A Lagrangian mechanics oriented approach to neural networks

A S Roberts
Department of Physics, University of Bath, Claverton Down, Bath BA2 7AY, UK

Abstract. Neural networks are transforming the landscape of physics research, enabling physicists to discover previously unseen patterns within the universe. Most neural networks, however, rely upon relatively simple loss functions to guide their training processes due to computational constraints. In this paper, we investigate a Lagrangian mechanics oriented approach to the creation of a new integrator-based loss function. Here, we aim to marry the similar behaviours exhibited by both Lagrangian mechanics and neural networks, in their minimisation attempts to find the simplest solution to a given problem. We develop a new version of the slimplectic integrator using the Google JAX framework, which is capable of integrating complex non-conservative systems in almost constant time. We also develop the foundation for a neural network capable of identifying Lagrangians from given observational motion data (q, π), uniquely using the low time-cost integrator within neural networks loss function. Amazingly, integration for $N < 10^6$ timesteps with an error given by Δt^{40} is performed in almost constant time $t \sim 3s$ whilst the neural network, when trained on a set of damped harmonic oscillators, could predict Lagrangians that produce q values accurate to ± 0.010 and π values accurate to ± 0.10 for a system with mean values of $q = 0.115, \pi = 0.65$.

1. Introduction

1.1. Neural Networks in Physics

Machine learning has become the latest addition to the modern physicist's toolkit with a broad range of applications, from deep learning for imaging in medical physics [1] to large-scale data analysis in astrophysics [2]. Machine learning and neural networks often allow physicists to tackle previously intractable problems due to the computation size and scale required. To understand how this tool can be used, one must first understand how the tool works.

Neural networks are one of the most common forms of machine learning and work by finding a generalised non-linear function (the neural network) between the input and output spaces. The input space contains the data in which patterns aim to be discovered and the output space is the set of possible outputs. For example, in astrophysics the input space may be a range of luminosity values attributed to specific wavelengths of light. The output space would be the set of possible classifications of stars combined with a set of confidence levels.

One of the main draws of neural networks is the ability to apply the function to any object in the input space, even if previously unseen. The neural network is represented as a connected graph where each node and connection has a scaling (called weights for the nodes and biases for the connections). The neural network is then trained to obtain a function that can accurately predict an outcome for any element of the input space. This training involves taking a known subset of the input and output spaces, called the training data set, along with a loss function f_{loss}. The loss function compares the neural networks calculated outputs with the known (true) outputs in the training data. Starting with randomised weights and biases to initialise neural network, the following process can be repeated to slowly optimise the network:

(i) Pass an element of the training data through the neural network
(ii) Calculate losses for the output
(iii) Take the gradient of the loss function with respect to the weights and biases in the neural network
(iv) Update the weights and biases according to $-\nabla f_{\text{loss}}$ to minimise the loss of the function.

Repetition of this process finds minima of the loss function, thereby optimising the neural network. Ideally, a global minima would be found but this is not guaranteed by the method outlined above due to the process becoming trapped in a local minima. Optimisation algorithms such as Adaptive Moment

Estimation (ADAM) optimiser [3] can be implemented to increase these chances by varying the size of the updates made when taking the gradient of the loss function.

With the ability to identify possibly unseen patterns in data, neural networks are an ideal tool for physicists. One area of neural network development related to this is that of physics-informed neural networks (PINNs). These neural networks have been given prior knowledge of basic physical laws, which can be encoded into the model in a multitude of ways. Physics-informed layers of the neural network are one such encoding with the example that for a fluid dynamics focused PINN, this could be a layer that enforces specific conservation laws. Another encoding is via loss function formulation, which incorporates terms derived from the governing physics equations.

1.2. Loss Functions

Loss functions play a crucial role in PINNs, often acting as a conduit for incorporating known physical laws. PINN loss functions usually have two purposes:

- Data Fitting: This ensures that the network accurately predicts values, much like in more general neural networks.
- Physics Information: An encoding of physical laws, such as ensuring specific PDEs are satisfied by penalising deviation from governing equations.

One issue that loss functions face is the requirement to be differentiable with respect to the weights and biases of a network. This limitation often leads to simple regression functions, such as mean squared error, being chosen over more complicated functions to guarantee this property. A regression function may be paired with a simple physics-informed term, such as a penalisation for deviation from conservation terms. For example, the conservation of mass within fluid dynamics for an incompressible fluid can be expressed as

$$\nabla \cdot \dot{q} = 0, \tag{1}$$

so a penalisation term in the loss function could be written as $= c|\nabla \cdot \dot{q}|^2$ with $c > 0$ chosen as an appropriate constant, thus penalising deviation from equation 1 when the NN minimises the loss.

Another more applied issue that loss functions face is the restraint of computational efficiency. As one of the most called functions within the neural network training process a loss function is constrained to mathematically quick calculations and therefore often cannot afford to implement longer more involved calculations.

1.3. Google JAX

Google JAX [4] is a Python machine learning framework developed by Google that collates the properties of `Autograd` [5] and XLA (Accelerated Linear Algebra) [6], which include just in time (JIT) compilation). `Autograd` allows for automatic differentiation of both Python and `NumPy` functions, a beneficial property when performing gradient descent during the training of neural networks. `Autograd` performs this seemingly analytical task by tracking each simple mathematical operation performed upon the input of a function. Each of these operations have precomputed derivative forms, which can be applied via the chain rule to produce the derivative of the larger, more complex initial function.

JIT compilation allows functionally written pieces of Python code to convert from interpreted to compiled code. By JIT compiling, code is converted into mathematically more efficient "black-box" functions which map inputs to outputs in a shorter time than traditional Python functions. JIT compiling is especially beneficial for code that is frequently called as it need only be compiled once before the "black-box" version is used thereafter.

1.4. Lagrangian Mechanics

Here we provide a brief overview of Lagrangian Mechanics, a formalism of classical mechanics prioritising the energy of a system as its initial reference point. Lagrange's formalism considers a set of independent generalised coordinates $\{q_1, q_2, \ldots, q_n, \dot{q}_1, \dot{q}_2, \ldots, \dot{q_n}, t\}$ (hereon denoted as $(\mathbf{q}, \dot{\mathbf{q}}, t)$), which are used to express the kinetic, $\mathcal{T}$, and potential, $\mathcal{V}$, energies of the system. The degrees of freedom of the system, n, is determined by the number of parameters with which one can completely specify its configuration. The Lagrangian

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}, t) - \mathcal{V}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{2}$$

is defined as the difference between the two energies of the system. The Lagrangian provides the final object of interest the action,

$$S = \int_{t_1}^{t_2} \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) dt \tag{3}$$

a functional of which stationary points are of interest. Hamilton's principle [7] states that these stationary points are where the action coincides with physical reality given by $\delta S = 0$.

From here, the equations of motion (EoMs) can be obtained by applying the Euler-Lagrange equation

$$\frac{\partial \mathcal{L}}{\partial q_i} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) = 0, \tag{4}$$

which is equivalent to minimising the action (and thus finding its stationary points). We obtain n differential equations for the n generalised coordinates. Equation 4 then provides a generalised form for the EoMs of a conservative system.

