Friedrich-Alexander-Universität
Erlangen-Nürnberg

Lehrstuhl für
Technische Dynamik
Prof. Dr.-Ing. habil. Sigrid Leyendecker

Master's thesis

On the identification of port-Hamiltonian models via machine-learning

Jiandong Zhao

September 5, 2022

Declaration

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

_____
Date

_____
Name

1 System Identification based on machine learning

1.1 Hamiltonian system

A Hamiltonian system is a triple (M, ω, H), where (M, ω) is a symplectic manifold which consists of a manifold M and a symplectic structure ω. The preservation of the ω preserves the Hamiltonian during the evolution of the system.

Hamiltonian or Hamiltonian function H is a smooth function on the manifold M that assigns the coordinates to conserved quantities of physical systems (Gerd section 9). Normally, the Hamiltonian of an isolated physical system represents the total energy. For instance, in mechanical systems, the total energy with k state variables q_i is $E(q) = E(q_1, ..., q_k)$, where the state variables are some natural variables. With the generalized coordinate $\mathbf{q}$ and generalized momentum $\mathbf{p}$, it holds that

$$E(q) = H(\mathbf{q}, \mathbf{p}) = T + V = \mathbf{q}^T \frac{1}{2c}\mathbf{q} + \mathbf{p}^T \frac{1}{2m}\mathbf{p}, \tag{1.1}$$

where T is kinetic energy and V is potential energy of the system.

Let the state variables $x = (\mathbf{q}, \mathbf{p}) \in \mathbb{X}$. Since the Hamiltonian is invariant with respect to time, it holds

$$\frac{dH}{dt} = \frac{\partial H}{\partial x}\frac{dx}{dt} = 0, \tag{1.2}$$

where $\frac{dx}{dt}$ can be written in the form:

$$\frac{dx}{dt} = J(\frac{\partial H}{\partial x})^T, \tag{1.3}$$

where

$$J = \begin{bmatrix} 0 & -I \\ I & 0 \end{bmatrix}$$

such that $\frac{dH}{dt} = \frac{\partial H}{\partial x}\frac{dx}{dt} = \frac{\partial H}{\partial x}J(\frac{\partial H}{\partial x})^T = 0$. A matrix in the form of J is called skew-symmetry matrix.

1.2 Port-Hamiltonian system

A port-Hamiltonian system is a Hamiltonian system endowed with geometric structure, which is known as Dirac structure. A Dirac structure is a subspace $\mathcal{D} \subset \mathcal{F} \times \mathcal{E}$, where $\mathcal{F}$ and $\mathcal{E}$ are the spaces of the port variables (i.e. flows f and efforts e), such that for all pair (f, e) in the Dirac structure $\mathcal{D}$ the power $< e \mid f >$ equal to zero (Vincent section 2.1.2). Note that $\mathcal{E}$ is the dual space of $\mathcal{F}$.

The concept of "ports" comes from the port-based physical system modeling approach (Vincent). It reveals that subsystems interact with each other via ports. In port-Hamiltonian system, such interactions are assumed to be exchanges of energy.

TODO: more details

1.3 Differential Equations

1.3.1 Ordinary Differential Equations

Consider a real-valued function x with k continuous derivatives: $x \in C^k(X)$, $X \subseteq \mathbb{R}$, $k \in \mathbb{N}_0$, $x : \mathbb{R} \to \mathbb{R}$. An implicit ordinary differential equation (ODE) is of the form:

$$f\left(t, x, x^{(1)}, \ldots, x^{(k)}\right) = 0. \tag{1.4}$$

Corresponding to implicit form, an explicit ordinary differential equation is of the form:

$$x^{(k)} = f\left(t, x, x^{(1)}, \ldots, x^{(k-1)}\right). \tag{1.5}$$

Suppose the function $x : \mathbb{R} \to \mathbb{R}^n$. Then the equation 1.5 should be rewritten in the form of system:

$$
\begin{aligned}
x_1^{(k)} &= f_1\left(t, x, x^{(1)}, \ldots, x^{(k-1)}\right) \\
x_2^{(k)} &= f_2\left(t, x, x^{(1)}, \ldots, x^{(k-1)}\right) \\
&\vdots \\
x_n^{(k)} &= f_n\left(t, x, x^{(1)}, \ldots, x^{(k-1)}\right).
\end{aligned}
\tag{1.6}
$$

The above form is a N-dimensional system, which has one or more ordinary differential equations. It is commonly called t as the independent variable and x as the dependent variable.

1.3.2 Initial Value Problems

A initial value problem(IVP) consists of a pair of equations:

$$\dot{x} = f(t, x), \quad x(t_0) = x_0 \tag{1.7}$$

Then the integral of the initial value problem is of the form:

$$x(t) = x_0 + \int_{t_0}^{t} f(s, x(s))\, ds. \tag{1.8}$$

Suppose t is infinite small such that $x_0(t) = x_0$. Interpolate it to the equation 1.8, the next state x_1 after the initial state x_0 can be written as:

$$x_1(t) = x_0 + \int_{t_0}^{t} f(s, x_0(s))\, ds. \tag{1.9}$$

By repeating this procedure, it is not difficult to deduce

$$x_{m+1}(t) = x_0 + \int_{t_0}^{t} f(s, x_m(s))\, ds. \tag{1.10}$$

for all approximating states.

TODO: more details

1.4 Harmonic Oscillator

1.4.1 Undamped Harmonic Oscillator

In the classical simple undamped harmonic oscillator model, a mass m is fixed to one end of a spring with spring compliance c and the other end of the spring is fixed to a fixed rigid body. The spring displacement q is in the positive direction of the direction of stretching by the mass. See figure 1.1.

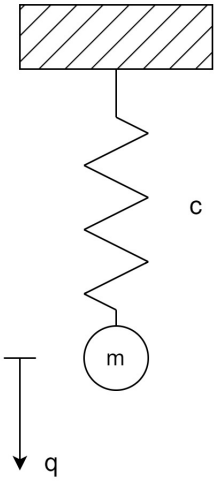


Figure 1.1: Undamped harmonic oscillator

According to Newton's second law $F = ma = m\ddot{x}$ and Hooke's law $F = -kx$, it holds $m\ddot{x} + kx = 0$, which is an implicit ODE in the form of the equation 1.4. As the highest order in the ODE is second order, it is called second order differential equation.

