Verification of Reinforcement Learning Models: Comparing Construction of Environment Models

Patrick Jeeva A

Verification of Reinforcement Learning Models: Comparing Construction of Environment Models

Dissertation submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Computer Science

by

Patrick Jeeva A
[Roll No: CS2220]

(Ericsson sponsored M.Tech. Dissertation)

under the guidance of

Dr. Swarup Mohalik
Prinicipal Engineer - Research
Ericsson Reserach India

ISI supervisor

Dr. Ansuman Banerjee
Professor
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

June 2024

DECLARATION

I, **Patrick Jeeva A**, hereby declare that the dissertation titled **"Verification of Reinforcement Learning Models: Comparing Construction of Environment Models"** submitted to Indian Statistical Institute, Kolkata, in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science** is is my original work. This dissertation has not been submitted to any other university or institute for the award of any degree or diploma. All sources of information and assistance received during the preparation of this dissertation have been duly acknowledged.

Patrick Jeeva A (CS2220)
Master of Technology in Computer Science
Indian Statistical Institute, Kolkata

CERTIFICATE

This is to certify that the dissertation titled **"Verification of Reinforcement Learning Models: Comparing Construction of Environment Models"** submitted by **Patrick Jeeva A** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under our supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in our opinion, has reached the standard needed for submission. This work was sponsored by Ericsson Research India.

Ansuman Banerjee
Advanced Computing and Microelectronics Unit(ACMU)
Indian Statistical Institute, Kolkata

Swarup K. Mohalik
Ericsson Research
Bangalore

Acknowledgement

I would like to thank my supervisors, *Dr. Swarup K. Mohalik*, Ericsson Research and *Dr. Ansuman Banerjee*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata for their continuous guidance and unwavering support. For the entire duration of the thesis, I have had many opportunities to learn and improve myself and my work. Their guidance has given me a much better appreciation of the research sphere and the value of good quality work.

My deepest thanks to the faculties of Indian Statistical Institute and scientists of Ericsson Research, for their support and assistance throughout the duration of the thesis.

Last but not the least, I would like to thank my family, friends and peers for their help and support. I thank all those, whom I have missed out from the above list.

Patrick Jeeva A
Roll No. CS2220
Indian Statistical Institute
Kolkata - 700108, India.

Abstract

In recent years of advancements in reinforcement learning (RL), utilizing neural network based models to make decisions in dynamic and complex environments has emerged as a powerful paradigm. In particular, model based reinforcement learning has been widely used for its ability to increase learning efficiency and performance. By constructing an environment model beforehand, the agent attains a prior knowledge of the dynamics of the model to take informed decisions and converge fast to optimal policies.

Real-world environments are often intricate and subject to external disturbances, posing substantial challenges for accurate modeling. Addressing these challenges requires the application of sophisticated neural network-based models that can effectively approximate the underlying environment dynamics.

In this work, we develop and evaluate extensive neural network models, specifically focusing on Gaussian Ensemble models, Bayesian neural networks, and Monte Carlo Dropout techniques, to approximate various standard gym environments. These models are trained on different numbers of samples to understand their efficiency and accuracy in capturing environment dynamics. Once trained, the neural network models are used to construct Markov Decision Processes (MDPs) with various discretization strategies. The constructed MDPs are then analyzed and compared to evaluate the performance of each neural network approach.

The purpose of this thesis is to present a comprehensive study on the construction of environment models using advanced neural network techniques. We aim to approximate the standard environments in the reinforcement learning setup, utilizing a variety of neural networks and compare the efficiency based on the reconstruction of MDPs.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the field of machine learning known as reinforcement learning (RL), agents interact with their surroundings to acquire the best possible behaviours. RL comprises a dynamic process of exploration and exploitation, unlike supervised learning, which trains models on a fixed dataset. This characteristic makes RL particularly well-suited to issues where sequential decision-making is essential. With applications ranging from resource management and games to robotics and autonomous vehicles, reinforcement learning (RL) is significant because it can handle complicated tasks that call for adaptive and autonomous decision-making. The idea of an agent interacting with an environment, where the agent acts in accordance with a policy and gets feedback in the form of incentives, is fundamental to reinforcement learning. Typically, a Markov Decision Process(MDP) is used to represent this interaction.

Even though model-free reinforcement learning techniques like Q-learning and deep Q-networks (DQN) have shown great success, they frequently necessitate a lot of interactions with the environment, which can be problematic in real-world situations where gathering data is costly or time-consuming. This problem is addressed by model-based reinforcement learning, which builds a model of the dynamics of the environment and uses it to plan and simulate actions without requiring direct interaction with it.

Deep learning has given rise to strong tools that can be used to represent complicated environment dynamics in reinforcement learning. Neural networks are especially well suited for this purpose because of their capacity to approximate nonlinear functions. Diverse approaches for capturing uncertainty and enhancing model accuracy are provided by different neural network topologies and techniques, such as Bayesian neural networks, Gaussian ensemble neural networks, and Monte Carlo dropout neural networks.

Our goal is to investigate and contrast several neural network-based methods for building MDPs in environments with continuous states. It specifically focuses on employing Gaussian ensemble neural networks, Monte Carlo dropout neural networks, and Bayesian neural networks to approximate the dynamics of non-deterministic environments such as cartpole, pendulum, mountain car, and mountain car continuous. Different neural network designs are put into practice and trained in order to produce an accurate model of the dynamics of the environment. The method of constructing and evaluating Markov Decision Processes (MDPs) involves building MDPs from the training models and evaluating their efficacy using pre-established comparison criteria. This entails weighing the relative benefits and downsides of the various approaches and determining which ones are most appropriate for reinforcement learning tasks.

1.1 Motivation of this dissertation

There are numerous neural network methods for approximating stochastic outputs, but these have not been extensively applied to approximate reinforcement learning (RL) environments. Furthermore, these methods are not evaluated based on their ability to recreate the dynamics of the original RL environment.

The lack of a methodology for building neural network-based models for stochastic gym environments and a standard basis for comparing them persists. These neural network environments can be easily adapted in a dynamic setup using transfer learning. However, there is insufficient work on analyzing how well they can recreate the dynamics of the original environment. To successfully reconstruct the dynamics of the environment after approximation, we need a robust methodology for both construction and comparison.

Existing neural network methods are primarily focused on static applications and often disregard the intricacies involved in dynamic RL environments. While these methods showcase promising capabilities in static scenarios, they do not address the challenges posed by stochastic and continuously evolving environments typically found in RL settings. The absence of comprehensive evaluation metrics for these methods further complicates the assessment of their effectiveness in approximating RL environments.

The current approaches in the field largely ignore the necessity of a standardized framework that would facilitate the construction of neural network-based models tailored for stochastic RL environments. Additionally, there is a significant gap in research concerning the comparative analysis of these models. Transfer learning, a technique that enables the adaptation of a pre-trained model to new tasks, has shown potential for dynamically updating neural network environments. However, the effectiveness of this approach in maintaining the fidelity of the original environment's dynamics remains underexplored.

To bridge this gap, it is crucial to develop a methodology that not only supports the construction of neural network models for stochastic gym environments but also provides a robust basis for their evaluation. This methodology should incorporate advanced techniques for model validation, ensuring that the approximated environments accurately reflect the dynamics of their original counterparts. Such an approach would involve defining clear metrics and benchmarks for performance assessment, thereby enabling a systematic comparison of different neural network methods.

Moreover, a detailed analysis of how well these models can replicate the intricate behaviors and responses of the original RL environments is essential. This would involve rigorous testing under various scenarios to evaluate the robustness and reliability of the neural network approximations. By establishing a comprehensive framework for both the construction and comparison of neural network models in stochastic RL environments, we can pave the way for more reliable and effective applications in this domain.

1.2 Contributions of this dissertation

The objective of this thesis is to investigate various neural network models, specifically Gaussian Ensemble models, Bayesian neural networks, and Monte Carlo Dropout techniques, for approximating diverse gym environments. The contributions of this thesis are briefly described below:

- *Neural Network Models*: This thesis explores employing models specifically Gaussian Ensemble models, Bayesian neural networks, and Monte Carlo Dropout techniques for approximating gym environments.
- *Training and Evaluation*: Trained models on varying sample sizes to evaluate their effectiveness and accuracy in capturing environment dynamics.
- *Markov Decision Processes (MDPs)*: Following training, the neural network models are employed to construct Markov Decision Processes (MDPs) that encapsulate the environment dynamics.
- *Performance Comparison*: The constructed MDPs are rigorously analyzed and compared to evaluate the efficacy of each neural network approach. Comparative studies highlight the strengths and weaknesses of each neural network model in accurately representing and simulating gym environments, providing insights into their applicability and performance in reinforcement learning tasks.

1.3 Organization of the dissertation

This dissertation is organized into 5 chapters. A summary of the contents of the chapters is as follows:

Chapter 1: This chapter contains an introduction and a motivation of this work.

Chapter 2: This chapter corresponds to the background and prerequisites of the work for all the topics discussed.

Chapter 3: This chapter describes the Methodology of model building and MDP construction.

Chapter 4: This chapter describes the experiments and results conducted on the models.

Chapter 5: We summarize with conclusions on the contributions of this dissertation.

Chapter 2

Background and Related Work

In this chapter, we describe the background concepts and prerequisites related to the systems that are discussed and referenced throughout the thesis. We also demonstrate the working of the systems discussed to have a better understanding of the work described in the subsequent chapters. We finally describe the software tools and programming libraries used in the coming chapters.

2.1 Reinforcement Learning

Reinforcement Learning is a type of machine learning where an agent of interest interacts with an environment to learn to take actions for maximizing the reward in long run. For each type of action taken by the agent with the environment, the agent receives a reward (maximizing objective). The objective of this methodology is to learn the actions that are supposed to be taken at a particular state such that the cumulative reward is maximized.