Much like generalised coordinates and velocity, generalised momenta is introduced here as

$$p_i = \frac{\partial \mathcal{L}}{\partial \dot{q}} \tag{5}$$

and is the fundamental basis for all forms of momentum, both linear and angular.

1.4.1. Example: Simple Harmonic Oscillator

The 1-dimensional simple harmonic oscillator can be used as a motivating example. With position x, mass m and spring constant k, the kinetic energy of this system is given by

$$\mathcal{T} = \frac{1}{2} m \dot{x}^2 \tag{6}$$

and the potential energy by

$$\mathcal{V} = \frac{1}{2} k x^2. \tag{7}$$

This gives the Lagrangian

$$\mathcal{L} = \frac{1}{2} m \ddot{x}^2 - \frac{1}{2} k x^2, \tag{8}$$

which when inputted into equation 4 again provides our known equation of motion

$$\ddot{x} = \frac{k}{m} x \tag{9}$$

as could be derived from Newtonian mechanics. Conservation laws can also be identified using Lagrangian mechanics via Noether's Theorem [7]. It states that every differentiable symmetry of the action for a conservative system has a corresponding conservation law. The conserved values are known as Noether charges and are often known as Noether currents when considered as functions of time.

1.4.2. Nonconservative Lagrangians

Galley, Tsang, and Stein [8] propose a generalisation of Noether's Theorem to allow for non-conservative systems. This works by generalising to the nonconservative Lagrangian

$$\Lambda = \mathcal{L}(\mathbf{q_1}, \dot{\mathbf{q}}_\mathbf{1}, t) - \mathcal{L}(\mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{2}, t) + K(\mathbf{q_1}, \dot{\mathbf{q}}_\mathbf{1}, \mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{2}, t) \tag{10}$$

where $\mathcal{L}$ is the Lagrangian as defined previously, and K is an additional coupling term physically similar to a non-conservative potential. The generalised coordinates are doubled, as explained by Galley [9], from $\mathbf{q}, \dot{\mathbf{q}}$ to $\mathbf{q_1}, \dot{\mathbf{q}}_\mathbf{1}, \mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{2}$. The cartoon in figure 1 depicts this doubling of coordinates to create the free endpoint. This doubling allows for a subtle but important change, the Euler-Lagrange equation provides an initial value problem (differential equation + initial conditions). In contrast, Hamilton's principle provides a boundary value problem in time (i.e. "minimise the action passing through given initial and final time values"). This is a slight difference and, in conservative systems, does not pose an issue. However, in non-conservative systems this breaks down due to non-conservative behaviours such as history dependence and non-linearity. The doubling of coordinates remedies this problem by converting the problem to one of an initial value problem with twice the degrees of freedom and two initial values $q_{i,1}$ and $q_{i,2}$. This process allows the creation of a free end point q_f which is no longer fixed. These new generalised coordinates must satisfy $q_1(t) = q_2(t)$ and $\dot{q}_1(t) = \dot{q}_2(t)$ for $t \in \{t_i, t_f\}$.

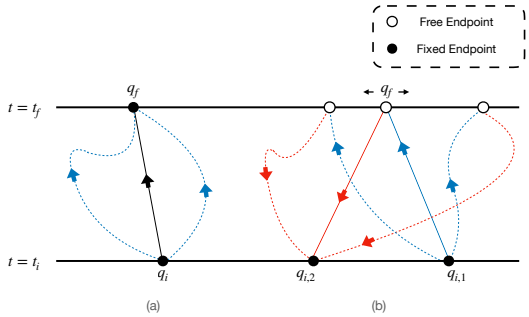

Figure 1. A cartoon depicting the doubling of coordinates posited by Galley [9]. (a) shows a conservative system where initial and end points are fixed such that the solid path between them minimises the Action and dashed paths represent paths that do not minimise the Action. (b) shows a system with doubled coordinates. the point on the $t = t_f$ line is now free to vary but both the $q_{1,i}$ and $q_{2,i}$ points are fixed on the $t = t_i$ line. Here the q_f point varies in order to globally minimise both paths. The physical limit occurs as $q_{2,i} \to q_{1,i}$ and, assuming the same system is modelled in both (a) and (b), rediscovers the path shown in (a).

Galley [9] also shows that K must be antisymmetric under swapping the indices and, therefore, must also vanish for $\mathbf{q}_1 = \mathbf{q}_2, \dot{\mathbf{q}}_\mathbf{1} = \dot{\mathbf{q}}_\mathbf{2}$.

For convenience, a change of coordinates is often performed due to physical motivation. These are $q_- = (q_1 - q_2)/2$ and $q_+ = (q_1 + q_2)/2$, where the former can be considered a hypothetical displacement that vanishes in the physical limit $q_- \to 0$ and the latter the physically relevant component of the coordinates. Note here that equation 10 continues to satisfy the Euler Lagrange equation, equation 4 when evaluated

in the physical limit (PL). This is formulated as

$$\frac{d}{dt}\left[\frac{\partial \Lambda}{\partial \dot{q_-}}\right]_{PL} - \left[\frac{\partial \Lambda}{\partial q_-}\right]_{PL} = 0. \tag{11}$$

1.4.3. Hamiltonian Mechanics

Another formulation of classical mechanics, similar to Lagrangian mechanics, is that of Hamiltonian mechanics. As the name suggests, the object of interest here is the classical Hamiltonian

$$\mathcal{H} = \mathcal{T} + \mathcal{V}. \tag{12}$$

Hamiltonian mechanics exchanges the use of generalised velocities ($\dot{q}$) in favour of generalised momenta p. This is sometimes, but not always, equivalent to momenta in classical mechanics in the same way that $\dot{\theta}$ could be considered a velocity (time derivative of an angle coordinate θ) but would not likely be ones initial idea for a velocity. This exchange does not affect the class of physical phenomena that Hamiltonian mechanics can describe compared to Lagrangian mechanics; that is to say these classes of phenomena are equivalent.

The Hamiltonian (equation 12) produces first-order ordinary differential equations (ODEs) as the equations of motion

$$\frac{dq_i}{dt} = \frac{\partial \mathcal{H}}{\partial p_i}, \frac{dp_i}{dt} = -\frac{\partial \mathcal{H}}{\partial q_i}. \tag{13}$$

This formulation is often useful for symplectic integrators [10] as discussed in section 2.

2. Slimplectic Integrator

Integrators are another tool within the modern physicist's toolbox. Many physics problems have non-analytic solutions, and this is where integrators make their appearance. Many integrators exist and are specialised for different applications. The Runge-Kutta (RKn) method [11] is a well-known and widely applied group of methods used for a balanced trade-off between accuracy and efficiency. RK45 is perhaps the most widely used of these as it is an efficient method of finding solutions to problems with stronger than 4th-order accuracy. The order, o, of the integrator provides an approximation for the error in the integrator's prediction as

$$|\mathrm{Err}(\Delta t)| \approx a(\Delta t)^o, \tag{14}$$

where the error is a function of the size of the timestep Δt. One issue that Runge-Kutta faces, however, is that of compounding error. Whilst any one term has a bounded error when using Runge-Kutta for integration, the total accumulation of error is not bounded, causing issues for systems that have evolved over longer time periods.

