BROWN
Computer Science

PhyDream: Enhancing Model-Based Reinforcement Learning with Jacobian Alignment in Differentiable Physics Engines

Mason Lee, Siddharth Boppana, Grant Landon, and Kaleb Newman

Brown University, Providence, RI, USA

Abstract

Model-based reinforcement learning (MBRL) agents rely on physics models to capture the dynamics of physical systems, enabling them to make informed decisions without facing real-world risks. While Physics-Informed Neural Networks (PINN) show promise in capturing system dynamics, their application to complex systems with rigid body mechanics remains challenging due to the absence of closed-form solutions. However, recent advancements in differentiable physics engines, exemplified by Brax, offer opportunities to bridge this gap. Leveraging the differentiability of Brax, we propose a novel approach aimed at enhancing the accuracy of world models in MBRL by incorporating physics priors. We introduce a novel loss function using Jacobian Alignment, which utilizes gradients from both Brax and the neural network to better understand how the environment changes relative to input states. By aligning these gradients, agents may be able to learn more nuanced representations of physical systems, thereby improving decision-making capabilities. Our approach demonstrates the potential to enhance the fidelity of world models in MBRL, paving the way for more effective and robust reinforcement learning systems. Although we had mixed results when it came to applying this to a Brax model, the fidelity of the Jacobian alignment loss on a PyTorch implementation is promising and can be utilized with many environments.

Keywords: Model Based Reinforcement Learning, Brax, Jacobian Alignment, JAX, PyTorch, Deep Learning, Physics Informed Neural Networks, World Models

Abbreviations: MBRL: Model Based Reinforcement Learning, JAL: Jacobian Alignment Loss, PINN: Physics Informed Neural Network, APG: Analytical Policy Gradient

1. Introduction

Model-based reinforcement learning (MBRL) has gained significant attention in recent years due to its potential for data-efficient learning. By leveraging a learned world model to simulate the environment dynamics, MBRL algorithms can imagine future outcomes of potential actions and reduce the number of real-world interactions required for policy learning [1, 2]. However, the accuracy of the world model is crucial for the success of MBRL. Approximated dynamics can lead to compounding errors, resulting in future states that deviate substantially from the truth or are physically impossible [3].

Physics-informed neural networks (PINNs) have shown remarkable promise in capturing the dynamics of various physical systems governed by closed-form partial differential equations (PDEs) [4, 5]. PINNs incorporate prior knowledge of the underlying physical laws into the neural network training process, enabling them to learn accurate and interpretable models. However, transitioning PINNs to more intricate systems with rigid body mechanics presents significant challenges. Unlike systems with succinct closed-form descriptions, the inherent complexity of rigid body mechanics makes it difficult to capture all the nuances of the dynamics without further information on the system [6].

Recent developments in differentiable physics engines offer a promising avenue for bridging the gap between traditional modeling using PINNs and data-driven approaches. Brax [3] is a fully dif-

ferentiable physics engine that allows the use of automatic differentiation to compute gradients of different parameters within the system. This breakthrough has led to successful implementations of analytical policy gradient (APG) methods [7], which use the physics engine's gradients to update the weights of a neural network control policy using the Jacobian of the output with respect to the input controls. While APG variants have shown performance improvements, the use of differentiable physics for learning dynamics models in MBRL world models has been under-explored. Jacobian matching techniques, such as JacNet [8] and Sobolev training [9], have been used for knowledge distillation between neural networks, but their application to knowledge distillation with a physics environment has not been extensively studied.

In this research we aim to investigate the potential of utilizing differentiable physics engines and Jacobian alignment techniques to improve the accuracy and sample efficiency of MBRL algorithms. We extend our method to use a popular stochastic world model framework that uses Variational-Autoencoder (VAE) and MDRNN and Recurrent Mixture Density Recurrent Neural Network (MDRNN) [10].

Our contributions are as follows:

- We introduce a framework for integrating differentiable physics and Jacobian alignment techniques into MBRL world models.
- We propose a novel Jacobian matching loss function that leverages the gradients provided by the differentiable physics engine to align the learned world model with the true dynamics.
- We demonstrate the effectiveness of our approach through extensive experiments on benchmark environments and compare it with existing MBRL methods.

2. Methods

2.1 World Modeling Techniques

Model-based reinforcement learning (MBRL) employs world models to simulate possible futures by predicting the next states of an environment from current observations and actions. Traditionally, world models compress observations through autoencoding, and a recurrent network for the prediction of future observations. We design or experiments around the landmark paper: 'World Models' [10] which designs their world model by utilizing a combination of Variational Autoencoders (VAE) and Mixed Density Network with Recurrent Neural Networks (MDN-RNN) as seen in Figure 1. This model takes advantage of the world model through the controller. The controller uses the compressed observation and the hidden state to make actions.

2.1.1 Role of VAE and MDN-RNN

The VAE component is typically employed to compress high-dimensional sensory inputs (like images found in Figure 2) into a lower-dimensional, meaningful latent space. This compression facilitates the handling of complex input data, making the system's dynamics more tractable for the predictive models that follow. The MDN-RNN effectively captures the temporal dynamics by predicting the gaussian mixture model (GMM) probability distribution of the next possible latent states given the current state and action. This allows the model to handle the inherent uncertainty in dynamics prediction, which is crucial for robust decision-making under varying conditions.

2.2 Error Propagation in MBRL

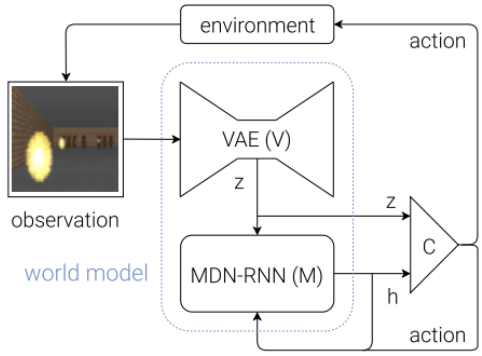


Figure 1. Flow Diagram of an Agent Model [1]

Figure 2. World Modeling Predictions [1]

2.2.1 Naive Dynamics Models

When treating our environment dynamics as a black box, conventional deep learning approaches train a neural network to be a function approximator of the state-action-to-next-state transition using the loss between predicted and true next states. The mean squared error (MSE) between the neural network's predicted next state and the simulated ground truth is the prevalent paradigm, as depicted in Figure 3. This loss function aims to capture the underlying dynamics of the environment.