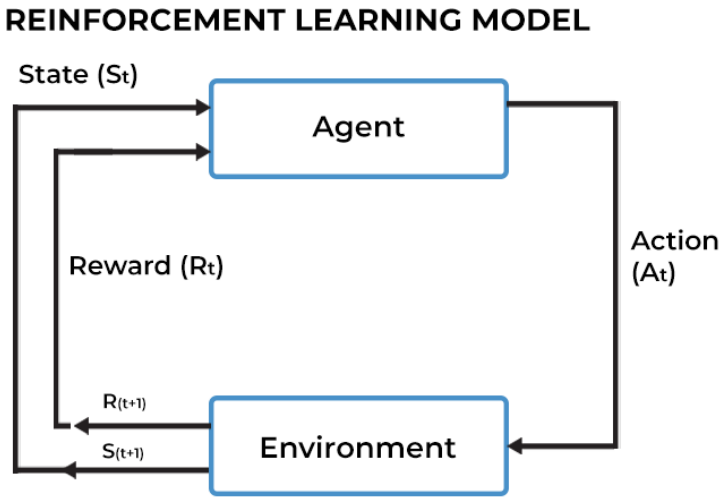


Figure 2.1: Reinforcement Learning Setup

Basic Concepts and Terminology :

- **Agent and Environment :** Agent is the object of interest that interacts with the environment over discrete time steps. The agent observes the present state of environment and for each time step, take action and observe the next state.
- **State (S) :** The State contains a set of variables that represents the situation information of agent in the environment.
- **Action (A) :** Action is the choice that is taken by the agent that changes the state of environment.
- **Reward (R) :** Reward is a scalar feedback that is provided for each action taken at each state. This reward function here is the maximising objective of that particular rl problem.
- **Policy (π):** Policy is the mapping between state and action such that for each state, the action taken maximizes the cumulative reward. The policy can be deterministic or stochastic.
- **Value function (V) :** The value function is the estimate of the long-term benefit of being in a particular state. It is expressed as the expectation of the cumulative reward starting from that state, following a particular policy.

2.2 Reinforcement Learning Algorithms

Reinforcement Learning (RL) algorithms can be categorized broadly into two main categories based on their approach to learning and utilizing environment dynamics:

- **Model Free Reinforcement Learning :** Model-free RL algorithms do not explicitly utilize a model of the environment dynamics. Instead, they rely on trial-and-error experiences, where the agent learns by interacting with the environment and receiving feedback in the form of rewards. By iteratively adjusting its actions based on past experiences, the agent gradually improves its policy without explicitly modeling how the environment behaves. Examples of model-free RL algorithms include Q-learning, SARSA, and Deep Q-Networks (DQN). These algorithms are often favored for their simplicity and ability to handle complex environments with unknown dynamics.
- **Model Based Reinforcement Learning :** Model-based RL algorithms involve learning and utilizing an explicit model of the environment dynamics. These algorithms aim to construct a model that accurately predicts how the environment will evolve in response to actions taken by the agent. By leveraging this model, the agent can plan ahead and make decisions that optimize long-term rewards more efficiently. Model-based approaches are typically more sample-efficient compared to model-free methods, as they can simulate multiple scenarios and predict outcomes without requiring extensive interaction with the real environment. Examples of model-based RL algorithms include Dyna-Q, Model Predictive Control (MPC), and some variants of Monte Carlo methods. These algorithms are advantageous in scenarios where data efficiency and planning capabilities are crucial.

2.3 Need for Model based Reinforcement Learning

Model based reinforcement learning overcomes various challenges that are inherently present in the Model free reinforcement learning approaches [7]. They are,

- **Planning and Decision Making :** By modelling the environment, the agent is enabled to predict future states and rewards that helps in taking informed decisions. This helps in optimising the actions that are not part of the real environment experience taken care by simulations of the modelled environment. This helps the agent provide a foresight on various sequences of actions and their consequences without actually taking them in the real world. So, the agent would exploit the environment effectively by taking dynamics aware decisions.
- **Non Stationary Environment :** Non-Stationary Environment: In environments that change over time, model-based RL techniques provide a way to continuously update the learned environment model with new experiences. This continuous updating allows model-based methods to maintain an accurate representation of the current state. In contrast, static policies learned through model-free approaches become obsolete in such scenarios [1].
- **Safety :** Since the agents has the idea of future states for action sequences, the agent is prevented from taking actions that could lead to risk prone regions. In safety critical domains, this methodology could help taking risk assessments and avoiding states that is unsafe for the agent.
- **Exploration vs Exploitation :** In reinforcement learning, balancing between taking unexplored actions (Exploration) and taking actions based on previous experience (Exploitation) is a fundamental challenge. Model-based approaches address this challenge in a nuanced way by identifying and prioritizing under-explored regions in the state spaces [10].
- **Convergence :** The environment models helps in targeted exploration that makes the learning effective and converges faster than model free approaches.

2.4 Neural Networks

Neural Networks (NN), often referred to as Artificial Neural Networks (ANN) or Deep Neural Networks (DNN), form the backbone of deep learning algorithms, a prominent subset of machine learning. A neural network is composed of interconnected processing units called neurons or nodes, which are linked by data transfer edges known as synapses. The terminology and structure of neural networks are inspired by the human brain's architecture, mirroring the way biological neurons signal to one another.

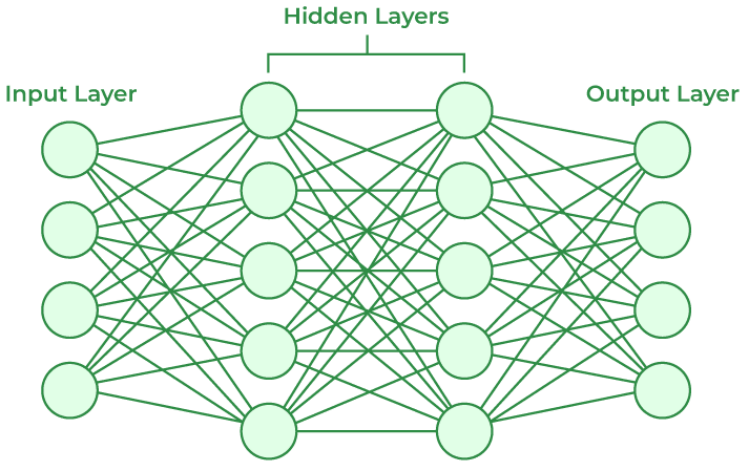


Figure 2.2: Artificial Neural Network

Unlike traditional algorithmic models that follow explicit programmed instructions, neural networks learn from data. This learning process involves training the network on a set of labeled data before it is deployed for actual tasks. During training, the network adjusts its parameters to minimize errors and improve accuracy through a method called backpropagation. This iterative process enables neural networks to learn complex patterns and make predictions, making them highly robust tools capable of solving problems that are challenging for conventional algorithms.

There are various types of neural networks, each with distinct architectures and applications. Among the most common are feed-forward neural networks, which consist of multiple sequential layers, each containing numerous nodes. In a feed-forward network, each layer is connected to the subsequent and preceding layers. When an input is received, the first layer processes it and passes the result to the second layer. This process continues through all layers until the final layer produces an output vector, representing the network's prediction or decision.

The operation of each layer in a feed-forward neural network involves two primary steps: the weighted sum of inputs followed by the application of an activation function. This mechanism allows the network to capture and model complex, non-linear relationships within the data.

Neural networks have a wide range of applications, including but not limited to classification, pattern recognition, value prediction, and decision modeling. Their versatility and capability to handle large, complex datasets have made them a focal point of research in data science and artificial intelligence. The study and development of neural networks and their variants continue to drive innovation and advancements in these fields, pushing the boundaries of what can be achieved with machine learning.

2.5 Need for Neural Network for environment construction

Model-based reinforcement learning (RL) methods have demonstrated remarkable prowess in tackling challenges posed by complex and high-dimensional state-action spaces. These approaches leverage models of the environment to plan and execute actions that optimize long-term rewards. However, constructing accurate environment models typically requires extensive data collection, which is often impractical in real-world scenarios due to the vastness of such spaces.

2.5.1 Neural Networks as Efficient Modeling Tools

Neural networks have emerged as powerful tools in this context, primarily due to their ability to approximate non-linear functions and effectively manage high-dimensional data. Unlike traditional methods that struggle with the complexity and dynamics of real-world environments, neural networks excel in capturing intricate patterns and dynamics inherent in data. This capability makes them particularly suitable for constructing environment models with minimal data samples [6].

2.5.2 Capturing Non-linear and Dynamic Environments

Real-world applications frequently encounter non-linear and continuously evolving environments. Neural networks are well-suited to model such complexities, offering a means to approximate the dynamics of these environments accurately. By fine-tuning neural network models, RL agents can derive valuable insights and make informed decisions, even when direct exploration of the environment is limited [9].

2.5.3 Versatility of Neural Network Architectures

The versatility of neural network architectures further enhances their utility in approximating non-deterministic environments encountered in various applications. Whether through convolutional networks for spatial data, recurrent networks for sequential data, or attention mechanisms for selective processing, neural networks offer flexible frameworks to adapt to diverse modeling requirements.

Various neural network architectures can effectively approximate such non-deterministic environments. In this work, we employ and compare Monte Carlo Dropout, Gaussian Ensemble Models, and Bayesian Neural Networks to model and analyze the dynamics of these environments. These architectures provide different methods of capturing uncertainty and variability in the environment, making them valuable tools for constructing robust and accurate models for RL.

2.6 Monte Carlo Dropout Model

Monte Carlo Dropout (MC Dropout) is a technique used to approximate Bayesian inference in neural networks, providing uncertainty estimates in predictions. It extends the traditional dropout regularization method by leveraging dropout during both training and inference phases to sample multiple predictions, thereby quantifying prediction uncertainty [4].

Core Concepts

Dropout is a regularization technique in neural networks to prevent overfitting. During training, dropout randomly ignores neurons with a specified probability, promoting robust feature learning. Traditional dropout turns off during inference, producing deterministic predictions.

In MC Dropout, dropout remains active during both training and inference phases. This introduces stochasticity during training, encouraging the network to learn a diverse set of robust features. During inference, MC Dropout generates multiple predictions for each input, allowing for uncertainty estimation.

2.6.1 Training Phase

1. **Regular Dropout Application**: During each training iteration, dropout is applied where neurons are randomly dropped with a probability p, creating different network architectures per iteration.

2. **Parameter Learning**: The network parameters (weights and biases) are updated based on the performance across these varied architectures. This regularization technique helps prevent overfitting and encourages the network to learn more generalizable features.

2.6.2 Inference Phase

1. **Dropout During Inference**: Unlike traditional dropout, MC Dropout keeps dropout active during inference. This means that for the same input, the network can produce different outputs due to different sets of active neurons, thus capturing model uncertainty.

2. **Monte Carlo Sampling**: To estimate uncertainty, MC Dropout involves running multiple forward passes with dropout enabled. Each pass yields a different prediction due to dropout's stochastic nature. These predictions can be averaged to obtain a final prediction and their variance used as a measure of uncertainty.