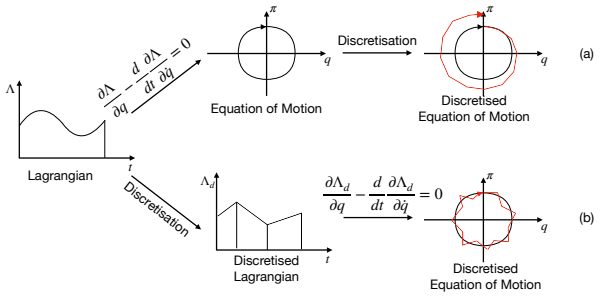

Figure 2. The difference between discretising (a) a standard integrator and (b) a variational integrator. In discretising a standard integrator the discretisation acts directly on the EoM's. However, a variational integrator discretises the action before determining the discretised EoM's.

A solution to the compounding error issue is instead the use of Symplectic integrators, a class of integrators that preserve the differential 2-form known as the symplectic form. This preservation, akin to the preservation of certain physical constants, makes Symplectic integrators particularly useful in integrating systems over long time scales. This is particularly common in orbital mechanics [12, 13] but also finds uses in statistical algorithms [14].

One way to construct symplectic integrators is through variational integrators, which are determined by the variation of a discretised action. This approach differs from traditional numerical integrators, such as Runge-Kutta, which discretise the equations of motion themselves. This unique construction method is a distinct advantage of symplectic integrators and is shown in figure 2 where the radial distance of the discretised EoM, or alternatively the maximum deviation from the true EoM, is conserved for the variational integrator. This is a property not conserved by standard integrators. Due to these conservation behaviours, symplectic integrators are often used to solve Hamiltonian systems numerically as the error in energy is one such bounded metric. One common group of approaches used for symplectic integrators is splitting methods such as the leapfrog and Verlet [15] methods. These work by splitting the Hamiltonian into smaller, more manageable parts and iteratively solving the system. By splitting the Hamiltonian like this the integrator can approximate solutions whilst preserving important geometric properties (such as the symplectic form, volume or energy) at a global level.

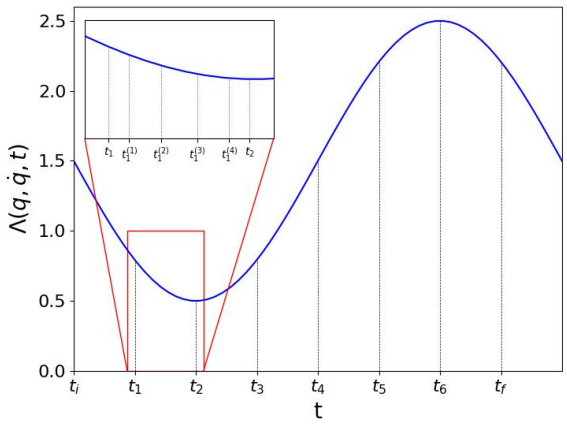

Figure 3. The discretisation of time used within the GGL method. This is used to approximate the action to N timesteps between t_i and t_f. Between t_n and t_{n+1} the quadrature discretises r times as according to equation 15.

2.1. The "Slimplectic" Integrator

Tsang et al. [16] have developed a variational integrator, the "Slimplectic" integrator, from the nonconservative action principle [8]. Starting with equation 11 the action integral $S = \int_{t_i}^{t_f} \Lambda(q_\pm, \dot{q_\pm}, t)dt$ is discretised using Galerkin-Gauss-Lobatto (GGL) quadrature [17] as shown in figure 3. The GGL quadrature is chosen as it is an even-order method and, therefore, symmetric under time reversal ($t \to -t$). The interval $[t_n, t_{n+1}]$ has $r+2$ quadrature points given by

$$t_n^{(i)} \equiv t_n + (1 + x_i)\frac{\Delta t}{2} \tag{15}$$

where $\Delta t = (t_{n+1} - t_n)/2$, $x_0 \equiv -1, x_{r+1} \equiv 1$ and x_i is the ith root of dP_{r+1}/dx, the derivative of the $(r+1)$th Legendre polynomial $P_{r+1}(x)$. To obtain approximations of values for $\dot{q}_{n,\pm}^{(i)}(t)$ at the quadrature points the derivative matrix [18]

$$D_{ij} = \begin{cases} -(r+1)(r+2)/(2\Delta t) & i = j = 0 \\ (r+1)(r+2)/(2\Delta t) & i = j = r+1 \\ 0 & i = j \notin \{0, r+1\} \\ \frac{2P_{r+1}(x_i)}{P_{r+1}(x_j)(x_i - x_j)\Delta t} & i \neq j \end{cases} \tag{16}$$

is used to give

$$\dot{q}_{n,\pm}(t_n^{(i)}) \simeq \sum_{j=0}^{r+1} D_{ij} q_{n,\pm}^{(j)} \tag{17}$$

where Tsang et al. [16] provides more explanation as to the reasoning for this approximation.

GGL quadrature allows any functional $\int F dt$ to be approximated over an interval of time by a discrete functional F_d given by

$$\begin{aligned} F_d^n &\equiv F_d\left(q_{n,\pm}, \left\{q_{n,\pm}^{(i)}\right\}_{i=1}^{r}, q_{n+1,\pm}, t_n\right) \\ &\equiv \sum_{i=0}^{r+1} w_i F\left(q_{n,\pm}^{(i)}, \dot{\phi}_{n,\pm}^{(i)}, t_n^{(i)}\right), \end{aligned} \tag{18}$$

where w_i is a weighting given by

$$w_i \equiv \frac{\Delta t}{(r+1)(r+2)[P_{r+1}(x_i)]^2}. \tag{19}$$

With a discretisation of $\dot{q}_\pm$ the action can be discretised by applying the methodology of equation 18 to equation 10 defined as

$$\mathcal{S}_d\left[t_0, t_{N+1}\right] \equiv \sum_{n=0}^{N} \Lambda_d\left(q_{n,\pm}, \left\{q_{n,\pm}^{(i)}\right\}_{i=1}^{r}, q_{n+1,\pm}, t_n\right). \tag{20}$$

This can then be applied to the Euler Lagrange equation to obtain discretised equations of motion

$$\left[\frac{\partial \Lambda_d^{n-1}}{\partial q_{n,-}} + \frac{\partial \Lambda_d^n}{\partial q_{n,-}}\right]_{PL} = 0 \tag{21a}$$

$$\left[\frac{\partial \Lambda_d^n}{\partial q_{n,-}^{(i)}}\right]_{PL} = 0. \tag{21b}$$

Taking the definition of generalised momenta in equation 5, discretised momenta can now be introduced by splitting equation 21a into

$$\pi_n \equiv -\left[\frac{\partial \Lambda_d^n}{\partial q_{n,-}}\right]_{PL} \tag{22a}$$

$$\pi_{n+1} \equiv \left[\frac{\partial \Lambda_d^n}{\partial q_{n+1,-}}\right]_{PL} \tag{22b}$$

$$0 \equiv \left[\frac{\partial \Lambda_d^n}{\partial q_{n,-}^{(i)}}\right]_{PL} \tag{22c}$$

which finally produces a set of discretised equations that can be computationally solved for. By taking initial conditions for q_0, π_0 the integrator can then forward integrate the system for N iterations whilst preserving a constant fractional error bound.

