right corner pixel. The input is a real valued scalar and the output is the softmax prediction of the number hand written in the image. Each sequence in this

data is 784 time-steps long where temporal correlations must be found to correctly classify the digit in the original image at the last time-step. Figures 2.10-2.13 below shows the training and validation error and accuracy on the Sequential MNIST task for ten seeds of all of the variants above. The mean and two standard deviations from the mean computed from all ten seeds for each variant at each epoch is depicted.

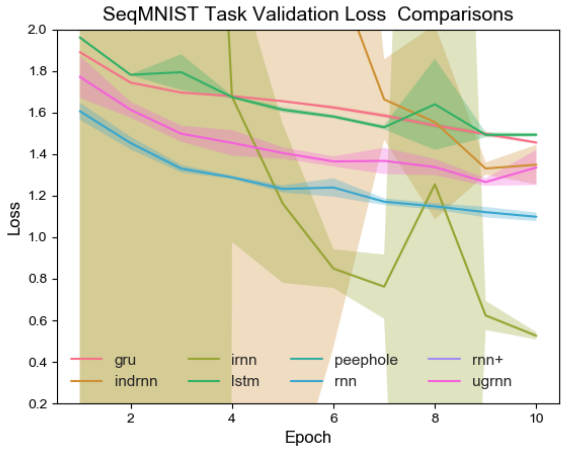

Figure 2.10: Validation Loss

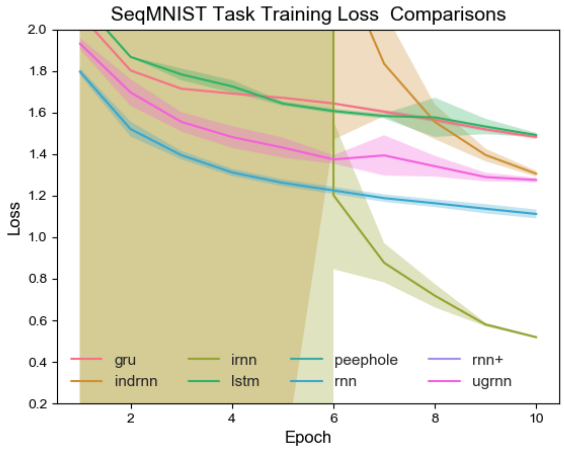

Figure 2.11: Training Loss

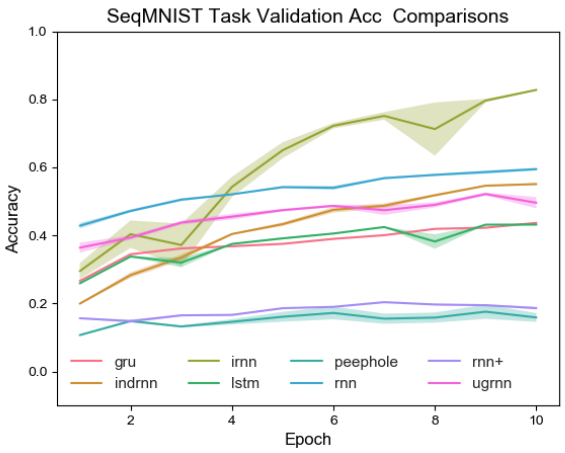

Figure 2.12: Validation Acc

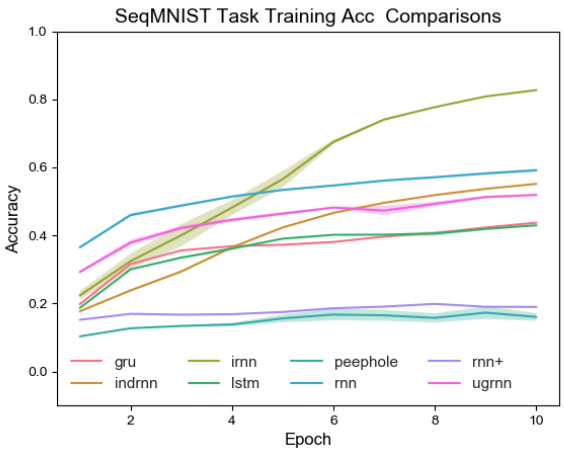

Figure 2.13: Training Acc

Task 4 – the Permuted Sequential MNIST Task

Task number four is Permuted Sequential MNIST (PSeqMNIST). PSeqMNIST, along with sequential MNIST, is one of the standard real-world datasets for validating RNN capability. The setup of this dataset is similar to that of Sequential MNIST. Before reading the sequence of pixels, however, a permutation matrix reassigns each pixel to a new location. The permutation matrix, while randomly generated before training, is constant for the entirety of the training and testing process. This dataset, while more difficult to learn, still contains all of the relevant information about the original image. Figures 2.14-2.17 below shows the training and validation error and accuracy on the Permuted Sequential MNIST task for ten seeds of all of the variants above. The mean and two standard deviations from the mean computed from all ten seeds for each variant at each epoch is depicted.

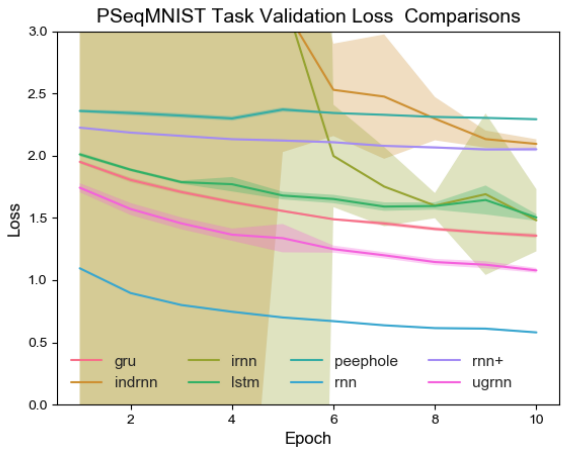

Figure 2.14: Validation Loss

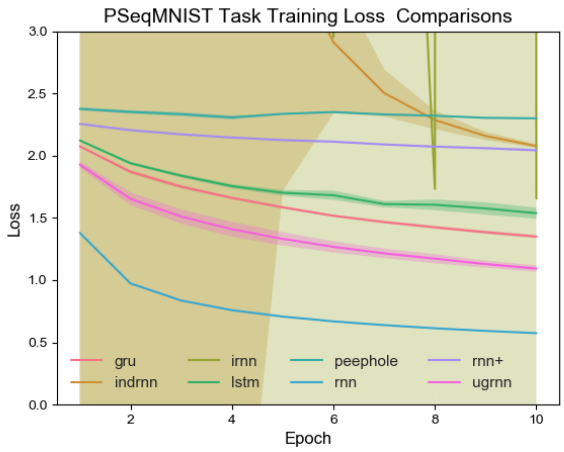

Figure 2.15: Training Loss

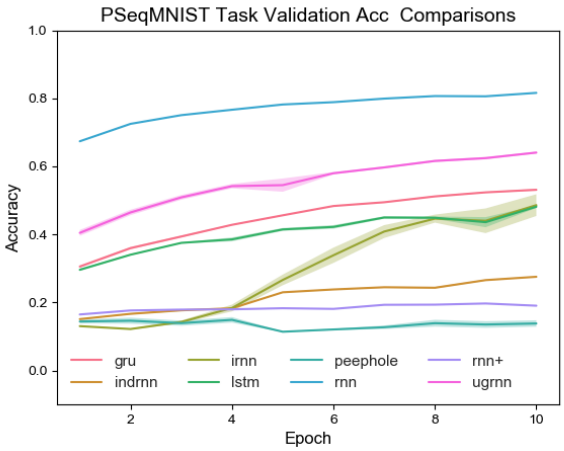

Figure 2.16: Validation Acc

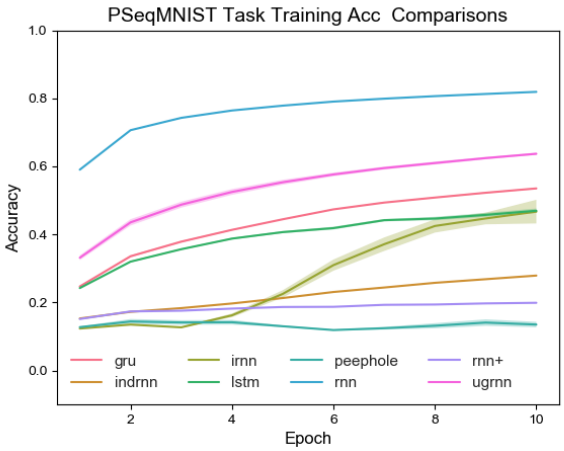

Figure 2.17: Training Acc

2.4 Disentangle Cell State Size for Recurrent Neural Networks

Recurrent neural network architectures generally have either a fixed memory state or multiple memory states. This section discusses a hypothesis that was tested and found to be false. The hypothesis was that if the long-term memory could be disentangled from the hidden vector's output size, then the recurrent memory vector would be able to store more information for more a more accurate inference capability.

2.4.1 SVDRNN Model and Theory

The model developed during my research to study the hypothesis above will be referred to as the SVDRNN model. This model uses the basic format of the LSTM. The SVDRNN, however, projects the input into a $\mathbb{R}^{m \times m}$ matrix where m is the hidden states size of the recurrent network. The network uses singular value decomposition on the cell state at every time-step and extracts singular values from the current long-term memory – $\mathbf{C}_t$ – at every time-step in order to produce output h_t. The activation applied to the singular values of Σ is changed from the hyperbolic tangent to a

sigmoid function in order to constrain h_t to be non-negative. $\mathbf{U}_{t-1}$ and $\mathbf{V}_{t-1}^T$ are both guaranteed to be orthonormal because they are the analytical outputs from the previous time-step's singular value decomposition. With both of these conditions satisfied, it is ensured that $\mathbf{U}_{t-1}diag(h_t^{l-1})\mathbf{V}_{t-1}^T$ represents the singular value decomposition of a unique matrix M.

$$\begin{aligned}
\mathbf{I}_t^l &= \sigma\left(\mathbf{W}_i^l diag(h_t^{l-1}) + \mathbf{U}_{t-1} diag(h_{t-1}^l)\mathbf{V}_{t-1}^T + b_i^l\right) \\
\mathbf{F}_t^l &= \sigma\left(\mathbf{W}_f^l diag(h_t^{l-1}) + \mathbf{U}_{t-1} diag(h_{t-1}^l)\mathbf{V}_{t-1}^T + b_f^l\right) \\
o_t^l &= \sigma(\mathbf{W}_o^l h_t^{l-1} + \mathbf{V}_o^l h_{t-1}^l + b_o^l) \\
\mathbf{C}_t^l &= \mathbf{F}_t^l \odot \mathbf{C}_{t-1}^l + \mathbf{I}_t^l \odot \sigma\left(\mathbf{W}_c^l diag(h_t^{l-1}) + \mathbf{U}_{t-1} diag(h_{t-1}^l)\mathbf{V}_{t-1}^T + b_c^l\right) \\
\mathbf{U}_t, \Sigma, \mathbf{V}_t^T &= SVD(\mathbf{C}_t^l) \\
h_t^l &= o_t^l \odot \sigma(\Sigma)
\end{aligned} \tag{2.12}$$

2.4.2 Results

This SVDRNN failed to improve upon previous approaches at sequential modeling. The model merely learned to predict the average of the dataset. This model performs no more accurately than a feed forward network trained on only the last input of the dataset.

2.5 Conclusions

Recurrent neural networks provide an incredibly strong framework for analyzing sequential data and learning temporal correlations between distant events. RNNs are currently one of the most promising frameworks for modeling sequential data. The primary problem to address in order to improve the performance of RNNs is the van-

ishing and exploding gradient problem. Numerous architectures have been developed to address this problem. Comparing these modern RNN architectures allowed for us to see that some solutions vastly outperform others. Specifically, the most ubiquitous variant, the LSTM did perform best on any of the tasks tested. Allowing for a larger cell state and using singular value decomposition to extract temporal features from the cell state does not improve upon the current state of the art for recurrent neural networks. Future research in this area should consider unsupervised ways to promote stable information flow in unrolled networks. Additional work should be undertaken to clarify mathematically the reasons for performance differences among RNN variants.

Chapter 3

PERTURBATION DETECTION

3.1 Introduction

The development of a robotic-pain system necessarily requires a means for the robot to detect harmful interactions with its environment. This aspect of robotic-pain is inspired by the nociceptive systems in biological organisms. The goal of the algorithm presented in this chapter is to determine if and when a negative harmful interaction occurred between a robot and its environment. As is the case with biological pain, the algorithm used does not need any external information outside of the robot's ability to internally query its own state. The feedback given from this algorithm is temporal and can be used as a conditioning signal that relates environmental information with harmful states. This chapter applies a perturbation detection algorithm to detect when a projectile has come into contact with a robot. The approach presented in this chapter uses a self-supervised Bayesian prediction method for determining harmful collisions. No external force sensors are needed to detect collisions. The approach used was previously presented by Sur and Ben Amor [33]. This section will describe the robot used for experimentation, the experimental task used to validate the strengths and weaknesses of the perturbation detection system, the theoretical aspects of the perturbation detection algorithm, the evaluation of the machine learning models trained, the thresholding selection considerations, and conclusions about the efficacy of this approach.

3.2 Task

For this research a novel scenario referred to in this paper as the dodge ball scenario was created. In this scenario, N_b different balls with colors C_b are sent into projectile motion at the beginning of the scenario. Each ball is initialized with a random normal position unique to each ball $P_b \in \mathbb{R}^3$ and random uniform velocity $V_b \in \mathbb{R}^3$. The robot is bimodal and can only move in two directions. There is a significant amount of noise in the robot movement, however, and slipping and falling off the robot's initial lateral axis is normal and frequent behavior. On average, it takes less than two seconds for the balls to reach the robot. The robot state vector and action vector are modeled by $s_t \in \mathbb{R}^S$ and $a_t \in \mathbb{R}^A$. Below is a depiction of the scene in action. S includes the xyz position, orientation, velocity and angular velocity of the robot. A is discrete and contains five possible actions, full speed right (action 0) half speed right (action 1) don't move (action 2) half speed left (action 3) and full speed left (action 4). V-REP was chosen as the robotic simulation platform and the Linetracer robot was selected due to its speed and ability to change its position and velocity quickly.

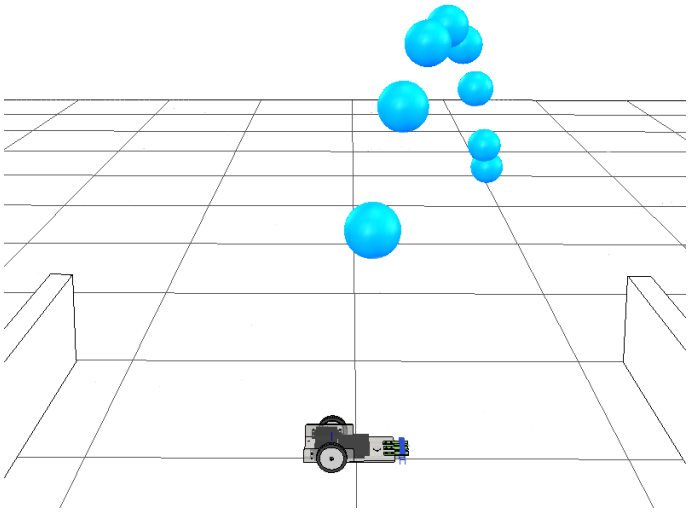

Figure 3.1: Dodge Ball Task

3.3 Deep Dynamics

The deep dynamics neural network model presented in this section is part of the final perturbation detection algorithm presented in this chapter.

3.3.1 Theory and Model

A feed forward neural network is used to regress future states of the robot. Specifically, the deep dynamics model used in this research is defined by the function $\hat{s}_{t+1} = f(s_t, a_t; \theta)$. Model parameters θ are learned using batch gradient descent via the Adam optimizer [19]. The input to the neural network is the current state and action of the robot at time t and the output of the network is the predicted next state of the robot. Dropout is used at every layer of the neural network except for the output layer [32]. The neural network has five layers and hyperbolic tangent activations were used in the selected model.

3.3.2 Training

The data for training this neural network was collected by allowing the robot to sample random actions in a safe environment. 200,000 state-action pairs were collected for training validation and testing. Of 200,000 state-action pairs, 80% were randomly selected for training, 10% were randomly selected for validation and the remaining 10% for testing. Three neuron architectures were used for θ. The number of neurons for the three variants tested, in order, from layer zero to layer four are as follows: $\theta_0 = [60, 40, 30, 20, 20]$; $\theta_1 = [120, 80, 60, 40, 40]$; $\theta_2 = [30, 20, 15, 10, 10]$. Two activation variants were analyzed, LeakyReLu, and the hyperbolic tangent function. Three dropout rates were tested for model selection, 0.6, 0.45 and 0.3. All combinations of the aforementioned variables were tested, yielding a total of 18 experiments. Mean squared error was used as the cost function for training. Each experiment used Xavier Normal Initialization and was trained for 250 epochs.

3.3.3 Evaluation

The model with the lowest validation loss over all 250 epochs for each of the model variants was selected for further analysis, as discussed in section 3.4.2. Model 17 was found to be the model that best met the requirements for perturbation detection. Figure 3.2 depicts the training and validation error over all 250 epochs for model 17.

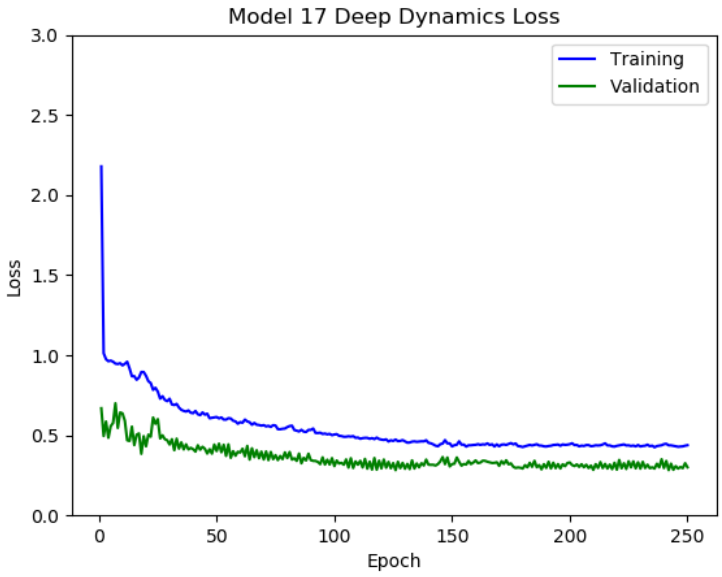

Figure 3.2: Training and Validation Loss

3.4 Detecting Perturbations

The deep dynamics model learns the functional mapping from the robot's current state and action to the robot's next state. This does not solve, however, the original problem which is to develop a system that detects significant and probably harmful perturbations caused by the environment. The following discusses the theory behind implementing such a system and validates use of such system on the dodge ball scenario.