2.6.3 Theoretical Understanding

Applying dropout during training introduces an element of stochasticity, effectively turning the neural network into an ensemble of different architectures. Each architecture sampled during training can be viewed as a sample from the approximate posterior distribution $q(\Theta|D)$, where Θ denotes the network parameters and D represents the training dataset.

This enables the approximation of the predictive distribution:

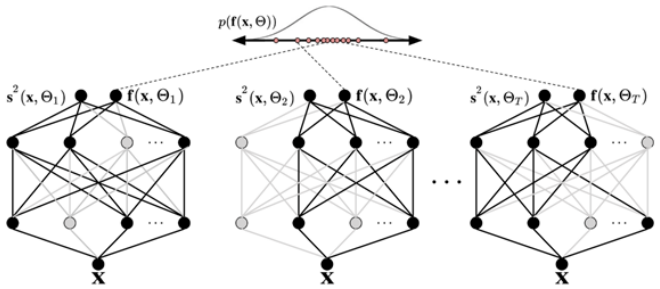


Figure 2.3: Dropout Configuration sampled from predictive distribution

$$p(y|x) \approx \int_{\Omega} p(y|x,\Theta)\, q(\Theta|D)\, d\Theta$$

Using Monte Carlo sampling, this integral can be approximated by averaging predictions over multiple samples $\Theta_t \sim q(\Theta|D)$:

$$p(y|x) \approx \frac{1}{T}\sum_{t=1}^{T} p(y|x,\Theta_t), \quad \text{where} \quad \Theta_t \sim q(\Theta|D)$$

Assuming the likelihood is normally distributed:

$$p(y|x,\Theta) = \mathcal{N}(f(x,\Theta), s^2(x,\Theta)),$$

where $f(x,\Theta)$ is the mean and $s^2(x,\Theta)$ is the variance of the predictions obtained through Monte Carlo dropout.

2.6.4 Advantages

1. **Uncertainty Quantification**: MC Dropout provides a principled way to estimate uncertainty in predictions, which is critical in applications such as medical diagnosis and autonomous driving where decision-making relies on confidence levels.

2. **Improved Generalization**: By maintaining dropout during inference, MC Dropout prevents over-reliance on specific neurons and encourages the network to generalize better, similar to its behavior during training.

3. **Simplicity and Efficiency**: MC Dropout can be implemented with minimal modifications to existing neural network architectures and training procedures, making it a practical approach for uncertainty-aware deep learning.

4. **Versatility**: This technique is applicable across various types of neural networks, including convolutional and recurrent networks, enhancing its utility across different domains and tasks.

By incorporating MC Dropout, neural networks not only improve their predictive accuracy but also gain the ability to provide confidence intervals for their predictions, thereby enhancing their reliability in real-world applications.

2.7 Gaussian Ensemble Model

The Gaussian Ensemble model is a method used to predict the mean and uncertainty of future states in a system. It leverages an ensemble of Gaussian neural network(GNN), each trained independently on the same dataset. This approach, known as Probabilistic Ensemble of trajectory sampling methods (PETS) [2], provides robust predictions by aggregating outputs from multiple models.

2.7.1 Training Phase

In the training phase, multiple Gaussian neural networks are trained on the dataset. Each network within the ensemble is initialized differently, which ensures that they evolve independently during training. This diversity in initialization helps in capturing different aspects of the data and improves the overall robustness of the ensemble.

During training, the objective is to optimize each network to predict the next state given the current state and possibly other contextual information. The training process typically involves minimizing a loss function that penalizes the difference between predicted and actual outcomes. By training multiple networks independently, the ensemble aims to cover a wide range of potential future states and their uncertainties.

2.7.2 Inference Phase

Once the ensemble of Gaussian models is trained, it is used for making predictions during the inference phase. Here's how the inference phase typically works:

- **Aggregation of Outputs :**
  For a given input x, each network in the ensemble produces its own prediction $f_i(x)$, where i ranges from 1 to N, the number of networks in the ensemble.
- **Mean Prediction (Expected Value) :**
  The mean prediction $\mu(x)$ is computed by averaging the outputs of all networks:

$$\mu(x) = \frac{1}{N}\sum_{i=1}^{N}\mu_i(x) \tag{2.1}$$

  where N is the number of networks in the ensemble and $\mu_i(x)$ denotes predicted mean of ith model. This mean represents the expected value or the best estimate of the output for the given input x.
- **Variance (Uncertainty Estimate) :**
  The variance $\sigma^2(x)$ is computed to quantify the uncertainty associated with the prediction. It measures the spread or variability of predictions across the ensemble:

$$\sigma^2(x) = \frac{1}{N}\sqrt{\sum_{i=1}^{n}\sigma_i(x)^2} \tag{2.2}$$

  where N is the number of networks in the ensemble and $\sigma_i(x)$ denotes predicted variance of ith model. A higher variance indicates higher uncertainty in the prediction, while a lower variance suggests greater confidence [5].

2.7.3 Need for Gaussian Ensemble Models

Embracing Uncertainty for Robust Decision Making

Unlike standard neural networks that churn out single point predictions, GNNs are designed to encode probability distributions. This capability elevates them from mere prediction machines to uncertainty-aware models. GNNs within PETS can capture two crucial types of uncertainties that plague the learning process:

- **Aleatoric Uncertainty :**
  This type of uncertainty reflects the inherent stochasticity or randomness present in the environment itself. For instance, consider an agent navigating a windy path. The wind introduces an element of chance, making the exact outcome of the agent's movement probabilistic.
- **Epistemic Uncertainty :**
  This uncertainty arises due to limitations in the data available for learning. When data is scarce, the model cannot be entirely certain about the true dynamics of the environment. Epistemic uncertainty essentially represents the model's own "confusion" about the world.

The Ensemble Advantage: A Wisdom of Crowds Approach

PETS employs an ensemble of GNNs, created using a technique called bootstrapping. Bootstrapping involves creating multiple datasets by sampling with replacement from the original data. Each GNN in the ensemble is trained on a unique bootstrapped dataset. As a result, each GNN learns a distinct distribution over the next state, given a current state and action.

The key lies in exploiting the disagreement between these ensemble members. This disagreement serves as a proxy for epistemic uncertainty. When the ensemble members produce significantly different predictions, it indicates that the model is unsure about the true outcome. Conversely, high agreement signifies greater confidence in the prediction.

2.8 Bayesian Neural Network

Neural networks typically produce single-point predictions, which limits their ability to express confidence in those predictions. Bayesian Neural Networks (BNNs), however, excel at capturing uncertainty, making them particularly useful in areas such as reinforcement learning (RL), where decision-making is crucial despite inherent unknowns.

2.8.1 Core Concepts

At the core of BNNs lies Bayesian inference, a statistical technique that updates probabilities based on new evidence. Applied to neural networks, this means the weights (parameters that influence the network's behavior) are not fixed values but rather probability distributions.

- **Prior Distribution**: Before any data is processed, a starting point is established for the weight distributions, potentially based on existing knowledge or a general distribution. This prior distribution encapsulates our initial beliefs about the parameters before observing any data.
- **Likelihood**: As data is fed into the network, the likelihood function calculates how well the current weight distributions explain the observed data. It represents the probability of the data given the parameters and is crucial for updating our beliefs.
- **Posterior Distribution**: Bayes' theorem combines the initial weight distributions (prior) with the data's influence (likelihood) to create a new distribution (posterior). This posterior distribution reflects the network's updated understanding of the weights after considering the data. The posterior encapsulates all information about the parameters after seeing the data.

The Bayesian update process is expressed mathematically as:

$$P(W|D) = \frac{P(D|W) \cdot P(W)}{P(D)}$$

where $P(W|D)$ is the posterior distribution, $P(D|W)$ is the likelihood, $P(W)$ is the prior, and $P(D)$ is the evidence.

2.8.2 Training Phase

Training a BNN involves estimating the posterior distribution of the weights given the training data. This process is often computationally intensive and may require approximations due to the intractability of exact Bayesian inference in high-dimensional spaces. Common techniques for approximating the posterior distribution include:

- **Variational Inference (VI)**: This method approximates the true posterior distribution with a simpler, parameterized distribution by optimizing a lower bound to the marginal likelihood. It converts the inference problem into an optimization problem, making it more tractable.
- **Markov Chain Monte Carlo (MCMC)**: This method samples from the posterior distribution to approximate it. Techniques such as Hamiltonian Monte Carlo (HMC) are popular for their efficiency in high dimensions. MCMC methods provide asymptotically exact samples from the posterior but can be computationally expensive.

- **Monte Carlo Dropout**: This technique uses dropout during both training and inference as an approximation to Bayesian inference, providing a practical and scalable approach to estimate uncertainty. By treating dropout as a form of approximate Bayesian inference, it allows for efficient uncertainty estimation.

2.8.3 Inference Phase

During inference, BNNs provide a distribution over possible outcomes rather than a single-point prediction. This allows for the estimation of uncertainty in the predictions. The process involves:

- **Sampling from the Posterior**: Multiple forward passes are performed using different samples from the posterior weight distribution. Each forward pass uses a different set of weights drawn from the posterior distribution.

- **Aggregating Predictions**: The outcomes of these forward passes are aggregated to produce a predictive distribution, capturing both the mean prediction and the uncertainty. This aggregation can be done by computing statistics such as the mean and variance of the predictions.

2.8.4 Advantages

BNNs offer significant advantages in stochastic reinforcement learning (RL) environments, where the system dynamics or reward structures are not fully known or are inherently random. These advantages include:

- **Uncertainty Quantification**: BNNs provide a measure of confidence in the predictions, which is crucial for making robust decisions in uncertain environments. This is particularly important in RL where the agent needs to balance exploration and exploitation.

- **Exploration-Exploitation Trade-off**: By quantifying uncertainty, BNNs can better manage the exploration-exploitation trade-off. High uncertainty can drive exploration, while low uncertainty can favor exploitation of known good strategies. This helps the agent to explore new strategies that might lead to better long-term rewards.

- **Robustness to Overfitting**: The probabilistic nature of BNNs helps prevent overfitting, which is particularly important in RL where the agent's actions influence future data distribution. By maintaining distributions over weights, BNNs can generalize better from limited data.

- **Improved Generalization**: By incorporating prior knowledge and updating beliefs based on observed data, BNNs can generalize better from limited data, enhancing performance in RL tasks. This is beneficial in environments where data is scarce or expensive to obtain.