Tsang et al. [16] have developed a Python code `slimplectic` which makes use of the `SymPy` package to include a computational solver using the slimplectic method as described in equation 22. The package takes initial conditions; a value for r; a number of timesteps, N, for which to iterate; the separation of the timesteps, Δt; and a Lagrangian, Λ, to determine values for q_n, π_n with $n \in \{1, N\}$.

3. Code and Method

Here we outline the two pieces of code developed and testing framework to identify strengths and weaknesses with each.

3.1. Slimplectic Integrator 2.0

Using the Google JAX package, we have developed an "autodiffable" version of the Slimplectic integrator (hereon referred to as the JAX-based integrator). The integrator follows the generation method outlined in section 2, which matches that of Tsang et al. [16], but uses the JAX implementations of `Autograd` and `NumPy` in place of the `SymPy` package used in the original version. Similar to the original Slimplectic integrator, the JAX-based integrator takes values $(r, N, \Delta t)$ along with initial conditions for q and π to forward integrate. In place of the `SymPy` expressions for $\mathcal{L}$ and K, a Python function with arguments (q, π) with the form of the Lagrangian is inputted.

Two examples (sections 3.1.1, 3.1.2) are used to compare the `SymPy`-based and JAX-based Slimplectic Integrators over two metrics. The first of these is the time complexity of the integrator measured by determining q and π for a large number of timesteps. The second is the order of the method, determined by varying r and observing the time taken to generate values for a fixed number of timesteps.

3.1.1. Damped Harmonic Oscillator

Using the format of equation 11, a one-dimensional damped harmonic oscillator (DHO) has a Lagrangian

$$\Lambda(\mathbf{q}_\pm, \dot{\mathbf{q}}_\pm, t) = \frac{1}{2}m\dot{q}^2 - \frac{1}{2}kq^2 - c\dot{q}_+ q_- \tag{23}$$

where m is the mass of the oscillating object, k is the spring constant of the oscillator, and c is a damping constant.

We choose a DHO as it allows us to identify non-conservative behaviour being correctly predicted at high levels of accuracy due to its simplicity.

3.1.2. Poynting-Robertson Drag

As with Tsang et al. [16], we also examine the orbital motion of a small dust particle being acted on by radiation from a solar-type star. This particle would experience Poynting-Robertson drag [19] which would give the system a conservative Lagrangian of

$$\mathcal{L} = \frac{1}{2}m\dot{\mathbf{q}}^2 + (1-\beta)\frac{GM_\odot m}{|\mathbf{q}|} \tag{24}$$

with $\mathbf{q}$ the particles position and m the particles mass. The β term represents a dimensionless quantity which is the ratio between forces due to radiation pressure and gravity given by

$$\beta \equiv \frac{3L_\odot}{8\pi c\rho G M_\odot d}. \tag{25}$$

Here, ρ and d are the density and diameter of the dust particle, and $L_\odot$ and $M_\odot$ are the solar luminosity and mass.

The Poynting-Robertson drag manifests itself as the non-conservative term

$$K = \frac{\beta G M_\odot m}{c\mathbf{q}_+^2}\left[\dot{\mathbf{q}}_+ \cdot \mathbf{q}_- + \frac{1}{\mathbf{q}_+^2}(\dot{\mathbf{q}}_+ \cdot \mathbf{q}_+)(\mathbf{q}_+ \cdot \mathbf{q}_-)\right]. \tag{26}$$

This example is chosen due to its use in Tsang et al. [16] to compare to the Runge-Kutta 4th order method.

3.2. Neural Network and Loss Function

The second piece of code is a neural network written using the Keras [20] and Tensorflow [21] frameworks. The network uses a simple, multi-layer LSTM model [22] combined with a novel loss function which implements the JAX-based slimplectic integrator. LSTM layers were chosen for their ability to retain "historical" information when parsing time-series data, a feature that should prove useful for the NN to learn how previous states of the system affect the current state. The NN is built to identify coefficients of the terms within the Lagrangian, which is part of a specific 'family'. A 'family' of Lagrangians is defined to be the set of all Lagrangians differing only by coefficient values. For example, the DHO Lagrangian in equation 23 would be part of the family of Lagrangians with the form

$$\Lambda = c_1\dot{q}^2 - c_2 q^2 - c_3\dot{q}_+ q_- \tag{27}$$

where the constant term has been omitted as Lagrangians differing by a constant term will produce the same equations of motion. As an input, the NN takes observational data (q, π) at sequential time-steps. The relationship between the input and output here highlights the motivation for creating such a NN. We aim to create a program that can correctly identify conserved quantities (symmetries) alongside other physical phenomena (e.g. energy decay in damped systems) without knowing any information about the underlying physics governing the system.

The loss function comprises terms akin to the L^2 norms in the position, q, and velocity, $\dot{q}$, spaces and has the form

$$\begin{aligned}\text{loss} = \alpha &\left(\sum_{i=0}^{N}(q_{\text{true}}(t_i) - q_{\text{pred}}(t_i))^2\right)^{1/2} \\ +\beta &\left(\sum_{i=0}^{N}(\dot{q}_{\text{true}}(t_i) - \dot{q}_{\text{pred}}(t_i))^2\right)^{1/2}\end{aligned} \tag{28}$$

where α and β are coefficients to be chosen and $q_{\text{pred}}, \dot{q}_{\text{pred}}$ are q and $\dot{q}$ values generated by taking the NN's current predictions for the Lagrangian coefficients and passing them through the JAX-based slimplectic integrator. $q_{\text{true}}, \dot{q}_{\text{true}}$ are the NN input values (i.e. the known observational data). This form was chosen to

allow experimentation of bias towards either the q or π coordinates through the α and β coefficients during training whilst aiming to minimise total deviance from the "true" values.

We note here the unconventional choice to perform loss calculations based upon values in the input space of the NN. One could choose to perform some variation of fitting, such as χ^2 (chi-squared) fitting, within the NN's output space (i.e. attempting to fit directly to the Lagrangian coefficient values). Whilst in simple situations this may provide beneficial we raise two points of consideration. Firstly, this places preferential bias upon specific Lagrangian forms, thereby not implementing the behaviour of Lagrangian mechanics where multiple Lagrangians may provide the same EoM's. Consider the example where two different Lagrangians L_a, L_b both produce the same equation of motion EoM_1. If the loss function were calculated using the L_a coefficients, the loss function would preferentially identify coefficients that are similar to those of L_a. Our loss function does not, however, see this issue due to no loss calculations being performed on the Lagrangian coefficients themselves.

Secondly, this NN is built as a "proof of concept" for implementing this type of physics-informed loss function and to verify the possibility of such a NN working in the first place. A loss function within the output space was trialled during development but frequently ran into issues of overfitting where the NN would simply learn specific behaviours of the dataset it was trained upon.