In the undamped harmonic oscillator model, referring to the equation 1.1, the Hamiltonian is the sum of the kinetic energy $T = \frac{1}{2m}p^2$ and the potential energy $\frac{1}{2c}q^2$:

$$H(q,p) = \frac{1}{2c}q^2 + \frac{1}{2m}p^2, \tag{1.11}$$

where p is the momentum of the mass. The state of the system (also known as canonical coordinates) $x = (q,p)$ moves along the symplectic gradient $\dot{x} = (\dot{q},\dot{p})$:

$$\dot{q} = \frac{\partial H}{\partial p}, \quad \dot{p} = -\frac{\partial H}{\partial q}. \tag{1.12}$$

Moving along the symplectic gradient holds the total energy of the conserve system, i.e. the Hamiltonian, constant. So that the derivative of the Hamiltonian in Equation 1.13 is hold

as zero.

$$\dot{H} = \frac{\partial H}{\partial q}\dot{q} + \frac{\partial H}{\partial p}\dot{p} = 0 \tag{1.13}$$

Referring to the equations 1.6, the ordinary differential equation system of the undamped harmonic oscillator could be written as

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} \frac{p}{m} \\ -\frac{q}{c} \end{bmatrix}. \tag{1.14}$$

1.4.2 Isothermal Damped Harmonic Oscillator

In contrast to undamped harmonic oscillator, the isothermal damped harmonic oscillator should be regarded as a port-Hamiltonian system.

TODO: more details

The ordinary differential equation system of the isothermal damped harmonic oscillator are written as

$$\begin{bmatrix} \dot{q} \\ \dot{p} \\ \dot{s_e} \end{bmatrix} = \begin{bmatrix} \frac{p}{m} \\ -\frac{q}{c} - d\frac{p}{m} \\ d(\frac{p^2}{m^2})\frac{1}{\theta_o} \end{bmatrix}, \tag{1.15}$$

where d is the damping coefficient, s_e is the entropy of the environment, θ_o is the environmental temperature.

1.4.3 Nonisothermal Damped Harmonic Oscillator

In a classical port-Hamiltonian system, such as isothermal damped harmonic oscillator, free energy flows into the energy dissipating component and eventually dissipates. However, the energy cannot disappear into thin air, so where does it ultimately go? The answer is clearly the environment. If the environment is also considered as a component, the system is called nonisothermal.

TODO: more details

The ordinary differential equation system of the nonisothermal damped harmonic oscillator

are written as

$$\begin{bmatrix} \dot{q} \\ \dot{p} \\ \dot{s_e} \\ \dot{s_d} \end{bmatrix} = \begin{bmatrix} \frac{p}{m} \\ -\frac{q}{c} - d\frac{p}{m} \\ d(\frac{p^2}{m^2})\frac{1}{\theta_o} \\ \alpha(\theta_d - \theta_o)/\theta_o \end{bmatrix}, \tag{1.16}$$

where α is the heat transfer coefficient, θ_d is the temperature of the damping.

1.5 Physical Model

1.5.1 Numerical Method

To eliminates the complicated procedures of solving differential equations (the procedure in 1.10), some numerical methods such as Euler's method, Runge–Kutta method are common to use.

TODO: also introduce Euler's method.

Consider a IVP problem in equations 1.7. Let $t_m = t_0 + hm$, where h is the time step size. A fourth order Runge–Kutta method is given by the following:

$$\begin{aligned} &x_{m+1} = x_m + \frac{h}{6}(k_{1,m} + 2k_{2,m} + 2k_{3,m} + k_{4,m}), \\ &\text{where} \\ &\quad k_{1,m} = f(t_m, x_m), \\ &\quad k_{2,m} = f(t_m + \frac{h}{2}, x_m + \frac{h}{2}k_{1,m}), \\ &\quad k_{3,m} = f(t_m + \frac{h}{2}, x_m + \frac{h}{2}k_{2,m}), \\ &\quad k_{4,m} = f(t_{m+1}, x_m + hk_{3,m}). \end{aligned} \tag{1.17}$$

Note that the higher-order Runge–Kutta method will converge faster, but it will also lead to more computations in each step. In general, fourth order is a suitable choice.

1.5.2 Simulation

To investigate the dynamics of the physical system, it is common to simulate the trajectories of the dynamical system with respect to time. The following block diagram illustrates this process.