2.9 Environment Modelling

2.10 Cartpole

2.10.1 System Description

The cartpole system is a classic problem in control theory and reinforcement learning, often referred to as the inverted pendulum problem. It consists of a cart that can move along a frictionless track with a pole connected to it via an unactuated joint. The objective is to balance the pole upright by applying forces to the cart, either to the left or right.

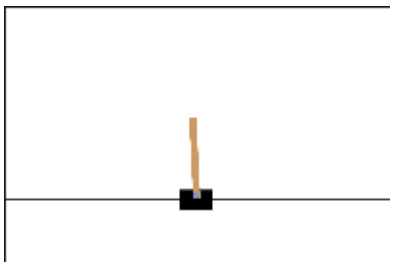


Figure 2.4: Cartpole Environment

2.10.2 Action Space

The actions correspond to the direction of the force applied to the cart:

Num	Action
0	Push cart to the left
1	Push cart to the right

Table 2.1: Cartpole Actions

2.10.3 Reward Structure

The reward structure is simple:

- **Reward:** +1 for every step taken, including the termination step. This encourages the agent to balance the pole for as long as possible.

2.10.4 State Space

The state of the system is represented by the following observations:

2.10.5 Episode Termination

The episode terminates when any of the following conditions are met:

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad($\sim -24°$)	~ 0.418 rad($\sim 24°$)
3	Pole Angular Velocity	-Inf	Inf

Table 2.2: Cartpole Observation Space

1. The pole angle exceeds 12° or is less than $-12°$.
2. The cart position exceeds 2.4 or is less than -2.4.

2.10.6 Physics and Dynamics of the Cartpole System

The cartpole system is inherently nonlinear due to the coupling between the cart and the pole dynamics. The equations of motion for the system can be derived using Newton's second law and considering the forces and torques acting on the cart and the pole.

Equations of Motion

Let:

- x be the position of the cart,
- $\dot{x}$ be the velocity of the cart,
- θ be the angle of the pole with the vertical,
- $\dot{\theta}$ be the angular velocity of the pole,
- F be the force applied to the cart,
- m be the mass of the pole,
- M be the mass of the cart,
- L be the length to the center of mass of the pole,
- g be the acceleration due to gravity.

The dynamics of the cartpole system can be described by the following set of nonlinear differential equations:

$$(M+m)\ddot{x} + mL\ddot{\theta}\cos(\theta) - mL\dot{\theta}^2\sin(\theta) = F$$

$$mL\ddot{x}\cos(\theta) + mL^2\ddot{\theta} - mgL\sin(\theta) = 0$$

These equations can be rearranged to solve for the accelerations $\ddot{x}$ and $\ddot{\theta}$:

$$\ddot{x} = \frac{F + mL(\dot{\theta}^2 \sin(\theta) - \ddot{\theta} \cos(\theta))}{M + m}$$

$$\ddot{\theta} = \frac{g \sin(\theta) - \cos(\theta) \left(\frac{F + mL\dot{\theta}^2 \sin(\theta)}{M+m} \right)}{L \left(\frac{4}{3} - \frac{m \cos^2(\theta)}{M+m} \right)}$$

2.10.7 Nonlinearity of the System

The nonlinearity of the cartpole system arises from the trigonometric terms $\sin(\theta)$ and $\cos(\theta)$ in the equations of motion. These terms create a complex relationship between the cart's position and velocity and the pole's angle and angular velocity. The system is nonlinear because the equations cannot be expressed as linear combinations of the state variables and their derivatives. The nonlinearity poses a significant challenge for control and requires sophisticated algorithms for stabilization and balance.

2.11 Mountain Car

The Mountain Car problem is a classic benchmark in control theory and reinforcement learning. It involves an underpowered car that must drive up a steep mountain road. The car's engine is not strong enough to directly drive up the steep slope, and the car must build up momentum by driving back and forth [8].

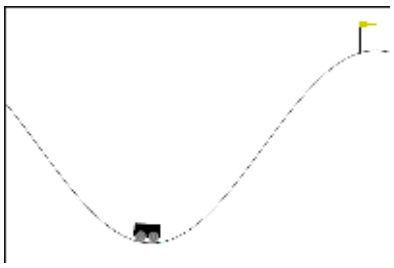


Figure 2.5: Mountain Car

2.11.1 Action Space

The actions correspond to the direction of the force applied to the car:

Num	Action
0	Accelerate to the left
1	No acceleration
2	Accelerate to the right

Table 2.3: Mountain Car Actions

2.11.2 Reward Structure

The reward structure is simple:

- **Reward:** -1 for every step taken until the car reaches the goal. This encourages the agent to reach the goal as quickly as possible.

2.11.3 State Space

The state of the system is represented by the following observations:

Num	Observation	Min	Max
0	Car Position	-inf	inf
1	Car Velocity	-inf	inf

Table 2.4: Mountain Car Observation Space

2.11.4 Episode Termination

The episode terminates when any of the following conditions are met:

1. The car reaches the goal position at 0.5.
2. The episode length reaches the maximum number of steps (usually 200).

2.11.5 Physics and Dynamics of the Mountain Car System

The Mountain Car system is inherently nonlinear due to the gravitational forces and the car's dynamics. The equations of motion for the system can be derived considering the forces acting on the car.

Equations of Motion

Let:

- x be the position of the car,
- $\dot{x}$ be the velocity of the car,
- F be the force applied to the car,
- g be the acceleration due to gravity,
- m be the mass of the car (typically normalized to 1 for simplicity).

The dynamics of the Mountain Car system can be described by the following nonlinear differential equations:

$$\ddot{x} = F - \frac{dV(x)}{dx}$$

where $V(x)$ is the potential energy function due to gravity, given by:

$$V(x) = \frac{1}{2}\cos(3x)$$

Thus, the force due to gravity can be expressed as:

$$\frac{dV(x)}{dx} = -\frac{3}{2}\sin(3x)$$

Substituting this into the equation of motion, we get:

$$\ddot{x} = F + \frac{3}{2}\sin(3x)$$

Since F is the control input, we typically have:

$$\ddot{x} = a + \frac{3}{2}\sin(3x)$$

where a is the acceleration applied by the agent (which can be -1, 0, or 1).

2.11.6 Nonlinearity of the System

The nonlinearity of the Mountain Car system arises from the $\sin(3x)$ term in the equation of motion. This term creates a complex relationship between the car's position and the force of gravity acting on it. The system is nonlinear because the equations cannot be expressed as linear combinations of the state variables and their derivatives. The nonlinearity poses a significant challenge for control and requires sophisticated algorithms for optimal performance.

2.12 Mountain Car Continuous

The Mountain Car Continuous problem is an extension of the classic Mountain Car problem in control theory and reinforcement learning. It involves an underpowered car that must drive up a steep mountain road. The car's engine is not strong enough to directly drive up the steep slope, and the car must build up momentum by driving back and forth. Unlike the discrete version, the Mountain Car Continuous problem features a continuous action space.

2.12.1 Action Space

The action space in Mountain Car Continuous is continuous:

Num	Observation	Min	Max	Unit
0	Position of the car along the x-axis	-Inf	Inf	position (m)
1	Velocity of the car	-Inf	Inf	position (m)

Table 2.5: Mountain Car Continuous Action Space

2.12.2 Reward Structure

The reward structure is as follows:

- **Reward:** $-0.1 \times \text{action}^2$ for each timestep. This encourages the agent to use minimal force to achieve the goal.

2.12.3 State Space

The state of the system is represented by the following observations:

Num	Observation	Min	Max
0	Car Position	-inf	inf
1	Car Velocity	-inf	inf

Table 2.6: Mountain Car Continuous Observation Space

2.12.4 Episode Termination

The episode terminates when any of the following conditions are met:

1. The car reaches the goal position at 0.5.
2. The episode length reaches the maximum number of steps (usually 200).

2.12.5 Physics and Dynamics of the Mountain Car Continuous System

The Mountain Car Continuous system is inherently nonlinear due to the gravitational forces and the car's dynamics. The equations of motion for the system can be derived considering the forces acting on the car.

Equations of Motion

Let:

- x be the position of the car,
- $\dot{x}$ be the velocity of the car,
- F be the force applied to the car,

- g be the acceleration due to gravity,
- m be the mass of the car (typically normalized to 1 for simplicity).

The dynamics of the Mountain Car Continuous system can be described by the following nonlinear differential equations:

$$\ddot{x} = F - \frac{dV(x)}{dx}$$

where $V(x)$ is the potential energy function due to gravity, given by:

$$V(x) = \frac{1}{2}\cos(3x)$$

Thus, the force due to gravity can be expressed as:

$$\frac{dV(x)}{dx} = -\frac{3}{2}\sin(3x)$$

Substituting this into the equation of motion, we get:

$$\ddot{x} = F + \frac{3}{2}\sin(3x)$$

Since F is the control input and can take any continuous value, we typically have:

$$\ddot{x} = a + \frac{3}{2}\sin(3x)$$

where a is the continuous acceleration applied by the agent.

2.12.6 Nonlinearity of the System

The nonlinearity of the Mountain Car Continuous system arises from the $\sin(3x)$ term in the equation of motion. This term creates a complex relationship between the car's position and the force of gravity acting on it. The system is nonlinear because the equations cannot be expressed as linear combinations of the state variables and their derivatives. The nonlinearity poses a significant challenge for control and requires sophisticated algorithms for optimal performance.

2.13 Pendulum

The Pendulum environment is a classic problem in control theory and reinforcement learning. It involves controlling a pendulum to keep it upright by applying torques at its pivot point. The goal is to swing the pendulum up to its upright position and keep it balanced there.

Figure 2.6: Pendulum Environment

2.13.1 Action Space

The action space in the Pendulum environment is continuous:

Num	Observation	Min	Max	Unit
0	Torque applied to the pendulum	-2.0	2.0	torque (N·m)

Table 2.7: Pendulum Action Space

2.13.2 Reward Structure

The reward structure is as follows:

- **Reward:** The reward at each time step is given by:

$$r = -(\theta^2 + 0.1 \cdot \dot{\theta}^2 + 0.001 \cdot \tau^2)$$

where θ is the angle of the pendulum from the upright position, $\dot{\theta}$ is the angular velocity, and τ is the torque applied. This encourages the agent to minimize the angle deviation, angular velocity, and torque.

