Deep Lagrangian Networks Applied to Underactuated Systems

Matt Philippi, Tom Power, Craig Knuth, Kaustav Chakraborty
Robotics Institute
University of Michigan - Ann Arbor
Ann Arbor, MI 48109
philimat@umich.edu

Abstract

Black box deep learning methods can fit arbitrarily complex functions. However this often comes at the cost of data efficiency and generalization to data outside of the training set. This project investigates how to specialize deep learning to learning the dynamics of a mechanical system by incorporating physics as a prior in a general way. We take an existing method from the literature which injects Lagrangian mechanics into the structure of a neural network. We aim to verify that the injection of Lagrangian mechanics has benefits in both data efficiency and generalization. We first implement the method and verify it on a two-bar linkage mechanism which was presented in the original paper. We then explore a modification to the method in order to apply it to an under-actuated system. We are able to reproduce the results of the original paper on the fully actuated system and see data efficiency and generalization benefits. However, in the under-actuated case we find that the proposed method is outperformed in terms of data efficiency and generalization by a standard feed-forward neural network.

1 Introduction

Deep learning has been one of the most promising areas of research in the recent years. Approximating functions or running model free inferences [1] have shown promise in simulation. Much of the success in such topics however depends on the availability of large amounts of data which can be prohibitive for physical systems. Due to this debilitating dependency on data, significant research has been conducted to make such systems require fewer training instances [2] or augment a smaller data-set with "meaningful" noise and corresponding labels [3] in order to replicate a large enough dataset for training. Including physics as a prior not only helps in providing a more informed model selection, but also in better extraction of features from the training samples in a way that conforms with the laws of the underlying model which enforces the physics.

Our project explores using physics as a prior on an underactuated system, specifically a cartpole. We are implementing the method presented in [4] which proposes learning the manipulator equations of a physical system via a Deep Lagrangian Network (DeLaN). The system is capable of following a given trajectory by producing torques from positions, velocities, and accelerations of the system at each timestep. The previous work is able to generalize to new trajectories which is a promising avenue of research allowing learning of physical systems with less data and interpretable learned quantites.

Applying the work to an underactuated system is strictly more challenging as it introduces an additional complexity in the manipulator equations. We not only need to consider the external forces acting on the system (as in [4]), but also how the control inputs are converted into forces. In this paper we contribute three items: 1) an implementation of the DeLaN, 2) an adaption for application

Preprint. Under review.

to an underactuated cartpole system, and 3) empirical results of application to fitting to cartpole trajectories.

2 Related Work

Using a model predictive control approach requires a model of the system in order to properly determine the relationship between the control input τ and system state q. The control law may require either the forward model (1) or the inverse model (2) depending on the approach being used.

$$f(q, \dot{q}, \tau) = \ddot{q} \tag{1}$$

$$f^{-1}(q, \dot{q}, \ddot{q}) = \tau \tag{2}$$

Models like these can either be derived from physics or generated using machine learning on data. Deriving the physical model is specific the system under study and requires detailed knowledge about its characteristics. Past examples using machine learning include use of methods such as Linear Regression [5], Gaussian Mixture Regression [6], Gaussian Process Regression [6], Support Vector Regression [7], feed-forward [8], and recurrent neural networks [9] to find the model parameters that accurately fit the data.

The majority of the methods used in the past do not incorporate the underlying physics of the system into the learning method. One approach was to input the graph representation of the kinematic structure into the network [8]. The standard system identification technique for robot manipulators [10], uses the Newton-Euler formalism to derive physics features using the kinematic structure and the joint measurements and solves for the parameters using linear regression. Hard-coding physics features within a neural network to learn the dynamics parameters has also been attempted, but the parameters learned are not constrained to agree with physics since inertia matrix may not be positive definite and/or the parallel axis theorem may not hold [11; 12]. Additionally, use of the linear regression can produce underdetermined systems, only allows to infer linear combinations of the dynamics parameters and cannot be applied to closed-loop kinematics [10].

Only within the last few years have certain learning approaches been successful at ensuring that the parameters learned adhere to the laws of physics. Examples of these approaches include Deep Lagrangian Networks (DeLaN) [13] which uses Lagrangian Mechanics to formalize the dynamical model and Hamiltonian Neural Networks [14], which uses the Hamiltonian Mechanics. In our work, we selected to expand on DeLaN with the intention to use an inverse model for control.