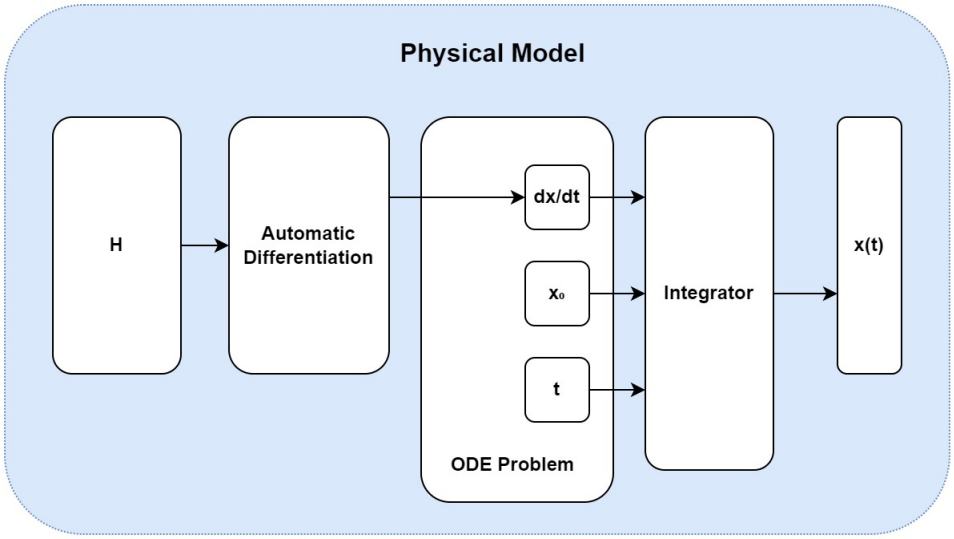
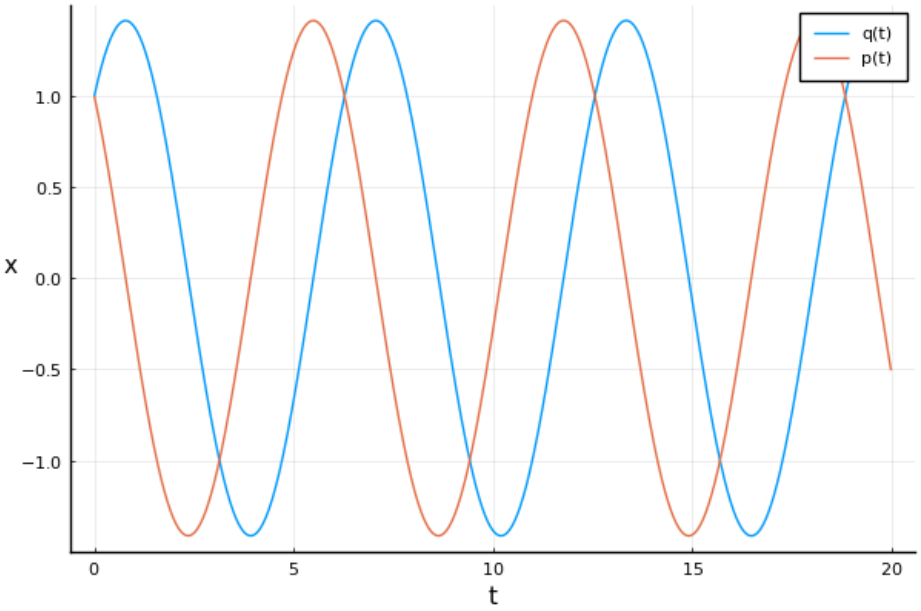
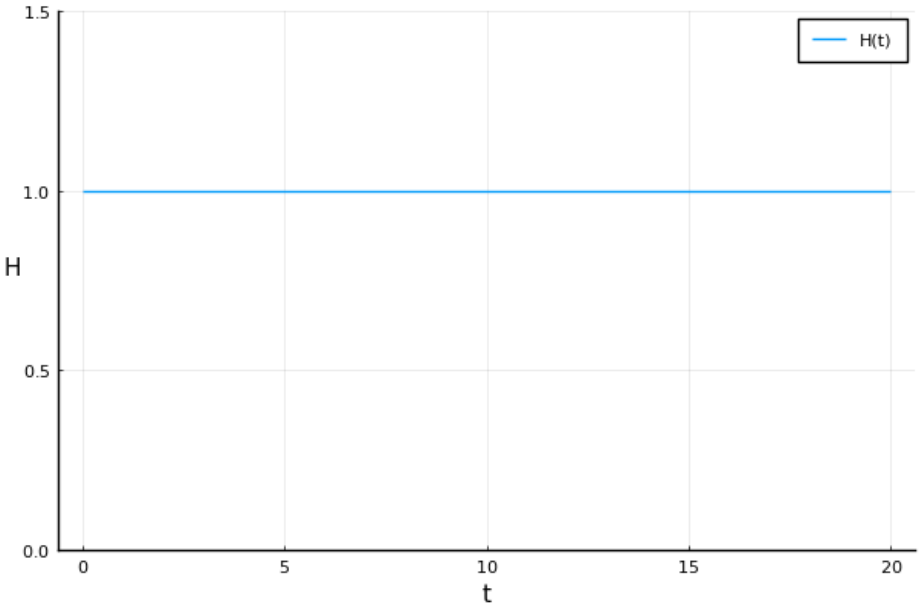


Figure 1.2: Block diagram of simulation

For simplicity, consider an undamped harmonic oscillator model. The Hamiltonian on the left in figure 1.2 is $H(q, p)$ in equation 1.11. The next block of automatic differentiation (AD) stands for a set of techniques that allow a computer program to yield the derivative of a function. Automatic differentiation uses the fact that any computer program that implements a vector-valued function could be decomposed into a sequence of basic specified operations, each of which can be easily differentiated by looking up a table. These basic partial derivatives that compute a particular term are combined into a differential form, such as gradient, Jacobian, etc. In the case of simple undamped harmonic oscillator, the output of the automatic differentiation block is the symplectic gradient $\dot{x} = (\dot{q}, \dot{p})$ in the equation 1.12. In fact, this symplectic gradient is consistent with the form of a 2-dimensional ordinary differential equations system mentioned in the equation 1.6, also see the equation 1.14. This ODE system together with initial condition x_0 and the independent variable t specifies an IVP problem (also called ODE problem in the block diagram). To evaluate the trajectories of the states, or in other words, to integrate the ODEs, it is efficient to introduce numerical methods to solve this ODE problem. For instance, the integrator block may embed a fourth order symplectic Runge–Kutta method, which integrates the symplectic 2-form $dq \wedge dp$ over a symplectic manifold. The solution of the integrator would be the time evolution of the canonical coordinates $x(t) = (q(t), p(t))$ with respect to time in the figure 1.3a.



(a) Time evolution of the canonical coordinates



(b) Time evolution of the Hamiltonian

Figure 1.3: Time evolution of a simple undamped harmonic oscillator model

In the figure 1.3b, the Hamilton is as time-invariant as one might imagine. Thus, the canonical coordinates in phase space would be an ellipse in the figure 1.4.