2.13.3 State Space

The state of the system is represented by the following observations:

Num	Observation	Min	Max
0	Cosine of the angle	-1.0	1.0
1	Sine of the angle	-1.0	1.0
2	Angular velocity	-8.0	8.0

Table 2.8: Pendulum Observation Space

2.13.4 Episode Termination

The episode terminates after a fixed number of time steps (usually 200).

2.13.5 Physics and Dynamics of the Pendulum System

The Pendulum system is inherently nonlinear due to the rotational dynamics and the gravitational forces acting on the pendulum. The equations of motion for the system can be derived using Newton's second law for rotational systems.

Equations of Motion

Let:

- θ be the angle of the pendulum from the upright position,
- $\dot{\theta}$ be the angular velocity of the pendulum,
- $\ddot{\theta}$ be the angular acceleration of the pendulum,
- τ be the torque applied to the pendulum,
- m be the mass of the pendulum,
- L be the length of the pendulum,
- g be the acceleration due to gravity.

The dynamics of the Pendulum system can be described by the following nonlinear differential equation:

$$\ddot{\theta} = \frac{\tau - m \cdot g \cdot L \cdot \sin(\theta)}{m \cdot L^2}$$

Rearranging, we get:

$$m \cdot L^2 \cdot \ddot{\theta} = \tau - m \cdot g \cdot L \cdot \sin(\theta)$$

Since $m \cdot L^2$ is the moment of inertia I of the pendulum, we can rewrite it as:

$$I \cdot \ddot{\theta} = \tau - m \cdot g \cdot L \cdot \sin(\theta)$$

Finally, solving for $\ddot{\theta}$:

$$\ddot{\theta} = \frac{\tau}{I} - \frac{m \cdot g \cdot L}{I} \cdot \sin(\theta)$$

2.13.6 Nonlinearity of the System

The nonlinearity of the Pendulum system arises from the $\sin(\theta)$ term in the equation of motion. This term creates a complex relationship between the angle of the pendulum and the torque applied. The system is nonlinear because the equations cannot be expressed as linear combinations of the state variables and their derivatives. The nonlinearity poses a significant challenge for control and requires sophisticated algorithms for stabilization and balance.

2.14 Continuous State Variables in Markov Decision Processes

In the realm of Markov Decision Processes (MDPs), one significant challenge arises when dealing with continuous state variables. MDPs are a fundamental framework in reinforcement learning and decision-making under uncertainty, where an agent interacts with an environment to maximize a cumulative reward [11]. However, the practical implementation of MDPs becomes intricate when state variables are continuous due to several reasons.

Challenges of Continuous State Variables

1. **State Space Representation**: Continuous state spaces are typically infinite in size, making it impractical to enumerate or store all possible states explicitly. This poses a significant computational challenge because algorithms that operate over discrete state spaces cannot directly apply to continuous ones.

2. **State Transition Dynamics**: In a continuous state space, the transition dynamics between states are described by continuous probability distributions. This requires methods to compute or approximate these transitions, which can be computationally intensive and complex.

3. **Policy Representation**: Defining a policy in a continuous state space involves specifying actions for every possible state, which is not feasible due to the infinite nature of the space. Thus, policies need to be parameterized or represented in a way that allows for generalization across similar states.

Discretizing Continuous State Variables

To overcome the challenges associated with continuous state variables, discretization is a common approach. Discretization involves dividing the continuous state space into a finite number of discrete states [12]. This transformation enables the application of algorithms designed for discrete MDPs and simplifies the representation and computation processes. Here's how discretization addresses the challenges:

1. **State Space Reduction**: Discretization reduces the infinite state space to a manageable finite set of states. Each discrete state represents a region in the original continuous space, aggregating similar states together.

2. **Transition Modeling**: With a discretized state space, the transition dynamics can be modeled using discrete probability distributions. Instead of computing continuous transitions, algorithms can estimate transition probabilities between discrete states.

3. **Algorithm Compatibility**: Many MDP algorithms such as value iteration, policy iteration, and Q-learning are designed for discrete state spaces. By discretizing the state space, these algorithms can be directly applied without modification.

Considerations and Limitations

While discretization facilitates the application of MDP algorithms to continuous state spaces, it introduces some trade-offs and considerations:

- **Loss of Precision**: Discretization may lead to a loss of information or precision, especially if the state space is divided too coarsely.
- **Curse of Dimensionality**: Discretization can exacerbate the curse of dimensionality in high-dimensional state spaces, where the number of discrete states grows exponentially with the number of dimensions.
- **Algorithm Sensitivity**: The choice of discretization method and parameters can significantly impact algorithm performance and convergence.

2.15 Kullback-Leibler Divergence

The Kullback-Leibler (KL) Divergence is a measure of how one probability distribution diverges from a second, expected probability distribution. It is a non-symmetric measure of the difference between two probability distributions P and Q.

Given two discrete probability distributions P and Q defined on the same probability space, the KL divergence from Q to P is defined as:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{2.3}$$

In the case of continuous probability distributions, the KL divergence is defined as:

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \, dx \tag{2.4}$$

Here, $p(x)$ and $q(x)$ are the probability density functions of P and Q, respectively.

2.16 Novelty of this Dissertation Work

In this work, we propose a novel methodology for modeling and analyzing non-deterministic approximations of reinforcement learning (RL) environments using neural networks. While significant research has focused on RL algorithms, relatively little work has been done on effectively approximating RL environments themselves. To address this gap, we apply and compare three distinct neural network architectures to approximate the dynamics of various RL environments. We then analyze and evaluate the construction of these models based on their ability to accurately reconstruct Markov Decision Processes (MDPs) and the implications of their dynamic properties. This comprehensive analysis provides new insights into the strengths and limitations of each neural network approach in the context of environment modeling for RL.

Chapter 3

Methodology

3.1 Data Collection and Format

Sampling from Environment :
Gym environments of our interest are Cartpole, Mountain car, Mountain Car Continuous and Pendulum. For the mentioned gym environments, generate data and store it of the tabular form with columns state, action, next state, reward and done.

Inputs for gym environment : state, action.
Outputs of gym environment : next state, reward and done.

For each episode, start with a random state, take an uniform random action, observe the next state, reward and done. if done is true or the length of episode is more than 200, then end the episode. All the environments are introduced with a gaussian noise disturbance less than one-tenth of the value of the state variables to produce non determinism. Generate four samples of sample size 10k, 20k, 30k, 40k and 50k for each one of these environments.

States are of the format list $[state\_var_0, state\_var_1, state\_var_2, \ldots, state\_var_n]$, where $state\_var_i$ denotes the corresponding state variable.

Samples are generated of 2 formats, dataframe and data loader. the Dataframe has columns state, action, next state, reward and done. This dataframe will further be used in our analysis for MDP construction. Two kinds of data loaders are stored, train loader and test loader. The train loader and test loader is stored as pickle files of the format input - $[state, action]$ and output = $[nextstate, reward, done]$. These dataloaders are the training samples that will be used for training and testing the neural network models.

Since our area of interest is discrete action space and continuous state spaces, We discretize the action spaces for Mountain Car Continuous and Pendulum.

Mountain Car Continuous : Action space is bounded by $[-1, 1]$. This is discretized to put it in four bins $[-1, -0.5]$, $[-0.5, 0]$, $[0, 0.5]$ and $[0.5, 1]$.

Pendulum : Action space is bounded by $[-2, 2]$. This is discretized to put it in four bins $[-2, -1]$, $[-1, 0]$, $[0, 1]$ and $[1, 2]$.

3.2 Monte Carlo Dropout Architecture

3.2.1 MC_Dropout_Net Class

The `MC_Dropout_Net` class defines a neural network architecture with Monte Carlo Dropout.

Initialization

The model is initialized with parameters such as input size, output size, number of hidden layers, number of nodes in each hidden layer, activation function (ReLU), dropout probability, and the number of stochastic passes (`num_network`) for Monte Carlo Dropout.

Layers

The model includes:

- An input layer (`nn.Linear`) for transforming input features.
- Multiple hidden layers (`nn.Linear`) followed by dropout layers (`nn.Dropout`).
- An output layer (`nn.Linear`) for producing predictions.

Forward Method

During the forward pass:

- The input data undergoes multiple stochastic forward passes (`num_network` times).
- Outputs from each pass are aggregated, and the final output is the average of these outputs, providing an ensemble prediction.

3.2.2 Training and Testing Procedure

Environment and Data Loading

The code loads datasets (`train` and `test`) for different reinforcement learning environments (`env_names`) and various sample sizes (`num_samples_list`). Data is loaded using `torch.utils.data.DataLoader`.

Model Initialization

For each environment and sample size combination:

- The model's architecture is set up based on the environment's action space and input dimensions.
- Hyperparameters such as number of hidden layers, nodes per layer, activation function, learning rate, dropout probability, and number of networks for Monte Carlo Dropout are defined.

Training Loop

- The model is trained over a fixed number of epochs (`num_epochs`).
- Each epoch iterates over batches from the training set (`train_loader`).
- Mean Squared Error (MSE) loss is computed between model predictions and true labels.
- Gradients are computed using backpropagation (`loss.backward()`) and model parameters are updated (`optimizer.step()`).

Evaluation Loop

- After each epoch, the model's performance is evaluated on the test set (`test_loader`).
- Test loss (MSE) is calculated to assess model generalization.

Saving and Visualization

- Model checkpoints (`torch.save(model.state_dict(), model_weights_file_path)`) are saved periodically.
- Evaluation loss curves (generated using `matplotlib`) are saved as PNG files (`montecarlodropout_env_name_r`

Sample Generation

- The trained model is used to generate samples from the environment.
- The `Sample` class simulates actions, records states, rewards, and next states, and saves generated samples as CSV files (`montecarlodropout_env_name_num_samples_sample.csv`).

Parameter	**Specification**
Input Size	(State dimension + Action dimension)
Output Size	(Next State dimension + 2)
Number of hidden layers	3
Number of nodes in hidden layer	20
Activation Function	Relu
Dropout Probability	0.3
Learning rate	0.01
Number of Networks	1

3.2.3 Generated Files

- **Model Weights:** Saved as `montecarlodropout_env_name_num_samples.pth`
- **Evaluation Loss Curve:** Plotted and saved as `montecarlodropout_env_name_num_samples_evaluation_loss_curve.png`
- **Sample Data:** Generated and saved as `montecarlodropout_env_name_num_samples_sample.csv`

3.3 Bayesian Neural Network Architecture

3.3.1 Model Definition (BayesianNN Class)

The `BayesianNN` class defines a Bayesian Neural Network architecture using PyTorch:

- **Initialization (__init__ method):**
- **Parameters:** input_size, output_size, num_hidden_layers, hidden_layer_nodes, activation.
- **Layers:**
  - input_layer: Bayesian linear layer (bnn.BayesLinear) with input_size and output_size.
  - hidden_layers: List of Bayesian linear layers (bnn.BayesLinear) for num_hidden_layers.
  - output_layer: Bayesian linear layer (bnn.BayesLinear) with input_size and output_size.