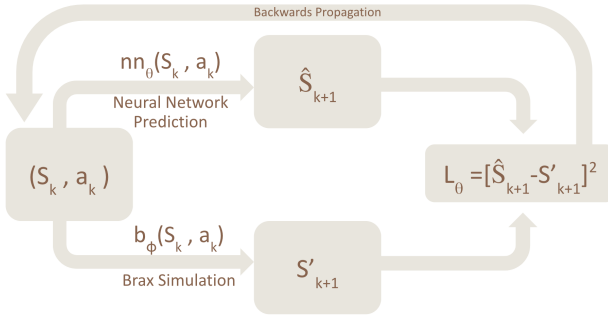


Figure 3. Standard Mean Squared Error Loss Between Neural Network Prediction and Brax Simulation

However, it does not inherently leverage the differentiability properties afforded by the Brax physics engine. To fully exploit Brax's gradient capabilities, it is imperative to shift focus from merely comparing predicted and true next states to examining how these states evolve in response to specific input states and actions. By analyzing the gradients of the state transition function with respect to the inputs, we can gain insights into the sensitivity of the dynamics to changes in the state and action variables, enabling a more nuanced understanding of the system's behavior.

2.2.2 Jacobian Alignment

In order to fully leverage the differentability of the Brax physics engine, the gradients must be directly calculated and utilized in the loss function from both Brax and the neural network. This allows for a more nuanced perspective into how the environment changes relative to the inputted states, as seen in Figure 4.

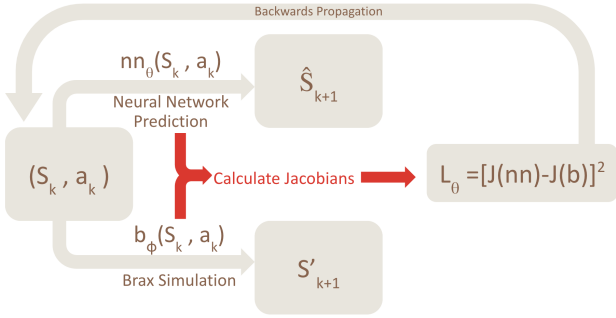


Figure 4. Novel Jacobian Alignment Loss Between Neural Network and Brax

The Jacobian alignment loss that was used in this study was element wise MSE between the neural network Jacobian and Brax Jacobian (1). We also experimented with other difference metrics such as the Frobenius Norm, and L1 Norm. Due to the architectural constraints of neural networks, the scale of values in the Jacobians were different than the Brax Jacobian. We applied min-max scaling before the loss calculation in order to normalize between the two Jacobians.

Jacobian Alignment Loss:

$$\boldsymbol{J}_{\text{predicted}} = \begin{bmatrix} \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial s_1} & \cdots & \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial s_n} & \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial a_1} & \cdots & \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial a_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial s_1} & \cdots & \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial s_n} & \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial a_1} & \cdots & \frac{\partial f_{\text{predicted}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial a_m} \end{bmatrix}$$

$$\boldsymbol{J}_{\text{true}} = \begin{bmatrix} \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial s_1} & \cdots & \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial s_n} & \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial a_1} & \cdots & \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_1}{\partial a_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial s_1} & \cdots & \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial s_n} & \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial a_1} & \cdots & \frac{\partial f_{\text{true}}(\boldsymbol{s},\boldsymbol{a})_k}{\partial a_m} \end{bmatrix}$$

$$\mathcal{L}_{\text{Jacobian}} = \frac{1}{k(n+m)} \sum_{i=1}^{k} \sum_{j=1}^{n+m} (\boldsymbol{J}_{\text{predicted}}(i,j) - \boldsymbol{J}_{\text{true}}(i,j))^2 \tag{1}$$

3. Workflow

3.1 PyTorch Dynamics: Double Pendulum

We start with a simple, closed form physics modeling problem: the double pendulum system. Taking two masses connected by idealized, massless rods, the system exhibits complex motion patterns influenced by gravitational forces.

Let the masses of the pendulums: m_1, m_2, lengths of the pendulum arms: l_1, l_2, gravitational acceleration: g. The angular accelerations $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are calculated using the equations:

$$\ddot{\theta}_1 = \frac{-g(2m_1 + m_2)\sin(\theta_1) - m_2 g \sin(\theta_1 - 2\theta_2) - 2\sin(\theta_1 - \theta_2)m_2(\dot{\theta}_2^2 l_2 + \dot{\theta}_1^2 l_1 \cos(\theta_1 - \theta_2))}{l_1(2m_1 + m_2 - m_2\cos(2(\theta_1 - \theta_2)))} + a_1,$$

$$\ddot{\theta}_2 = \frac{2\sin(\theta_1 - \theta_2)((\dot{\theta}_1^2 l_1(m_1 + m_2) + g(m_1 + m_2)\cos(\theta_1) + \dot{\theta}_2^2 l_2 m_2 \cos(\theta_1 - \theta_2)))}{l_2(2m_1 + m_2 - m_2\cos(2(\theta_1 - \theta_2)))} + a_2.$$

3.1.1 Numerical Integration

The next state of the system is computed using the Euler method for numerical integration, with a timestep $\Delta t = 0.1$ seconds:

$$\begin{aligned}
\theta_1' &= \theta_1 + \Delta t \dot{\theta}_1, \\
\theta_2' &= \theta_2 + \Delta t \dot{\theta}_2, \\
\dot{\theta}_1' &= \dot{\theta}_1 + \Delta t \ddot{\theta}_1, \\
\dot{\theta}_2' &= \dot{\theta}_2 + \Delta t \ddot{\theta}_2.
\end{aligned}$$

In the context of the 2D double pendulum dynamics, the Jacobian matrix represents the partial derivatives of the next state vector with respect to the current state and action vectors. This matrix is crucial for understanding how small changes in angles and angular velocities, as well as applied torques, affect the evolution of the system's state over time. Moreover, with the use of PyTorch, we can leverage automatic differentiation (autodiff) through numerical integration steps, enabling the computation of gradients efficiently for optimization and learning tasks.