We use the DHO example in section 3.1 once again to study the behaviour of the NN. Here, we generate two subsets of the DHO family of Lagrangians, the training and validation datasets. The NN is trained upon the training dataset before verifying the NN against the unseen validation dataset. Validation involves passing the validation data into the trained NN to obtain predicted Lagrangians. These predicted Lagrangians can then be compared with the true Lagrangians within the validation dataset via the loss function and statistical analysis. Hereby, identifying whether or not the neural network has accurately predicted Lagrangians that govern the systems and thus identified the equations of motion.

4. Results

4.1. Integrating for N timesteps

We first focus on the JAX-based slimplectic integrator and look at its behaviour for large numbers of iterations. Figure 4 compares the time taken to forward integrate a DHO system for N iterations with $r = 2, \Delta t = 0.1$ for both the original and JAX-based integrators. The JAX-based integrator is further split by compilation and integration time. The JAX-based integrator performs the total integration in approximately constant time for $N < 10^6$ due to the fixed time cost of JIT compiling in this domain.

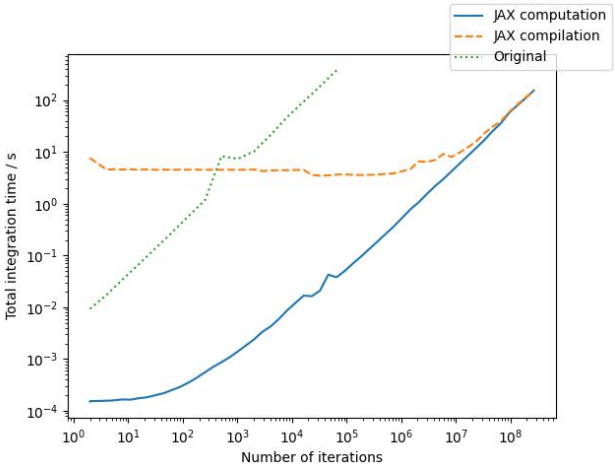

Figure 4. Time taken to integrate a simple DHO system for N timestep. The system has the parameters $r = 4, \Delta t = 0.01$s and has a Lagrangian with the same form as equation 27 with $c_1 = c_2 = c_3 = 1$. Note that the number of iterations, N, is reported against a logarithmic scale and therefore the exponential growth seen for $N > 10^6$ is a linear relationship between N and total integration time for N in this region.

The total integration time for the integrator can, therefore, be seen as $t_{\text{tot}} = t_{\text{comp}} + t_{\text{int}}$. For $N < 10^6$ we have $t_{\text{comp}} \gg t_{\text{int}}$, which exhibits itself as approximately constant time. For $N > 10^6$ we have $t_{\text{compile}} \propto t_{\text{int}}$ which produces the observed linear growth in t_{tot}. Note that the exact value at which t_{tot} transitions from the constant to the linear domain is dependent on the specific family of Lagrangians. A Lagrangian defined by four coefficients will differ from one defined by twelve coefficients due to the increased complexity in the mathematical calculations performed.

We observe that the original slimplectic integrator performs integration for all values of N in a manner such that $t_{\text{tot}} \propto N$. For $N < 10^2$ we observe that the original integrator performs the integrations quicker than the JAX-based version due to the lack of compile time. We observe that for all values of N, $t_{\text{int,original}} \propto 10^4 \times t_{\text{int,JAX}}$, which for large values of N represents a hundred-fold increase in the computation speed of the JAX-based version when compared to the original.

4.2. Changing the order of the Integrator

The order of the Slimplectic integrator [16] is $r+2$ and allows us to approximate the error in the integrators predictions as in equation 14. We can, therefore, increase the accuracy of the integrator by increasing r. Figure 5 shows the effect of varying this for a

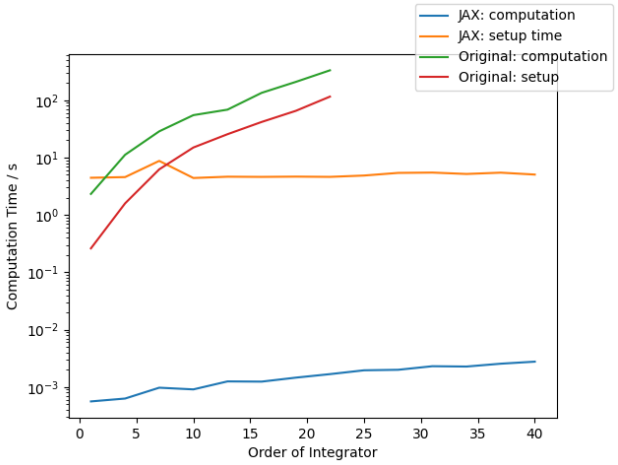

Figure 5. A comparison of the order of the integrator and the time taken to integrate a PRD system. Time is split by basis of the model (original vs JAX) and by setup/computation calculations. Here we observe the fixed compiling cost once again dominating the total time for the JAX-based integrator effectively providing a fixed computation time regardless of integrator order. Times for order > 25 were not recorded for the original integrator due to extreme computation time.

Poynting-Robinson Drag system. The compile time dominates the JAX-based integrator's total integration time, leading to an effective fixed-cost computation for $r \sim 10^1$. The JAX-based integrator cannot currently discretise $r > 40$ due to numerical instability in the Legendre polynomial root finding used in equation 15.

4.3. Bounding the Integrator Error

As with the original slimplectic integrator we investigate the fractional energy error relative to the analytic solutions energy. Here, we observe almost identical behaviour to the original integrator. Mirroring figure 2 from Tsang et al. [16], we compare the fractional energy error with the system evolution time for the JAX-based integrator and Runge-Kutta 2 and 4 methods. This is shown in figure 6 and almost directly mirrors the original slimplectic integrators version of this figure.

4.4. Approximating DHO Functions

We now move to the NN, which was trained on a dataset of 5.0×10^5 randomly generated non-conservative Lagrangians within the DHO family given by equation 27. For each generated set of Lagrangian coefficients, $\{c_1, c_2, c_3\}$, the conditions $c_1 > 0$ and $c_2, c_3 \geq 0$ were applied. Approximately 5% of c_2 and c_3 values were randomly set to 0 such that the training dataset was guaranteed to have systems with $\mathcal{V}, K = 0$. Predictions of negative coefficient values were discouraged by implementing an additional term in the loss function penalising values < -0.1. This

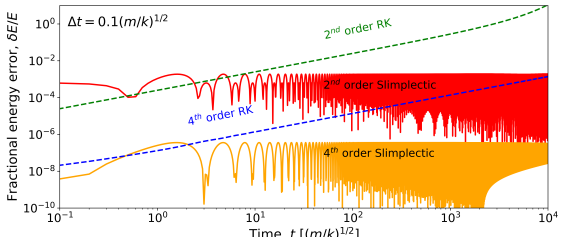

Figure 6. The energy error as a fraction of the analytic solution for the energy at specified times. As with the original slimplectic integrator we observe initially roughly equivalent energy errors between RKn and a slimplectic integrator of order n. However for long time periods this diverges with the error in RK increasing whilst the error in the slimplectic integrator remains bounded.

limit was chosen so as not to discourage the neural network from choosing zero-valued coefficients whilst still discouraging non-physical systems. We note here that this should not bias the loss function to select one lagrangian over another as any system discouraged would be unphysical due to the negative coefficient values.