3.4.1 Theory

In order to determine if a sufficiently significant perturbation has occurred to warrant being deemed harmful, the deep dynamics model is used to create a set of beliefs about the next state which is compared to the ground truth. In order to create a set of beliefs over $\hat{s}_{t+1}$, a method commonly termed stochastic forward passes was

used [12]. This method uses dropout at inference time in order to sample subnetwork predictions over $\hat{s}_{t+1}$. A total of 64 stochastic forward passes were used at inference time. If the difference between the predicted belief set over $\hat{s}_{t+1}$ and the ground truth is sufficiently large –with the threshold being defined by the user– that time-step is labeled as a harmful perturbation. Using the set of predictions collected by using stochastic forward passes, we can approximate the expected value for $\hat{s}_{t+1}$ and the variance of $\hat{s}_{t+1}$. Equations 3.1 and 3.2 below define how to calculate the mean and variance of our belief set of $\hat{s}_{t+1}$.

$$\mathbb{E}[\hat{s}_{t+1}] \approx \frac{1}{N_s} \sum_{i=1}^{N_s} \hat{s}_{t+1}^{i} \tag{3.1}$$

$$Var[\hat{s}_{t+1}] \approx \mathbb{E}[(\hat{s}_{t+1})^2] - \mathbb{E}[\hat{s}_{t+1}]^2 \tag{3.2}$$

Finally an exponentially smoothed norm, as depicted in equations 3.3 and 3.4, is used to produce the final predicted error output. This predicted error output represents the difference between the predicted belief set and the ground truth. The norm calculated from equation 3.4 provides insight into the severity of the perturbation that occurred.

$$\delta = \mathbb{E}(\hat{s}_{t+1}) - s_{t+1} \tag{3.3}$$

$$\begin{aligned} \tau_i &= e^{\delta_i - 2Var(\hat{s}_{t+1})_i} \\ \Delta &= ||\tau_i|| \end{aligned} \tag{3.4}$$

3.4.2 Evaluation

Most of the 18 models trained had significant deviations in their perturbation output –in other words, the predicted error– when undergoing a collision. There were some differences between models and ultimately model 17 was found to be the best performing model based on the following standards. In the dodge ball scenario, let a false positive be defined by classifying a simulation where no collision occurred as a hit simulation. Let a false negative be defined by a simulation where a hit occurred and it was labeled a miss. This research based its model selection on the following criteria. First, a threshold was selected above the largest reported predicted error for each model on its evaluation set, essentially eliminating false positives. Then all models were compared by the reported false negatives. Ultimately, model 17 was chosen with 0 false positives reported and 10 false negatives reported over 150 simulations. This model had the fewest false negatives given the threshold determined. The histogram shown below is model 17's reported largest predicted error output for all full trajectories for 150 simulations. The false positive threshold selected for this model was 250. Any output above this threshold, would be classified as damaging and harmful to the robot. Figure 3.3's x axis logarithmically scaled by 2^x due to the predicted error's large outputs.

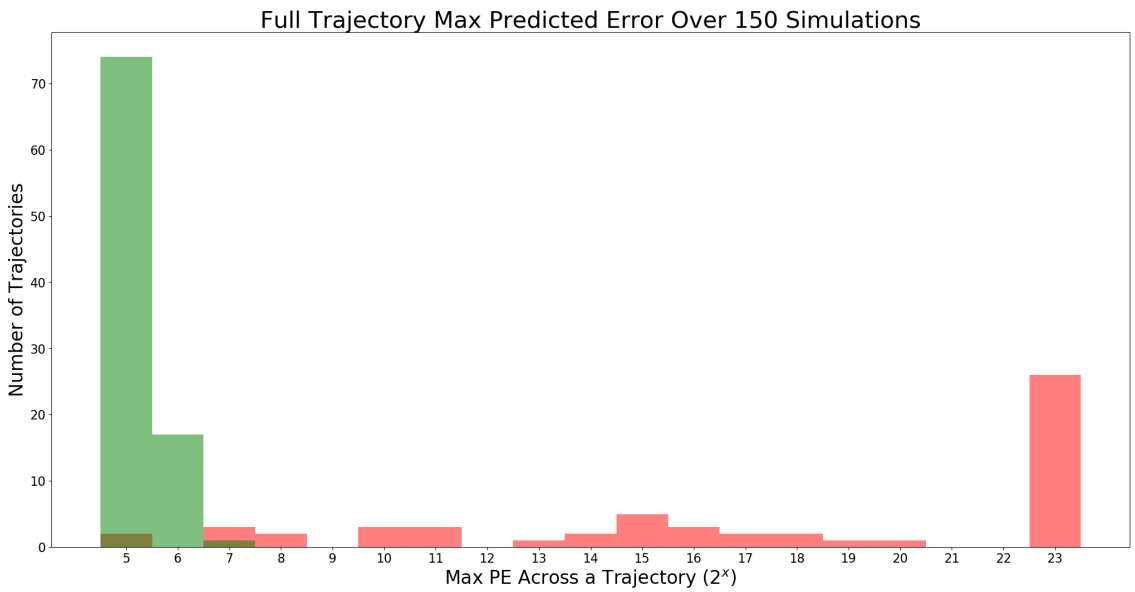

Figure 3.3: Model 17's Max Predicted Errors Over 150 Simulations

Figures 3.4 - 3.8 represent the output of the predicted error (in blue) vs the scaled ground truth (in red). In order to properly visualize the time-steps where the ball was in contact with the robot along side the predicted error output, the ground truth was scaled. In figure 3.7, a collision occurred but the output of the deep dynamics model did not go above the threshold of 250. This was a false negative and it is possible that this collision was not harmful. It is also possible, if not probable, that this model misclassified this trajectory and the collision was harmful. What is significant about model 17, however, is that model 17 properly classified the vast majority of collisions and the predicted error output showed significant spread between hits and misses – in other words, hits were usually orders of magnitude larger in their predicted error than misses.

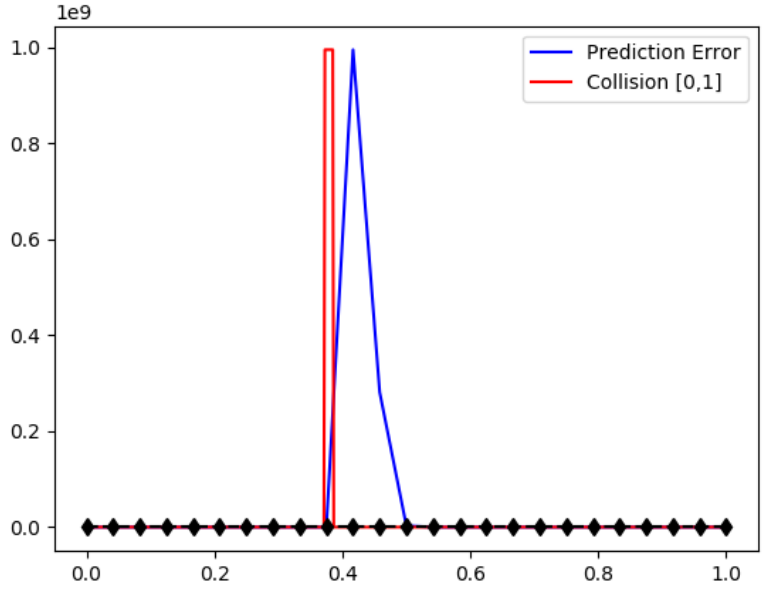

Figure 3.4: Predicted Error Vs Ground Truth

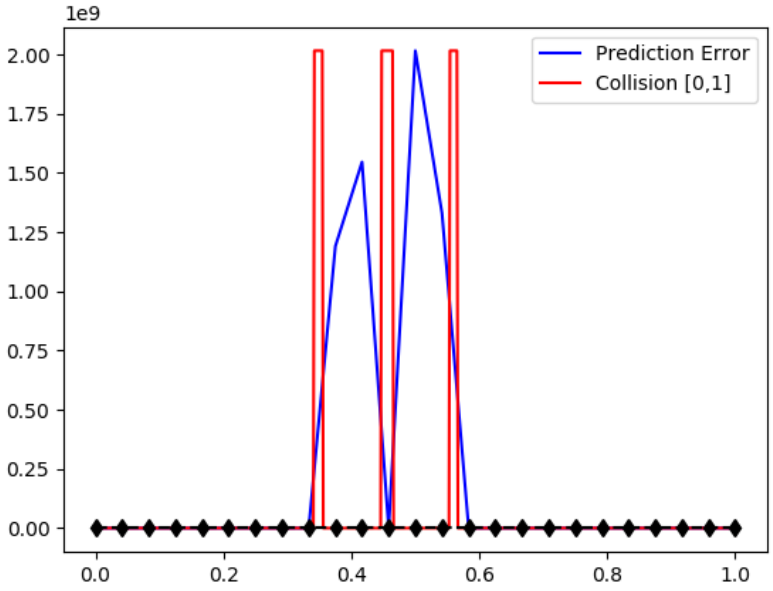

Figure 3.5: Predicted Error Vs Ground Truth

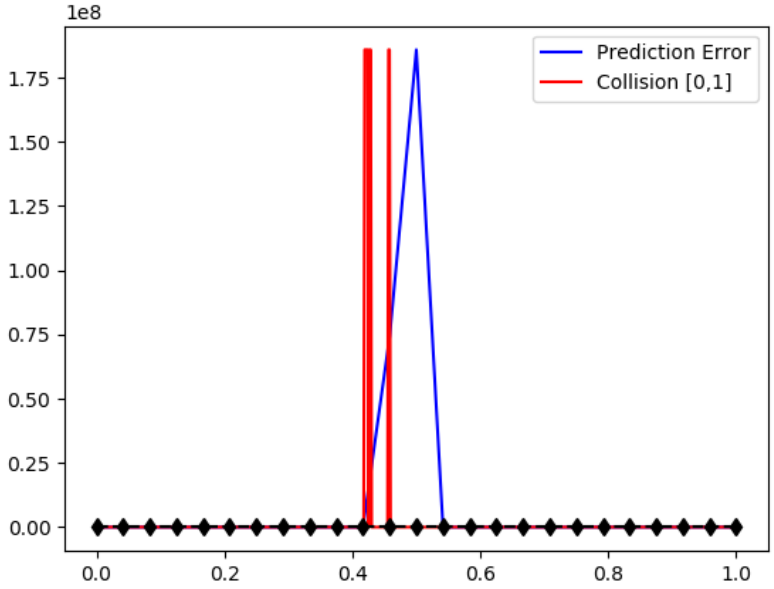

Figure 3.6: Predicted Error Vs Ground Truth

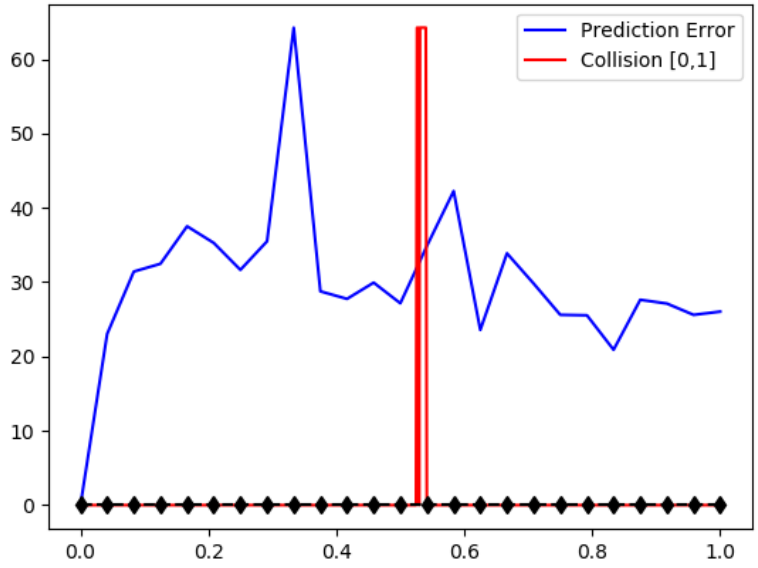

Figure 3.7: Predicted Error Vs Ground Truth

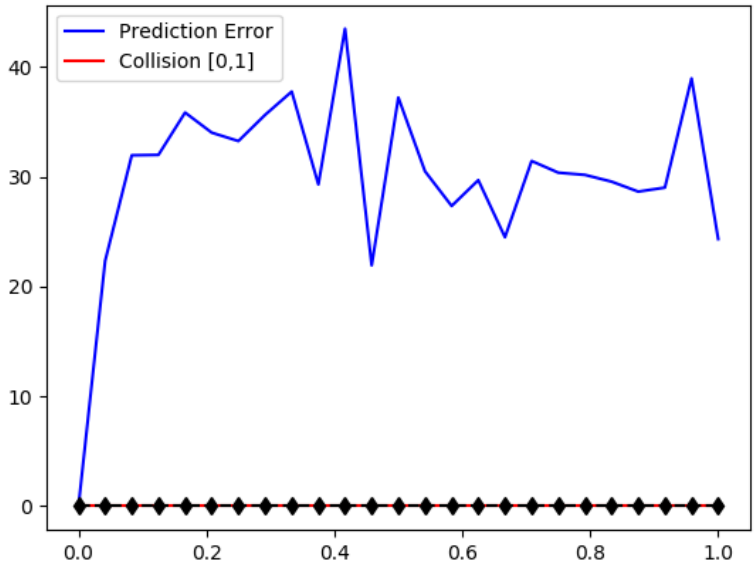

Figure 3.8: Predicted Error Vs Ground Truth

3.5 Conclusions

In this chapter, a perturbations detection algorithm was described, implemented and evaluated using the dodge ball scenario. The dodge ball scenario was created as part of this research for this purpose. Variants of the deep dynamics model were tested and the model selection technique was described. It has been shown that this algorithm can effectively detect collisions as external perturbations despite the randomness present in the robot's movement. The clear division between the average hit predicted error and miss predicted error trajectories provided an accurate threshold. Some drawbacks to this approach include the fact that a threshold was defined using the ground truth. Future research should focus on eliminating this limitation and instead use unsupervised methods to distinguish which values are sufficiently large to be considered harmful collisions. Future research should also consider including using

sensors on the robot to detect different types of harmful environments such as heat, chemical makeup, forces and or electrical signals. Finally, future research should also look at how to design datasets in order for the perturbation detection algorithm to generalize to new and different environments. One can imagine this system has the potential to report many false negatives if the deep dynamics model is trained on flat ground and the robot finds its way to a very different environment such as sand.

Chapter 4

COLLISION ANTICIPATION

4.1 Introduction

Pain allows biological organisms to make important associations and engage in conditioned responses to varying stimuli in their environment. Fear-based conditioning and avoidance learning enhance an organism's ability to navigate through dangerous environments. Inspired by homeostatic behavior mechanisms in the brain, a deep predictive model that associates visual queues with future internal harm was proposed by Sur and Ben Amor [33]. Using a sliding window of images as input, their model uses convolutional recurrent neural networks to reason about a sequence of images and predict the probability of future collisions. In the research described in this paper, an alternative model – termed the stateful collision anticipation model – is proposed. This model reasons about and predicts the probability of future events given only a current image and recurrent memory vectors of previous images. The stateful collision anticipation model described herein is a supervised model. It uses the outputs from the self-supervised perturbation detection method described in Chapter 3 as label data for the video input data. This model offers three primary improvements as compared to the previous system. First, the stateful collision anticipation model is orders of magnitude faster with an inference rate of 65Hz compared to the previous, which operates at an inference rate of 5Hz. Second, the run-time memory requirement for the stateful collision anticipation model is twenty percent of the same requirement for the prior model. Third, the stateful collision anticipation model is capable of learning relationships between inputs arbitrarily spaced in time whereas the

previous model could only reason about temporal relationships within five consecutive images. The construction of a stateful collision anticipation model was necessary in order to solve the dodge ball task with one projectile and make significant improvements towards solving the general dodge ball task or n projectiles. Speed, accuracy, minimal memory requirements and spatiotemporal feature extraction were all necessary to achieve these results. Discussed below are the theoretical aspects, training procedures and evaluative results for the stateful collision anticipation model applied to the dodge ball task.

4.2 Theory

The objective of the collision anticipation model is to approximate the probability of a future collision, as seen in equation 4.1, given a sequence of images and access to the internal state of the robot.

$$f(x_t;\theta) \approx P(collision|x_t, x_{t-1}, x_{t-2}, ...x_1, x_0) \tag{4.1}$$

The labels for the stateful collision anticipation model emanate from the self-supervised method described in chapter 3. A multilayer ConvLSTM neural network was chosen to as the primary component of the function approximation depicted in equation 4.1 for this task due to its ability to learn a mapping between spatiotemporal data and their labels [37]. The ConvLSTM, depicted by equation 4.2, builds on the LSTM architecture that is referenced in Chapter 2. Inputs $\mathbf{X}_t$ and $\mathbf{H}_{t-1}$, however, are now sets of activation maps in the domain $\mathbb{R}^{H\times W\times C}$ where H, W and C represent, respectively, the height of the input activation map, the width of the input activation map, and the number of input channels. The ConvLSTM architecture also includes element

wise multiplications with weight matrix $\mathbf{U}$ and its cell state or long term memory.

$$
\begin{aligned}
\mathbf{I}_t^l &= \sigma(\mathbf{W}_i^l * \mathbf{H}_t^{l-1} + \mathbf{V}_i^l * \mathbf{H}_{t-1}^l + \mathbf{U}_i^l \odot \mathbf{C}_{t-1}^l + b_i^l) \\
\mathbf{F}_t^l &= \sigma(\mathbf{W}_f^l * \mathbf{H}_t^{l-1} + \mathbf{V}_f^l * \mathbf{H}_{t-1}^l + \mathbf{U}_f^l \odot \mathbf{C}_{t-1}^l + b_f^l) \\
\mathbf{O}_t^l &= \sigma(\mathbf{W}_o^l * \mathbf{H}_t^{l-1} + \mathbf{V}_o^l * \mathbf{H}_{t-1}^l + \mathbf{U}_o^l \odot \mathbf{C}_{t-1}^l + b_o^l) \\
\mathbf{C}_t^l &= \mathbf{F}_t^l \odot \mathbf{C}_{t-1}^l + \sigma(\mathbf{W}_c^l * \mathbf{H}_t^{l-1} + \mathbf{V}_c^l * \mathbf{H}_{t-1}^l + \mathbf{U}_c^l \odot \mathbf{C}_{t-1}^l + b_c^l) \\
\mathbf{H}_t^l &= \mathbf{O}_t^l \odot tanh(\mathbf{C}_t^l)
\end{aligned}
\tag{4.2}
$$

4.3 Training

In contrast to the sliding window approach, use of a stateful ConvLSTM is proposed to predict future collisions. This proved to be necessary to give the robot sufficient time to dodge incoming projectiles. On average, the projectiles reach the robot within two seconds from the start of the simulation. Therefore, if the robot is going to be able to avoid collisions with the projectiles, it needs to reason about its safety quickly and often. In order to accomplish this, a custom-built ConvLSTM was coded in both Tensorflow and PyTorch. Ultimately, due to the ease of recording gradients through time in PyTorch, PyTorch was selected as the library to use for the final model. No gradients were clipped, and as a result, the loss at any time-step t updated the weights with respect to all hidden outputs across time. The dataset used consisted of 2800 videos, 2274 for training, 263 for validating and 263 for testing. The video data collected was of the form $D_i \in \mathbb{R}^{T \times L \times W \times C}$ where T represents the number of input images in the simulation. The number of channels was three due to the fact that the robot had an RGB video sensor on its body. The model consisted of five layers, the first three being ConvLSTM layers. The fourth layer flattened the output of the convolved activation maps from layer three and used a dropout rate of 0.3. The final layer used softmax activation and cross-entropy as the loss function. The

only input to this model was video input and the labels were generated by the self-supervised perturbation detection model described in Chapter 3. The robot's state still needed to be queried as inputs to the perturbation detection algorithm in order to generate the labels for the stateful collision anticipation model. The final model was trained on 81.2% of the data, validated on 9.4% of the data, and tested on 9.4% of the data. The final per-frame test accuracy of the model trained was 91.183%. The per frame accuracy of the stateful model meant that the robot was able to determine the probability of collision with extremely high accuracy by frame 10 in a video sequence of well over 100 frames. Recalling that the ball takes on average about 2 seconds to collide with the robot and the algorithm inferences at 65 Hertz, the model was able to make an accurate prediction within 0.1 seconds, giving the robot ample time to dodge the ball and solve the problem for the one projectile scenario.

4.4 Evaluation

The collision anticipation convolutional recurrent neural network was able to predict future collisions in a testing set of size of 600 at each time-step in the video with an average accuracy of 91.183%. Figure 4.1 below reports the training and validation error for the collision anticipation model.