3.2 Low-Dimensional Brax Environments

Building upon the foundational insights gained from PyTorch dynamics modeling, we transition to utilizing Brax, to simulate simple environments like the inverted double pendulum. In this phase, we focus on integrating Brax into our existing framework and adapting our neural network models to leverage the differentiability provided by the physics engine. By interfacing with Brax's API, we gain access to high-fidelity simulations, enabling us to train our models on realistic environments while maintaining differentiability for gradient-based optimization. This will allow us to utilize this framework on other Brax environments in the future.

3.3 Rigid-Body Brax Environments

Having established proficiency in modeling dynamics within simple Brax environments, we progressively scale up the complexity of our simulations to encompass more intricate scenarios. This involves changing parts of the framework to learn a larger system like half-cheetah. The goal is to challenge the robustness and generalization capability of our models. Through iterative experimentation and refinement, we aim to iteratively enhance the fidelity of our simulations while ensuring computational tractability and scalability.

3.4 Integrating Jacobian Alignment Loss in MBRL World Model

In this final phase of our workflow, we integrate the Jacobian Alignment Loss (JAL) into the framework of Model-Based Reinforcement Learning (MBRL). By incorporating JAL into the training objective of our world model, we seek to promote the convergence of the learned dynamics towards the true underlying physics, leading to improved sample efficiency and robustness in reinforcement learning tasks. To compute the JAL, we need to backpropagate through the decoder, Mixture Density

Network - Recurrent Neural Network (MDN-RNN), and encoder components of the world model. Let s_{pred} denote the predicted state, s the input state, a the action, z the latent representation, y the Gumbel-Softmax sample, π the mixture weights, μ the means, and σ the standard deviations of the Gaussian mixture model. The gradient of the predicted state s_{pred} with respect to the input state s can be computed using the chain rule:

$$\frac{\partial s_{\text{pred}}}{\partial s} = \frac{\partial s_{\text{pred}}}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial \pi}\frac{\partial \pi}{\partial z_{\text{enc}}}\frac{\partial z_{\text{enc}}}{\partial s} = f'_{\text{dec}}(z) \cdot \left(\sum_{i=1}^{K} y_i \sigma_i\right) \cdot f'_{\text{gs}}(\pi) \cdot f'_{\text{mdn-rnn}}(z_{\text{enc}}, a) \cdot f'_{\text{enc}}(s) \tag{2}$$

where f_{dec}, f_{gs}, $f_{\text{mdn-rnn}}$, and f_{enc} represent the decoder, Gumbel-Softmax, MDN-RNN, and encoder functions, respectively. The gradient of the predicted state s_{pred} with respect to the action a incorporates direct influences from the action on the MDN-RNN outputs:

$$\frac{\partial s_{\text{pred}}}{\partial a} = \frac{\partial s_{\text{pred}}}{\partial z}\left(\frac{\partial z}{\partial \mu}\frac{\partial \mu}{\partial a} + \frac{\partial z}{\partial \sigma}\frac{\partial \sigma}{\partial a} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial \pi}\frac{\partial \pi}{\partial a}\right) \tag{3}$$

$$= f'_{\text{dec}}(z)\left(\sum_{i=1}^{K} y_i \epsilon_i \cdot \frac{\partial \mu_i}{\partial a} + \sum_{i=1}^{K} y_i(\mu_i + \sigma_i \epsilon_i) \cdot \frac{\partial \sigma_i}{\partial a} + \left(\sum_{i=1}^{K} \sigma_i \epsilon_i\right) \cdot f'_{\text{gs}}(\pi) \cdot \frac{\partial \pi}{\partial a}\right) \tag{4}$$

where $\epsilon_1, \ldots, \epsilon_K$ are i.i.d. samples drawn from the standard normal distribution. To sample from the Gaussian mixture model output by the MDN-RNN, we employ the Gumbel-Softmax trick. The Gumbel-Softmax distribution allows for differentiable sampling from a categorical distribution by adding Gumbel noise to the logits and applying a softmax function with a temperature parameter τ:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{K} \exp((\log(\pi_j) + g_j)/\tau)} \tag{5}$$

where $g_1, \ldots, g_K$ are i.i.d. samples drawn from the Gumbel(0, 1) distribution. The latent representation z is then computed as a weighted sum of the means and standard deviations of the Gaussian mixture components, using the Gumbel-Softmax samples as weights:

$$z = \sum_{i=1}^{K} y_i(\mu_i + \sigma_i \epsilon_i) \tag{6}$$

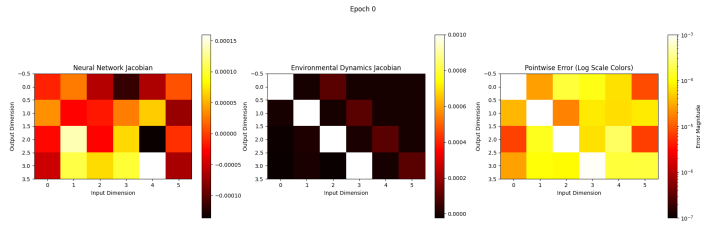
By incorporating these equations into the computation of the Jacobian Alignment Loss, we can effectively backpropagate through the entire world model and align the learned dynamics with the true physics captured by the differentiable physics engine.