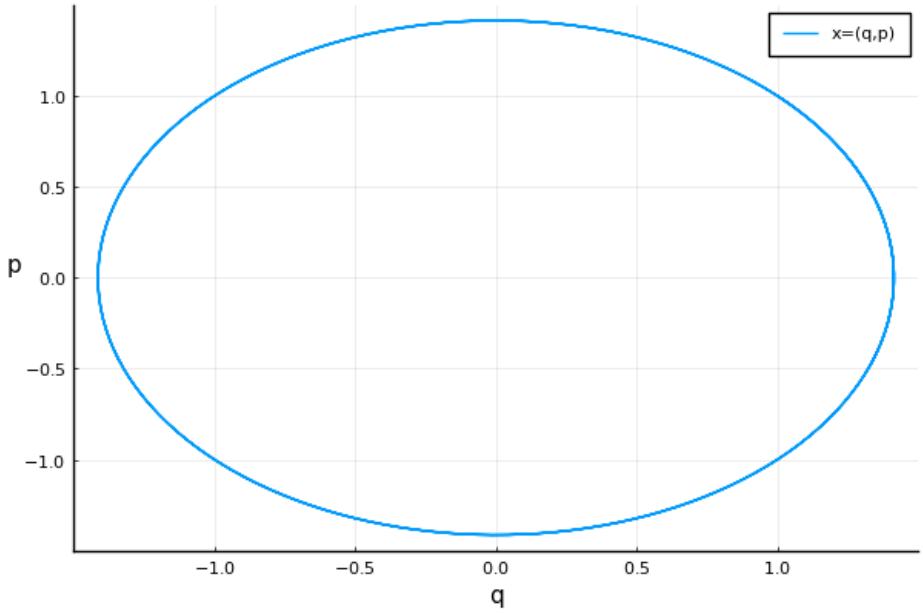


Figure 1.4: Phase portrait of a simple undamped harmonic oscillator model

1.6 Neural Network

Artificial neural networks (ANNs), also referred to as neural networks (NNs), are algorithmic mathematical models that mimic the behavioral characteristics of animal neural networks for distributed parallel information processing, even though this mimicry is superficial (Stuart). Such networks rely on the complexity of the system to process information by adjusting the relationship between a large number of internal nodes connected to each other.

1.6.1 Perceptron

Neural network technology originated in the 1950s as perceptron by McCulloch and Pitts. The characteristics of perceptrons are strongly contemporary: their inputs and outputs are in binary form. Each of these perceptrons is characterized as either "on" or "off", with an "on" response occurring when stimulated by a sufficient number of neighboring perceptrons.

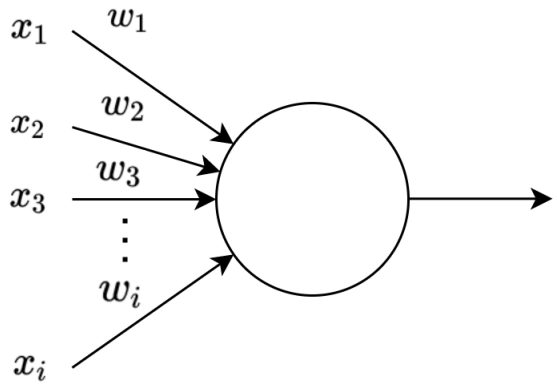


Figure 1.5: Perceptron

In the figure 1.5, x denotes multiple inputs to the perceptron, w denotes the weight corresponding to each input and the arrow to the right of the perceptron indicates that it has only one output. Each input is multiplied by the corresponding weight and then summed, and the result is compared with a threshold, with 1 being output if it is greater than the threshold and 0 being output if it is less than the threshold:

$$f(x) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i \leq \text{threshold} \\ 1 & \text{if } \sum_{i=1}^{n} w_i x_i > \text{threshold} \end{cases} \tag{1.18}$$

Let $b = -\text{threshold}$, the formula 1.18 can be rewritten as:

$$f(x) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i + b \leq 0 \\ 1 & \text{if } \sum_{i=1}^{n} w_i x_i + b > 1 \end{cases}, \tag{1.19}$$

where b is also known as bias.

1.6.2 Activation Function

Following the designers of the perceptron, McCulloch and Pitts, a node in neural network computes the weighted sum of inputs and then applies a activation function to yield the output $g(z)$, where $z = \sum_{i=1}^{n} w_i x_i + b$. For instance, the perceptron in the equation 1.19 applied the Heaviside step function (also known as binary step function)

$$g(z) = \begin{cases} 0 & \text{if } z \leq 1 \\ 1 & \text{if } z > 0 \end{cases} \tag{1.20}$$

as its activation function.

The figure 1.6 shows the relationship between perceptron and activation function.

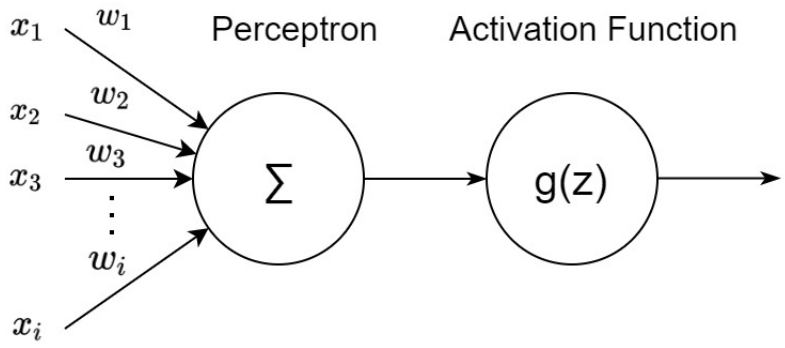


Figure 1.6: The structure of a classical neuron

A good activation function should be continuous and differentiable, such that the derivative of the activation function can be learned directly by using numerical optimization methods for updating parameters. In addition, the domain of derivatives should be in a suitable range, not too large or too small. Otherwise, it will affect the efficiency and stability of the training. Despite some of the limitations mentioned above, there are still a wide variety of activation functions available. One of them is logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{1.21}$$

Sigmoid is a classical saturating function. It is common to use sigmoid as activation function when the output is expected to be probabilistic. Since the probability in the real world is always limited to the range of 0 to 1, this is consistent with the range of sigmoid.

TODO: plot sigmoid

Another activation function similar to the sigmoid function is the tanh function:

$$f(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})} \tag{1.22}$$

Compared to sigmoid, tanh function has the output range of -1 to 1:

TODO: plot tanh

One characteristic of the tanh function is zero-centered, while the sigmoid function is non-zero-centered. The non-zero-centered output causes a bias shift in the input of the neurons in the later layer and further slows down the convergence of gradient descent. That is the reason why the tanh function converges faster than sigmoid.

The ReLU function (Rectified Linear Unit):

$$ReLU(x) = max(0, x). \tag{1.23}$$

Neurons with ReLU only need to perform addition, multiplication and comparison operations, which is more efficient in computation. However, the ReLU function is still non-zero-centered, which may affect the efficiency of gradient descent. Furthermore, the use of ReLU activation functions may lead to the well-known dying ReLU problem. One may need to try other functions in ReLU family to avoid this problem.

TODO: plot ReLU

1.6.3 Architecture Of Neural Network

So far, neural networks have evolved a variety of architectures. The following three types of neural network architectures are commonly used: feedforward neural network, feedback neural network and graph neural network. Take one of the simplest neural network models as an example: a feedforward neural network could be seen as a directed acyclic graph (DAG) with specified input and output nodes, which are fully connected to each other, i.e., the nodes in the later layer are all connected to each node in the former layer. Each node computes its input from the former layer with the parameters and activation function and passes the result to the nodes in the latter layer as their inputs. By definition of a directed acyclic

graph, these nodes will never form a closed loop. The figure 1.7 is an example of feedforward neural network.