3 Method and Approach

We seek to incorporate prior knowledge from the study of classical mechanics into the deep learning process, while maintaining the ability to fit to a wide range of mechanical systems. We do this by embedding the formalism of Lagrangian Mechanics within the structure of the neural network. In this section we will start with a brief primer on Lagrangian Mechanics before explaining how this structure is incorporated into the neural network.

3.1 Lagrangian Mechanics

Lagrangian mechanics is a formulation of classical mechanics from which we can derive the equations of motion of any mechanical system subject to holonomic constraints [1]. In the case of non-conservative forces and generalized coordinates the dynamics of a system are completely defined by the Lagrangian L. Generalized coordinates q are any set of coordinates that uniquely define the system configuration. The Lagrangian is a function of q and is typically defined by the potential and kinetic energies V and T respectively:

$$L(q, \dot{q}) = T(q, \dot{q}) - V(q) \tag{3}$$

[1]A holonomic constraint is any constraint which can be expressed as $f(q, t) = 0$ where q is configuration and t is time

If we further assume that the system in question is composed of multiple particles[2] then the kinetic energy can be expressed as:

$$T(q, \dot{q}) = \frac{1}{2}\dot{q}^T H(q)\dot{q} \tag{4}$$

Where $H(q)$ is the symmetric positive definite inertia matrix. By applying the calculus of variations we arrive at the Euler-Lagrange equation with non-conservative forces:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i \tag{5}$$

Since L is a function of q and $\dot{q}$ this is forms the basis for our inverse model. We can substitute L, T and $dV/dq = g(q)$ into the above expression to arrive at the manipulator equations with generalized forces which define the inverse model f^{-1}:

$$f^{-1}(q, \dot{q}, \ddot{q}) = H(q)\ddot{q} + \dot{H}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T H(q)\dot{q}\right)\right)^T + g(q) = \tau \tag{6}$$

It is common to group the components describing Centripetal and Coriolis forces into $c(q, \dot{q})$ such that:

$$f^{-1}(q, \dot{q}, \ddot{q}) = H(q)\ddot{q} + c(q, \dot{q}) + g(q) = \tau \tag{7}$$

Where:

$$c(q, \dot{q}) = \dot{H}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T H(q)\dot{q}\right)\right)^T \tag{8}$$

3.2 Incorporating Lagrangian Mechanics into Deep Learning

We see from equation 6 that the inverse model is fully parameterized by $H(q)$ and $g(q)$. DeLaN incorporates the structure of Lagrangian Mechanics by finding approximations $\hat{H}(q)$ and $\hat{g}(q)$ using feed-forward neural networks with parameters θ and ϕ respectively. Since $H(q)$ is symmetric positive definite, DeLaN instead seeks to approximate the lower triangular matrix square root of $\hat{H}(q)$ denoted as $\hat{L}(q)$.

$$\hat{H}(q) = \hat{L}(q;\theta)\hat{L}(q;\theta)^T \qquad \hat{g}(q) = \hat{g}(q;\phi) \tag{9}$$

We can find these approximations by minimizing the inverse model predictive loss. This also minimizes the violation of the physical law described by Lagrangian mechanics, which helps enforce physical plausibility of the system. The optimization is:

$$(\theta^*, \phi^*) = \arg\min_{\theta,\phi} \mathcal{L}\left(f^{-1}(q, \dot{q}, \ddot{q}; \theta, \phi), \tau\right) + \lambda\left(\theta^T\theta + \phi^T\phi\right) \tag{10}$$

$$\text{s.t.} \quad x^T\hat{L}\hat{L}^T x > 0 \ \ \forall x \in (x \in \mathcal{R}^n : x \neq 0) \tag{11}$$

Where $\mathcal{L}$ is a differentiable loss function, and λ is the ridge coefficient. We use mean squared error (MSE) for our loss function. The ridge loss is added to mitigate the effect of the Lagrangian not being unique. We train the network on a dataset of reference trajectories for the given system $\mathcal{D} = \left[\{(q_t, \dot{q}_t, \ddot{q}_t), \tau_t\}_{t=1}^T\right]^N$. The final form of the inverse model is thus:

$$f^{-1}(q, \dot{q}, \ddot{q}) = \hat{L}\hat{L}^T\ddot{q} + \frac{d}{dt}\left(\hat{L}\hat{L}^T\right)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T\hat{L}\hat{L}^T\dot{q}\right)\right)^T + \hat{g} = \tau \tag{12}$$

[2]Commonly for robotic manipulators we assume linked rigid bodies.