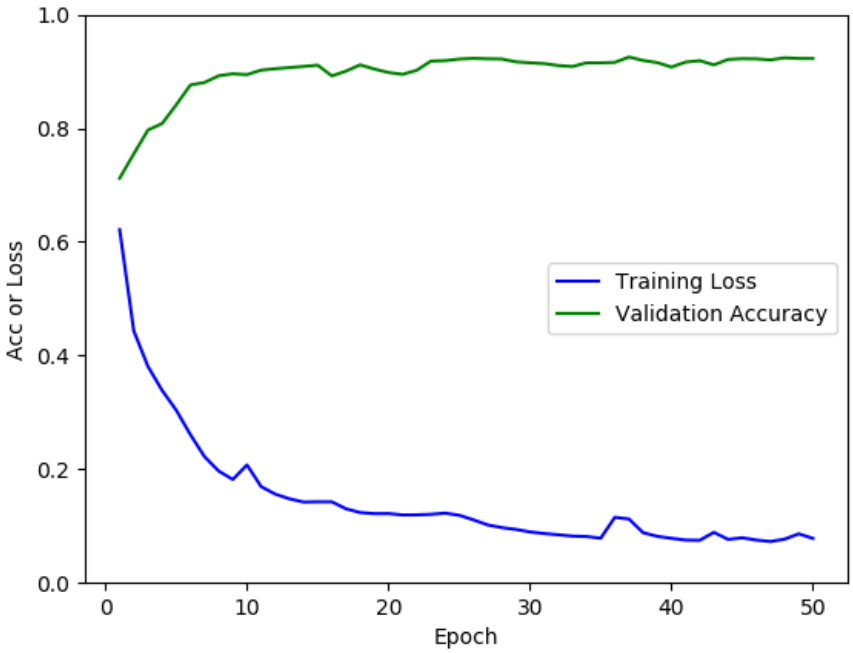

Figure 4.1: Per-Frame Training Loss and Validation Accuracy

In order to better understand what the stateful collision anticipation model was able to learn, the hidden activation maps and cell state activation maps are visualized in figures 4.2 through 4.5. For figures 4.2 through 4.5 below, the top left image is what the robot vision sensor observes at time-step 33. Figure 4.2 represents the hidden activation maps outputted from layer zero at time-step 33 for a randomly selected collision trajectory. White represents high activations (i.e. high spatial attention) given to specified spatial regions of the activation map. Black represents low activation (i.e. low spatial attention) to that region of the activation map. While it is difficult to rigorously define these spatial attention results, it can clearly be seen that different filters learn to focus on the ball, the floor, and the background in the first ConvLSTM layer.

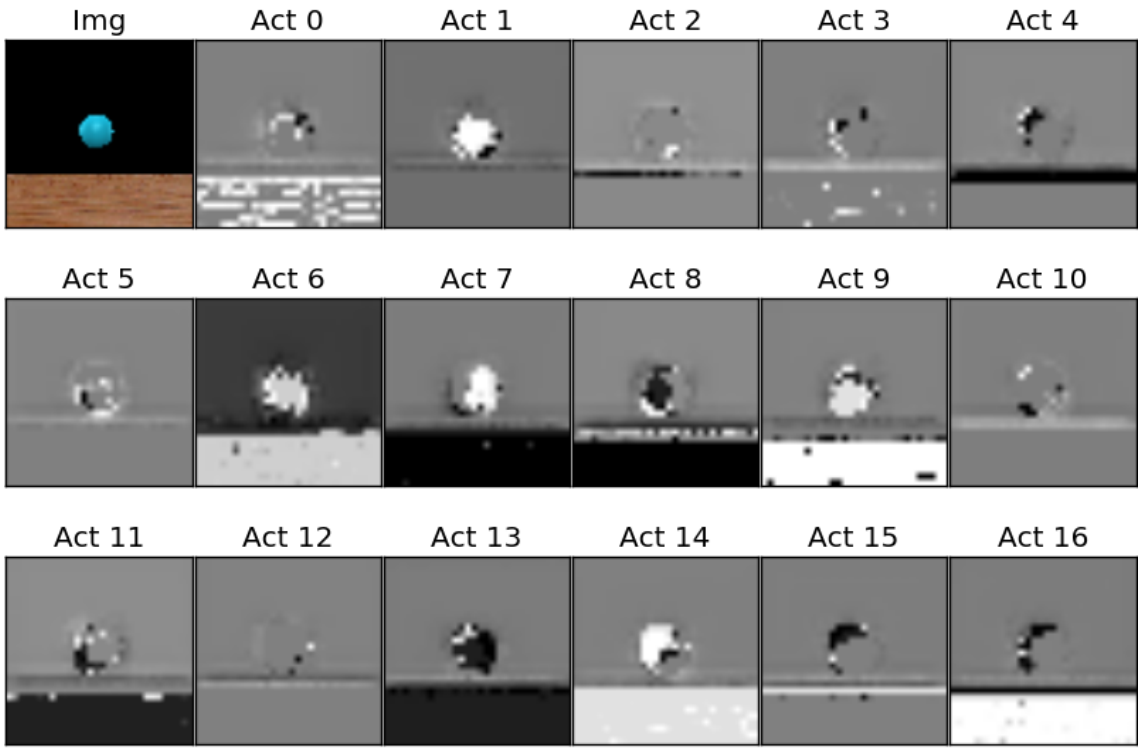

Figure 4.2: Hidden Activations for Layer 0

In order to visualize what the long term memory has learned, the cell states for layer zero at time-step 33 are visualized in figure 4.3. These images indicate that that certain filters learn different aspects about the input feature map.

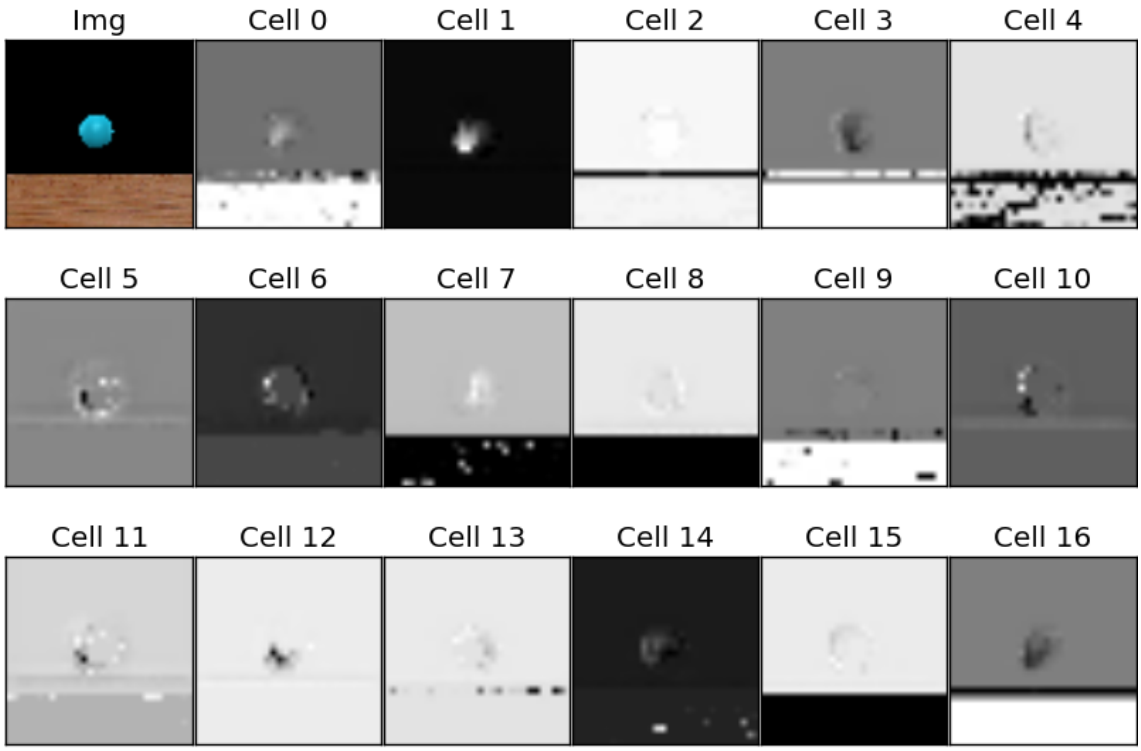

Figure 4.3: Cell State for Layer 0

Activation maps 1 and 7 are especially interesting because they show learned attention current projectile location as well as historical projectile location. The high accuracy of the model allowed for the development of simple deterministic algorithm used for collision avoidance in the one projectile dodge ball task. The simple algorithm developed queried the output of the stateful collision anticipation model at every time-step after the tenth time-step. If the probability of a future collision was significant then the robot would randomly move to the left or the right. If it did not predict the future collision, the robot would remain in place.

One unexpected and interesting result was that the robot was able to generalize its predictions even when the robot was in motion. The robot was trained to predict

future collisions while stationary, but the stateful collision anticipation algorithm maintained accurate predictions during robotic evasive action.

The ability to interpret the activation maps becomes more difficult when higher layers of the neural network are analyzed. This is likely due to the fact that the higher layers encode higher level features of the data in order to optimize its inference capability. These learned encodings are not intended to be inherently descriptive to humans. Figures 4.4 and 4.5 depict the hidden activation outputs and cell states for the final ConvLSTM layer in the stateful collision anticipation model. These layers encode the necessary information to predict future hit with a 91.183% per-frame testing accuracy.

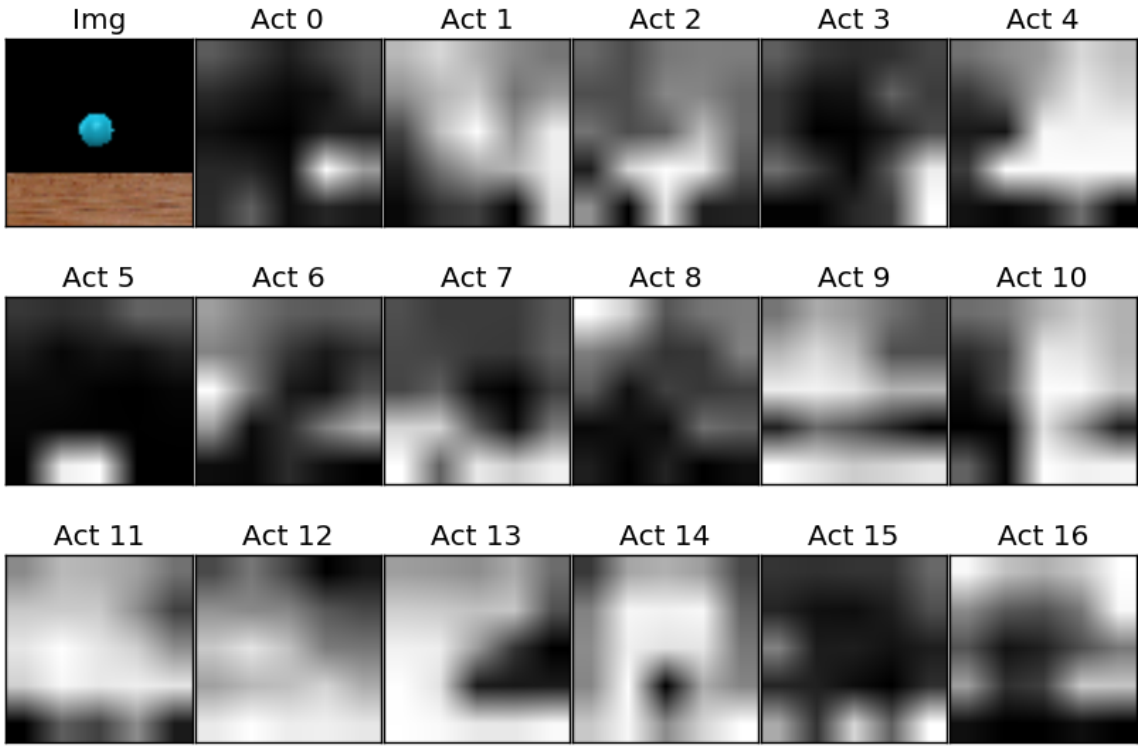

Figure 4.4: Hidden Activations for Layer 2

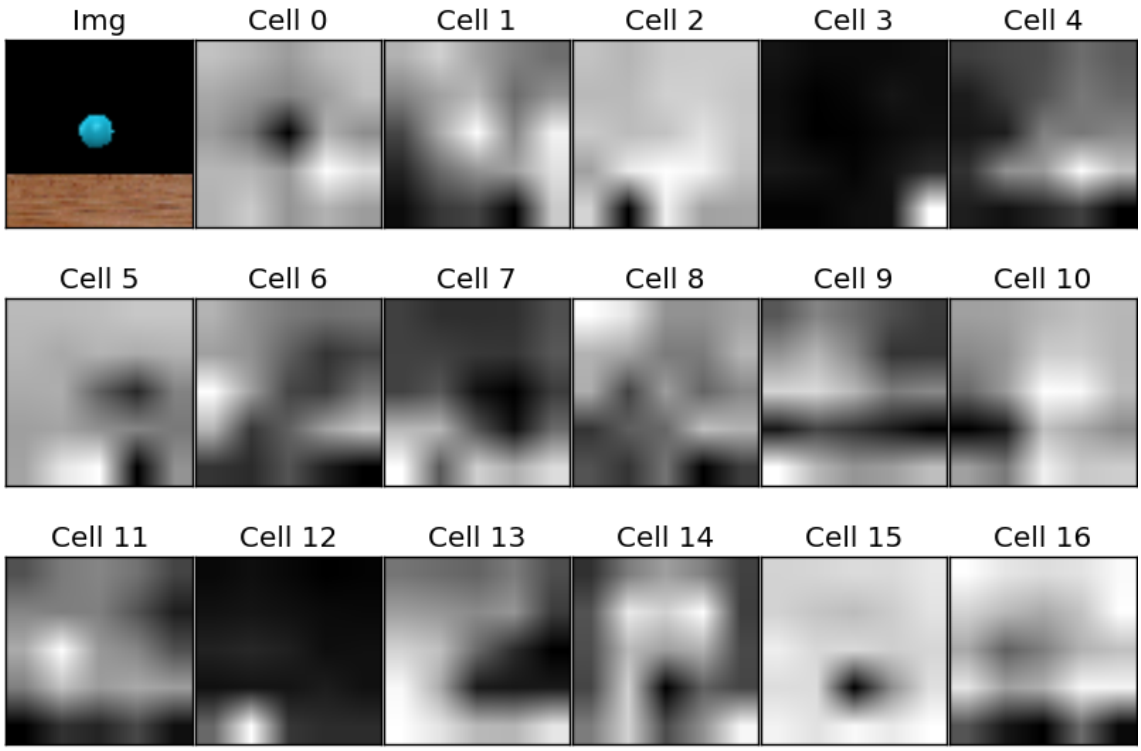

Figure 4.5: Cell State for Layer 2

4.5 Conclusions

The stateful collision anticipation model gains its ability to be useful because the perturbation detection algorithm produces informative labeling for the generated video data. No human hand-crafted labeling technique is needed for the video data. The model previously proposed utilized a sliding window of five time-steps. This model suffered from a slow inference rate of 5 Hertz, large memory requirements and limited capability for learning long-term temporal correlations. This previous model failed to solve the single projectile dodge ball task. Motivated by this model, however, a new model was developed to attempt to address the earlier model's limitations. The new stateful collision anticipation architecture was trained on full video sequences and

therefore could learn temporal correlations during training across the entire timescale. This model also had significant savings in on-board memory as well as inference speed. Admittedly, the stateful collision anticipation model requires many instances of harmful collisions in order to learn the functional mapping between inputs and the probability of collision. Future research should therefore focus on how to minimize the number of collisions required to learn this functional mapping while maintaining the speed, low memory costs, and high accuracy of the stateful collision anticipation model.

Chapter 5

INTRINSIC REWARD POLICY GRADIENT METHODS

5.1 Introduction

Reinforcement learning (RL) is a powerful methodology for teaching autonomous agents complex behaviors and skills. A critical component in most RL algorithms is the reward function – a mathematical function that provides numerical estimates for desirable and undesirable states. This chapter explores and discusses a new way to train policy gradient algorithms using the output from the self-supervised perturbation detection algorithm discussed in Chapter 3 as the reward for each action in a given state. The motivation for this approach is based upon developing new learning algorithms for robots that minimize damaging interactions during machine learning in new environments. This section proposes a new deep reinforcement learning algorithm to attempt to solve the dodge ball task with n projectiles rather than just one. Specifically, the research below was tested on eight projectiles. Hereafter, I will refer to this as the eight projectile dodge ball task or the eight projectile dodge ball scenario. The Monte-Carlo (i.e. stochastic) policy gradient algorithm method was used in the research described in this chapter; however, the proposed intrinsic reward strategy is not specific to the Monte-Carlo policy gradient algorithm. The intrinsic reward strategy presented in this chapter is general to any policy gradient method and may be usable in other reinforcement learning frameworks.

A solution to the eight projectile dodge ball task was not obtained. This research, however, did yield valuable insights. For example, the intrinsic reward policy gradient method was able to converge on a consistent strategy. Also, the task itself may

need further refinement to provide the appropriate platform to test the method proposed. Finally, due to the complexity of the problem space, a simulation environment that allows for running experiments headless, synchronous, and in parallel efficiently appears to be required.

5.2 Model and Theory

The model proposed in this section is called the intrinsic reward deep reinforcement learning method. This method combines the self-supervised perturbation detection model, the semi-supervised collision anticipation model, and policy gradient methods. Specifically, the model used in this research was tested with the stochastic gradient policy method. Policy gradient methods are a family of reinforcement learning methods that minimize an objective function $J(\theta)$ where θ are the parameters of policy π_θ as depicted in equation 5.1.

$$J(\theta) = \int_A \pi_\theta(a_t|s_t, a_{t-1})R(s_t, a_t) \tag{5.1}$$

In the eight projectile dodge ball task, policy π_θ is represented as a deep neural network that approximates $P(a_{t+1}|s_t, a_t; \theta)$. The neural network updates its weights θ in order to maximize equation 5.1. We will use gradient based update methods to maximize $J(\theta)$. Equation 5.2 below shows the gradient for $J(\theta)$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log\left(\pi_\theta(s, a)\right) Q^{\pi_\theta}(s_t, a_t)] \tag{5.2}$$

The value in the equation 5.3 below is described by the sum of the discounted rewards for every action in a trajectory. The value function's relationship to the reward in the eight projectile dodge ball scenario is that the reward is the negative scaled output of our thresholded perturbation detection model.

$$V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \tag{5.3}$$

Assuming $V_t^{\pi_\theta}(s)$ is an unbiased sample of $Q_t^{\pi_\theta}(s_t, a_t)$ the parameter update equation used in this research for theta is of the form depicted in equation 5.4.

$$\theta_t \leftarrow \theta_{t-1} + \alpha \nabla_\theta log\left(\pi_\theta(s_{t-1}, a_{t-1})\right) V_{t-1}^{\pi_\theta}(s_{t-1}) \tag{5.4}$$

The proposed intrinsic reward method uses the output of the perturbation detection algorithm as the instantaneous reward signal. Therefore, the value of a policy gradient method is no longer hand-engineered for the eight projectile dodge ball task. A deep neural network was used to define the policy π_θ.

5.3 Training and Evaluation

In order to evaluate the performance of the given models, a baseline experiments were undertaken. The baseline methods shown in figure 5.1 were deterministic action policies. In the eight projectile dodge ball scenario, the robot has the option of selecting an action from a set of the following five actions: full speed left, half speed left, stop, half speed right and full speed right. The baseline experiments tested six different deterministic policies, five of which were the continual selection of a single action from the available list above and the sixth policy used a random action selection from the options above. Ten seeds of fifty simulations were run for all of the baseline deterministic algorithms and the box-plot below shows average number of collisions that occurred for each of the policies.