4. Results

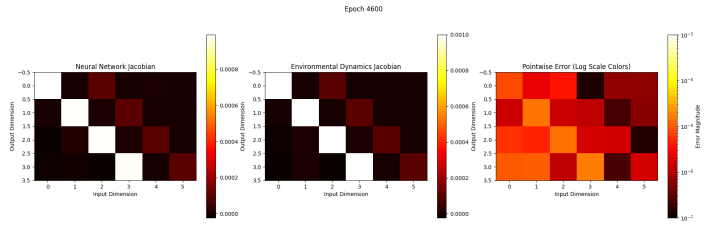
4.1 Double Pendulum: PyTorch

We began our investigation with a simple, closed-form physics modeling problem: the double pendulum system. We trained a two-layer fully connected neural network to predict the next state, given the previous state and action, using the PyTorch dynamics model. Our results demonstrate that the use of the Jacobian Alignment Loss (JAL) leads to a successful alignment over time, as seen in Figures 5a and 5b. These figures illustrate the progressive improvement in the alignment of the learned Jacobian with the true Jacobian as training progresses from epoch 0 to epoch 4600.

To further evaluate the effectiveness of JAL, we conducted experiments with varying rollout lengths. As shown in Figure 6, when rolling out for more time steps, the use of JAL led to a divergence in



(a) Jacobian Alignment at Epoch 0



(b) Jacobian Alignment at Epoch 4600

Figure 5. Comparison of Jacobian Alignment from Epoch 0 to Epoch 4600

the loss curves with respect to the rollout length. The incorporation of JAL resulted in a direct improvement in loss when using longer rollouts, indicating that JAL enhances the trajectories of a learned dynamics model over time. This observation suggests that JAL helps in capturing the long-term dynamics of the system more accurately.

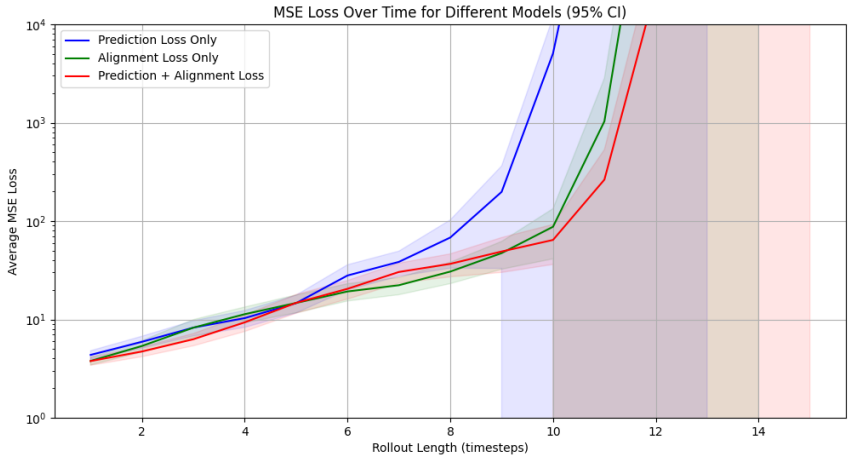


Figure 6. MSE Loss Over time for Different Models (95% CI)

We also assessed the generalization capabilities of the trained dynamics models by evaluating their performance on increasingly out-of-bounds testing rollouts. Figure 7 illustrates that the models trained with JAL exhibit better generalization compared to those trained without JAL. The JAL-trained models maintain lower loss values even when tested on states that lie outside the training distribution, demonstrating their ability to extrapolate and handle unseen scenarios more effectively.

Furthermore, we analyzed the training dynamics of the models by examining the training loss across

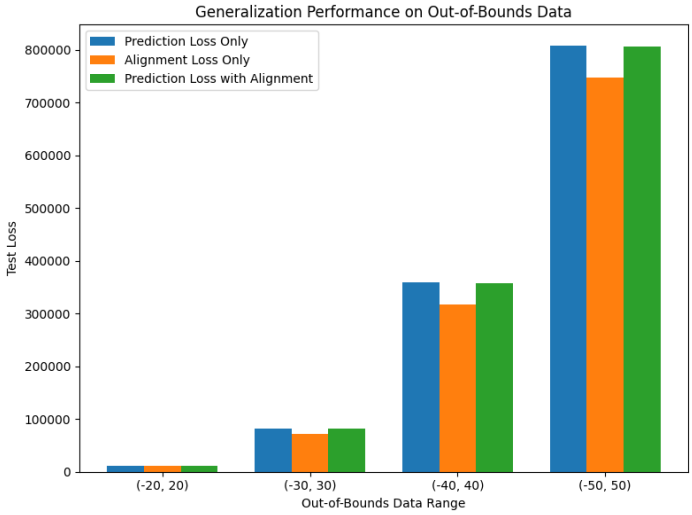


Figure 7. Generalization of Trained Dynamics Models: Evaluating on increasingly out-of-bounds testing rollouts

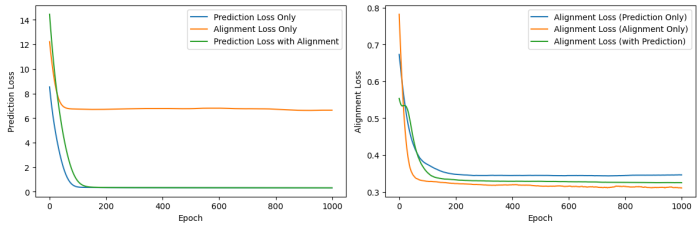


Figure 8. Training Loss Across

different epochs. Figure 8 shows the evolution of the training loss for models trained with and without JAL. The model trained with JAL achieves lower training loss and exhibits more stable convergence compared to the model trained without JAL. This observation highlights the beneficial impact of JAL on the optimization process and its ability to guide the model towards a better solution.