3.2.1 Enforcing positive definiteness of H

As seen in equation 11, we have a positive definite constraint on $\hat{L}\hat{L}^T$ which must be enforced. We do this by ensuring that all diagonal elements of $\hat{L}$ are positive. In order to do this, the diagonal elements l_d and the off-diagonal elements l_o are computed separately. The layer which computes the diagonal elements uses a softplus activation which outputs only positive values. We use a softplus here as when using ReLU we encountered the dying ReLU problem [15] where during training the ReLU layer outputs only zeros. This then gives zero gradient and the ReLU layer cannot recover. In addition we add a small scalar $b = 1 \times 10^{-9}$ to the diagonal elements which ensures positive definiteness.

3.2.2 Computing derivatives

From equation 12 we can see that in order to evaluate the inverse model we must compute the following derivatives in the forward pass:

$$\frac{d}{dt}\hat{L}\hat{L}^T = \hat{L}\frac{d\hat{L}^T}{dt} + \frac{d\hat{L}}{dt}\hat{L}^T \tag{13}$$

$$\frac{\partial}{\partial q_i}\dot{q}^T\hat{L}\hat{L}^T\dot{q} = \dot{q}^T\left(\frac{\partial \hat{L}}{\partial q_i}\hat{L}^T + \hat{L}\frac{\partial \hat{L}^T}{\partial q_i}\right)\dot{q} \tag{14}$$

The time derivative must be calculated manually since we cannot use automatic differentiation to differentiate with respect to time. The second derivative with respect to q_i can be differentiated via automatic differentiation, however this leads to issues when using automatic differentiation during the backward pass of the model. The derivatives in the forward pass can be calculated analytically and computed during the forward pass. The full derivations of the analytical gradients can be found in the original paper and are excluded for brevity.

3.2.3 Extending to underactuated systems

The current form of equation 12 assumes we are able to directly input the generalized forces to the system. For underactuated systems, this is no longer the case. By introducing a control matrix $B(q)$ such that $B(q)u = \tau$ we get the standard manipulator equations:

$$H(q)\ddot{q} + \dot{H}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T H(q)\dot{q}\right)\right)^T + g(q) = B(q)u \tag{15}$$

For an underactuated system B is not full row rank. Thus the inverse model becomes:

$$(B^TB)^{-1}B^T\left(H(q)\ddot{q} + \dot{H}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T H(q)\dot{q}\right)\right)^T + g(q)\right) = u \tag{16}$$

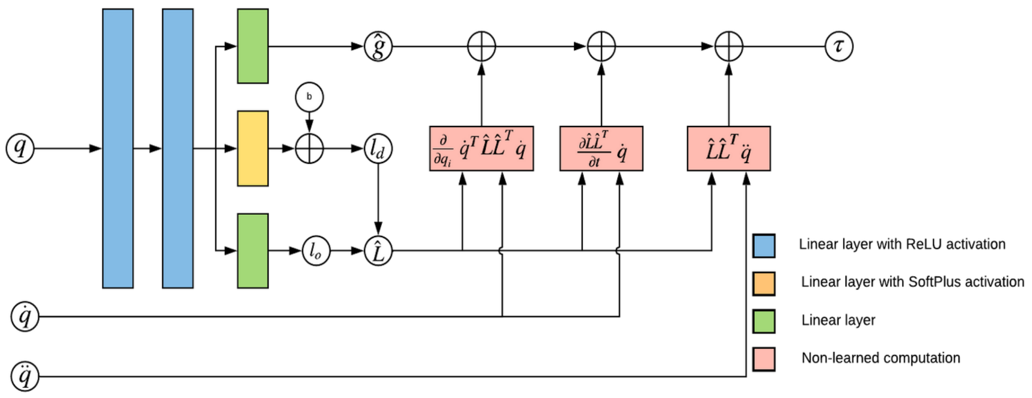
Where we project the output generalized forces onto the space of the actual system inputs. We evaluate two methods of dealing with the underactuated case. In the first we simply set the non-actuated dimensions to have zero force and DeLaN must learn to produce zero for these dimensions. In the second we set B explicitly and calculate the inverse model as in equation 16.