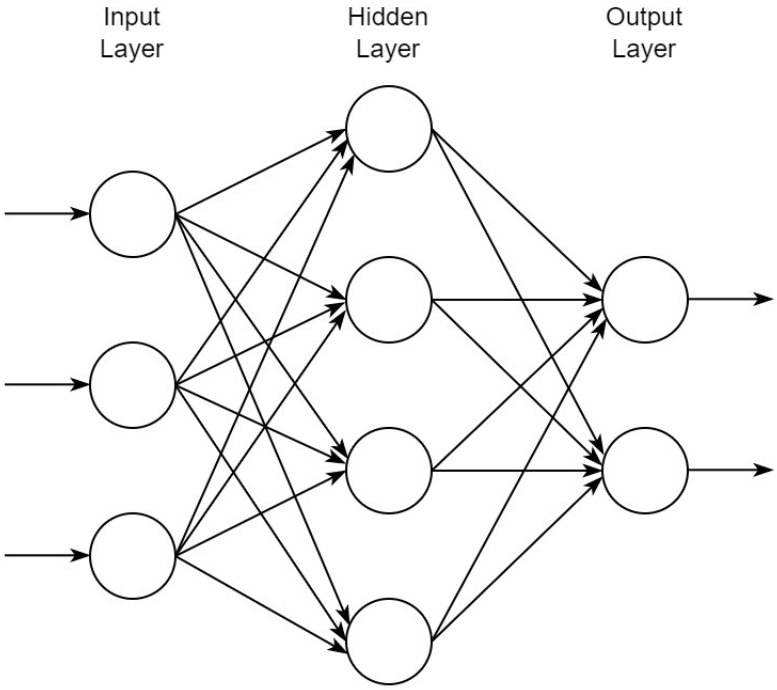


Figure 1.7: Architecture of a feedforward neural network.

To clarify the principle of feedforward neural network, there are some notations to declare first. The uppercase L stands for the number of total layers (input layer not included) in the feedforward neural network, the lowercase l for the lth layer and M_l for the number of neurons in lth layer. The same as the equation 1.20, $g_l(z^{(l)})$ denotes the activation function in lth layer, where $z^l \in \mathbb{R}^{M_l}$. The parameters weight and bias are represented by the weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$ and the bias matrix $\mathbf{b}^{(l)} \in \mathbb{R}^{M_l}$. Thus, the input $\mathbf{z}^{(l)}$ and output $\mathbf{a}^{(l)}$ of a neuron in lth layer could be written as:

$$\begin{aligned} \mathbf{z}^{(l)} &= \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{z}^{(l)} \in \mathbb{R}^{M_l} \\ \mathbf{a}^{(l)} &= g_l(\mathbf{z}^{(l)}), \quad \mathbf{a}^{(l)} \in \mathbb{R}^{M_l}. \end{aligned} \tag{1.24}$$

In fact, the whole feedforward neural network can be seen as a function $f(\mathbf{x};\theta)$, where $\mathbf{x}$ is the input of the neural network and θ is the set of the parameters (including weight and bias). Note that the input $\mathbf{x}$ here is the output of the neurons in the input layer, i.e., $\mathbf{x} = [x_1, ..., x_n] = \mathbf{a}^{(0)}$ and the function itself is the output of the neurons in the ouput layer, i.e., $f(\mathbf{x};\theta) = \mathbf{a}^{(L)}$. In general, one can use the feedforward propagation algorithm as the first step to yield initial output to compute the loss function for further optimization. The following is the algorithmic procedure for feedforward propagation:

Algorithm 1 The feedforward propagation algorithm

Require: neural network $f(\mathbf{x}; \mathbf{W}, \mathbf{b})$ with lth layer
Require: weight matrix $\mathbf{W} \in \mathbb{R}^{M_l \times M_{l-1}}$, bias matrix $\mathbf{b} \in \mathbb{R}^{M_l}$
Require: activation function $g(z)$
Require: input $\mathbf{x}$
  $\mathbf{a}^{(0)} = \mathbf{x}$ ▷ The output of the input layer
  for $n = 1...N$ **do**
    for $l = 1...L$ **do**
      $\mathbf{z}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ ▷ Compute the input of a neuron in lth layer
      $\mathbf{a}^{(l)} \leftarrow g_l(\mathbf{z}^{(l)})$ ▷ Compute the output of a neuron in lth layer
      ▷ See the equation 1.24
    end for
  end for
  return $f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{a}^{(L)}$ ▷ Return the initial output

1.6.4 Loss Function

The loss function $\mathcal{L}(y, f(\mathbf{x};\theta))$ is a non-negative real-valued function that quantifies the error between the truth y and the model prediction $f(\mathbf{x};\theta)$. A small error is always expected, as it implies the prediction is very close to the truth. The following are three commonly used loss functions: zero-one loss function, absolute error loss function and squared error loss function.

Zero-one loss function:

$$\mathcal{L}_{0/1}(y, f(\mathbf{x};\theta)) = \begin{cases} 0 & \text{if } y = f(\mathbf{x};\theta) \\ 1 & \text{if } y \neq f(\mathbf{x};\theta) \end{cases} \tag{1.25}$$

The zero-one loss function is pretty intuitive for judging whether the result is good or bad. However, the binary output does not provide some room for further optimization.

Absolute error loss function, also called $\mathcal{L}_1$ loss:

$$\mathcal{L}_1(y, f(\mathbf{x};\theta)) = |y - f(\mathbf{x};\theta)| \tag{1.26}$$

Squared error loss function, also called $\mathcal{L}_2$ loss:

$$\mathcal{L}_2(y, f(\mathbf{x};\theta)) = (y - f(\mathbf{x};\theta))^2 \tag{1.27}$$

The $\mathcal{L}_1$ and $\mathcal{L}_2$ loss are widely used in linear regression models to evaluate the error. In general, the $\mathcal{L}_2$ loss is preferred since it is more sensitive. Yet the $\mathcal{L}_1$ loss performs better than the $\mathcal{L}_2$ loss in terms of robustness, especially when there are some outliers in the data.