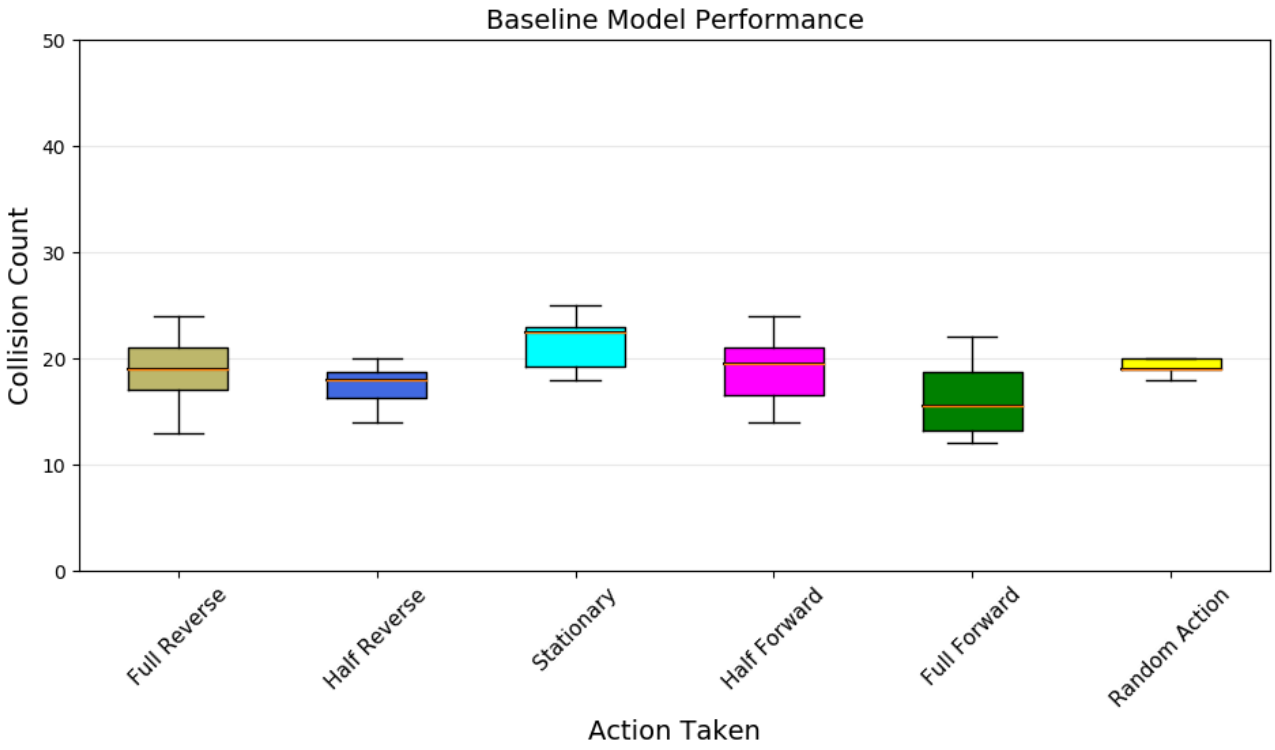

Figure 5.1: Baseline Hit Count Across 10 Seeds Of 50 Simulations

The models that were trained and evaluated compared multiple architectural policy networks and used varying reward signals. The first experiments compared potential architectures as the policy network and used the scaled thresholded perturbation detection output as the reward for time-step t. If the perturbation detection algorithm predicted a value above threshold d (d was set using the equations 3.1 through 3.4 in Chapter 3), then the negative scaled predicted error was used as the reward for that time-step. If the predicted error was below the threshold d, then the reward for time-step t was zero. For a trajectory where all rewards are zero, a positive one was used as the reward for every time-step t.

Four variants were compared for this algorithm. All of these variants used two final feed forward layers with a softmax activation in the output layer in order to

produce an action distribution at time-step t. Variant 0 was a feed forward policy network that flattened the two most recent input images into vectors and appended the state and action of the robot to the flattened images as input x_t. Variant 1 used three layers of convolutional neural networks on two input images appended across the channel dimension and three feed forward layers for the state and action of the robot. The output of the convolutional layers was flattened and both branches' outputs were appended in order to produce the input to the second to last layer of the neural network. Variant 2 used ConvLSTM layers discussed in Chapter 4 as the means to extract spatiotemporal features from the video input and LSTM layers to extract temporal features from the state and action input from the robot. These two branches' outputs were reshaped and appended in order to be input for the final two layers of the neural network. Variant 3, based on intuition gained from Chapter 2, used IRNNs as the major source of temporal feature extraction. Variant 3 used 3 convolutional layers to extract spatial features from the current image viewed by the robot's sensor and reshaped that output as input to an IRNN layer. The state and action data for Variant 3 was input to the IRNN cell and then appended to the output of the reshaped convolutional branch of the neural network to be fed into the last two layers. Notably, and as shown Figure 5.2 below, the Variant 3 or the ConvIRNN architecture outperformed the other architectures achieving the highest reward.

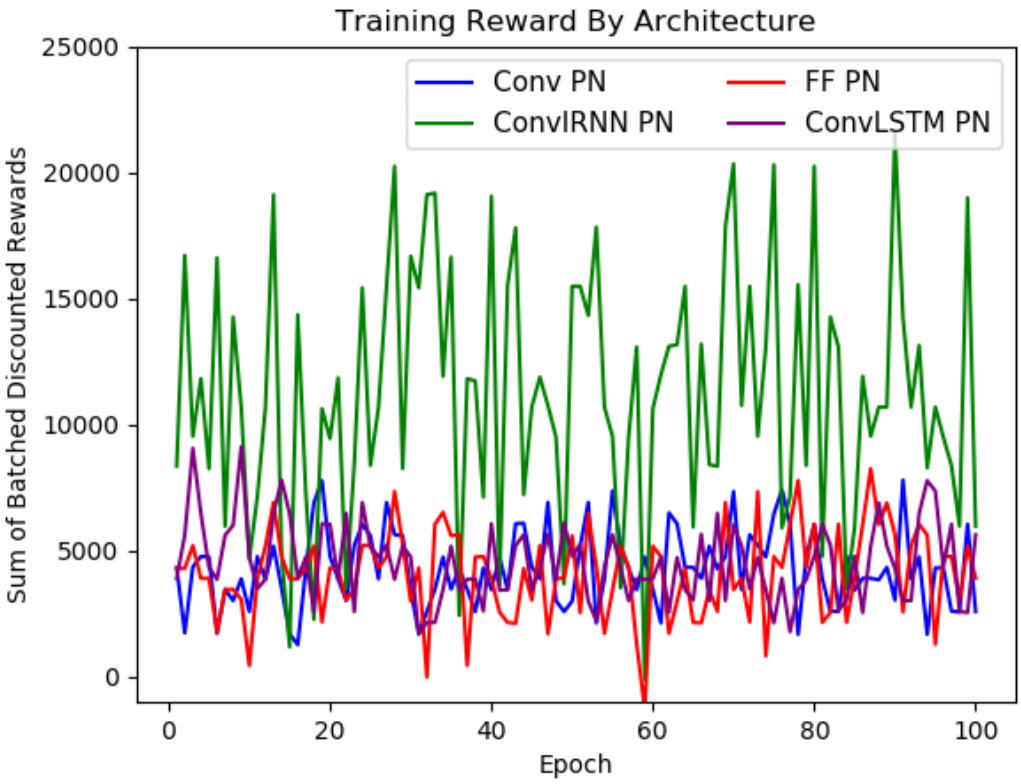

Figure 5.2: 8 Ball Scenario Training Rewards

The best performing policy networks for the ConvIRNN variant were validated according to the same criteria as the baseline methods. The number of collisions across fifty simulations were recorded for ten different random seeds. The results are depicted in 5.3.

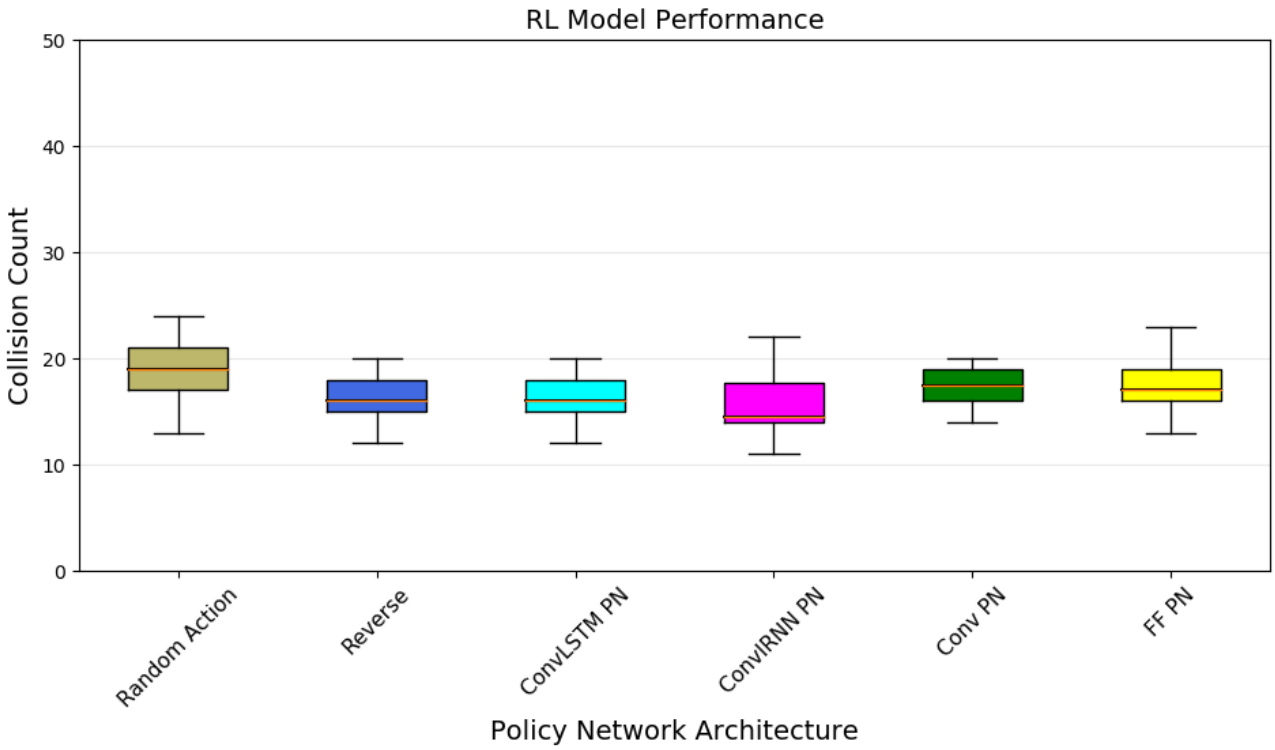

Figure 5.3: Hit Count By Architecture

5.4 Results and Insights

The best performing ConvIRNN models do not yield a solution to the eight projectile dodge ball task. One of the deterministic baseline algorithms was approximately as effective as the best-trained ConvIRNN models. Importantly, the best-trained ConvIRNN models converged to a general strategy for collision avoidance and outperformed five out of the six deterministic baseline strategies. Unfortunately, unlike the one projectile dodge ball task solved in Chapter 4, the solution to the eight ball dodge ball task remains elusive. More research is needed to determine whether, and if so to what extent, the approach proposed in this research can be applied to robotic collision avoidance problems. Nothing in the research undertaken to date suggests

that such additional efforts would be futile. Insights gained from this research on the eight projectile dodge ball task include the following:

1. Convergence

The intrinsic reward policy gradient method using the scaled perturbation detection output showed convergence to consistent behaviors or policies. The best-trained solutions did outperform five of the six deterministic baseline policies and did not perform more poorly than the optimal deterministic baseline policy. Therefore, the intrinsic reward policy gradient method proposed did not abjectly fail. Divergence from optimal policies was not observed. Even though the tested intrinsic reward policy gradient method did not solve the eight projectile dodge ball task, it showed potential in its ability to find converging behaviors that attempted to solve the problem.

2. Dodge Ball Task Refinement

An important aspect of being able to validate the efficacy of the intrinsic reward policy gradient approach is having a testing environment that is difficult enough to require complex learning, while at the same time being reasonably solvable. From an RL perspective, whether the eight projectile dodge ball task, as currently configured, is reasonably solvable merits further study. A significant amount of time was applied to designing the eight projectile dodge ball task. The task was developed with the intention of being difficult to solve. Because the eight projectile dodge ball task was constructed specifically during and for this research, no other researchers have attempted to solve it. Therefore, it is difficult to determine the effectiveness of the best-trained models resulting from the proposed method. This research proposes that these solutions offer the first baselines for this difficult task.

3. Simulation Platform Requirements

The simulation platform selected to implement the eight projectile dodge ball task presented many problems. In order to properly explore reinforcement learning solu-

tions, a simulation environment needs to be able to run experiments in a time-efficient manner. V-REP does not run synchronous, headless and in parallel on a single computer. Reinforcement learning requires adequate hyper-parameter searches in order to produce stable solutions. Because of these two facts, V-REP limits a researchers capability to test, in any reasonable time frame, the validity of reinforcement learning approaches with image input on a problem as relatively difficult as the eight projectile dodge ball task. Another problem faced in this research was that V-REP does not use the physics engine's collision detection algorithm to determine the ground truth; the software does its own collision detection. This would not have posed a problem if V-REP did not use a 50ms step for its own rendering and updates, while the physics engine uses a different – 5ms – step for its physics updates. In order to solve this discrepancy, which was required to access the ground truth for the dodge ball task, one must set V-REP's update step to be 5ms. Unfortunately, this means that the limiting factor for the time it takes to train each RL algorithm is no longer the computation required from the RL algorithm but rather the rendering step required by V-REP. For the eight projectile dodge ball task, one cannot increase the simulation rendering rate of V-REP without losing the ground truth.

5.5 Conclusions

The intrinsic reward deep reinforcement learning method proposed in this section was motivated by the potential of designing reward functions that promote robot safety and longevity in new environments. The eight projectile dodge ball scenario was designed in order to test the proposed intrinsic reward policy gradient method. Unfortunately, the proposed method did not solve the task. Despite the lack of a solution, however, more research needs to be done to determine the effectiveness and potential benefits of the proposed approach. Insights were gained from this

research. One is that the intrinsic reward policy gradient method converges to local minimum. Also, the eight projectile dodge ball task requires further investigation and possible refinement to ensure that it is reasonably solvable. Finally, another simulation environment should be considered in order to better allow researchers to evaluate, within a reasonable time frame, the intrinsic reward policy gradient methods. Future research can also apply the intrinsic reward policy gradient method using other policy gradient algorithms such as TRPO and DDPG. Research can also be undertaken using the intrinsic reward policy gradient method on other tasks with common baselines. Future research should be undertaken using other algorithms in order to set more baselines for the eight projectile dodge ball task created in this research. Different simulation environments – ones that can be run synchronous, headless and in parallel – should be considered.

REFERENCES

[1] A. V. Apkarian. Pain perception in relation to emotional learning. *Current opinion in neurobiology*, 18(4):464–468, 2008.

[2] A. I. Basbaum and H. L. Fields. The origin of descending pathways in the dorsolateral funiculus of the spinal cord of the cat and rat: further studies on the anatomy of pain modulation. *Journal of Comparative Neurology*, 187(3): 513–531, 1979.

[3] M. A. Bedau and C. E. Cleland. *The nature of life: classical and contemporary perspectives from philosophy and science*. Cambridge University Press, 2010.

[4] A. Beese and S. Morley. Memory for acute pain experience is specifically inaccurate but generally reliable. *Pain*, 53(2):183–189, 1993.

[5] D. M. Broom. Evolution of pain. *Vlaams Diergeneeskundig Tijdschrift*, 70(1): 17–21, 2001.

[6] J. Chen, Z. Li, Y. Lv, C. Li, Y. Wang, R. Wang, K. Geng, and T. He. Empathy for pain: A novel bio-psychosocial-behavioral laboratory animal model. *Sheng li xue bao:[Acta physiologica Sinica]*, 67(6):561–570, 2015.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[8] J. Collins, J. Sohl-Dickstein, and D. Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

[9] A. Craig. A new view of pain as a homeostatic emotion. *Trends in neurosciences*, 26(6):303–307, 2003.

[10] A. D. Craig. How do you feel? interoception: the sense of the physiological condition of the body. *Nature reviews neuroscience*, 3(8):655, 2002.

[11] R. Dawkins. *River out of Eden: A Darwinian view of life*. Basic books, 2008.

[12] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[13] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999.

[14] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug): 115–143, 2002.

[15] C. Gramsch, J. Kattoor, A. Icenhour, M. Forsting, M. Schedlowski, E. R. Gizewski, and S. Elsenbruch. Learning pain-related fear: neural mechanisms mediating rapid differential conditioning, extinction and reinstatement processes in human visceral pain. *Neurobiology of learning and memory*, 116:36–45, 2014.

[16] S. Haddadin. *Towards safe robots: approaching Asimovs 1st law*, volume 90. Springer, 2013.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] M. Hunter, C. Philips, and S. Rachman. Memory for pain. *Pain*, 6(1):35–46, 1979.

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

[20] J. Kuehn and S. Haddadin. An artificial robot nervous system to teach robots how to feel pain and reflexively react to potentially damaging contacts. *IEEE Robotics and Automation Letters*, 2(1):72–79, 2017.

[21] C. Lamm, J. Decety, and T. Singer. Meta-analytic evidence for common and distinct neural networks associated with directly experienced pain and empathy for pain. *Neuroimage*, 54(3):2492–2502, 2011.

[22] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] V. Legrain, G. D. Iannetti, L. Plaghki, and A. Mouraux. The pain matrix reloaded: a salience detection system for the body. *Progress in neurobiology*, 93 (1):111–124, 2011.

[25] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5457–5466, 2018.

[26] E. M. Nagasako, A. L. Oaklander, and R. H. Dworkin. Congenital insensitivity to pain: an update. *Pain*, 101(3):213–219, 2003.

[27] S. Pearce, S. Isherwood, D. Hrouda, P. Richardson, A. Erskine, and J. Skinner. Memory and pain: tests of mood congruity and state dependent learning in experimentally induced and clinical pain. *Pain*, 43(2):187–193, 1990.

[28] P. A. Roche and K. Gijsbers. A comparison of memory for induced ischaemic pain and chronic rheumatoid pain. *Pain*, 25(3):337–343, 1986.

[29] N. Schwartz, C. Miller, and H. L. Fields. Cortico-accumbens regulation of approach-avoidance behavior is modified by experience and chronic pain. *Cell reports*, 19(8):1522–1531, 2017.

[30] T. Singer, B. Seymour, J. O'doherty, H. Kaube, R. J. Dolan, and C. D. Frith. Empathy for pain involves the affective but not sensory components of pain. *Science*, 303(5661):1157–1162, 2004.

[31] C. G. Skibinsky. Changes in store for the livestock industry-canada's recurring proposed animal cruelty amendments. *Sask. L. Rev.*, 68:173, 2005.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[33] I. Sur and H. B. Amor. Robots that anticipate pain: Anticipating physical perturbations from visual cues through deep predictive models. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 5541–5548. IEEE, 2017.

[34] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[35] D. Talmi, P. Dayan, S. J. Kiebel, C. D. Frith, and R. J. Dolan. How humans integrate the prospects of pain and reward during choice. *Journal of Neuroscience*, 29(46):14617–14626, 2009.

[36] K. Wiech and I. Tracey. Pain, decisions, and actions: a motivational perspective. *Frontiers in neuroscience*, 7:46, 2013.

[37] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[38] J. Zaman, V. J. Madden, J. Iven, K. Wiech, N. Weltens, H. G. Ly, J. W. Vlaeyen, L. Van Oudenhove, and I. Van Diest. Biased intensity judgements of visceral sensations after learning to fear visceral stimuli: a drift diffusion approach. *The Journal of Pain*, 18(10):1197–1208, 2017.