3.2.4 Network structure

We implement the method in PyTorch [16]. The structure of the network is shown in figure 1. We use two hidden layers of ReLU activations before the networks diverge to compute $\hat{g}, l_d, l_o$ separately. Both of these layers have 64 hidden units. From here l_o and $\hat{g}$ are computed with a linear layer, whereas l_d is computed with a softplus activated linear layer and a small offset as detailed in section 3.2.1. We then compute $\hat{L}$ and $\hat{g}$ and the relevant derivatives are calculated as in section 3.2.2.

4 Experiments and Results:

We evaluate our implementation by attempting to learn the dynamics for two example systems. The first example system is a fully actuated double pendulum system which was explored in the original



q
$\dot{q}$
$\ddot{q}$
b
$\hat{g}$
l_d
l_o
$\hat{L}$
$\frac{\partial}{\partial q_i}\dot{q}^T\hat{L}\hat{L}^T\dot{q}$
$\frac{\partial \hat{L}\hat{L}^T}{\partial t}\dot{q}$
$\hat{L}\hat{L}^T\ddot{q}$
τ
Linear layer with ReLU activation
Linear layer with SoftPlus activation
Linear layer
Non-learned computation

Figure 1: Structure of the deep lagrangian network

paper. We call this system *Reacher* to maintain consistency with the original paper. We use this experiment to verify our implementation.

The second system we explore is the cartpole environment. We explore this system to explore the application of our implementation to the underactuated case.

4.1 Reacher

The reacher system is two-link arm which has double pendulum dynamics. The configuration is given by $q = [\theta_1, \theta_2]^T$. The system is fully actuated in that we can apply torques independently to both links.

4.1.1 Data Collection

We use the same dataset as as in [4], where we consider trajectories of handwritten characters[3]. The dataset provides values for $\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}$ where $\mathbf{x} \in \mathbb{R}^2$ is the position of the pen point. We treat the bottom of the double pendulum as the pen point. In order to apply our method, we first convert the positions, velocities, and accelerations of the pen point to positions, velocities and accelerations in the joint space of the two-link reacher via inverse kinematics and then subsequently generate the corresponding torques necessary for the trajectories.

4.2 Cartpole

The second system we consider is the cartpole, shown in Figure 2. For the cartpole we have a pendulum attached to a cart, and the configuration is $q = [x, \theta]^T$. The system is underactuated in that we can only apply a horizontal force F to the cart, and cannot apply a torque directly to the pendulum. We apply our method to learn the dynamics of the system via 4 different types of trajectories.

4.2.1 Data Collection

The first type is to keep the cartpole upright while moving the cartpole from the zero position to some positive x value, e.g. between 1 and 2. The second type is cartpole swingup where the pendulum starts hanging straight down, and the goal is to swing the pendulum upright and balance. The third type is to revolve the pendulum. The pendulum starts straight up and the goal is to perform one full revolution before balancing straight up again. The fourth type is to keep the cartpole upright while moving the cartpole to some negative x value, e.g. between -2 and -1. Each of these trajectory types significantly differ from each other in terms of control effort and regions of the state space (aside from types 1 and 4). This differs from the character trajectories for the reacher environment as they are more or less are composed of straight and curved lines.

[3]http://archive.ics.uci.edu/ml/datasets/Character+Trajectories

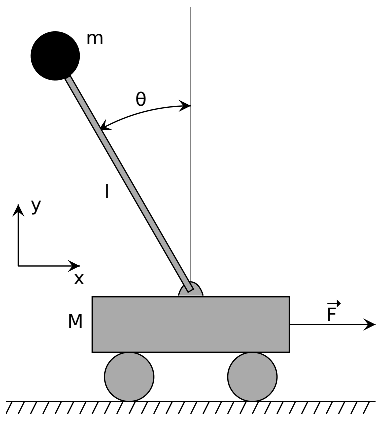


Figure 2: Standard cartpole environment. Here m, l, θ are the mass, length, and angle of the pendulum, M is the mass of the cart, and F is the force applied to the cart.