Training of the model was temperamental and often susceptible to random permutations of the dataset causing diverging loss values which eventually broke models. By increasing batch sizes and introducing loss caps, this was eventually remedied to produce a model that identifies trends within the data. Although not able to perfectly reproduce equations of motion as yet, figure 7 shows promising progress for the NN. For mean values (RMS) of $q = 0.115, \pi = 0.65$ the model produced loss error value of ± 0.050 in q and ± 0.10 in π. The loss function was also varied during the training process, initially weighting more in favour of the RMS error in q to compensate for the fact that a majority of systems produced larger π-values than q-values.

Biasing the loss function only to use RMS error in π produced results of interest, initially reducing loss to 10^{-4}, a before unseen value. However, this was accomplished by the model choosing $c_1 \approx c_2 \approx c_3$ and then varying these values by small quantities to appease the loss function, thereby not producing physically accurate predictions.

Another result of interest was the model's inability to train to an accuracy greater than that used to produce figure 7. The training process frequently approached a validation loss ≈ 0.20 but often struggled to become more accurate than this, with the loss soon after "blowing up", possibly due to the complexity of the loss function below this resolution. In figure 7, we can see that the phase of q has been matched, but the predicted amplitude is smaller than the true amplitude. Remedying this would require

scaling multiple coefficients whilst conserving a shared property. If one of these parameters were to change but not the other, the model would see a "de-syncing" of the phase thereby increasing loss. Therefore the complexity in the loss function here may be due to carefully balancing the scaling of multiple coefficients to conserve the phase. This issue was identified after exploring gradient descent in the embedding space to study the loss function. Figure 8 identifies this behaviour by observing the behaviour of the loss function about the global minima, i.e. the true embedding of the function.

We observe a generally decreasing behaviour of the loss function towards the global minima although the spike for $\Delta c_1 = -2$ produced by a divide by zero problem in the equation of motion may cause issues. The equation of motion for a harmonic oscillator confirms this with c_1 proportional to the mass of the harmonic oscillator in equation 9.

A similar term will exist within the DHO equation of motion but is not shown due to its non-analyticity.

This may be the cause of the neural networks lack of ability to find the global minima whilst training causing rapid growth in the loss if come into contact with. The loss function in the far-regime plots for c_2 and c_3 clearly show the non-negativity component creating an antisymmetric behaviour in the plots.

5. Discussion

5.1. Slimplectic Integrator

As reported in section 3.1 the JAX-based integrator runs in what is experienced by the user as constant time for the number of timesteps $N < 10^6$. The computation time was fit to the function $\log(t) = a\log(N)$, which for $N > 10^4$ returned $a = 1.0140 \pm 0.0042$, suggesting the growth in the integration time is linear. For all values of r tested, similar behaviour was observed, but the growth is curtailed before it notably contributes to computation time due to float precision affecting the ability to determine roots of the k^{th} Legendre polynomial precisely for $k > 40$. This may be remedied with the use of higher precision values within the program; however, XLA (and therefore JAX) does not yet (as of time of writing) support values more precise than float64 (doubles).

An area of interest is the small spike observed for $7 \leq r \leq 9$ in figure 5. This was reproduced with multiple sets of data but is of uncertain origin. One explanation is that this occurs due to some computational attempt at an optimisation occurring for data larger than 8 bits ($r = 7 \implies \text{order} = 9 > 8$). Although this does not produce much of a detrimental effect on the overall calculation time. A higher order of integration could also be used to generate more accurate results in a quicker time circumventing this effect. Potential improvement to the integrator could be made via the implementation of varying timesteps as this would allow for preserving both symplectic-momentum conservation and momentum-energy conservation [23]. Adaptive timestep methods do, however, allow this [24].

5.2. Neural Network

The JAX-based integrator's fixed JIT compile cost allows for the application of the integrator to multiple areas of physics. The integrator can be compiled once and then used over a spectra of initial conditions or physical parameters (Lagrangian coefficients) allowing for quick exploration of a problem space at a high level of accuracy in a small amount of time. This could be taken further and, by compiling a larger family of Lagrangians (in the sense that the set of harmonic oscillator Lagrangians is a subset of the set of DHO Lagrangians), parts of systems such as charge, damping, and nuclear forces could be switched on and off or varied through a spectrum of values.

This process could also be applied to field-theory-like systems which model chains of smaller subsystems, each similar to one another. Here the low computation time once again allows for the modelling of much larger systems than previously available with a similar style of integrator. The Lagrangian in this example is a sum of the Lagrangians of the smaller coupled systems which, after the first element of the sum was compiled, could be included in the calculation at a lower computational cost. Examples of this could be molecular dynamics modelling, such as those discussed by Tuckerman and Martyna [25], or statistical mechanics systems.

The neural network produced promising results but was not able to entirely predict accurate Lagrangians from the q and π values inputted. We propose some areas of future development in order to better train the network with the possibility of producing a fully functioning model. As with most neural networks, a larger, better-curated dataset may allow for training to a more optimised level. The trained model performs worse than average at attempting to classify systems comprised entirely of kinetic energy (KE) ($c_2, c_3 = 0$). This is an issue that may be solved by training the model on a dataset comprising more functions of this form, as the training dataset had approximately 1000 such samples spread across many batches (an average of 1 KE-only system per batch). This suggests that further training datasets would need to take care to include large subsets of families with zero-valued coefficients when physically relevant, in order to better highlight the effect of each term on the physical behaviour of the system.