In addition to the mentioned loss function, it is worth noting one more concept in statistics: the mean squared error (MSE). In machine learning, the following errors are commonly introduced for optimization:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_2(y, f(\mathbf{x};\theta)) = \frac{1}{N}\sum_{n=1}^{N}(y - f(\mathbf{x};\theta))^2 \tag{1.28}$$

1.7 Optimization

An optimization algorithm is ordinarily an algorithm that seeks the optimum. In general, finding the global optimum on a convex objective function is the simplest problem. However,

if the objective function is unfortunately a non-convex function, then one has to settle for second best, i.e., seeking the local optimum.

1.7.1 Gradient Descent

Gradient descent (GD) is the most usual optimization algorithm in machine learning. The idea is to iterate the following procedure:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{N} \sum\nolimits_{n=1}^{N} \nabla \mathcal{L}\left(y_n, f(\mathbf{x_n}; \theta)\right) \tag{1.29}$$

where θ_t is the parameters at the time step t, α is the learning rate and $\nabla \mathcal{L}\left(y^{(n)}, f(\mathbf{x^{(n)}}; \theta)\right)$ is the gradient of the loss function:

$$\nabla \mathcal{L}\left(y_n, f(\mathbf{x_n}; \theta)\right) = \frac{\partial \mathcal{L}\left(y_n, f(\mathbf{x_n}; \theta)\right)}{\partial \theta} \tag{1.30}$$

The parameters θ_t in each iteration are implemented to the neural network model, and the error (e.g. the mean squared error in the equation 1.28) is also updated simultaneously with the update of the neural network model. Once the error is small enough to be a desirable value, then the model is considered to be well trained.

1.7.2 Back Propagation

Back propagation is an algorithm for efficient computation of gradients. The kernel of this algorithm lies in the computation of the partial derivatives of the loss function with respect to parameters, i.e., the gradient in the equation 1.30.

Consider a neural network $f(\mathbf{x}; \theta) = f(\mathbf{x}; \mathbf{W}, \mathbf{b})$. Based on the chain rule, the partial derivatives of the loss function could be written as the form:

$$\begin{aligned} \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{W}^{(l)}} &= \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}} \\ \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{b}^{(l)}} &= \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \end{aligned} \tag{1.31}$$

where $\mathbf{z}^{(l)}$ is the input of a neuron in lth layer, which is already mentioned above in the equation 1.24, i.e., $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$. Notice that in the equation 1.31 only three terms to be computed: $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}}$, $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}$ and $\frac{\partial \mathcal{L}(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b}))}{\partial \mathbf{z}^{(l)}}$.

The first two terms are relatively simple to deduce:

$$\begin{aligned} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}} &= \mathbf{a}^{(l-1)} \\ \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} &= \mathbf{I}, \end{aligned} \tag{1.32}$$

where $\mathbf{I}$ is an identity matrix. The third term is called the error, which reflects the sensitivity of the loss to neurons in lth layer. It is denoted by $\delta^{(l)}$. Applying the chain rule, the error $\delta^{(l)}$ in terms of the error in the later layer $\delta^{(l+1)}$ can be written as:

$$\begin{aligned} \delta^{(l)} &= \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{z}^{(l+1)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \\ &= (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot g_l'(\mathbf{z}^{(l)}), \end{aligned} \tag{1.33}$$

where $\mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}$, $\mathbf{a}^{(l)} = g_l(\mathbf{z}^{(l)})$ from the equation 1.24 are plugged into the second line in 1.33 and the operation $\odot$ stands for Hadamard product.

After the above deduction, the equation 1.31 together with 1.32 and 1.33 are rewritten in the following form:

$$\begin{aligned} \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{W}^{(l)}} &= \delta^{(l)}(\mathbf{a}^{(l-1)})^T \quad \in \mathbb{R}^{M_l \times M_{l-1}} \\ \frac{\partial \mathcal{L}\left(y, f(\mathbf{x}; \mathbf{W}, \mathbf{b})\right)}{\partial \mathbf{b}^{(l)}} &= \delta^{(l)} \quad \in \mathbb{R}^{M_l}. \end{aligned} \tag{1.34}$$

The algorithm 2 presents the gradient descent training procedure using the back propagation algorithm.

Algorithm 2 Back propagation in the case of gradient descent

The back propagation algorithm requires initial parameters θ_0, which is computed by the feedforward propagation algorithm 1. After the neural network model adapts to the initial parameters θ_0:

Require: neural network $f(\mathbf{x}; \mathbf{W}, \mathbf{b})$ with lth layer
Require: weight matrix $\mathbf{W} \in \mathbb{R}^{M_l \times M_{l-1}}$, bias matrix $\mathbf{b} \in \mathbb{R}^{M_l}$, initial parameters θ_0
Require: activation function $g(z)$
Require: regularization parameter λ
Ensure: $\nabla \mathcal{L}\left(y_n, f(\mathbf{x}; \theta)\right) = 0$
  while $\nabla \mathcal{L}\left(y_n, f(\mathbf{x}; \theta)\right) \neq 0$ **do**
    for $n = 1...N$ **do**
      for $l = 1...L$ **do**
        $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ ▷ Compute the input of neurons in the lth layer
        $\mathbf{a}^{(l)} = g_l(\mathbf{z}^{(l)})$ ▷ Compute the output of neurons in the lth layer
        ▷ See the equation 1.24
        $\delta^{(l)} = (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot g_l'(\mathbf{z}^{(l)})$ ▷ Compute the error in the lth layer
        ▷ See the equation 1.33
        $\frac{\partial \mathcal{L}(y_n, f(\mathbf{x}; \mathbf{W}, \mathbf{b}))}{\partial \mathbf{W}^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^T$ ▷ Compute the gradient on the weight matrix
        $\frac{\partial \mathcal{L}(y_n, f(\mathbf{x}; \mathbf{W}, \mathbf{b}))}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$ ▷ Compute the gradient on the bias matrix
        ▷ See the equation 1.34
        $\mathbf{W}^{(l+1)} \leftarrow \mathbf{W}^{(l)} - \alpha(\delta^{(l)}(\mathbf{a}^{(l-1)})^T + \lambda \mathbf{W}^{(l)})$ ▷ Update the weight matrix
        $\mathbf{b}^{(l+1)} \leftarrow \mathbf{b}^{(l)} - \alpha\delta^{(l)}$ ▷ Update the bias matrix
        ▷ See the equation 1.29
      end for
    end for
  end while
  return $\mathbf{W}, \mathbf{b}$ ▷ Return the parameters

The regularization parameter λ is used to prevent overfitting. The reason for overfitting is that some of the parameters are way too large. Adding a regularization term can restrict the parameters in order to prevent overfitting. Note that the regularization parameter λ may also appear in the step of computing the gradient on the weight and bias matrices in some other back propagation algorithms.