To generate the cartpole trajectories, we apply Model Predictive Control (MPC) techniques [17]. Briefly, to set up a MPC problem, we specify a dynamics model, any state and control constraints, a quadratic cost function of the state and control, and a horizon N. MPC computes the optimal control sequence over the horizon N, applies the first control action, and then recomputes the optimal control sequence at the next timestep. To introduce variability in the the four different types of trajectories we adjust the state and control constraints, cost function, and horizon, but keep the model fixed in order to have fixed values for H and g.

4.3 Evaluation Methodology

The primary metric of interest is mean-squared error (MSE) between the calculated torques and the predicted torques. MSE is a common metric in regression models and is what was used in [13], so we believe it is most appropriate for this task. As in [13], we chose to also develop a feed-foward neural network (FF-NN) as the baseline to compare against. The FF-NN has an input of 6 units for $\{q, \dot{q}, \ddot{q}\}$, two hidden layers of 64 units with ReLU activation function, and an output layer of two units for τ. Our core hypothesis is that the Deep Langrangian network will generalize better with less data to novel trajectories than a standard FF-NN. To first test this, we adopt a similar evaluation strategy as in [4]. We select n types of trajectories at random and train on a fixed number of each type of trajectory for both DeLaN and the FF-NN. Then, we evaluate the performance of DeLaN and FF-NN on a novel trajectory type. By iterating on this process, we expect to approximate the performance of our network on a novel trajectory.

4.4 Results

4.4.1 Reacher

We first introduce the double pendulum system with torque control at both joints as the "Reacher". Sample results of DeLaN evaluated for the Reacher system on the character trajectory **a** can be see in Figure 3. With only 1 sample of 1 random character, DeLaN is able to generalize well to a novel trajectory and starts to fit the system parameters. As the number of character types increase, DeLaN continues to improve the fit to the ground truth. Figure 4 shows a comparison between DeLaN and the FF-NN trained under the same conditions and evaluated on the same novel character **a**. It can be seen that DeLaN is able to fit the ground truth better than the FF-NN. Figure 5 shows the results of a more comprehensive test where DeLaN and the FF-NN were trained on multiple numbers of unique characters, evaluated on the remaining characters. This was done at 10 random seeds per level of unique training characters to provide the average and 95% confidence interval shown in the figure. DeLaN clearly outpeforms the FF-NN at all intervals of training data which highlights its ability to generalize better at least for the Reacher system and on the character dataset.

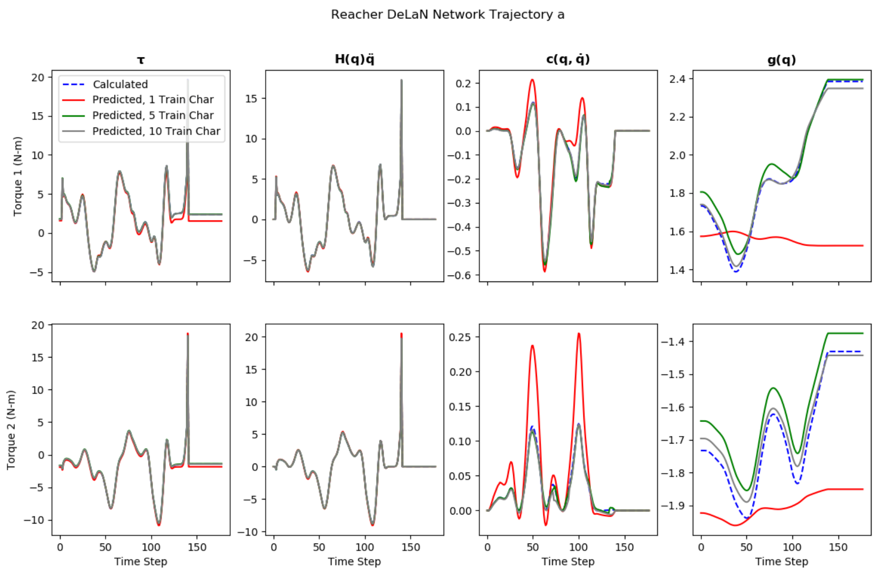


Figure 3: Reacher DeLaN predicted vs. calculated torques for character **a**. DeLaN was trained on 1 sample of 1, 5 and 10 random characters excluding **a** for 200 epochs.