APPENDIX A

DATA COLLECTED

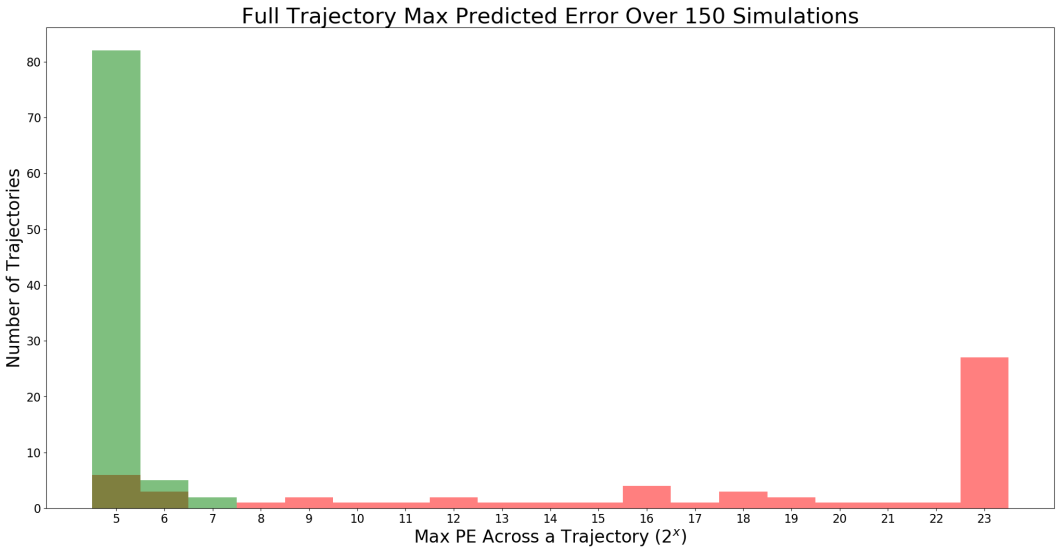

Figure A.1: Model 0 Max Predicted Error Per Trajectory

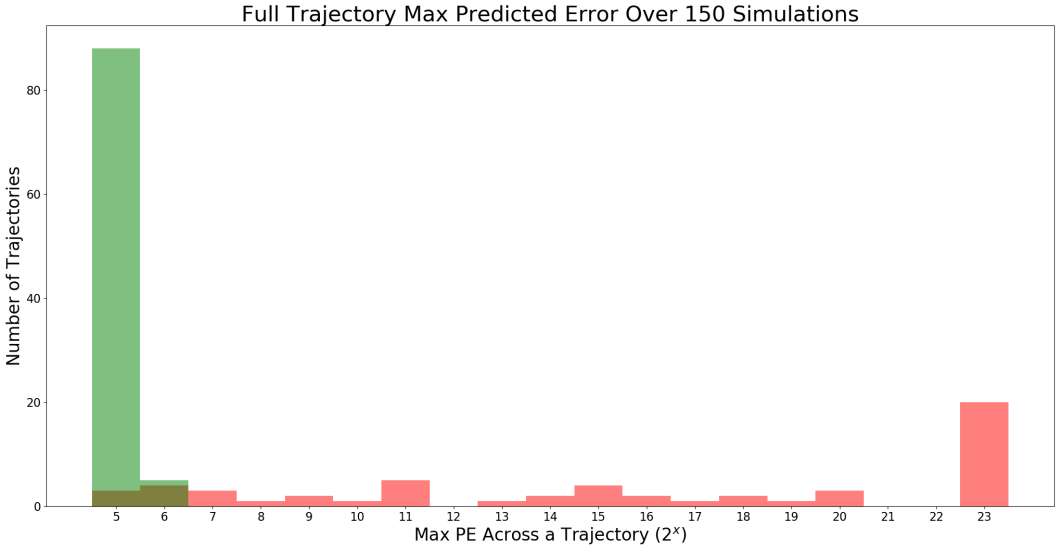

Figure A.2: Model 1 Max Predicted Error Per Trajectory

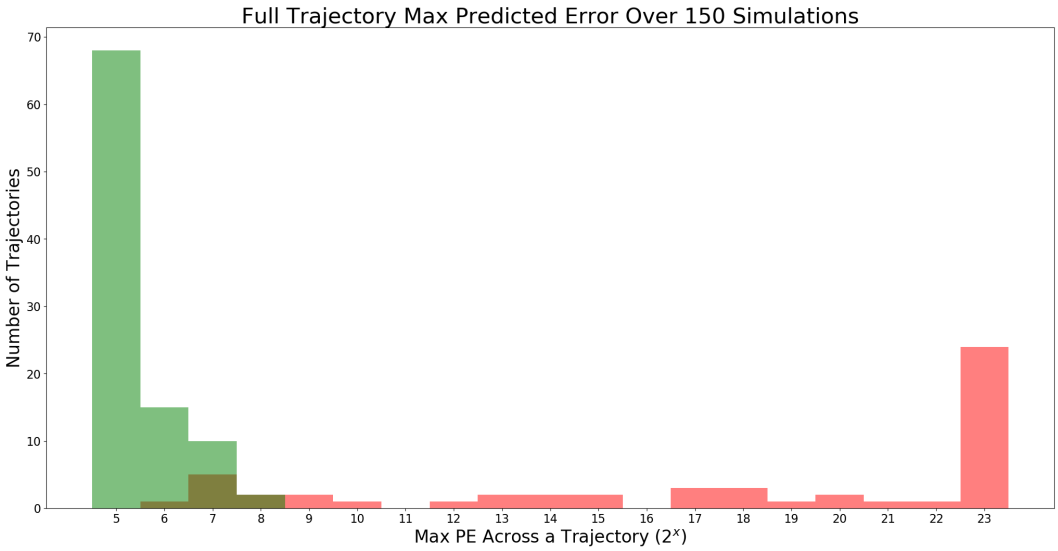

Figure A.3: Model 2 Max Predicted Error Per Trajectory

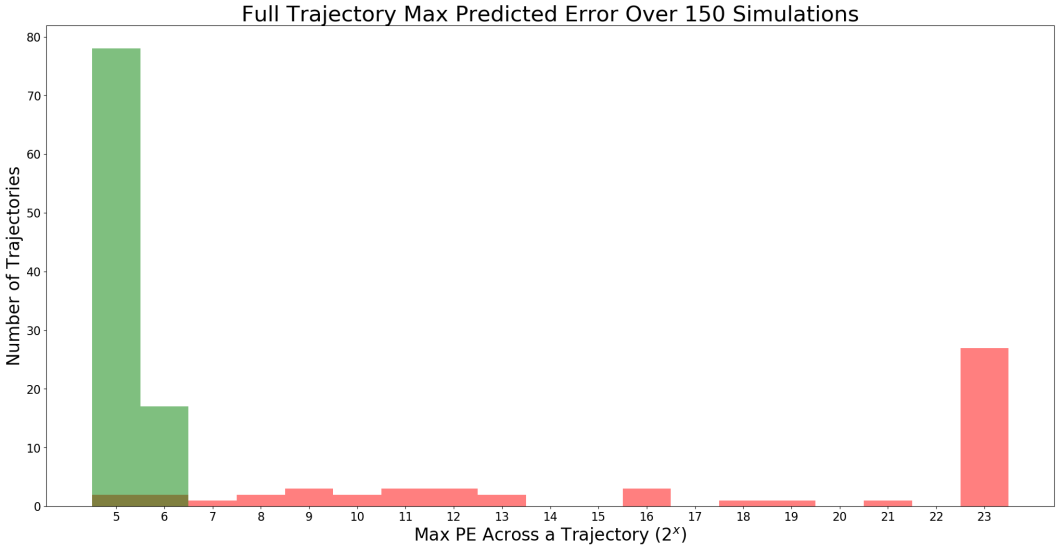

Figure A.4: Model 3 Max Predicted Error Per Trajectory

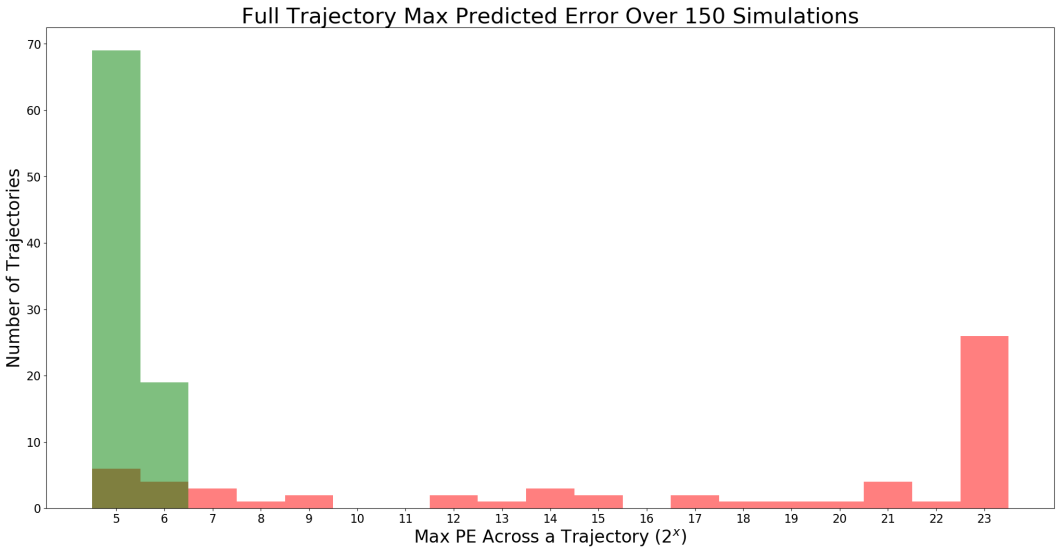

Figure A.5: Model 4 Max Predicted Error Per Trajectory

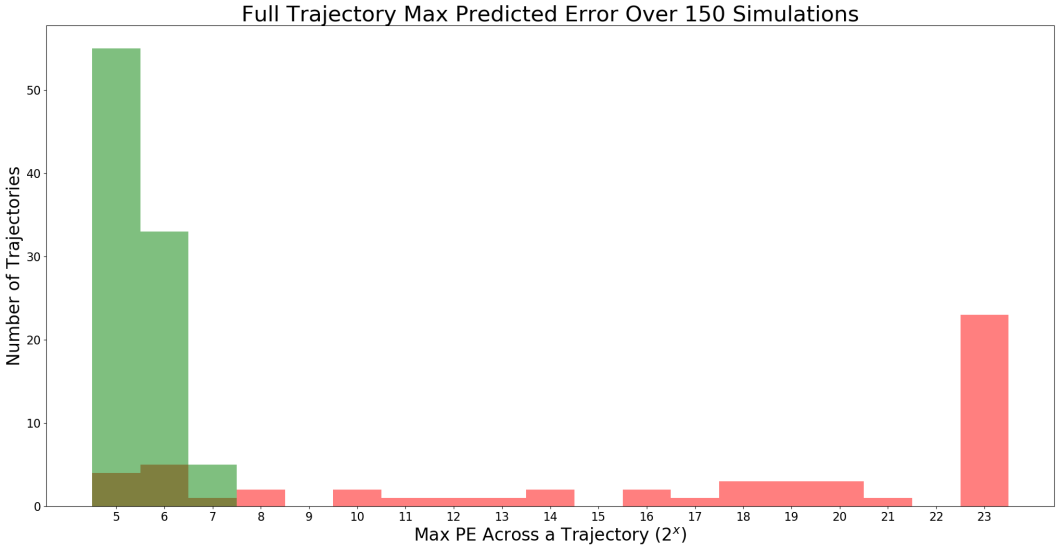

Figure A.6: Model 5 Max Predicted Error Per Trajectory

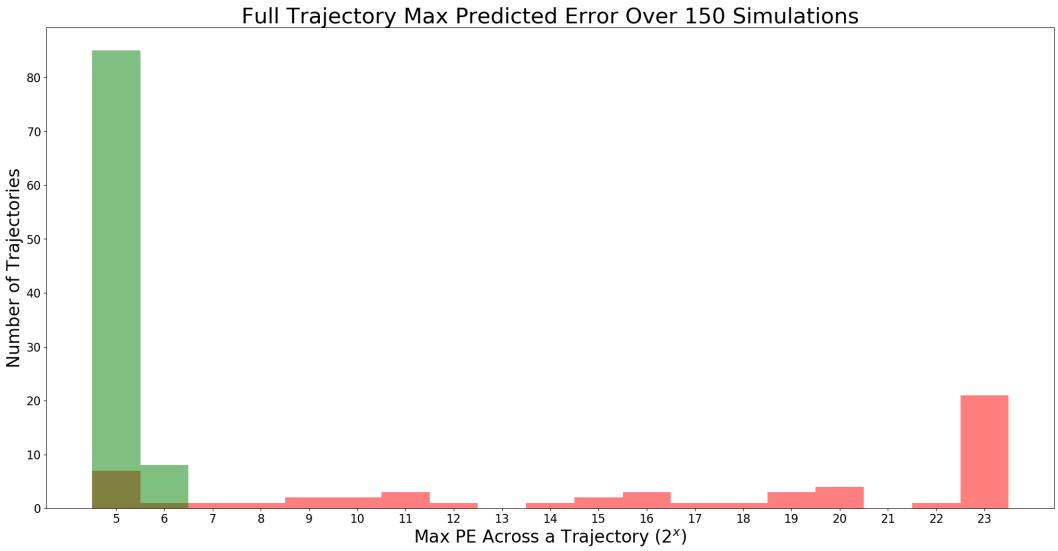

Figure A.7: Model 6 Max Predicted Error Per Trajectory

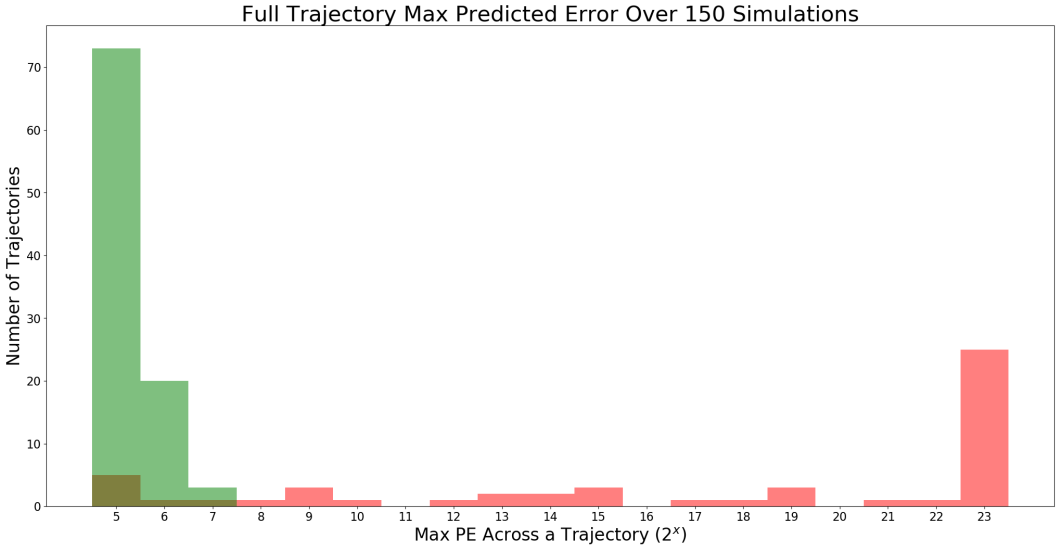

Figure A.8: Model 7 Max Predicted Error Per Trajectory

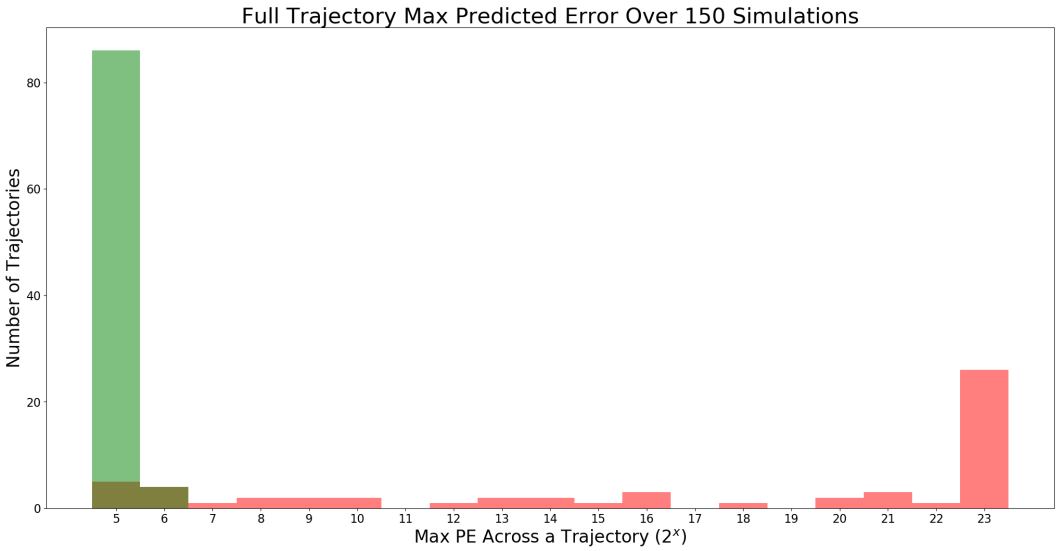

Figure A.9: Model 8 Max Predicted Error Per Trajectory

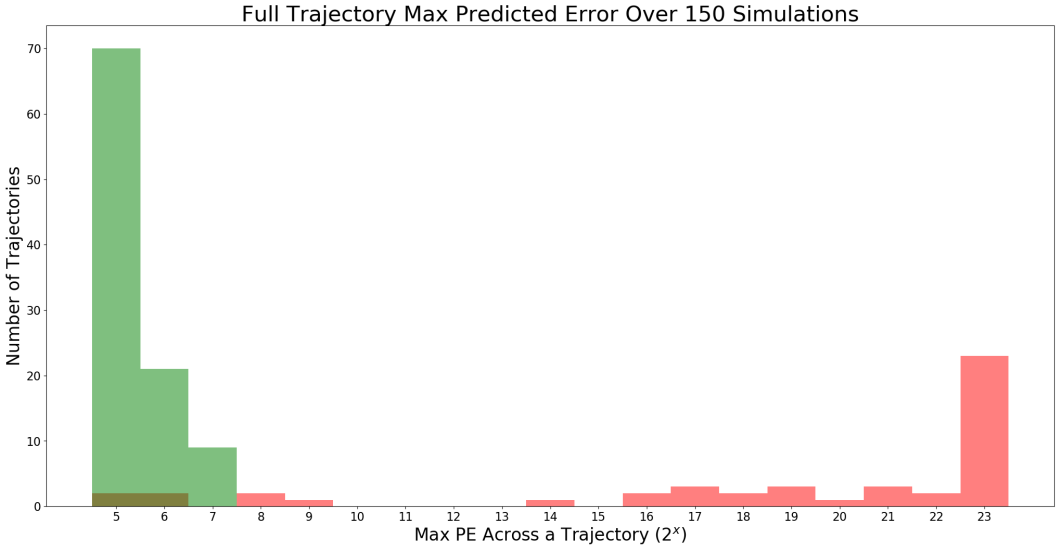

Figure A.10: Model 9 Max Predicted Error Per Trajectory

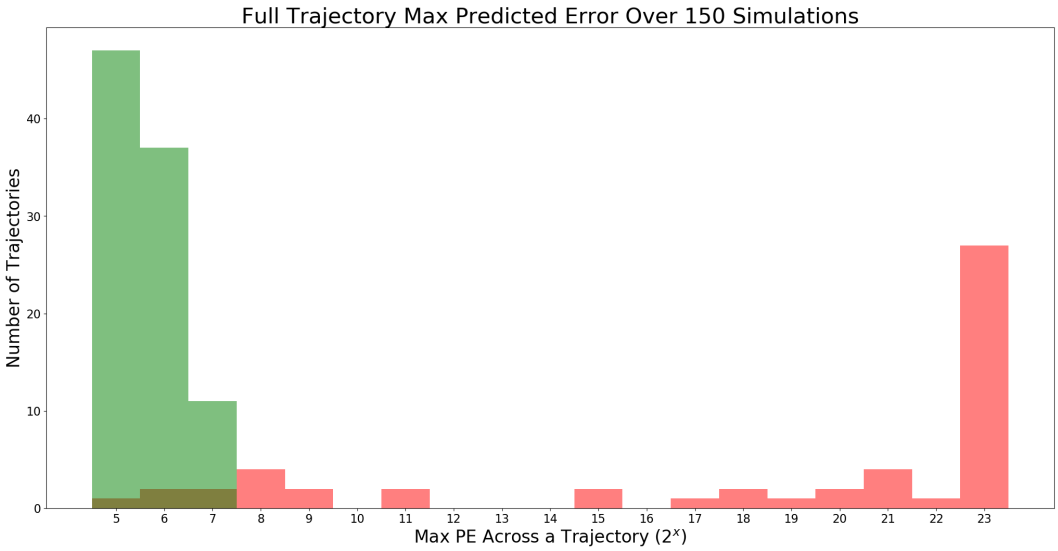