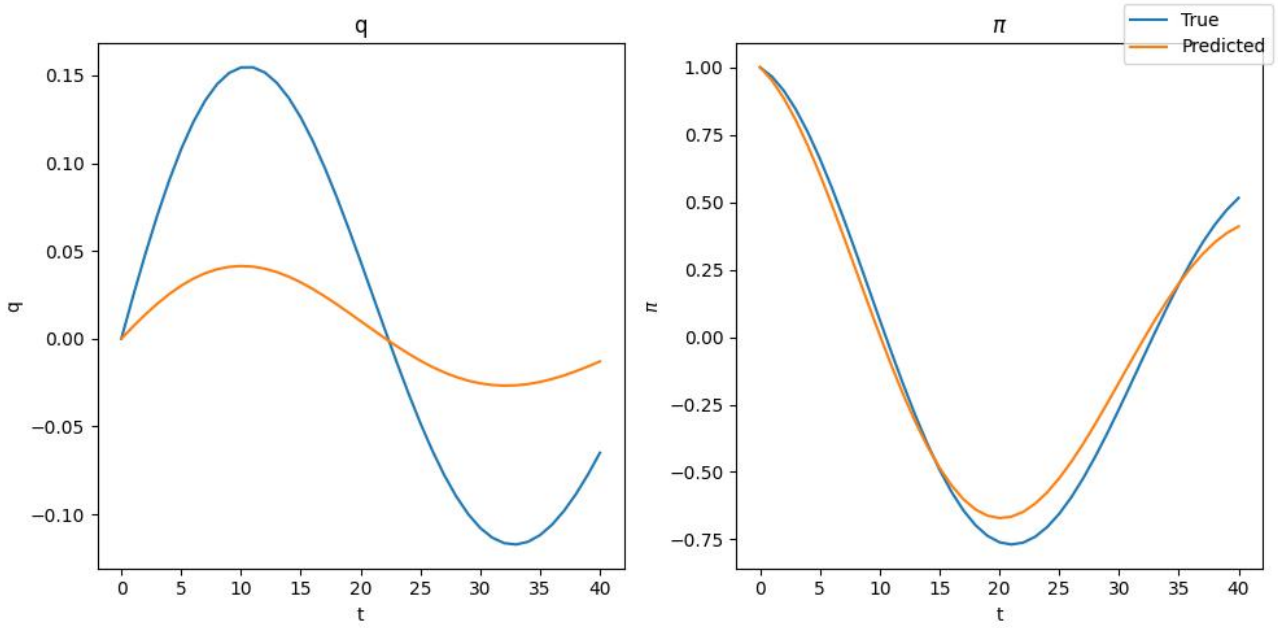

Figure 7. Particle motion as predicted by the Neural Network when trained on a dataset of DHO systems. The true coefficient embedding was $\{c_1 = 2, c_2 = 4, c_3 = 1\}$ and the predicted embedding was$\{c_1 = 6.88, c_2 = 14.19, c_3 = 5.46\}$. Both embeddings were then forward integrated 40 timesteps to produce these figures. We note the high level of correspondence in the π value whilst the q value differs by a matter of amplitude, a common issue with the neural network.

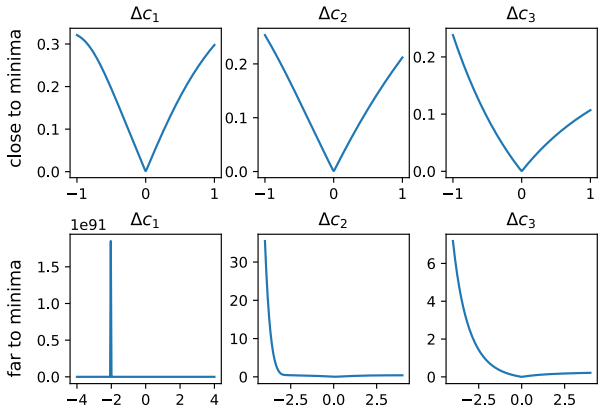

Figure 8. Observations of the Loss function about the global minima (true embedding) for the DHO embedding $c_1 = 2, c_2 = 3, c_3 = 1$. Δc_i shows the deviance from the true embedding with fixed values for c_j with $j \neq i$. Of note is the magnitude of the "spike" at $\Delta c_1 = -2$ which corresponds to a predicted value of $c_1 = 0$. This is therefore a division by zero issue at this point and can be shown through the harmonic oscillator equation of motion $\ddot{q} = (k/m)q$.

5.3. Loss Function and Model Variants

Another area of improvement could come with the development of a more complex model. Either through a larger neural network or through a different choice of loss function. A more complicated network would allow the model to encapsulate more information; this does, however, increase the chance of the model overfitting to data. Layer choices could also be varied from the LSTM layers chosen within this model to lead the model to focus on more specified behaviours. As with the analysis in section 4.4, the loss function could further have the weighting between the RMS calculations varied or perhaps a different calculation could be performed. Examples of these calculations could be mean absolute error (MAE, akin to the L_1 norm) or Huber loss (a cross between MAE and RMS in different regions). However, the RMS error/L^2 norm used perhaps still provides the best loss calculation from a physical sense. The RMS error in q is akin to the total distance deviated from the original path of motion and would scale well into multiple dimensions. The RMS error in π provides a term proportional to $\dot{q}^2$, which in turn is proportional to the KE of the system, thus providing (in some sense) a measure proportional to the difference in kinetic energies of the predicted and true systems. The RMS error would also be more susceptible to outliers in predictions which would punish the NN more strongly than MAE would. This is not to say that MAE or Huber loss may not be beneficial when training and these functions are yet to be explored.

The other term introduced to the loss function was the non-negativity requirement which was encoded as the continuous function

$$\text{loss}_{\text{neg}} = \begin{cases} 20(c_i + 0.1)^2, & c_i < -0.1 \\ 0 & c_i \geq -0.1. \end{cases} \tag{29}$$

During the development of the neural network, we discussed different implementations of the loss function, such as using exponential growth in the negative region, the implementation of the "buffer region" for $-0.1 < c_1 < 0$ and whether or not there should be penalisation for negative values at all. The resultant loss function, as in equation 29, came about as a choice to penalise strongly non-physical (negative) values. The buffer region was implemented to ensure this penalisation without deterring the NN from 'exploring" low values of coefficients. The choice of quadratic growth of the loss in this region was made to maintain an entirely C_1 (differentiable) loss function. Here, the choice of exponential growth would have led to a discontinuity in the derivative of the loss function at $c_i = 0.1$.

An issue discovered here was due to the model discovering 0 valued masses, which cause "explosions" in the behaviour of the predicted function. This issue is highlighted in figure 8 and implies that the loss function may need more prior physics assumptions. The assumptions could take the form of term-specific non-negativity penalisation when the coefficient is known to take strictly positive values compared to a coefficient that can equal 0. A simple example here would be the difference between coefficients depending on k, the spring constant which can take the value 0, and m, the mass of a particle which cannot.

A specific use for systems with zero-valued coefficients comes in the aim for the NN to identify and preserve conservation laws. Tests for this can be performed by validating the model on unseen systems (of the trained Lagrangian family) that adhere to certain conservation laws, such as harmonic oscillators within the DHO family. Here, we would have $c_3 = 0$, indicating the lack of a damping term within the oscillator. However, this behaviour likely only comes through a well-optimised and trained model.

One such reason for the lack of optimisation in the presented model was the NN's inability to correctly obtain amplitudes, as seen in figure 7. Manually varying coefficients outputted from the NN discovered that, often, a larger c_1 coefficient would produce more accurate motion data. The under-prediction of this coefficient may be related to the discussion about the loss function, as mentioned previously within this section, as the c_1 coefficient is mass dependent. This may, therefore, be remedied by a more specialised loss function incorporating more physical knowledge. Alternatively, a term akin to an L^n norm for a large value of n could be implemented. This norm would more strongly penalise the maximal deviation in the predicted values when compared with RMS or MAE.

We consider one other solution to the amplitude-scaling issue, which was not tried during the development of the NN. By varying the DHO Lagrangian embedding we can provide parameters that scale multiple terms. Potential embeddings of this format could be represented as

$$\Lambda = c_1\mathcal{T} - c_2\left(c_3\mathcal{V} + c_4 K\right) \tag{30a}$$

$$\Lambda = c_1\left(c_2\mathcal{T} - c_3\left(c_4\mathcal{V} + c_5 K\right)\right) \tag{30b}$$

where $\mathcal{T} = \dot{q}^2, \mathcal{V} = q^2, K = \dot{q}_+ q_-$ for notational compactness and more parameters are introduced to allow grouped varying of coefficients. Each of these attempts to solve the issue at hand but, in doing so, may introduce more issues. Both of these introduce more coefficients requiring the neural network to accurately predict upwards of four values as opposed to the three required previously which will likely require more computational work and training.