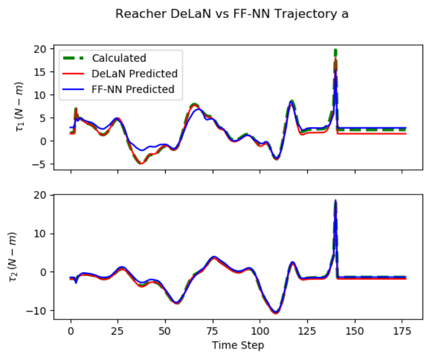


Figure 4: Reacher DeLaN and a FF-NN predicted vs. calculated torques for character **a**. Both networks were trained on 1 samples of a random character excluding **a** for 200 epochs.

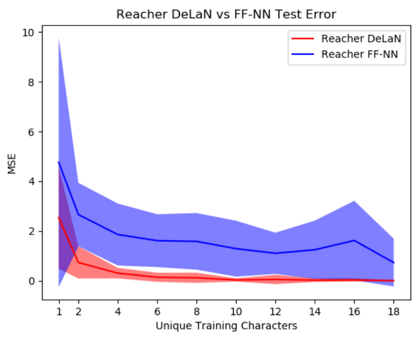


Figure 5: Average MSE for Reacher DeLaN and a FF-NN when trained on 1 sample of various numbers of unique characters for 200 epochs averaged out over 10 different seeds. The shaded region is the 95% confidence interval.

4.4.2 Cartpole

Sample results for DeLaN on CartPole system can be seen in Figure 6. The test trajectory shown is the revolution trajectory and the training trajectories are the remaining three types. DeLaN has a difficult time fitting the ground truth especially in the flat regions where torque limits are in place for the simulated trajectory. Figure 7 shows a comparison of DeLaN and the FF-NN trained under the same conditions. The FF-NN is able to fit this particular trajectory better than DeLaN and does not struggle as much in the flat regions. Figure 8 shows the same type of test as in Figure 5, but for the CartPole system and trajectories. This analysis shows that the FF-NN actually generalizes better than DeLaN to this combination of system and trajectories. We also evaluated DeLaN in a mode where we set $\tau_2 = 0$ at the end of the forward pass to mimic what the case would be in the true underactuated system. This did perform any better than the version which was free to choose τ_2. A summary of the results of the three different networks when tested on a novel trajectory and trained on the remaining three is shown in Table 1. Each network was trained for 200 epochs, 10 different seeds and 5 samples

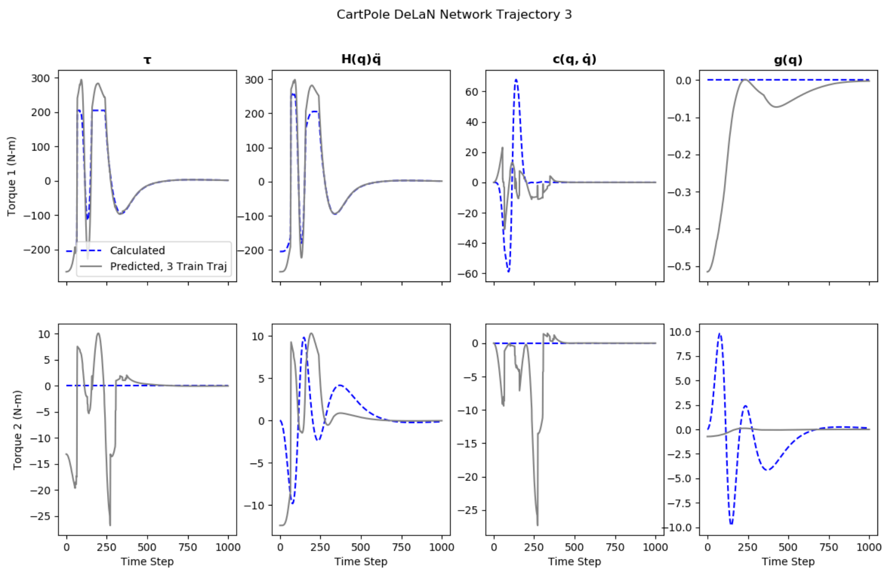


Figure 6: CartPole DeLaN predicted vs. calculated torques for the **revolution** trajectory. DeLaN was trained 5 samples of the each of the other 3 trajectory types excluding **revolution** for 200 epochs.