4.2 Brax Environments with NN

Building upon the insights gained from the PyTorch experiments, we ventured into more complex scenarios using the Brax physics engine, starting with the inverted pendulum environment. We experimented with three fully connected networks (FCNs), each subjected to different training regimes: MSE loss only, JAL only, and a combination of both.

Figure 9 presents the training dynamics, where the model leveraging both MSE and JAL showcased the lowest prediction loss and a relatively stable alignment loss. However, the models trained solely on JAL not only demonstrated higher prediction losses but also showcased an increase in alignment loss over time, contrary to expectations. This pattern suggests that while JAL may refine certain aspects of model training, it does not consistently lead to correct or stable dynamics, particularly under complex conditions where the physical interactions are more intricate.

The experiments further revealed a critical observation: all three networks empirically learned similar Jacobians, yet the alignment loss did not correlate with an accurate learning of dynamics. This anomaly suggests that the JAL might be exploiting certain dynamics incorrectly to achieve favorable

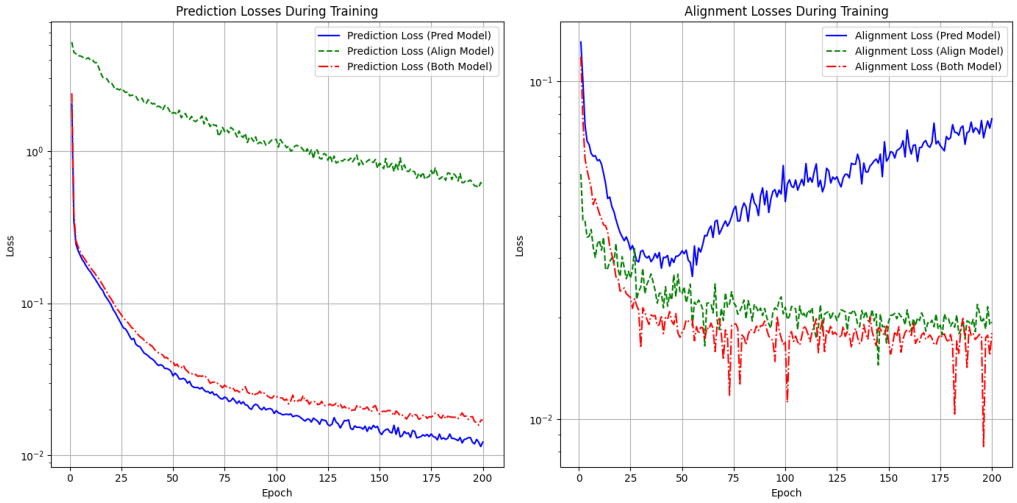
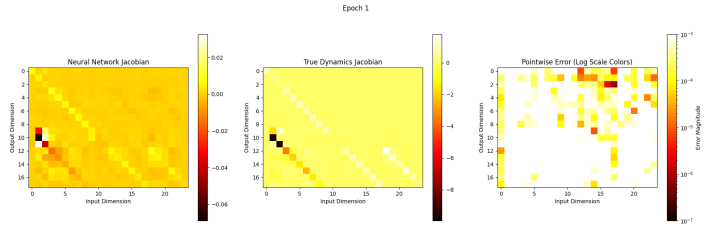
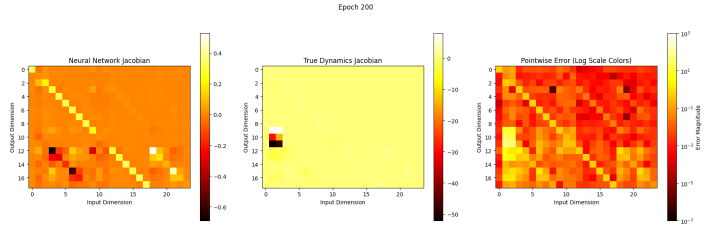


Figure 9. Comparison of prediction and alignment losses during training on the Brax inverted pendulum environment, illustrating different training dynamics across the models.

gradients, without truly capturing the underlying physical laws.



(a) Brax Half-Cheetah Jacobian at Epoch 1



(b) Brax Half-Cheetah Jacobian at Epoch 200

Figure 10. Visualization of Brax Half-Cheetah Jacobian changes from Epoch 1 to Epoch 200, demonstrating the challenges in capturing dynamic intricacies.

As we extended our investigation to the more demanding Brax half-cheetah environment, as visualized in Figure 10, the models struggled significantly to learn the relative scales of the Jacobian accurately. This difficulty underscores the complexities involved in modeling such dynamic systems and suggests that our JAL approach,a may require reevaluation or enhancement to better address these challenges.

The findings indicate that using JAL alone in rigid Brax environments does not guarantee proper alignment of Jacobians. Often, it converges to suboptimal solutions, indicating a highly sensitive and

challenging optimization process. Figure 11 showcases that despite these challenges, there are slight indications of enhanced stability in long rollout periods for the network trained with JAL, similar to observations in simpler PyTorch pendulum cases.

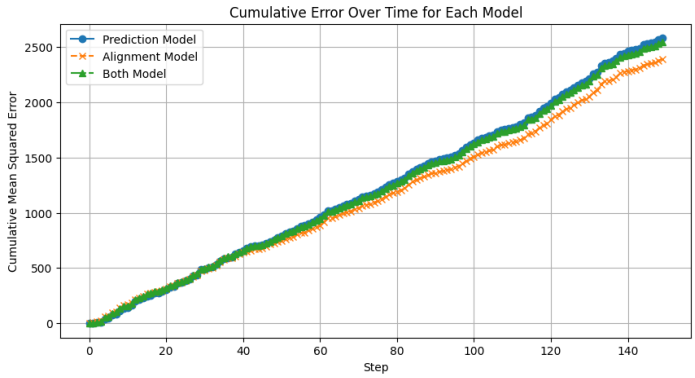


Figure 11. Cumulative loss for neural network dynamics models over rollouts, highlighting slight signs of stability from the JAL-trained network.