As a final note on the loss function, we discuss the development of the NN. Here we have attempted to bias the NN as little as possible with human formulations of physics. We have instead taken a holistic approach to this issue, providing the neural network with as little physics knowledge as is necessary to tackle the problem at hand, namely the non-negativity conditions which are required to allow the integrator to function and that of the integrator itself with its embedded formulation of discretised non-conservative Lagrangian mechanics. The aim of this ties to perhaps the end goal for a physics AI, one that can discover laws of the universe that we have yet to discover ourselves. By training a ML model heavily on human knowledge of physics one would expect the ML to produce results similar to that of humans.

5.4. Further Applications

Although not a fully optimised model, the neural network does represent progression in a good direction. We envisage a working model having a multitude of physics and more generalised applications such as identifying celestial dynamics behaviour within astrophysics. A trained model may be able to identify equations of motion corresponding to unseen bodies such as planets affecting stellar movement.

This could also be used at small scales to classify the motion of particles in complex systems and identify the main contributors to the particle's behaviour. The classification and identification behaviour may have applications in a variety of other fields. Experimental studies of fluid dynamics and climate sciences stand out as interesting candidates for application due to the complex behaviour of the systems involved. Limitations may exist here when applied to chaotic systems due to the nature of discretisation when studying small changes in initial conditions. More generalised applications could exist within engineering

disciplines where the neural network could be used to improve designs based on testing feedback or within robotics, where advanced perception techniques could be developed, such as the ability to identify and predict the motion of objects. The loss function could also be adapted to help train other NNs, such as a large language model approach to a similar problem [26]. Clearly, these future applications will require a neural network that is trained on a larger set of data. However, the underlying training process will be similar across any family of Lagrangians but may require more complex networks and much larger datasets to produce satisfactory results.

6. Conclusion

We have successfully developed a GPU/generally optimised version of the slimplectic integrator, a non-conservative variational integrator, using the JAX framework. Which, for many purposes, runs in constant time allowing for highly accurate, large systems to be integrated and in linear time for systems integrated for $N > 10^6$ timesteps. This integrator was then implemented within the loss function of a neural network that has produced promising results when trained to identify the Lagrangian governing a system from motion data. The loss function is defined in equation 28.

Trained on a dataset of damped harmonic oscillators the neural network predicts equations of this form with the scope to train on larger datasets in order to predict a greater scope of functions. This neural network does not perfectly reproduce these equations of motion but could possibly with future development and investigation into the behaviour of the loss function. The neural network predicts a Lagrangian that produces q values accurate to ± 0.010 and π values accurate to ± 0.10 for a system with mean values of $q = 0.115, \pi = 0.65$. The loss function and model training method ran into issues in the low loss domain, possibly due to dividing by 0 errors as highlighted in figure 8. This project was ultimately time-constrained, and further research on and development of the loss function will follow shortly.

Acknowledgments

I would like to thank my project partner Joshua Coles for his indispensable coding abilities, ideas and overall support throughout this project. Without him I would still probably be stuck setting up the correct Python environment. I would also like to thank my supervisor Dr. David Tsang. His advice and mentorship have been invaluable and I have very much enjoyed our often wandering discussions during project meetings.

References

[1] Sahiner B, Pezeshk A, Hadjiiski L M, Wang X, Drukker K, Cha K H, Summers R M and Giger M L 2019 *Med. Phys.* **46** e1–e36
[2] Vanderplas J, Connolly A, Ivezić Ž and Gray A 2012 Introduction to astroml: Machine learning for astrophysics *Conference on Intelligent Data Understanding (CIDU)* pp 47 –54
[3] Kingma D P and Ba J 2017 Adam: A method for stochastic optimization (*Preprint* 1412.6980)
[4] Bradbury J, Frostig R, Hawkins P, Johnson M J, Leary C, Maclaurin D, Necula G, Paszke A, VanderPlas J, Wanderman-Milne S and Zhang Q 2018 JAX: composable transformations of Python+NumPy programs URL http://github.com/google/jax
[5] Maclaurin D, Duvenaud D and Adams R P 2015 Autograd: Effortless gradients in numpy *ICML 2015 AutoML Workshop* vol 238 p 5
[6] Openxla https://openxla.org accessed: 2024-03-12
[7] Goldstein H 1980 *Classical Mechanics* (Addison-Wesley)
[8] Galley C R, Tsang D and Stein L C 2014 Publisher: arXiv Version: 1 URL https://arxiv.org/abs/1412.3082
[9] Galley C R 2013 *Phys. Rev. Lett.* **110**(17) 174301 URL https://link.aps.org/doi/10.1103/PhysRevLett.110.174301
[10] Sanz-Serna J 1992 *Acta Numerica* **1** 243 – 286
[11] DeVries P and Hasbun J 2011 *A First Course in Computational Physics* (Jones & Bartlett Learning) p 215 ISBN 9780763773144
[12] Wisdom J and Holman M 1991 *The Astronomical Journal* **102** 1528–1538
[13] Rein H and Tamayo D 2015 *Monthly Notices of the Royal Astronomical Society* **452** 376–388 ISSN 0035-8711
[14] Brooks S, Gelman A, Jones G and Meng X L 2011 *Handbook of Markov Chain Monte Carlo* (Chapman and Hall/CRC) ISBN 9780429138508 URL http://dx.doi.org/10.1201/b10905
[15] Verlet L 1967 *Phys. Rev.* **159**(1) 98–103 URL https://link.aps.org/doi/10.1103/PhysRev.159.98
[16] Tsang D, Galley C R, Stein L C and Turner A 2015 *The Astrophysical Journal Letters* **809** L9 URL https://dx.doi.org/10.1088/2041-8205/809/1/L9
[17] Farr W M and Bertschinger E 2007 *The Astrophysical Journal* **663** 1420–1433 (*Preprint* astro-ph/0611416)
[18] Boyd J P 2001 *Chebyshev and Fourier spectral methods* (Courier Corporation)
[19] Burns J A, Lamy P L and Soter S 1979 *Icarus* **40** 1–48 ISSN 0019-1035 URL https://www.sciencedirect.com/science/article/pii/0019103579900502
[20] Chollet F *et al.* 2015 Keras https://keras.io
[21] Abadi M *et al.* 2015 TensorFlow: Large-scale machine learning on heterogeneous systems software available from tensorflow.org URL https://www.tensorflow.org/
[22] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (MIT Press) http://www.deeplearningbook.org
[23] Zhong G and Marsden J E 1988 *Physics Letters A* **133** 134–139 ISSN 0375-9601 URL https://www.sciencedirect.com/science/article/pii/0375960188907736
[24] Kane C, Marsden J E and Ortiz M 1999 *Journal of Mathematical Physics* **40** 3353–3371 ISSN 0022-2488 (*Preprint* https://pubs.aip.org/aip/jmp/article-pdf/40/7/3353/19017050/3353_1_online.pdf) URL https://doi.org/10.1063/1.532892
[25] Tuckerman M E and Martyna G J 2000 *The Journal of Physical Chemistry B* **104** 159–178 ISSN 1520-6106 publisher: American Chemical Society URL https://doi.org/10.1021/jp992433y
[26] Tenachi W, Ibata R and Diakogiannis F I 2023 (*Preprint* arXiv:2303.03192)