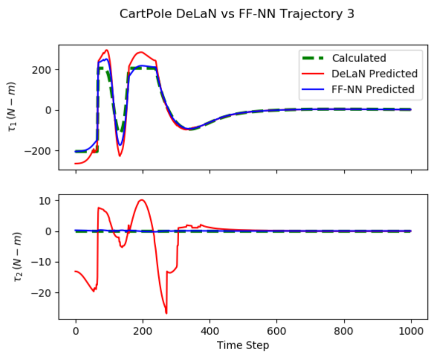


Figure 7: CartPole DeLaN and a FF-NN predicted vs. calculated torques for the **revolution** trajectory. Both networks were trained on 5 samples of the each of the other 3 trajectory types excluding **revolution** for 200 epochs.

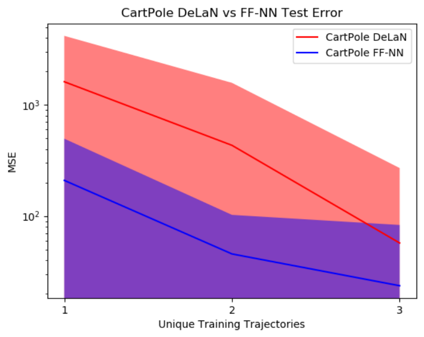


Figure 8: Average MSE for CartPole DeLaN and a Feed Forward Neural Network when trained on 5 samples of various numbers of unique trajectory types for 200 epochs averaged out over 10 different seeds. The shaded region is the 95% confidence interval.

of each trajectory type to obtain the mean and standard deviation MSE shown. The FF-NN was able to generalize better to trajectories 2 & 3 (swing-up and revolution) than DeLaN.

Table 1: MSE mean and standard deviation of Networks for Testing on a Novel Trajectory

Train Traj.	Test Traj.	DeLaN		DeLaN ($\tau_2 := 0$)		FF-NN	
		μ	σ	μ	σ	μ	σ
2,3,4	1	0.18	0.04	0.06	0.04	1.1	1.3
1,3,4	2	85.8	58.3	93.5	63.4	55.3	13.4
1,2,4	3	299	130	589	458	118	6.5
1,2,3	4	0.40	0.18	0.23	0.13	6.2	2.8

5 Discussion

Our code and dataset can be found at https://anonymous.4open.science/r/1996758e-d545-46c0-b84c-ddf72a554701/. All of the code was written by us and the networks were built using the PyTorch library.

Perhaps the most surprising result is that DeLaN did not generalize more successfully to the cartpole system. As mentioned in the dataset collection section, the trajectory types of the cartpole system did vary significantly in terms of control effort and regions of state space. Additionally, some trajectories were induced by a system under control constraints which was not present in the reacher system. However, in this case we expect that both the feed-forward and the DeLaN would both generalize poorly while the DeLaN would still maintain a slight advantage. One possible explanation is that the cartpole system (being underactuated) is more difficult to learn and requires more training trajectory types to see DeLaN eventually improve over the feed-forward network.

To improve performance on cartpole, our team attempted several changes without success. One, the DeLaN system is biased to learn continuous dynamics whereas the feed-forward network is free to learn discrete dynamics. To test the importance of the bias, our team increased the trajectory frequency from 20 to 200 Hz. However, this did not improve the results of the DeLaN. Two, as the manipulator equations include a $\mathbf{B}$ matrix converting torques into generalized forces, we tried training the network to learn the generalized forces (i.e. both τ_1 and τ_2) or simply the torque applied to the system (just τ_1). Unfortunately, neither of these approaches improved performance. Three, we originally trained solely on balancing and swingup trajectory types for the cartpole system so we generated more trajectory types (another type of balancing and revolution) to evaluate whether or not more training data would result in better performance. Even with three training trajectory types, which was sufficient for the reacher system, the performance did not improve past the feed-forward network.