This nuanced exploration suggests that while the JAL framework holds promise, its current application in complex, real-world scenarios like those offered by Brax presents significant challenges that need to be addressed to fully realize its potential.

4.3 World Modeling

In the final phase of our experiments, we aimed to integrate JAL into the framework of Model-Based Reinforcement Learning (MBRL) using a stochastic world model. We extended our method to utilize a popular framework that combines a Variational Autoencoder (VAE), a Mixture Density Recurrent Neural Network (MDN-RNN) and an MLP as controller. Under this framework, the VAE is trained using reconstruction loss. The MDN-RNN is trained using a defined loss function for the MDN layer parametrized by number of mixtures. The controller is trained with CMA-ES (Covariance Matrix Adaptation Evolution Strategy). Finally, each of these modules are trained in succession - the world model portion with rollouts under a random policy. Our adaptation of this paper, along with its design choices, could contribute to the negative results we will detail. Firstly, in order to work with the Brax environment, we needed to repurpose the model in PyTorch. Additionally, the original model was trained on 2D gym environments, which necessitated a Convolutional VAE. For our Brax environments, we instead used a VAE with linear layers. In our implementation, we decided to train the VAE for 20 epochs, the MDN-RNN on 15 epochs, and then finetune the MDN-RNN with the JAL for 10 epochs.

As seen in our results (Figures 12, 13, and 14) the VAE cannot learn an effective compression of the observations. We belive that this is because the Brax states already include a minimal amount of information needed to describe the observation. Thus, the learning of the MDN-RNN is affected. Because of our world model's inability to learn, and the simplicity of our controller, we do not believe we had good testing grounds for our JAL loss. As can be seen from our results, the agent after fine-tuning the MDN-RNN with JAL loss does not get comparable results.

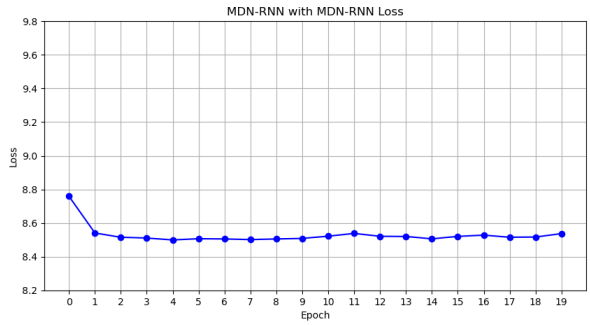


Figure 12. Epochs vs Loss for VAE

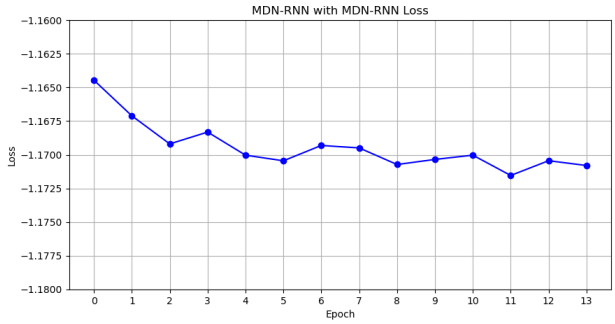


Figure 13. Epochs vs Loss for MDN-RNN (normal loss)

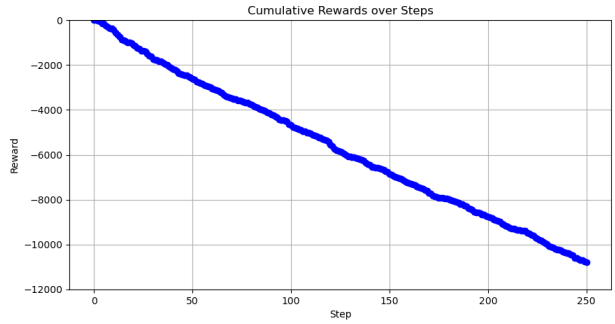


Figure 14. Steps vs Return after JAL Fine-tuning

5. Conclusion

In this paper, we demonstrate a novel method for aligning the trajectories of model based reinforcement learning algorithms. We introduce Jacobian alignment loss to constrain the learning of a next state prediction to the gradients of a step function. While we are able to demonstrate the ability of JAL to improve training in a simple dynamics model, as seen in Figures 5 and 6, we have found limited success in adapting the alignment for fully differentiable Brax physics environments. Despite previous work that found success in using JAL [11, 9], we believe that framing our problem as a physics-engine knowledge distillation presents several considerable differences. First, the scales do

not align between the learned Jacobian from a neural network and Brax's auto-differentiated Jacobian, as evident in Figure 10. Although different scaling methods were implemented to address this issue for the sake of the loss function, this may point to an underlying problem in approximating Jacobians with deep learning. When using only prediction loss regardless of scaling, we found that the relative values of the two Jacobians matched somewhat even though the magnitudes were off by a factor of ten or higher, as shown in Figure 9.

Second, the construction of our method to have better aligned training trajectories does not directly constrain the Jacobians of a neural network. The model is tasked with predicting the next state, while simultaneously being constrained on the gradients of this state. In an effort to directly constrain the Jacobian of the neural network, we also experimented with predicting the Jacobian matrix as a direct output of the model. By training the network to explicitly output the Jacobian, we aimed to enforce a stronger alignment between the learned dynamics and the true physics. However, despite providing the ground truth Jacobian as a target during training, this approach did not yield significant improvements in the model's ability to capture the underlying dynamics accurately. The challenges in aligning the scales and magnitudes of the learned Jacobian with the true dynamics persisted, suggesting that predicting the Jacobian directly does not necessarily resolve the fundamental differences between the neural network approximation and the physics engine. The discrepancy in scales between the learned Jacobian and the true dynamics Jacobian is a notable challenge in our experiments. While we successfully trained alignment models using min-max scaling across batches, z-score normalization did not yield similar results. The effectiveness of min-max scaling suggests that the relative relationships within the Jacobians are more important for alignment than matching their absolute magnitudes. However, the failure of z-score normalization indicates that the distributions of the Jacobian values differ significantly between the neural network and the physics engine, making it challenging to align them based on statistical properties alone.