Figure A.11: Model 10 Max Predicted Error Per Trajectory

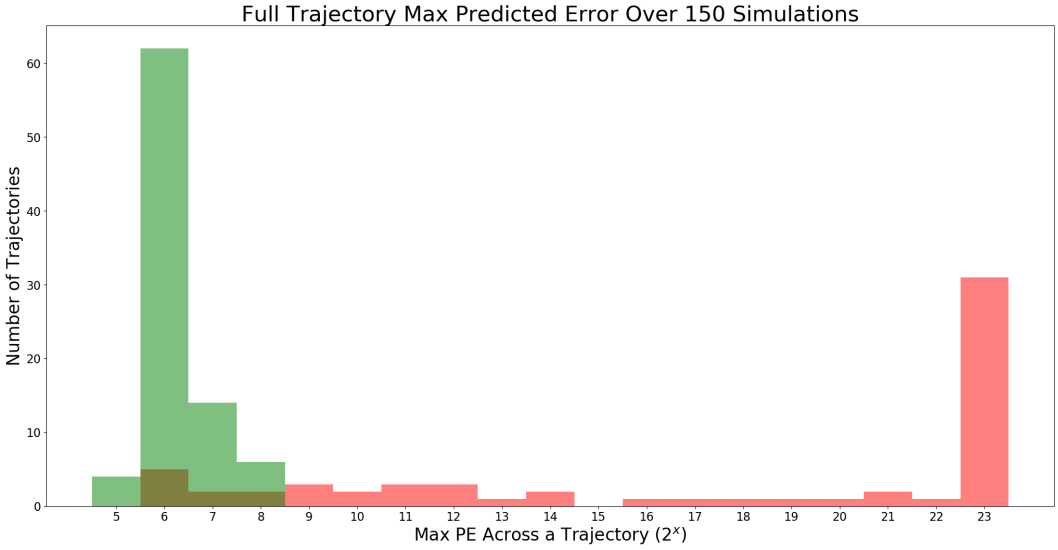

Figure A.12: Model 11 Max Predicted Error Per Trajectory

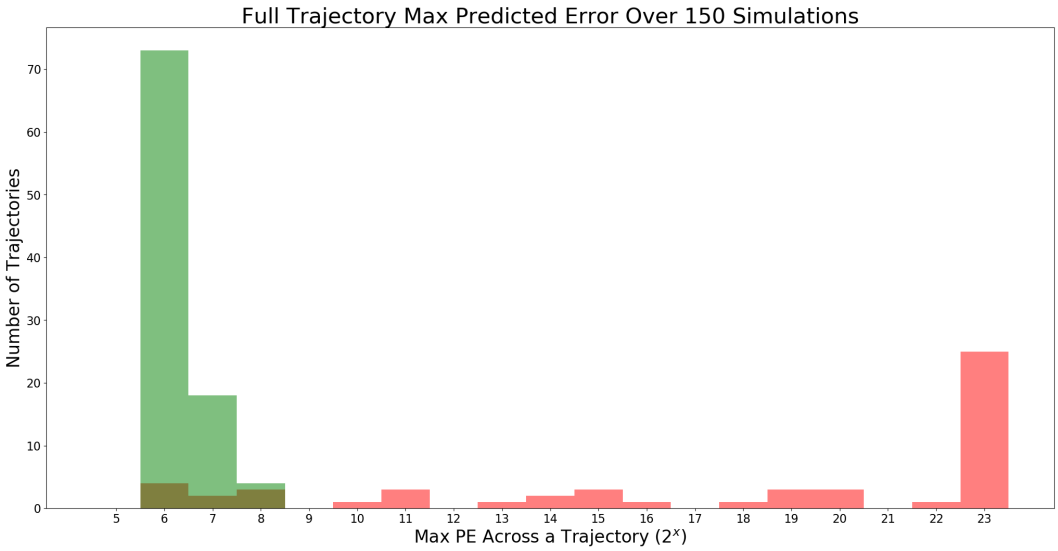

Figure A.13: Model 0 Max Predicted Error Per Trajectory

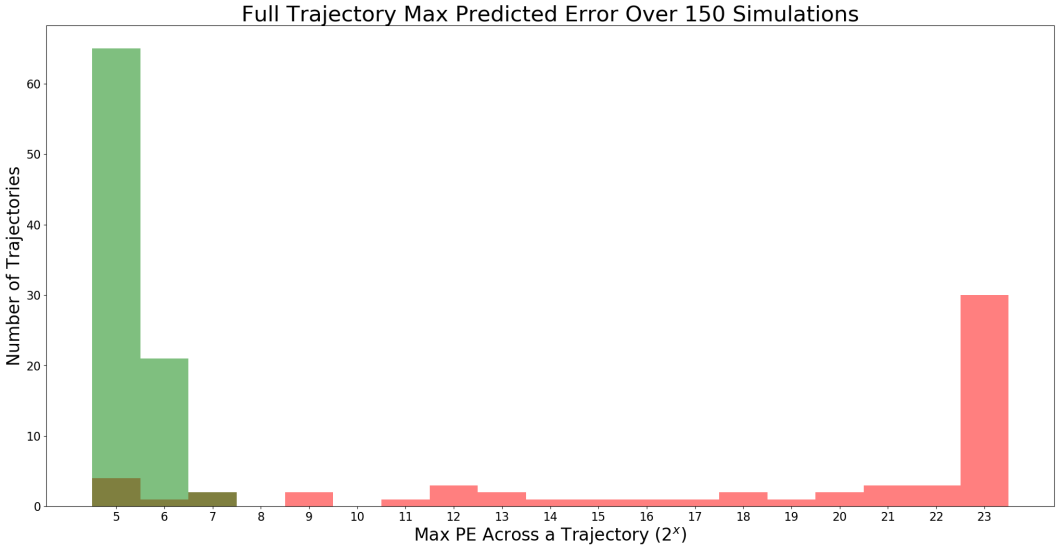

Figure A.14: Model 13 Max Predicted Error Per Trajectory

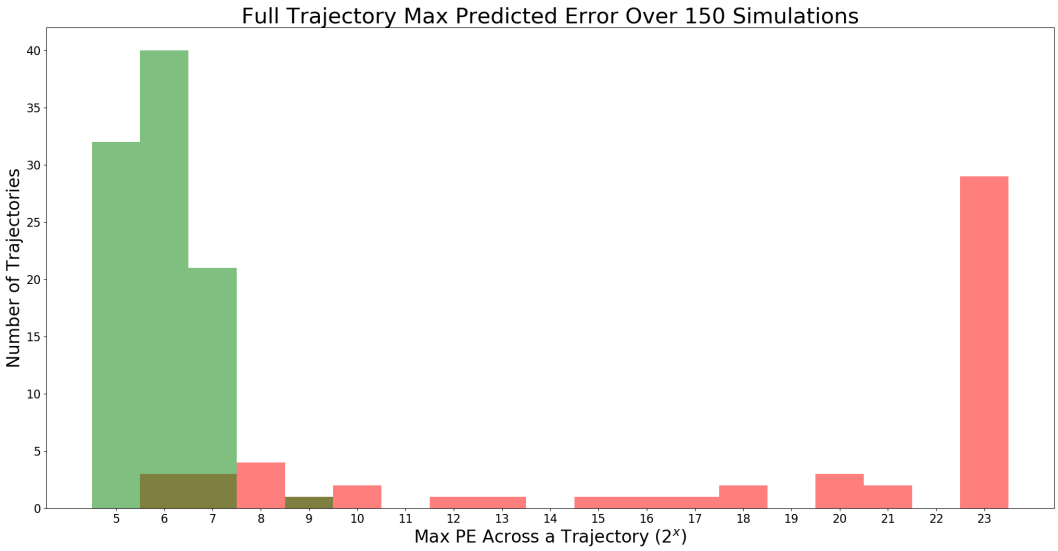

Figure A.15: Model 14 Max Predicted Error Per Trajectory

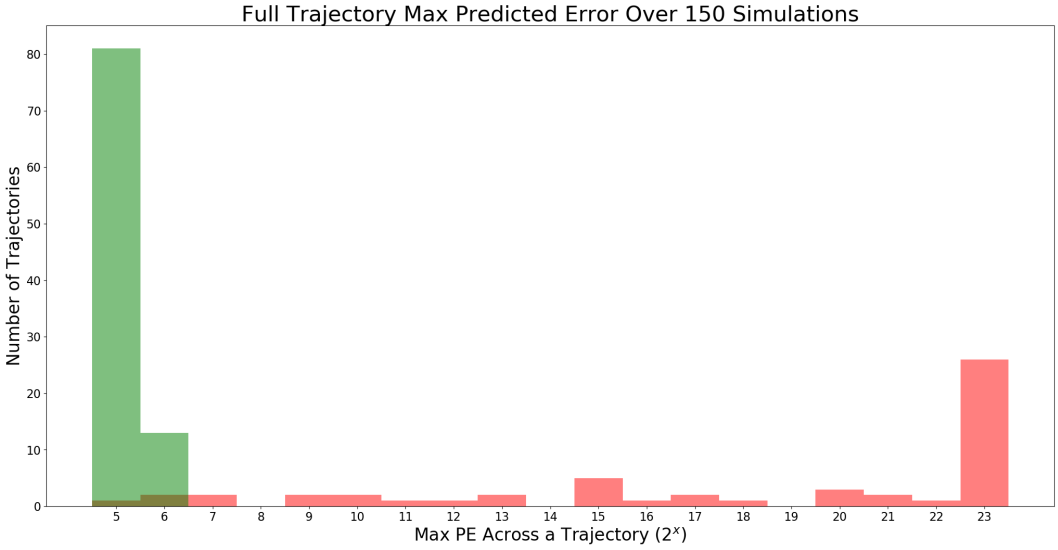

Figure A.16: Model 15 Max Predicted Error Per Trajectory

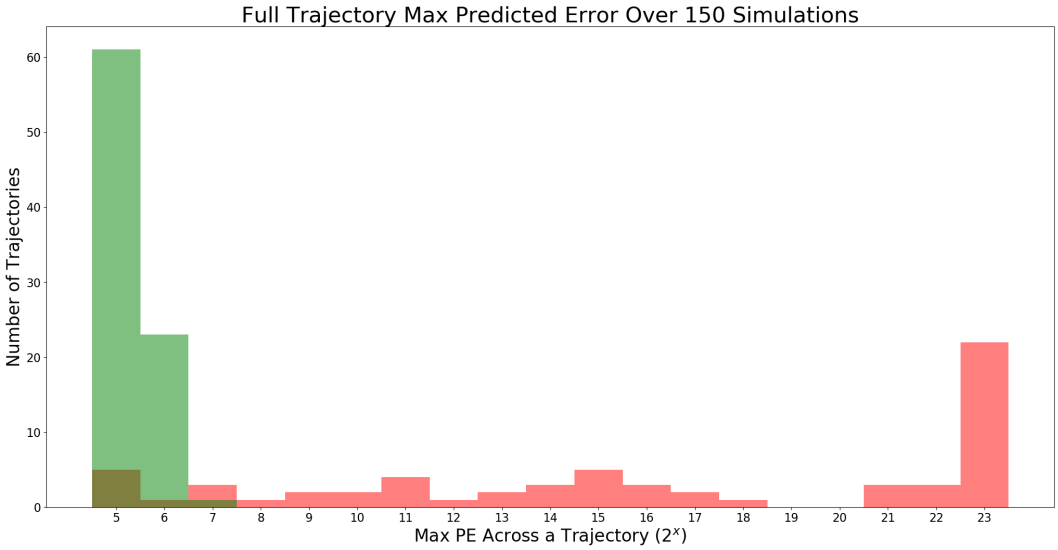

Figure A.17: Model 16 Max Predicted Error Per Trajectory

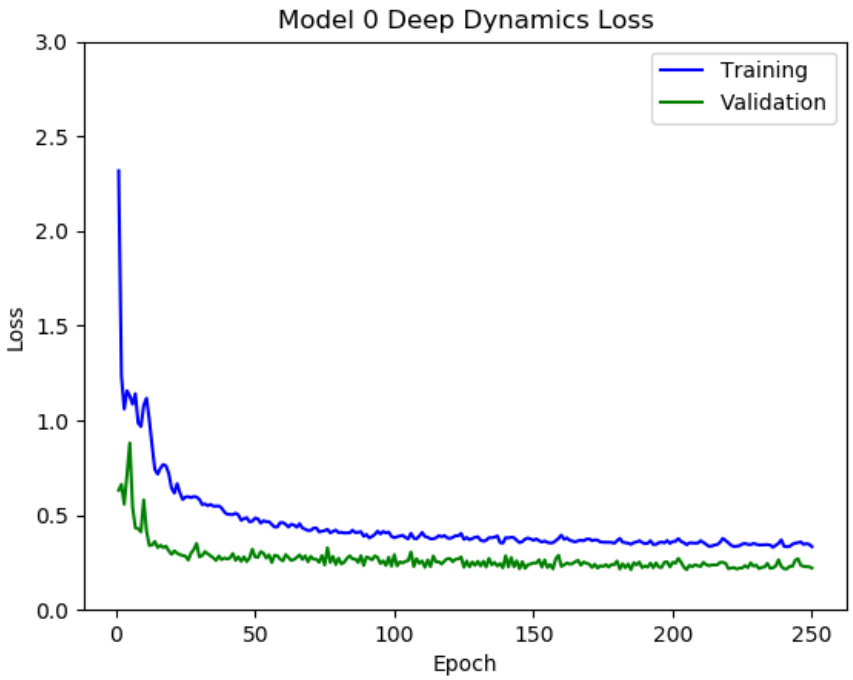

Figure A.18: Model 0 Deep Dynamics Training and Validation Loss

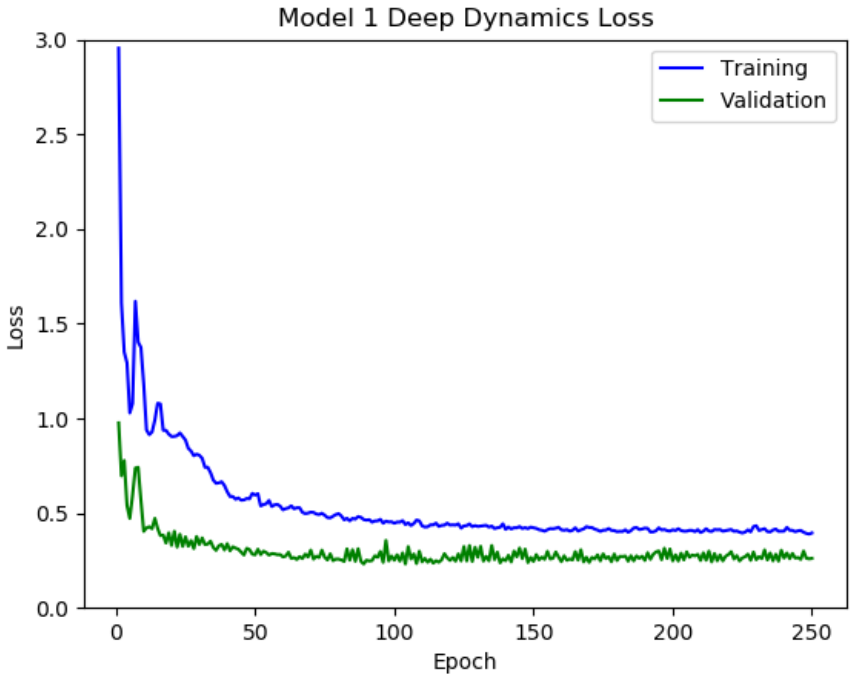

Figure A.19: Model 1 Deep Dynamics Training and Validation Loss

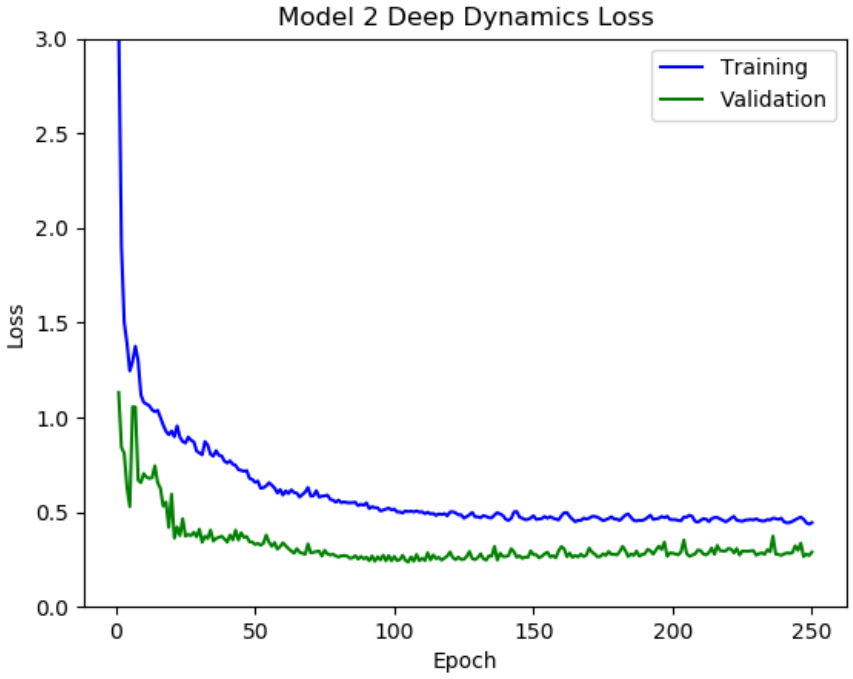

Figure A.20: Model 2 Deep Dynamics Training and Validation Loss

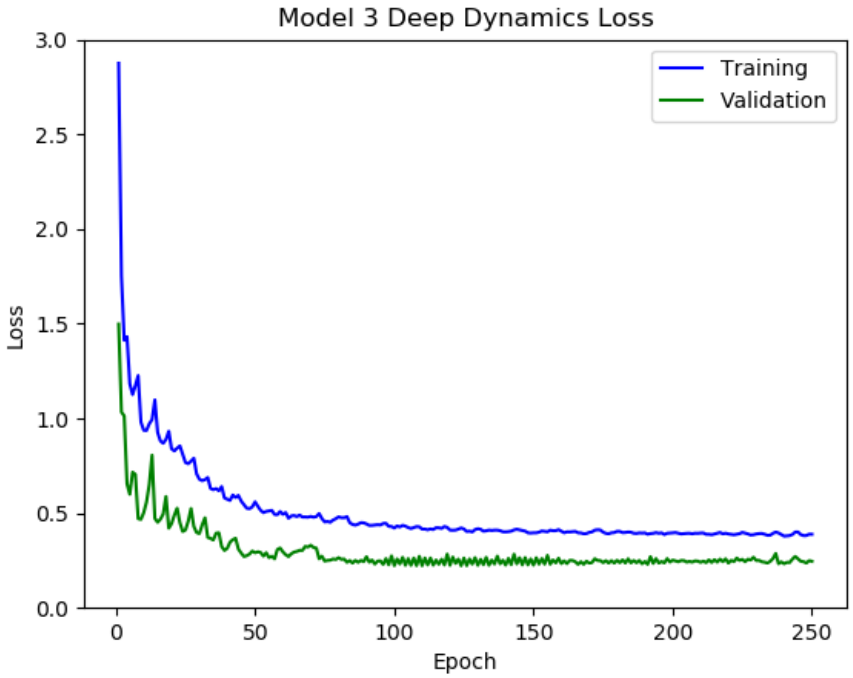