3.3.2 Training and Testing Procedure

Environment Setup (env_names):

- Reinforcement learning environments are iterated over (env_names like 'CartPole-v1', 'MountainCar-v0', etc.).
- Each environment (env = gym.make(env_name)) is created using OpenAI's Gym library.

Dataset Loading (num_samples_list):

- For each environment and each dataset size (num_samples in num_samples_list), loads:
- Training dataset (train_dataset_loaded) using pickle.
- Testing dataset (test_dataset_loaded) using pickle.

Model Initialization and Hyperparameters

- num_hidden_layers = 3: Specifies 3 hidden layers in the neural network.
- hidden_layer_nodes = 20: Each hidden layer contains 20 nodes.
- activation = F.relu: ReLU (Rectified Linear Unit) activation function is used throughout the hidden layers.
- learning_rate = 0.01: Adam optimizer with a learning rate of 0.01 is employed.
- kl_weight = 0.01: Weight for the Kullback-Leibler (KL) divergence loss (bnn.BKLLoss) regularization term.
- num_epochs = 50: Number of training epochs set for training the model.

Model Training

- Instantiates the Bayesian Neural Network model (model) using the BayesianNN class with the specified hyperparameters.
- Defines the loss function (mse_loss) as Mean Squared Error (MSE) and the KL divergence loss (kl_loss) using bnn.BKLLoss.

Optimization:

- Sets up the optimizer (optimizer) using Adam optimizer (optim.Adam) to optimize the model parameters with the specified learning rate.

Training Loop:

- Iterates over num_epochs epochs.
- Sets the model in training mode (model.train()).
- Iterates over batches from the train_loader.
- Computes predictions (outputs) from the model and calculates the combined loss (mse_loss + kl_weight * kl_loss).
- Backpropagates the loss and updates the model parameters using the optimizer.

Evaluation on Test Data

- After each epoch, evaluates the model on the test data (test_loader):
- Sets the model in evaluation mode (model.eval()).
- Computes predictions (outputs) for the test data.
- Calculates the test loss using the same combined loss function as in training.
- Tracks and stores both training and test losses (train_losses, test_losses).

Performance Metrics:

- After training completes.
- Computes Mean Squared Error (MSE) and R-squared (r2_score) to assess the model's predictive performance on the test data.

Saving Results and Model

- Saves the trained model's state dictionary (model.state_dict()) using torch.save.
- Saves evaluation loss curves as plots (matplotlib.pyplot).

- Prints and saves MSE and R-squared values for each combination of environment and dataset size.

Generating Sample Predictions

- Loads the saved model (torch.load(model_weights_file_path)) to perform inference or generate sample predictions.
- Uses an unspecified function (generateSample) to generate sample predictions and saves them to a CSV file (sample_filename).

Parameter	Specification
Input Size	(State dimension + Action dimension)
Output Size	(Next State dimension + 2)
Number of hidden layers	3
Number of nodes in hidden layer	20
Activation Function	Relu
Learning rate	0.01
Number of Networks	1

Objective Loss function :

MSE(inputs and outputs) + KL Divergence(model weights with N(0,1)).

Optimizer : Adam.

Number of training epochs : 50.

3.4 Gaussian Ensemble Model Architecture

This section provides a detailed description of the procedure for training and testing a Gaussian Neural Network (GNN) ensemble. The GNN ensemble is designed to predict outputs as Gaussian distributions, leveraging multiple models to capture uncertainty and improve robustness in predictions.

3.4.1 Gaussian Network Architecture

The core architecture consists of the following classes:

- **Architecture**:
  - `fc1`, `fc2`: Fully connected layers processing input (`input_dim`) with `hidden_dim` neurons and ReLU activation (`torch.relu`).
  - `fc_mean`: Outputs the mean of the Gaussian distribution.
  - `fc_log_std`: Outputs the log standard deviation, exponentiated and clamped for stability (`torch.exp`, `torch.clamp`).

3.4.2 Ensemble Gaussian Network

The ensemble of Gaussian Networks is defined as:

- **Ensemble**:
  - Contains `num_models` instances of `GaussianNetwork`.
  - During forward pass, computes means and standard deviations from each model in the ensemble and stacks them.

3.4.3 Training and Testing Procedure

The procedure involves training and evaluating the ensemble on different environments and dataset sizes.

Initialization and Setup

- **Environment and Dataset Setup**:
  - Iterates over predefined environment names (`env_names`) and dataset sizes (`num_samples_list`).
  - Loads preprocessed datasets (`train_dataset_loaded`, `test_dataset_loaded`).
- **Model Initialization**:
  - Defines architecture-specific parameters: `input_size`, `output_size`, and hyperparameters (`num_layers`, `num_nodes`, `activation`, `num_ensembles`, `learning_rate`).
  - Instantiates `EnsembleGaussianNetwork` model and moves it to GPU if available.

Training Loop

- **Training**:
  - Executes a loop over `num_epochs`.
  - Sets model to training mode (`model.train()`).
  - Computes means and standard deviations (`means`, `stds`) for each batch from `train_loader`.
  - Calculates negative log likelihood loss using `torch.distributions.Normal`.
  - Optimizes model parameters using Adam optimizer (`optim.Adam`).

Testing Loop

- **Testing**:
  - Switches model to evaluation mode (`model.eval()`).
  - Evaluates model on `test_loader`, computes test loss.
  - Computes and logs test losses for each epoch.

Logging and Visualization

- Logs and saves training and testing losses (`train_losses`, `test_losses`).
- Plots evaluation loss curve (`test_losses` and `train_losses`) and saves as images.

Model Saving and Performance Logging

- Saves trained model state dictionary (`model.state_dict()`) periodically.
- Logs total training time for each session.

Hyper Parameter	Specification
Input Size	(State dimension + Action dimension)
Output Size	2*(Next State dimension + 2)
Number of hidden layers	3
Number of nodes in hidden layer	20
Activation Function	Relu
Learning rate	0.01
Number of Networks	5

Objective Loss function: Gaussian Likelihood.
Optimizer: Adam.
Number of training epochs: 50.

3.5 MDP construction

Train the mentioned neural network models on all the respective environment training samples present in the dataloader format.

Continuous State Variables in RL environment :

Cartpole, Mountain Car, Mountain Car continuous and Pendulum are those environments with continuous state variables. Since the models are non deterministic with continuous state variables, defining the dynamics of the environment poses a challenge. Due to the continuous nature of state variables, they can take any value between the defined bounds which leads to infinite number of possible states. For the purpose of interpretability and taking valuable insights out of them, we discretize them to form more meaningful states [3].

State Discretization : State discretization is a process of dividing the continuous state into discrete set of bins. Here each of these bins would consists of an interval of continuous state values to a manageable set of state space. By this way, we limit the size of state space leveraging the benefit of construction of Markov Decision Processes(MDP).

To discretize the bins, the bounds of each state variable has to be fixed. There are variables with theoritical infinite bounds, so we fix bounds for these variables by finding maximum and minimum observed values of each state space from all the original 10k, 20k, 30k, 40k and 50k datasets. The following are the observed bounds fixed for the state variables.

- **Cartpole :**

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-5.2	5.2
2	Pole Angle	-0.418	0.418
3	Pole Angular Velocity	-5.1	5.1

Table 3.1: Cartpole State Space Bounds

- **Mountain Car :**

Num	Observation	Min	Max
0	position of the car along the x-axis	-4.3	4.3
1	velocity of the car	-4.4	4.4

Table 3.2: Mountain Car State Space Bounds

- **Mountain Car Continuous:**

Num	Observation	Min	Max
0	position of the car along the x-axis	-4.1	4.1
1	velocity of the car	-3.9	3.9

Table 3.3: Mountain Car Continuous State Space Bounds

- **Pendulum:**

Num	Observation	Min	Max
0	$x = \cos(\theta)$	-1.0	1.0
1	$y = \sin(\theta)$	-1.0	1.0
2	Angular Velocity	-8.0	8.0

Table 3.4: Pendulum State Space Bounds

Discretizing Methods :

The following are the different methods we utilize to discretize the state space.

- **No of bins :**
  - **Description:** Divide the state space into a specified number of equally-sized bins within the given bounds.
  - **Example:** Suppose you have a state variable x with bounds [0, 1] and you choose 5 bins. Each bin would cover a range of $\frac{1-0}{5} = 0.2$. Therefore, the bins would be: [0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0).
- **No of bins(without bounds) :**
  - **Description:** Similar to the first method but includes bins that cover values outside the specified bounds.

  - **Example:** Using the same example with bounds [0, 1] and 5 bins. Bins would cover: $(-\infty, 0), [0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0), (1.0, \infty)$. This method ensures all possible values are accounted for, even those outside the specified bounds.

- **Bin size :**

  - **Description:** Divide the state space into bins of a specified size within the given bounds.
  - **Example:** If you choose a bin size of 0.1 for a state variable x with bounds [0, 1], the bins would be: [0, 0.1), [0.1, 0.2), ..., [0.9, 1.0). This method allows you to control the granularity of the discretization directly.

- **Interval :**

  - **Description:** Divide the state space based on specified intervals for each bin.
  - **Example:** Suppose you define intervals [0, 0.25, 0.6, 0.9, 1.0] for a state variable x with bounds [0, 1]. This would result in the bins: [0, 0.25), [0.25, 0.6), [0.6, 0.9), [0.9, 1.0). Any value of x falling within a specified interval is discretized into the corresponding bin.

After discretization, the states will of the format of list of discretized state variables(Whole numbers). For example, [1,2,2,1] is a discretized state in a cartpole environment that denotes the position and angular velocity are part of first bins of their respective state space. Similarly angle and velocuty are part of second bins of their respective state space.