Overall, our method shows that in certain environments, such as the double pendulum system in PyTorch (Figure 5), learning the Jacobian of a step function can be valuable information to an agent and may benefit the training process. We hope that future work will leverage the differentiability of Brax to implement gradient-based methods of learning world models, as explored in [3, 12]. We now shift our focus to potential theoretical problems with the JAL optimization problem. To backpropagate through the JAL and update the neural network's parameters, we need to compute the gradient of the loss with respect to the parameters:

$$\frac{\partial \mathcal{L}\text{Jacobian}(\theta)}{\partial \theta} = \frac{2}{k(n+m)} \sum i = 1^{k} \sum j = 1^{n+m} (\boldsymbol{J}\theta(s,a)i,j - \boldsymbol{J}\text{true}(s,a)i,j) \frac{\partial \boldsymbol{J}\theta(s,a)i,j}{\partial \theta}$$

$$\frac{\partial \boldsymbol{J}\theta(s,a)i,j}{\partial \theta} = \frac{\partial}{\partial \theta} \left(\frac{\partial f_\theta(s,a)i}{\partial x_j} \right) = \frac{\partial^2 f\theta(s,a)i}{\partial x_j \partial \theta} = \boldsymbol{H}\theta(s,a)_{i,j}$$

where $\boldsymbol{H}\theta(s,a)$ is the Hessian matrix of the function $f\theta(s,a)$ with respect to the input variables s and a, evaluated at the current parameter values θ. The Hessian matrix captures the second-order partial derivatives of the function output with respect to the input variables and the model parameters. Backpropagating through the Jacobian calculation means computing the gradient of each element in the Jacobian matrix with respect to the neural network's parameters. This involves taking the second-order derivatives of the network's output with respect to both the inputs and the parameters. We are implicitly doing two passes of backpropagation and inadvertently calculating the Hessian of the neural network weights. This can easily be seen in the context of a simple FCN backward pass, the unrolled gradient computation would involve the following steps:

1. Forward pass: Compute the neural network's output $f_\theta(s,a)$.
2. Jacobian computation: Compute the Jacobian matrix $\boldsymbol{J}\theta(s,a)$ by taking the first-order derivatives of the output with respect to the inputs.

3. Jacobian alignment loss: Compute the MSE loss between the neural network's Jacobian and the true Jacobian.
4. Backpropagation through the Jacobian: Compute the second-order derivatives of the network's output with respect to both the inputs and the parameters to obtain $\frac{\partial \boldsymbol{J}\theta(s,a)_{i,j}}{\partial \theta}$.

For non-linear dynamics we argue that the JAL loss introduces a highly non-convex optimization landscape. The loss function depends on second order derivatives of the neural network weights as we are backproping through the Jacobian partial w.r.t to the weights $\frac{\partial \boldsymbol{J}\theta(s,a)i,j}{\partial \theta}$. We believe that this can create an ill-conditioned optimization objective, especially when the state and action space are large and dense. If the eigenvalues of the Jacobian matrices are small, the gradients can vanish exponentially as they propagate backward through the layers. Conversely, if the eigenvalues are large, the gradients can explode, leading to unstable training. This problem is exacerbated in high-dimensional spaces, where the Jacobian matrices can have a wide range of eigenvalues. This is contrasted to MSE between predicted states. The MSE loss induces a relatively smooth optimization landscape, as it is a convex linear function with respect to the predicted next state. The gradients of the MSE loss with respect to the network parameters θ can be computed using standard backpropagation:

$$\nabla_\theta \mathcal{L}\text{MSE}(\theta) = \frac{2}{N} \sum i = 1^N (f\theta(s_i, a_i) - s_{i+1}) \cdot \nabla_\theta f_\theta(s_i, a_i)$$

Additionally, in highly non-linear systems, the Jacobian is only a local approximation in the state-action phase plane but does not necessarily extrapolate beyond an infinitesimally close region. The equation for the derivative of the Frobenius norm in the gradient alignment loss involves a term that can potentially amplify or diminish gradients, depending on the eigenvalues of the Jacobian matrices involved. This can lead to instability in training or difficulties in achieving convergence. We believe that this is reflected within our experimentation. Training on a pendulum environment, as shown in Figure 5b, we see that training only on JAL causes an increase in alignment error over time. We believe the overfitting to local jacobians causes a subsequent compounding error effect in the dynamics. This may introduce a recursive nature to the optimization process. The loss function depends on the Jacobian, which in turn depends on the network's parameters. During backpropagation, the gradients of the loss with respect to the parameters involve the second-order derivatives of the network's output, creating a feedback loop between the loss and the Jacobian. This recursive interaction can further complicate the optimization landscape and make it more challenging to find a stable and optimal solution.

6. Next Steps

6.1 Improved Optimization and Regularization

Regularization techniques that promote smoother and more stable Jacobians, such as spectral normalization or gradient penalties, can help alleviate some of the optimization difficulties. Additionally second-order optimization methods may handle the JAL loss optimization landscape in a more stable regime. Second-order optimization methods, such as Newton's method or quasi-Newton methods (e.g., BFGS, L-BFGS), explicitly incorporate second-order derivative information in the optimization process. These methods use the Hessian matrix or an approximation of it to capture the curvature of the loss function and can converge faster and handle second-order interactions more effectively compared to first-order methods.