Figure A.21: Model 3 Deep Dynamics Training and Validation Loss

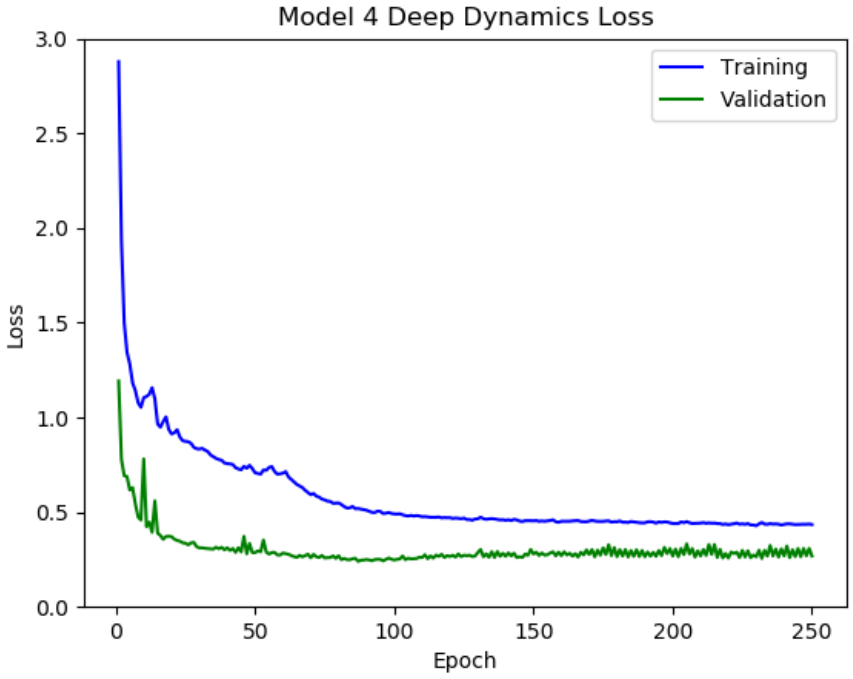

Figure A.22: Model 4 Deep Dynamics Training and Validation Loss

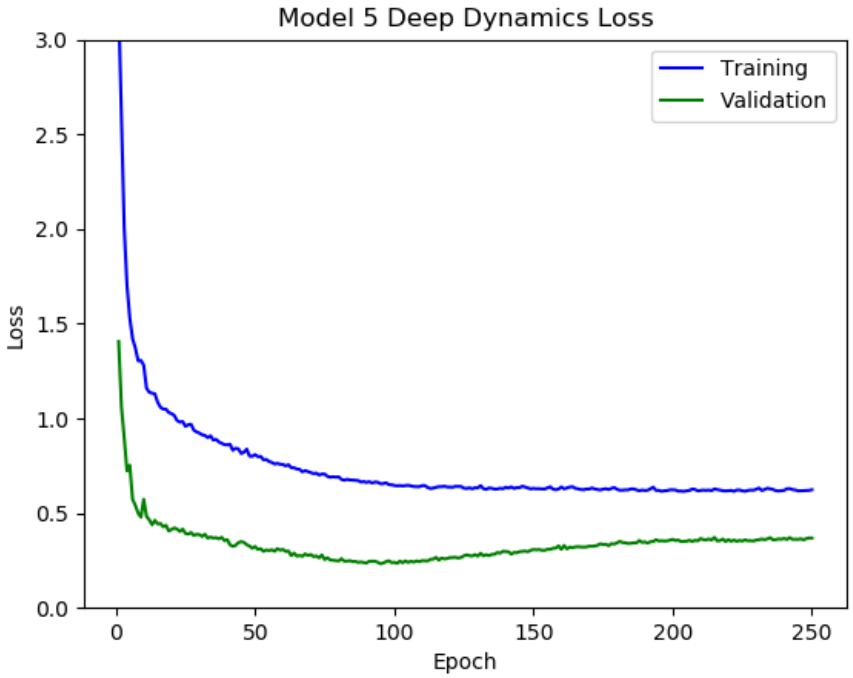

Figure A.23: Model 5 Deep Dynamics Training and Validation Loss

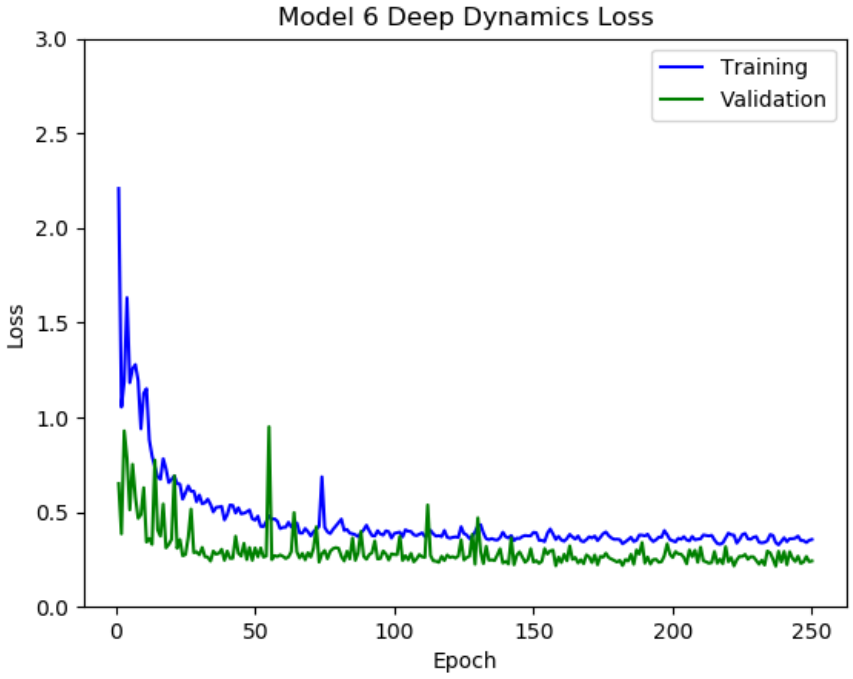

Figure A.24: Model 6 Deep Dynamics Training and Validation Loss

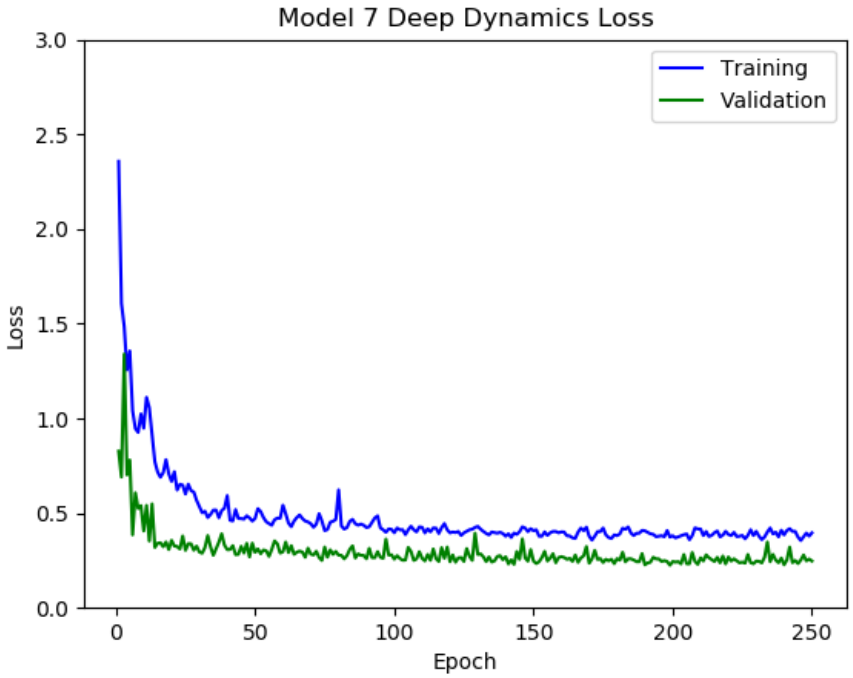

Figure A.25: Model 7 Deep Dynamics Training and Validation Loss

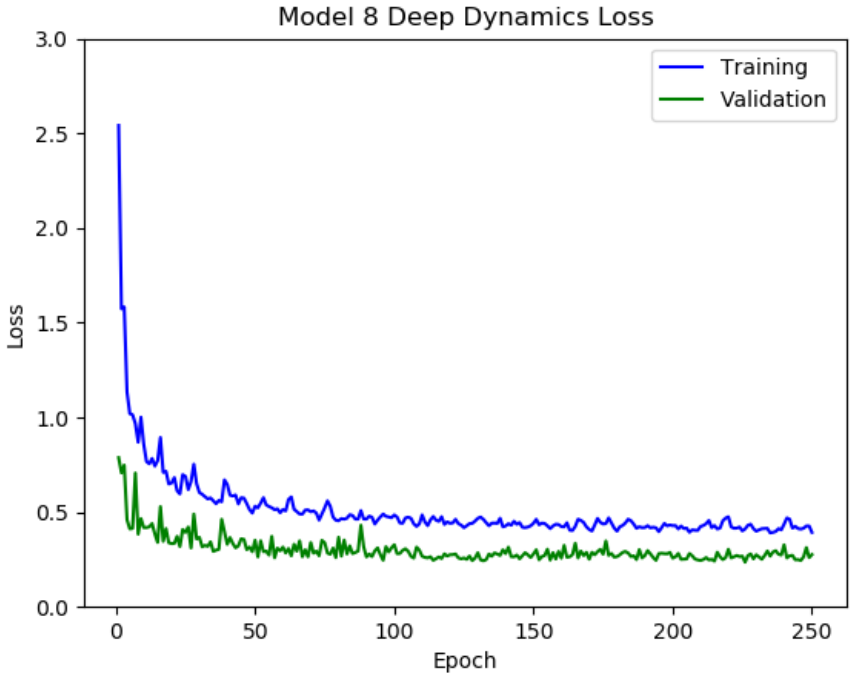

Figure A.26: Model 8 Deep Dynamics Training and Validation Loss

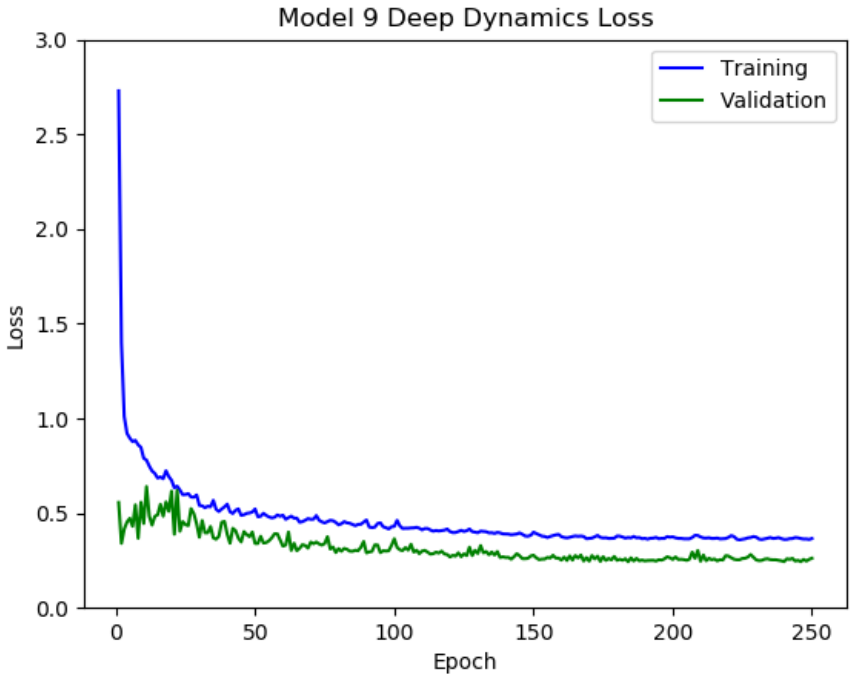

Figure A.27: Model 9 Deep Dynamics Training and Validation Loss

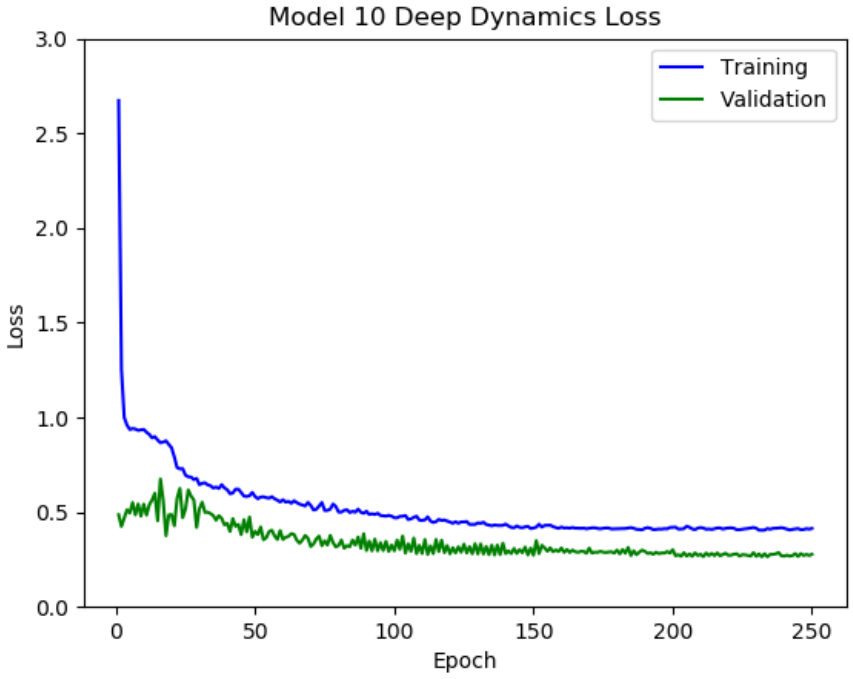

Figure A.28: Model 10 Deep Dynamics Training and Validation Loss

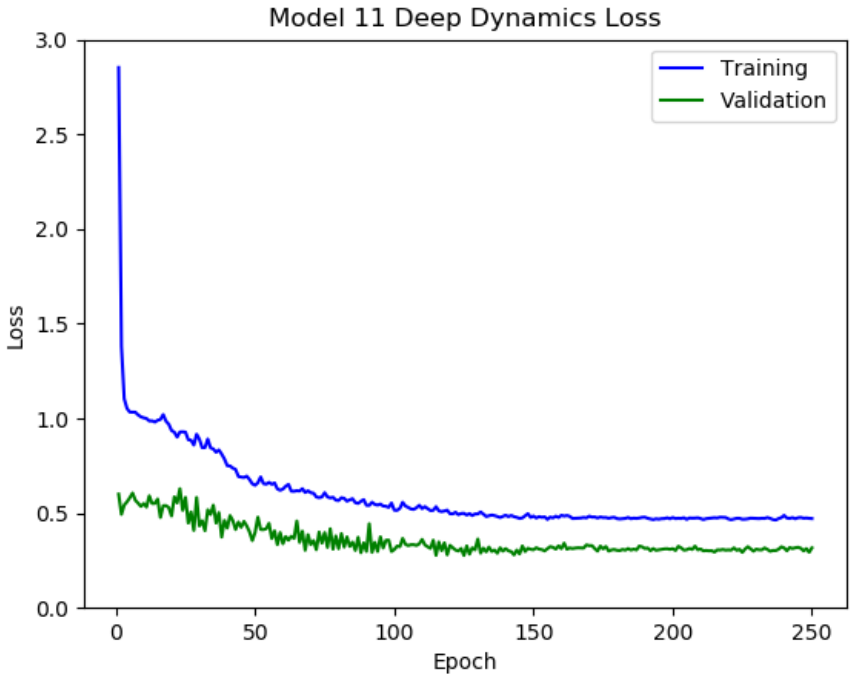

Figure A.29: Model 11 Deep Dynamics Training and Validation Loss

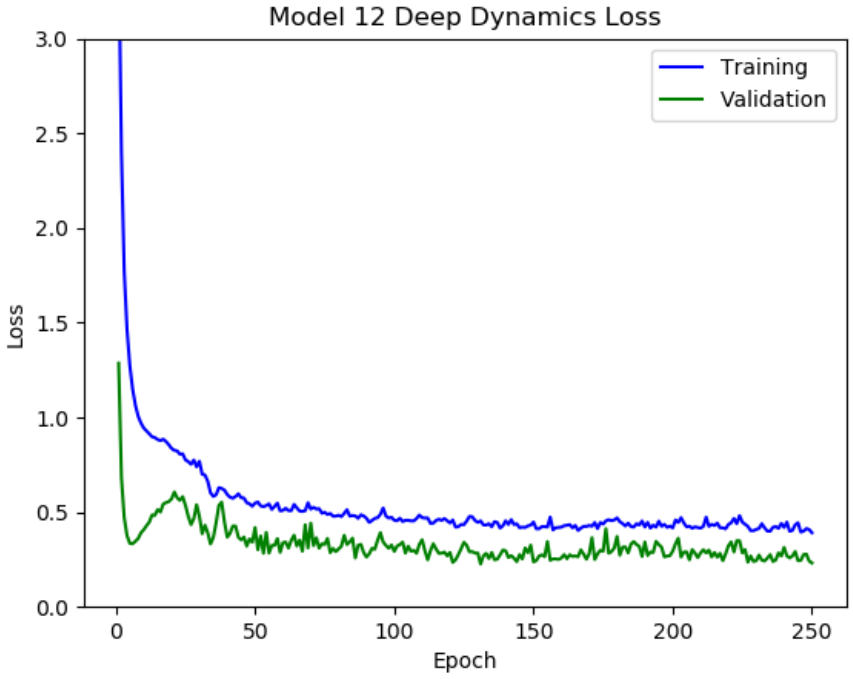

Figure A.30: Model 12 Deep Dynamics Training and Validation Loss