Sample Averaging :

For discretized samples with data of state, action and next state, Sample averaging is a method for assigning transition probabilities for each (state, action, next state) pairs. For each (state, action) pair, find the probability of going to a particular next state. The MDPs for the discretized samples is obtained from the below formula.

$$T(s, a, s') \approx \frac{\text{Total number of transitions from } s \text{ under action } a}{\text{Number of transitions from } s \text{ to } s' \text{ under action } a}$$

3.6 Comparison Metrics

We use various set of metrics to understand and assess each neural network model with different number of training samples. These metrics helps assessing in performance, efficiency and reliability of all the non deterministic models. The key metrics include, training time, evaluation loss and Kullback-Leibler(KL) divergence of mdps.

3.6.1 Training Time

Training time is the time taken for a neural network model to train the model for given training samples. This helps us to measure the computational efficiency of model training process. This also states how long does a model takes to approximate the dynamics of environment. Training time is an important aspect that needs to be considered due to its practical implications, especially in cases where the environments are supposed to be adaptive and demand real time performance.

Training time here is measured in seconds. Lower the training time, more efficient the model considering the optimal values in evaluation metrics.

3.6.2 Evaluation Loss Graph

Evaluation loss measures the error obtained from the model's prediction with respect to the actual observed data. The evaluation loss graph is a plot of loss vs training epochs. This provides us the idea on convergence of each of these models over time and their capability to generalize. By this, we can also understand whether the model is overfitting or underfitting the training data. Models that have lower evaluation loss and smooth convergence are desired to best approximate the training samples.

3.6.3 MDP Comparison

Comparison of MDPs across various models provide the understanding of how far the models learn the dynamics of the environment. This indeed helps in understanding the transitions across state discretization. The comparison is based on the following,

Size of the MDPs :

Size of the MDPs denote the number of unique discretized states that are present in the samples generated from neural network models.

Kullback-Leibler (KL) divergence :

We use KL divergence to find the the difference in distribution of the MDPs. This tells us how far the probability distribution of neural network models align with the original environment.

3.7 Procedure

- **Collection of Data :**

  Collect the state, action, next state, reward, done data of all the afore mentioned environments and store them as mentioned in 3.1.

- **Construction of Environment Models :**

  Build neural network based models for all the training samples of size 10k, 20k, 30k, 40k and 50k for all the environments with architecture specified in 3.2, 3.3 and 3.4.

- **Construction of MDPs :**

  Collect 10k training samples for each of the neural network models built, similar to method mentioned in 3.1. After Collecting the training samples for all the neural netwrok models, we create 5 MDPs for 5 different discretizations on each model(3.5).

- **Comparison of MDPs :**

  Compare and analyze the MDPs formed by the comparison metrics provided in 3.6.

Chapter 4

Experiments and Results

4.1 Training Time :

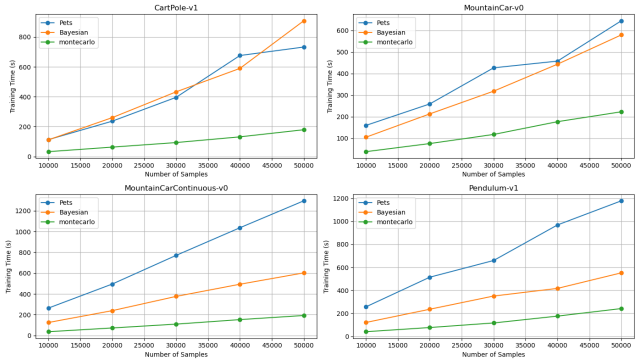


Figure 4.1: Training Time vs Number of Training Samples

Training Time Trends :

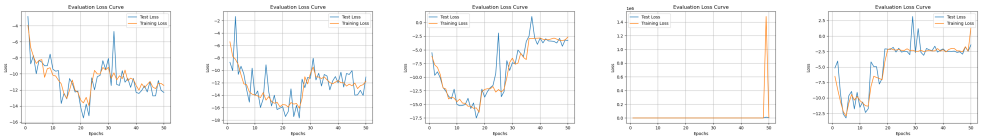
- For all the environments, the training time is increasing gradually with respect to the training samples.
- For all the three models, the scaling laws exhibited are different with respect to training time.

Model Comparison :

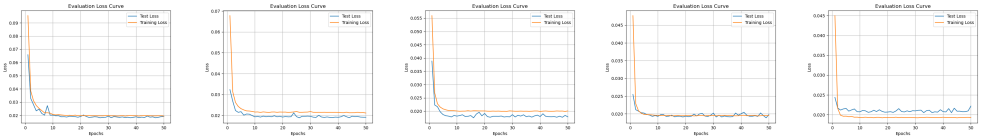
- **Pets Model :** Its is consistently the most time consuming model to train across all environments. This could be due to more complex gradient calculation involved.
- **Bayesian Neural Network Model :** Generally Bayesian model takes less time than Pets and more time than Monte Carlo Dropout model. Bayesian model provides middle ground between pets and Monte Carlo Dropout in terms of training time.
- **Monte Carlo Dropout Neural Network Model :** The least time consuming model in all the environments is Monte Carlo Dropout. It is the most effective model in terms of training time making it suitable for limited computational resource and time.

4.2 Evaluation Loss vs epochs :

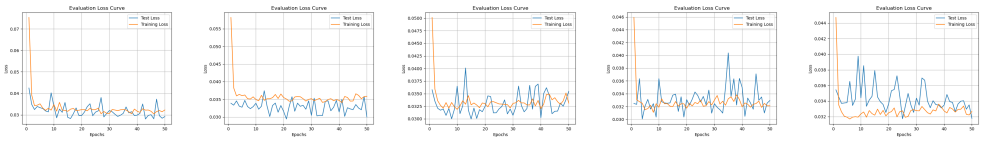
Cartpole :



(a) Pets 10k (b) Pets 20k (c) Pets 30k (d) Pets 40k (e) Pets 50k



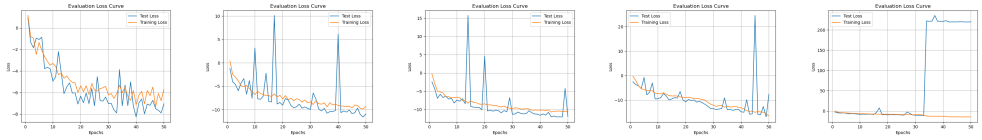
(f) Bayesian 10k (g) Bayesian 20k (h) Bayesian 30k (i) Bayesian 40k (j) Bayesian 50k



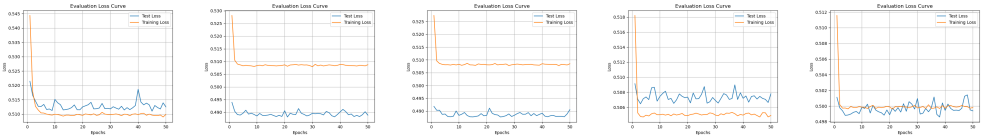
(k) MCD 10k (l) MCD 20k (m) MCD 30k (n) MCD 40k (o) MCD 50k

Figure 4.2: Evaluation Loss vs epochs on all models approximating Cartpole Environment.

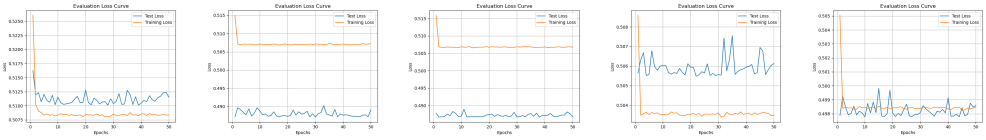
Mountain Car :



(a) Pets 10k (b) Pets 20k (c) Pets 30k (d) Pets 40k (e) Pets 50k



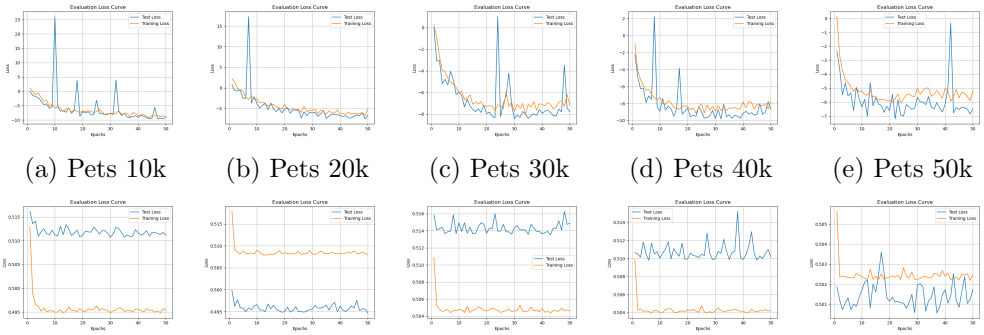
(f) Bayesian 10k (g) Bayesian 20k (h) Bayesian 30k (i) Bayesian 40k (j) Bayesian 50k



(k) MCD 10k (l) MCD 20k (m) MCD 30k (n) MCD 40k (o) MCD 50k

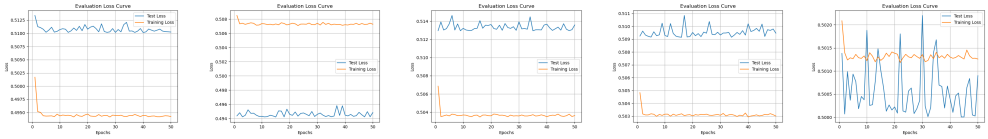
Figure 4.3: Evaluation Loss vs epochs on all models approximating MountainCar Environment.

Mountain Car Continuous :



(a) Pets 10k (b) Pets 20k (c) Pets 30k (d) Pets 40k (e) Pets 50k

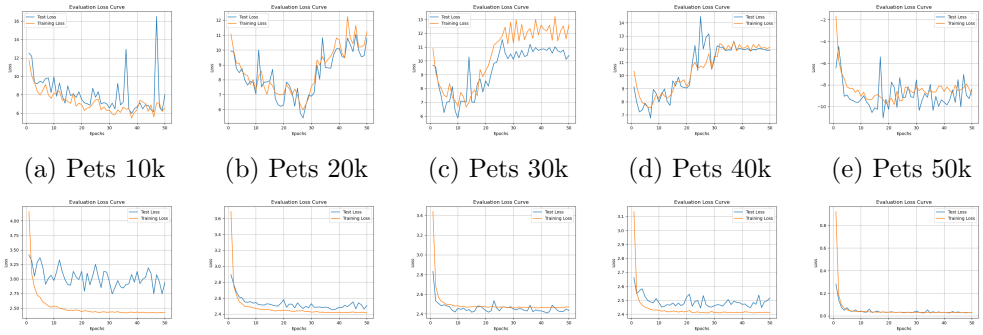
(f) Bayesian 10k (g) Bayesian 20k (h) Bayesian 30k (i) Bayesian 40k (j) Bayesian 50k



(k) MCD 10k (l) MCD 20k (m) MCD 30k (n) MCD 40k (o) MCD 50k

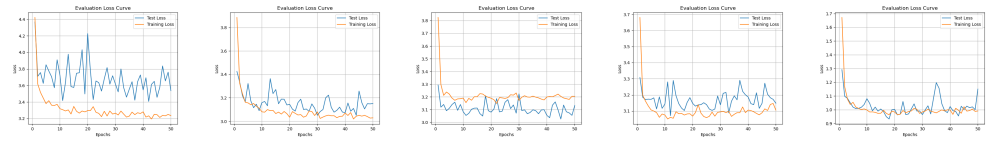
Figure 4.4: Evaluation Loss vs epochs on all models approximating MountainCarContinuous Environment.