6.2 World Models

Integrating the Jacobian alignment loss into stochastic autoencoder-based world models presents several challenges that warrant further exploration. The latent space physics may struggle with

the optimization landscape induced by the Jacobian alignment loss, as the second-order interactions and recursive nature of the loss can complicate the learning of stable and consistent latent dynamics. Moreover, we often seek to train World-Models end-to-end instead of sequential latent space then dynamics model training; the ability to incorporate physics into a changing latent basis is still not solved. The Gumbel-Softmax trick employed for differentiable sampling from the Gaussian mixture model introduces additional complexity, as the sample efficiency and effectiveness of this method in updating the probability distributions within the context of Jacobian alignment remain unclear. Moreover, the presence of the hidden state as a mixture of Gaussians poses backpropagation challenges, as the stochastic nature of sampling from this distribution can introduce variance and instability in the gradient estimates, potentially hindering the convergence and stability of the optimization process. How this scales with the number of mixture experts is still an open problem.

In the future we plan to use a larger model that will get better out-of-the-box performance on the Brax environment. This will allow us to have a better evaluation of our loss.

6.3 Implicit Differentiation in Bi-level Optimization

In our exploration of Jacobian alignment techniques within differentiable games, we propose further investigation into implicit differentiation (ID) as a pivotal tool for handling the nested structures of bi-level optimization problems. Bi-level optimization, where one set of optimization tasks is nested within another, can be effectively approached using ID to derive gradients for the upper-level task while respecting the solution structure of the lower-level task.

$$\begin{aligned} \text{Upper level:} \quad & \min_x f(x, y^*(x)) \\ \text{Lower level:} \quad & y^*(x) = \arg\min_y g(x, y) \end{aligned}$$

Where f and g are objective functions of the upper and lower levels, respectively. $y^*(x)$ is the solution to the lower level problem parameterized by x.

6.4 Jacobian Alignment as Bi-level Optimization

In the context of Jacobian alignment, the upper-level problem can be formulated as minimizing the discrepancy between the predicted Jacobian and an ideal Jacobian derived from the game dynamics. The lower-level problem involves optimizing the model parameters to accurately predict the system dynamics.

$$\begin{aligned} \text{Upper level:} \quad & \min_\theta |0\boldsymbol{J}_\theta - \boldsymbol{J}_{\text{true}}|0^2 \\ \text{Lower level:} \quad & \boldsymbol{J}_\theta = \nabla_\theta f_\theta(s, a) \end{aligned}$$

Here, θ represents the parameters of the model, f_θ the model's output, and $\mathcal{J}_\theta$ the Jacobian of f_θ with respect to its inputs.

6.5 Theoretical Enhancements via Implicit Differentiation

Implicit differentiation provides a robust framework for computing gradients through equilibrium points defined by implicit functions, which is crucial for the convergence and stability of Jacobian alignment techniques. By leveraging the properties of ID, we can ensure that the gradients used in

the upper-level optimization are accurate and stable, even when the lower-level problems involve complex, non-linear dynamics.

$$\frac{dy^*}{dx} = -\left(\frac{\partial^2 g}{\partial y^2}\right)^{-1} \frac{\partial^2 g}{\partial x \partial y}$$

This expression allows us to compute the derivative of the optimal y^* with respect to x, facilitating the gradient descent in the upper level.

In summary, our next steps will focus on harnessing implicit differentiation to refine our Jacobian alignment approach, ensuring rigorous convergence analysis and enhancing the stability of our optimization algorithms. This direction not only strengthens the theoretical aspects of our method but also enhances its applicability to complex, real-world systems where differentiable games play a crucial role.

Github Access

If you are interested in replicating our results, it will be hosted on github at https://github.com/PhyDreamV1/PhyDream.

Acknowledgement

A special thanks to our wonderful TAs David Tao and Saket Tiwari for giving guidance throughout the whole process of writing this paper. Thank you to Professor Ronald Parr for running such a wonderful class this semester.

References

[1] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. *DayDreamer: World Models for Physical Robot Learning*. arXiv:2206.14176 [cs]. June 2022. URL: http://arxiv.org/abs/2206.14176 (visited on 03/17/2024).

[2] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. *Mastering Diverse Domains through World Models*. arXiv:2301.04104 [cs, stat]. Jan. 2023. DOI: 10.48550/arXiv.2301.04104. URL: http://arxiv.org/abs/2301.04104 (visited on 03/17/2024).

[3] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. *Brax – A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. arXiv:2106.13281 [cs]. June 2021. DOI: 10.48550/arXiv.2106.13281. URL: http://arxiv.org/abs/2106.13281 (visited on 03/17/2024).

[4] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[5] George Em Karniadakis et al. *Physics-informed machine learning*. en. arXiv:1706.04859 [cs]. 2021. eprint: 1907.04490. URL: https://doi.org/10.1038/s42254-021-00314-5 (visited on 05/02/2024).

[6] Michael Lutter, Christian Ritter, and Jan Peters. *Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning*. 2019. arXiv: 1907.04490 [cs.LG].

[7] Nina Wiedemann, Valentin Wüest, Antonio Loquercio, Matthias Müller, Dario Floreano, and Davide Scaramuzza. *Training Efficient Controllers via Analytic Policy Gradient*. arXiv:2209.13052 [cs]. May 2023. URL: http://arxiv.org/abs/2209.13052 (visited on 03/17/2024).

[8] Jonathan Lorraine and Safwan Hossain. "JacNet: Learning Functions with Structured Jacobians". en. In: ().

[9] Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Świrszcz, and Razvan Pascanu. *Sobolev Training for Neural Networks*. en. arXiv:1706.04859 [cs]. July 2017. URL: http://arxiv.org/abs/1706.04859 (visited on 05/02/2024).

[10] David Haet et al. "World Models". In: *Neural Information Processing Systems* (2018), pp. 1–21.

[11] Suraj Srinivas and Francois Fleuret. "Knowledge Transfer with Jacobian Matching". en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2018, pp. 4723–4731. URL: https://proceedings.mlr.press/v80/srinivas18a.html (visited on 05/02/2024).

[12] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. "A Differentiable Physics Engine for Deep Learning in Robotics". English. In: *Frontiers in Neurorobotics* 13 (Mar. 2019). Publisher: Frontiers. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00006. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2019.00006 (visited on 03/17/2024).