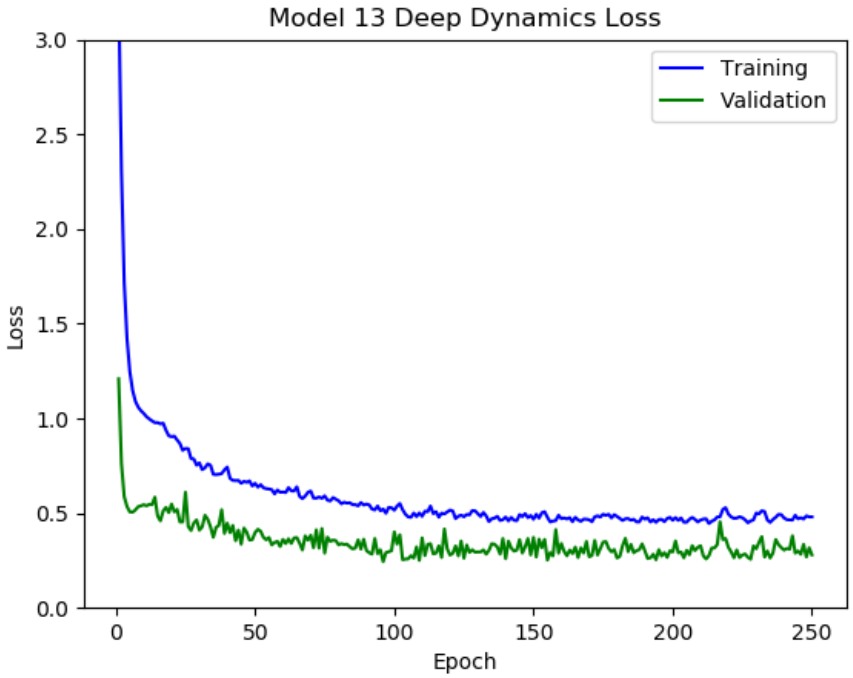

Figure A.31: Model 13 Deep Dynamics Training and Validation Loss

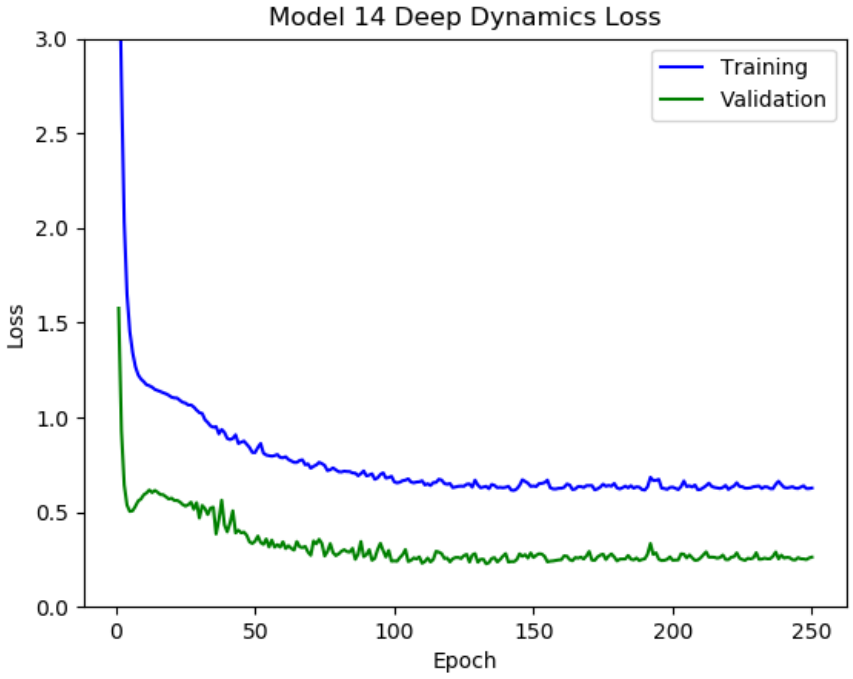

Figure A.32: Model 14 Deep Dynamics Training and Validation Loss

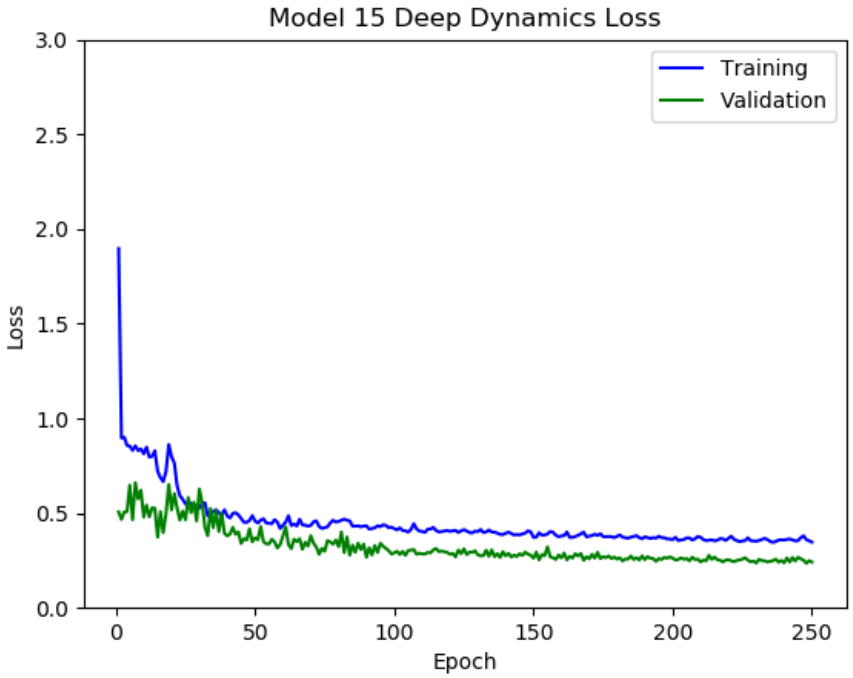

Figure A.33: Model 15 Deep Dynamics Training and Validation Loss

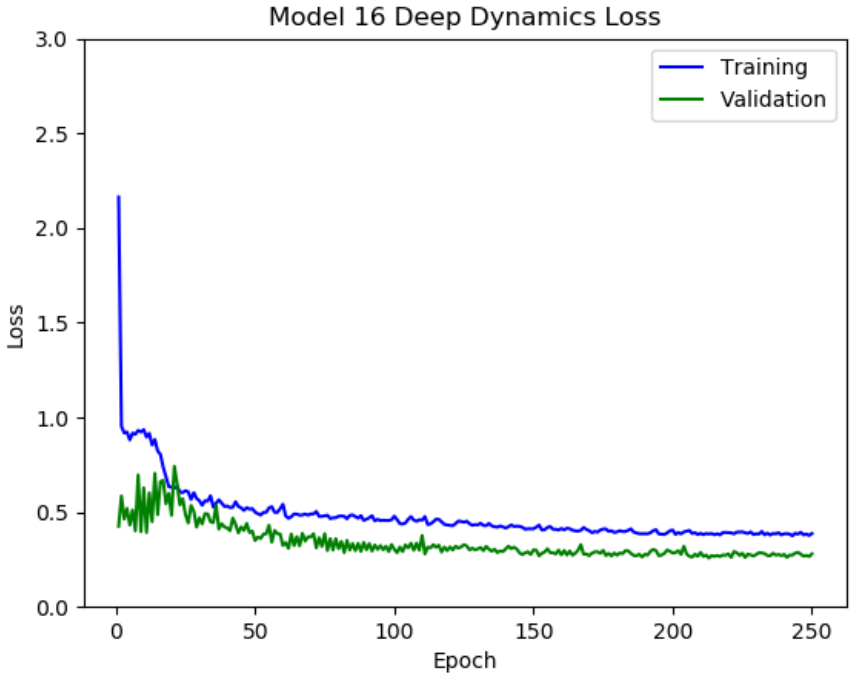

Figure A.34: Model 16 Deep Dynamics Training and Validation Loss

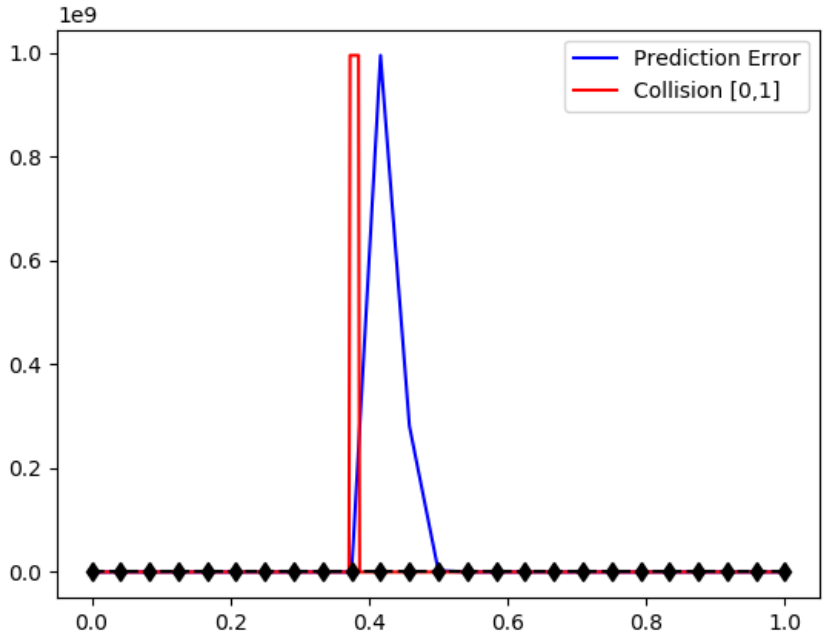

Figure A.35: Predicted Error Vs Ground Truth

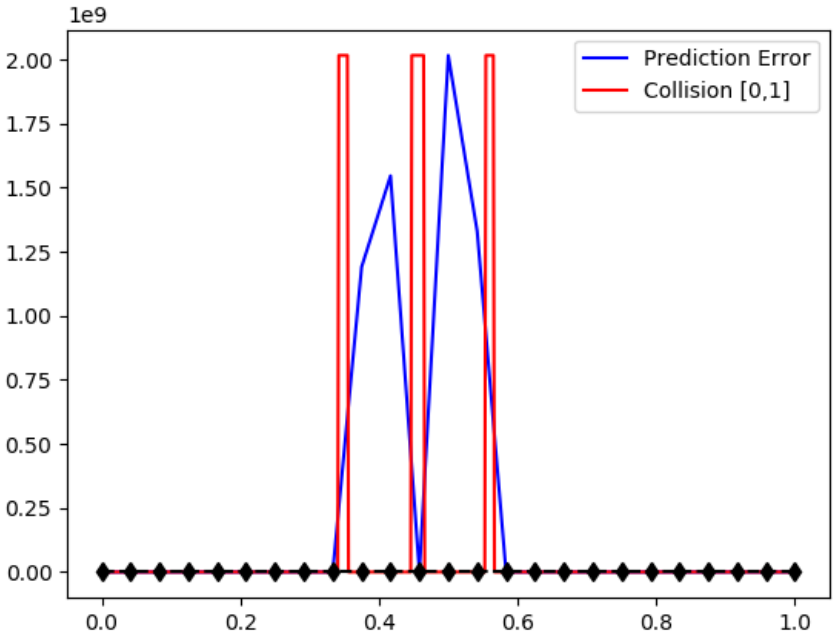

Figure A.36: Predicted Error Vs Ground Truth

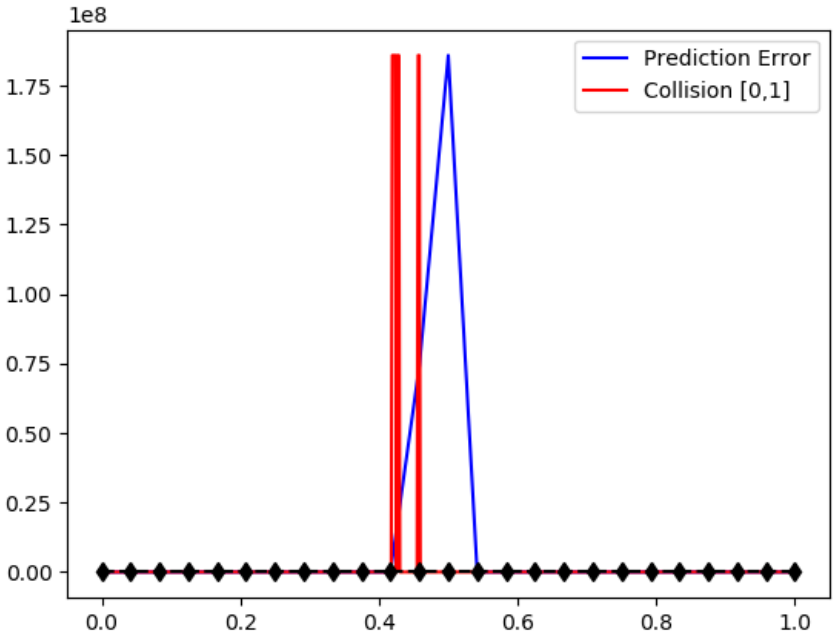

Figure A.37: Predicted Error Vs Ground Truth

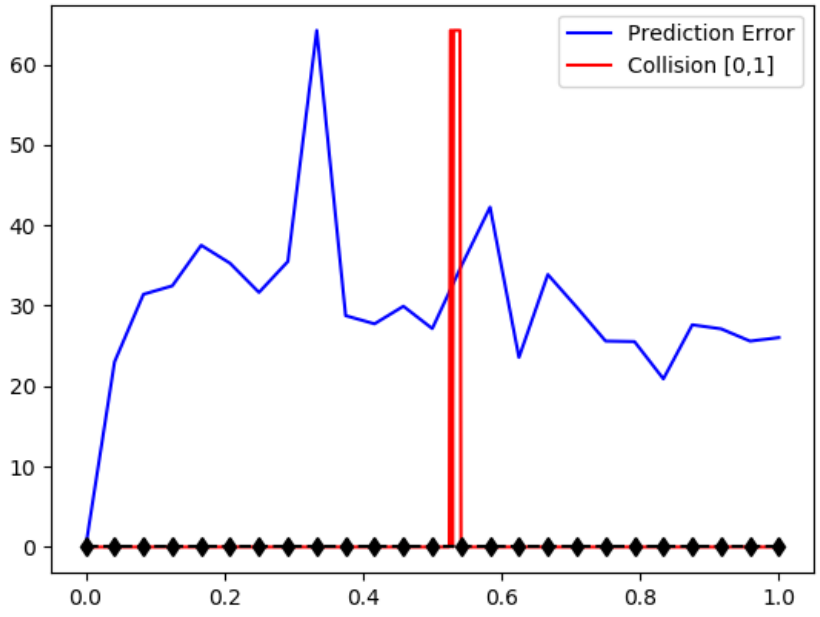

Figure A.38: Predicted Error Vs Ground Truth

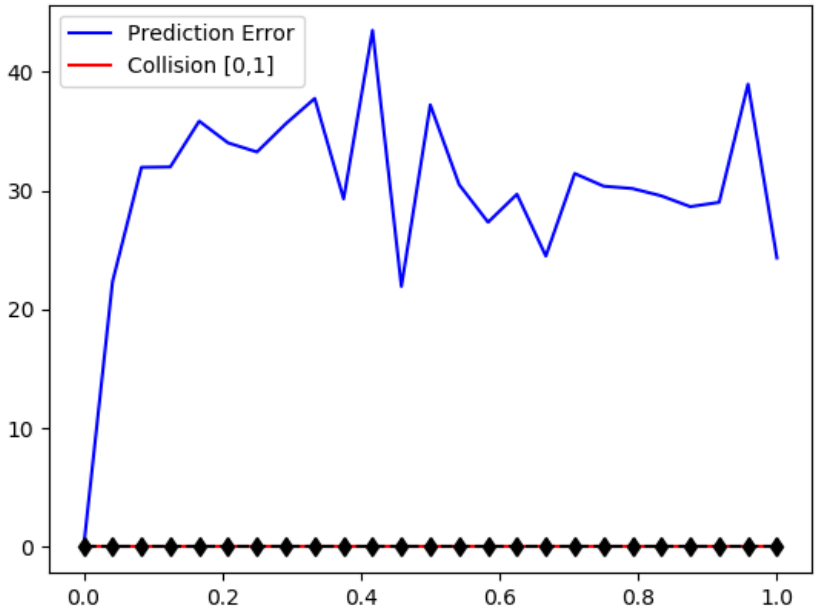

Figure A.39: Predicted Error Vs Ground Truth

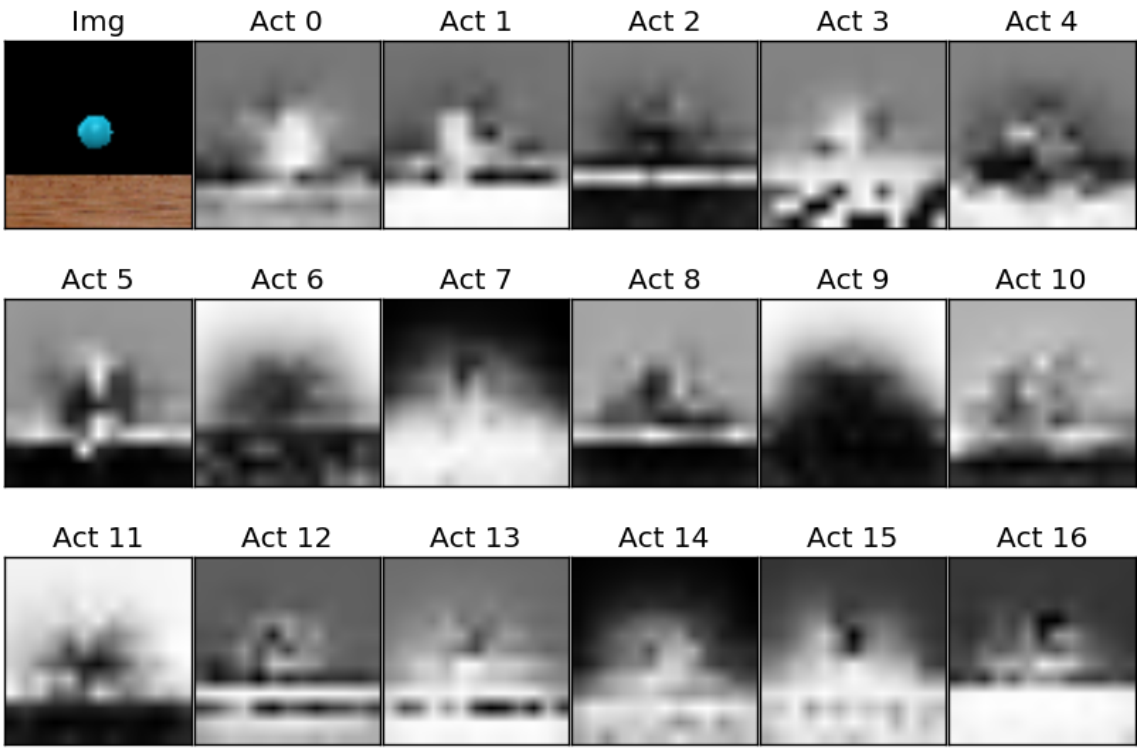

Figure A.40: Hidden Activations for Layer 1

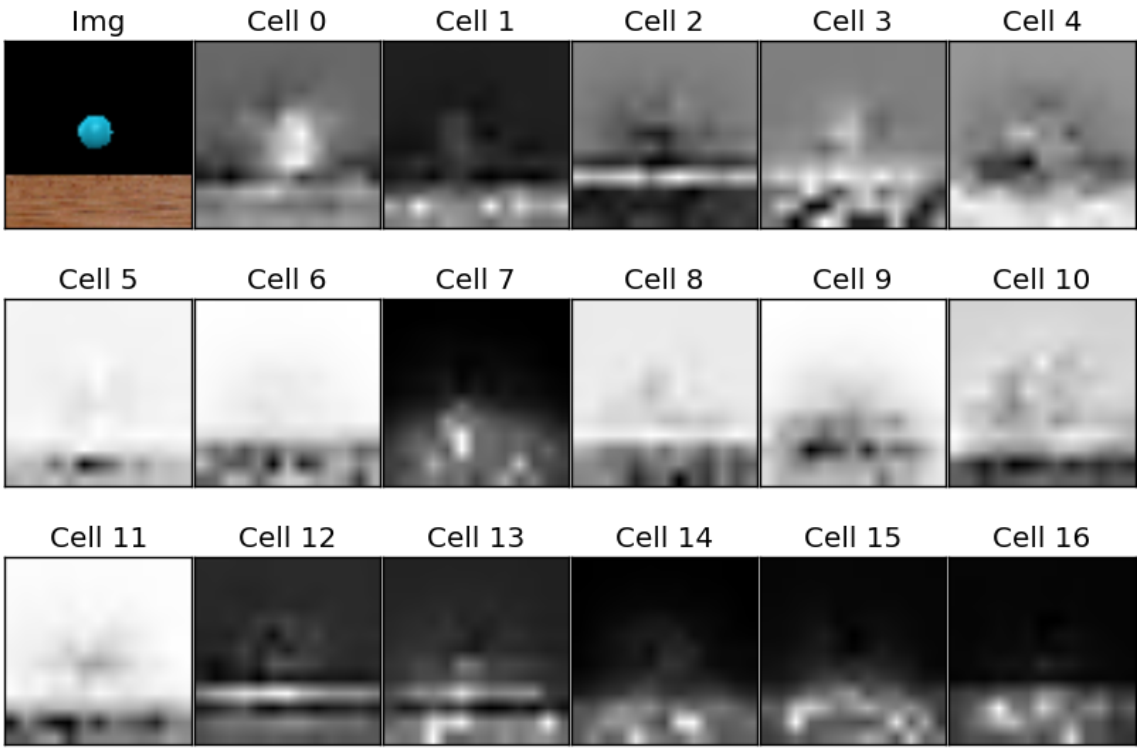

Figure A.41: Cell State for Layer 1