Furthermore, none of our approaches to learning the second generalized force τ_2 of the cartpole system worked well since the second force is rarely zero. In this case, enforcing the positive definiteness of H by adding a small constant b, while helpful for training, may have prevented the second force from always being zero. Our results suggest that a more principled approach to handling underactuated systems is necessary in order to learn the dynamics, such as replacing the positive definite requirement of H with a positive semi-definite requirement. Adopting a different approach may also improve the DeLaN's ability to generalize beyond that of a feed-foward neural network.

6 Conclusion

In this paper, we presented an adapted DeLaN from [4], an implementation of that network, and results applied to an underactuated cartpole system. As the cartpole system is underactuated, it presents more complicated dynamics than the reacher system evaluated in the previous work. We found that, despite being able to replicate the results on the reacher environment, the same property of generalization did not hold for the cartpole environment. This may be due to too few training trajectory types. However, the feed forward network's performance will also improve with more training data, so it is unclear if the DeLaN would ever improve over the feed-forward network. In future work, we may adapt the problem formulation to explicitly handle underactuated systems by, for instance, removing the positive definite requirement of H with a positive semi-definite requirement.

Our project team considers our effort to be moderate success as we implemented an existing work, replicated the results of the paper, and applied the method to a novel and challenging dataset. The final result is a reduction in content from the goals in the project proposal which additionally intended to produce a forward model of the system in order to produce new trajectories instead of following a given trajectory. However, given the difficulties in performance on the cartpole system, the project team decided to focus on improving performance before moving on to significantly more challenging work.

Our team worked diligently to explore options to improve performance on cartpole, such as improving the controller frequency from 20 to 200 Hz, producing additional trajectory types beyond swingup and balancing, and learning both generalized forces and input torques. Without seeing any improvement in these three categories, we suspect the best course of action is a reformulation of the method to handle underactuated systems which is beyond the scope of this project.

References

[1] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, “Imitation learning: A survey of learning methods,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.

[2] O. Nachum, S. S. Gu, H. Lee, and S. Levine, “Data-efficient hierarchical reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[3] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, “Dart: Noise injection for robust imitation learning,” *arXiv preprint arXiv:1703.09327*, 2017.

[4] M. Lutter, C. Ritter, and J. Peters, “Deep lagrangian networks: Using physics as model prior for deep learning,” in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=BklHpjCqKm

[5] S. Schaal, C. G. Atkeson, and S. Vijayakumar, “Scalable techniques from nonparametric statistics for real time robot learning,” *Applied Intelligence*, vol. 17, no. 1, pp. 49–60, Jul 2002. [Online]. Available: https://doi.org/10.1023/A:1015727715131

[6] S. Calinon, F. D’halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, “Learning and reproduction of gestures by imitation,” *IEEE Robotics Automation Magazine*, vol. 17, no. 2, pp. 44–54, June 2010.

[7] Y. Choi, S.-Y. Cheong, and N. Schweighofer, “Local online support vector regression for learning control,” in *2007 International Symposium on Computational Intelligence in Robotics and Automation*. IEEE, 2007, p. 13–18.

[8] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, “Graph networks as learnable physics engines for inference and control,” 2018.

[9] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, “Learning inverse dynamics models in o(n) time with lstm networks,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov 2017, pp. 811–816.

[10] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[11] F. Díaz Ledezma and S. Haddadin, “First-order-principles-based constructive network topologies: An application to robot inverse dynamics,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov 2017, pp. 438–445.

[12] J.-A. Ting, M. Mistry, J. Peters, S. Schaal, and J. Nakanishi, “A bayesian approach to nonlinear parameter identification for rigid body dynamics,” 08 2006.

[13] M. Lutter, C. Ritter, and J. Peters, “Deep lagrangian networks: Using physics as model prior for deep learning,” in *7th International Conference on Learning Representations (ICLR)*, 2019.

[14] S. Greydanus, M. Dzamba, and J. Yosinski, “Hamiltonian neural networks,” 2019.

[15] S. C. Douglas and J. Yu, “Why relu units sometimes die: Analysis of single-unit error backpropagation in neural networks,” in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, Oct 2018, pp. 864–868.

[16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in PyTorch,” in *NeurIPS Autodiff Workshop*, 2017.

[17] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 01 2017.