1.7.3 Adam Algorithms

As can be seen from the back propagation algorithm 2 above, the gradient descent algorithm is actually applied in the step of updating the parameters. So are there any other algorithms to update the parameters besides the gradient descent algorithm? The answer is yes. For example, one can use the momentum method to replace the gradient $\nabla \mathcal{L}(\cdot)$ with momenta and the RMSProp algorithm (Root Mean Squared Propagation algorithm) [Tieleman et al., 2012] to update the learning rate α. An algorithms that combines the ideas of the momentum method and the RMSProp algorithm is the Adam algorithms (Adaptive Moment Estimation Algorithm)[Kingma et al., 2015]. In practice, the Adam algorithm shows better performance,

i.e. fewer parameters to tune, resulting in higher computational efficiency.

Similar to the idea of gradient descent (see 1.29), the point of the Adam algorithm is to iterate the following procedure:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{s}}{\sqrt{\hat{r}} + \epsilon}, \tag{1.35}$$

where ϵ is a very small constant for stabilization (avoiding a denominator of zero). And the two terms $\hat{s}$ and $\hat{r}$ are known as correct moment estimate. It corrected the bias of the original moment estimate s and r at the beginning of the iteration:

$$\begin{aligned} \hat{s_t} &= \frac{s_t}{1 - \beta_1^t} \\ \hat{r_t} &= \frac{r_t}{1 - \beta_2^t}, \end{aligned} \tag{1.36}$$

where β_1^t and β_2^t are learning rate decay. The moment estimate s and r are defined as following:

$$\begin{aligned} s_t &= \beta_1 s_{t-1} + (1 - \beta_1)\mathbf{g_t} \\ r_t &= \beta_2 r_{t-1} + (1 - \beta_2)\mathbf{g_t} \odot \mathbf{g_t} \end{aligned} \tag{1.37}$$

where $\mathbf{g_t}$ is the mean gradient over all training data:

$$\mathbf{g_t} = \frac{1}{N} \sum\nolimits_{n=1}^{N} \nabla \mathcal{L}\left(y_n, f(\mathbf{x}; \theta)\right). \tag{1.38}$$

Algorithm 3 The training procedure of the Adam algorithm

The Adam algorithm also requires initial parameters θ_0, which is computed by the feedforward propagation algorithm 1. The decay rates β_1 and β_2 should be within the range of $[0, 1)$, f.e., $\beta_1 = 0.9$ and $\beta_2 = 0.99$. The learning rate α is recommended to be 0.001. And the very small constant ϵ is set to 10^{-8}.

Require: neural network $f(\mathbf{x}; \mathbf{W}, \mathbf{b})$ with lth layer
Require: weight matrix $\mathbf{W} \in \mathbb{R}^{M_l \times M_{l-1}}$, bias matrix $\mathbf{b} \in \mathbb{R}^{M_l}$, initial parameters θ_0
Require: activation function $g(z)$
Require: decay rate β_1 and β_2
Require: learning rate α
Require: constant ϵ
Ensure: $\nabla \mathcal{L}\left(y_n, f(\mathbf{x}; \theta)\right) = 0$
  while $\nabla \mathcal{L}\left(y_n, f(\mathbf{x}; \theta)\right) \neq 0$ **do**
    for $n = 1...N$ **do**
      for $l = 1...L$ **do**
        $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ ▷ Compute the input of neurons in the lth layer
        $\mathbf{a}^{(l)} = g_l(\mathbf{z}^{(l)})$ ▷ Compute the output of neurons in the lth layer
        ▷ See the equation 1.24
        $\delta^{(l)} = (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot g_l'(\mathbf{z}^{(l)})$ ▷ Compute the error in the lth layer
        ▷ See the equation 1.33
        $\frac{\partial \mathcal{L}(y_n, f(\mathbf{x}; \mathbf{W}, \mathbf{b}))}{\partial \mathbf{W}^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^T$ ▷ Compute the gradient on the weight matrix
        $\frac{\partial \mathcal{L}(y_n, f(\mathbf{x}; \mathbf{W}, \mathbf{b}))}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$ ▷ Compute the gradient on the bias matrix
        ▷ See the equation 1.34
      end for
    end for
    $\mathbf{g_t} \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla \mathcal{L}\left(y_n, f(\mathbf{x}; \theta)\right)$ ▷ Compute the mean gradient, see the equation 1.38
    $s_t \leftarrow \beta_1 s_{t-1} + (1 - \beta_1)\mathbf{g_t}$ ▷ Compute the fist moment estimate
    $r_t \leftarrow \beta_2 r_{t-1} + (1 - \beta_2)\mathbf{g_t} \odot \mathbf{g_t}$ ▷ Compute the second moment estimate
    ▷ See the equation 1.37
    $\hat{s_t} \leftarrow \frac{s_t}{1-\beta_1^t}$ ▷ Correct the bias
    $\hat{r_t} \leftarrow \frac{r_t}{1-\beta_2^t}$ ▷ Correct the bias
    ▷ See the equation 1.36
    $\theta_{t+1} \leftarrow \theta_t - \alpha \frac{\hat{\mathbf{s}}}{\sqrt{\hat{r}} + \epsilon}$ ▷ Update the parameters
    ▷ See the equation 1.35
  end while
  return $\mathbf{W}, \mathbf{b}$ ▷ Return the parameters