Pendulum :



(a) Pets 10k (b) Pets 20k (c) Pets 30k (d) Pets 40k (e) Pets 50k

(f) Bayesian 10k (g) Bayesian 20k (h) Bayesian 30k (i) Bayesian 40k (j) Bayesian 50k



(k) MCD 10k (l) MCD 20k (m) MCD 30k (n) MCD 40k (o) MCD 50k

Figure 4.5: Evaluation Loss vs epochs on all models approximating Pendulum Environment.

Observation :

1. Variance of Pets model is more across all the training samples of all environments.

2. As we can clearly see for Bayesian Model, the training loss is more that the test loss for Mountain Car and Mountain Car Continuous till 40k sample. This denotes the model tends to underfit for Mountain and Mountain Car Continuous environment.

3. Evaluation loss of Monte Carlo is generally decreasing and bounded for cartpole and pendulum.

4.3 MDP Comparison :

Since, for 50k models of all the environments has the least evaluation loss, We analyze their MDPs. For comparing the MDPs constructed with one another, we use KL Divergence with baseline MDP. The Baseline MDP mentioned is the MDP constructed from original environment samples.

Cartpole 50k :

Discretization	Model Name	Size of MDP	KL Divergence with original MDP
[2,1,1,1]	Pets	2	2.29295
[2,1,1,1]	Monte Carlo	2	3.92593
[2,1,1,1]	Bayesian	2	2.40487
[1,2,1,1]	Pets	2	1.55957
[1,2,1,1]	Monte Carlo	2	3.07561
[1,2,1,1]	Bayesian	2	5.41390
[1,1,2,1]	Pets	4	2.25362
[1,1,2,1]	Monte Carlo	4	2.60839
[1,1,2,1]	Bayesian	4	3.14935
[2,2,1,1]	Pets	4	7.33485
[2,2,1,1]	Monte Carlo	4	4.95135
[2,2,1,1]	Bayesian	4	14.12805
[2,4,4,2]	Pets	63	70.16448
[2,4,4,2]	Monte Carlo	16	10.34798
[2,4,4,2]	Bayesian	28	43.67062
[2,2,4,4]	Pets	61	67.64601
[2,2,4,4]	Monte Carlo	16	10.34789
[2,2,4,4]	Bayesian	28	43.66563
[2,4,4,2]	Pets	54	69.35098
[2,4,4,2]	Monte Carlo	16	10.46850
[2,4,4,2]	Bayesian	16	35.04773

Size of Model : For simpler discretization, the all the models capture the states present in baseline MDP. More complext the discretization, Pets has MDP size greater than the baseline MDP. Pets tends to capture states which aren't part of baseline MDP. MDP size of Bayesian and Monte Carlo are lesser than the baseline MDP size.

KL Divergence : Pets model shows the best overall performance in terms of KL divergence for simpler discretizations. Generally, Bayesian model has the maximum value for KL divergence with baseline MDP. This tells that Pets captures the dynamics better than Monte Carlo and Bayesian. Monte Carlo model is more effective for more complex discretizations. Meanwhile Bayesian isn't able to retain the distribution of the original MDP to the level of Monte carlo and Pets.

Mountain Car 50k :

Discretization	Model Name	Size of MDP	KL Divergence with original MDP
[1,2]	Pets	2	0.00144
[1,2]	Monte Carlo	2	2.07414
[1,2]	Bayesian	2	7.36147
[2,4]	Pets	4	0.38331
[2,4]	Monte Carlo	4	2.09790
[2,4]	Bayesian	4	7.00234
[4,4]	Pets	8	0.38331
[4,4]	Monte Carlo	8	2.09790
[4,4]	Bayesian	8	7.00234
[4,6]	Pets	12	4.90049
[4,6]	Monte Carlo	12	2.17675
[4,6]	Bayesian	12	5.48467
[6,4]	Pets	8	0.71916
[6,4]	Monte Carlo	8	2.10099
[6,4]	Bayesian	8	6.95459

Size of Model : Mostly for the discretizations, Monte Carlo and Bayesian has lesser number of states than the original MDP. This denotes that Monte Carlo doesn't generalize properly to explore all discretized state values.

KL Divergence : Pets has the minimum value of divergence among other models in all five discretizations. Generally, Bayesian model has the maximum value for KL divergence with baseline MDP. This tells that Pets captures the dynamics better than Monte Carlo and Bayesian. Meanwhile Bayesian isn't able to retain the distribution of the original MDP to the level of Monte carlo and Pets.

Mountain Car Continuous 50k :

Discretization	Model Name	Size of MDP	KL Divergence with original MDP
[1,2]	Pets	2	0.00241
[1,2]	Monte Carlo	2	53.50166
[1,2]	Bayesian	2	0.00569
[2,1]	Pets	2	0.02616
[2,1]	Monte Carlo	2	2.89611
[2,1]	Bayesian	2	0.43969
[2,2]	Pets	4	0.15274
[2,2]	Monte Carlo	3	25.35256
[2,2]	Bayesian	4	0.92253
[2,4]	Pets	8	5.99659
[2,4]	Monte Carlo	4	24.05824
[2,4]	Bayesian	5	0.09665
[4,2]	Pets	8	15.21703
[4,2]	Monte Carlo	4	24.87557
[4,2]	Bayesian	5	0.35507

KL Divergence : Pets consistently outperforms Monte Carlo and Bayesian models in approximating the Mountain Car Continuous MDP across various discretizations. Monte Carlo offers a stable performance with slightly higher KL divergence compared to Pets. Bayesian model, while less accurate

overall, still provides reasonable approximations especially in simpler configurations.

Pendulum 50k :

Discretization	Model Name	Size of MDP	KL Divergence with original MDP
[1,2,1]	Pets	2	1.59560
[1,2,1]	Monte Carlo	2	49.59054
[1,2,1]	Bayesian	2	5.32252
[2,1,1]	Pets	2	1.54365
[2,1,1]	Monte Carlo	2	4.88494
[2,1,1]	Bayesian	2	7.35603
[2,2,1]	Pets	4	7.18580
[2,2,1]	Monte Carlo	4	145.42147
[2,2,1]	Bayesian	4	44.41007
[1,2,2]	Pets	4	7.87571
[1,2,2]	Monte Carlo	4	47.95672
[1,2,2]	Bayesian	4	25.31453
[2,1,2]	Pets	8	6.67920
[2,1,2]	Monte Carlo	8	5.28048
[2,1,2]	Bayesian	8	77.28758

KL Divergence : Pets and Bayesian models offer reasonable approximations for the Pendulum problem, with Pets generally performing better in simpler configurations. Monte Carlo consistently performs poorly across all tested configurations, indicating it may not be suitable for this problem. The choice between Pets and Bayesian models would depend on the specific requirements of the discretization complexity and the desired accuracy in approximating the original MDP for solving the Pendulum problem.

Chapter 5

Conclusion and Future Work

To sum up, this thesis has investigated the creation of Markov Decision Processes (MDPs) utilising neural network-based techniques, delving into the field of model-based reinforcement learning (RL). We have investigated the effectiveness of Bayesian neural networks, Gaussian ensemble neural networks, and Monte Carlo dropout neural networks in simulating the dynamics of continuous state environments through careful testing and analysis. Our analysis has highlighted the trade-offs between different discretizations on the advantages and disadvantages of each strategy.

Our results show that every neural network-based strategy has advantages and disadvantages. While robust uncertainty estimates are provided by Gaussian ensemble neural networks, they may have higher computational cost. While Monte Carlo dropout neural networks provide effective measurement of uncertainty, they need to be carefully regularised to avoid overfitting. While they can offer logical uncertainty estimates, training and assessing Bayesian neural networks can be computationally demanding.

There are a number of intriguing directions that model-based reinforcement learning research and development can go in the future. Analysing reachability in MDPs with an emphasis on identifying and mitigating undesired states or failure modes is one such avenue. Through comprehending the reachability of these "bad" states from starting circumstances, we may create more resilient reinforcement learning algorithms that can steer clear of or circumvent such obstacles.

Furthermore, integrating formal methods approaches like model testing and verification has a lot of potential to improve the safety and dependability of RL systems. We can make sure that RL agents behave within intended behavioural bounds, especially in safety-critical areas like autonomous driving and healthcare, by explicitly checking attributes of MDPs and implementing safety restrictions.

Bibliography

[1] Amadio, F., Dalla Libera, A., Carli, R., Nikovski, D., and Romeres, D. Model-based policy search for partially measurable systems, 01 2021.

[2] Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems 31* (2018).

[3] Dulac-Arnold, G., Evans, R., Sunehag, P., and Coppin, B. Reinforcement learning in large discrete action spaces.

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

[5] Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592* (2018).

[6] Luo, F.-M., Xu, T., Lai, H., Chen, X.-H., Zhang, W., and Yu, Y. A survey on model-based reinforcement learning, 06 2022.

[7] Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning 16*, 1 (2023), 1–118.

[8] Moore, A. W. Efficient memory-based learning for robot control. Tech. rep., University of Cambridge, Computer Laboratory, 1990.

[9] Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 7559–7566.

[10] Osband, I., and Roy, B. Why is posterior sampling better than optimism for reinforcement learning.

[11] Otterlo, M., and Wiering, M. Reinforcement learning and markov decision processes. *Reinforcement Learning: State of the Art* (01 2012), 3–42.

[12] Van Hasselt, H. Reinforcement learning in continuous state and action spaces.