1.8 Neural ODE

TODO: introduce System Identification with physical priors

The idea of neural ODE

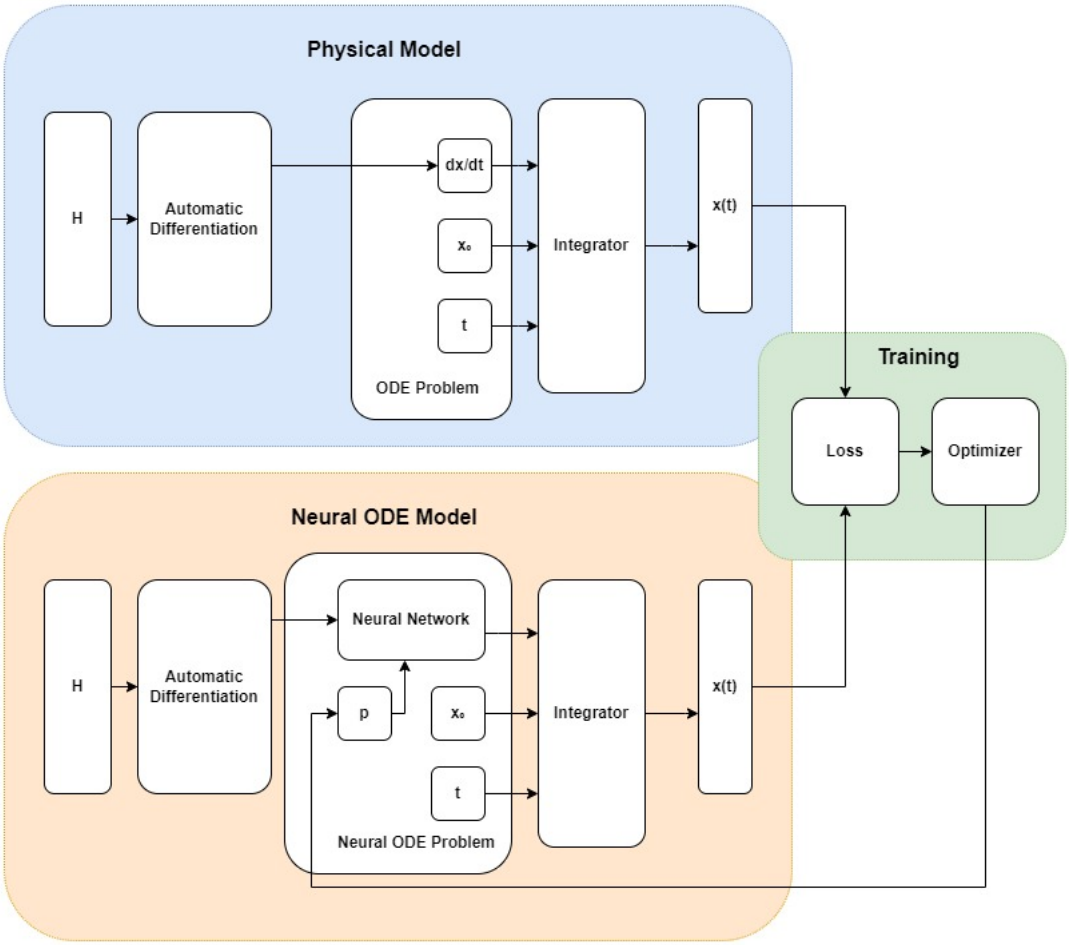


Figure 1.8: Computation graph of Neural ODE model

The model generated by neural ODE is considered as the baseline model. The baseline model is used to benchmark the machine learning results.
In hamiltonian dynamics, the states of a system (q, p) are represented as points in the phase space. Let $x = [q, p]$, $dx/dt = RHS(x)$ where the RHS could be replaced with a neural network. In this case, regardless of the system, the neural network always tries to approximate the system without any physics priors.

TODO: data driven, discrete time model(2017, Maziar)

1.8.1 Undamped Harmonic Oscillator

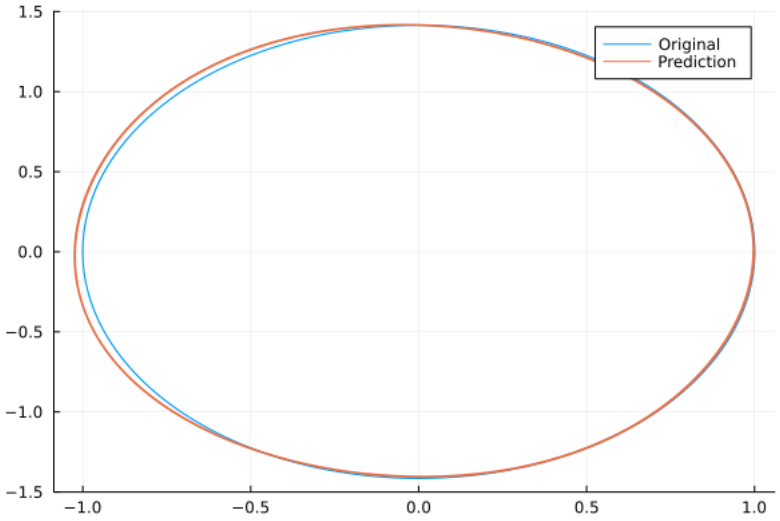


Figure 1.9: Neural network prediction and origin of undamped damped harmonic oscillator

1.8.2 Isothermal Damped Harmonic Oscillator

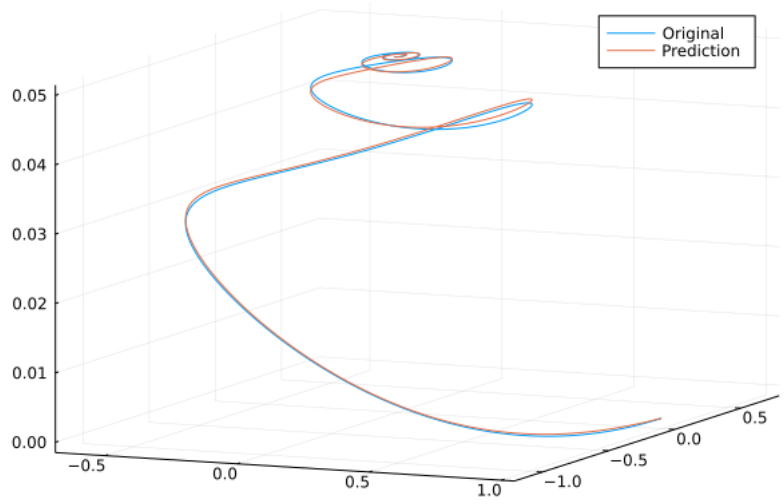


Figure 1.10: Neural network prediction and origin of isothermal damped harmonic oscillator

1.8.3 Nonisothermal Damped Harmonic Oscillator

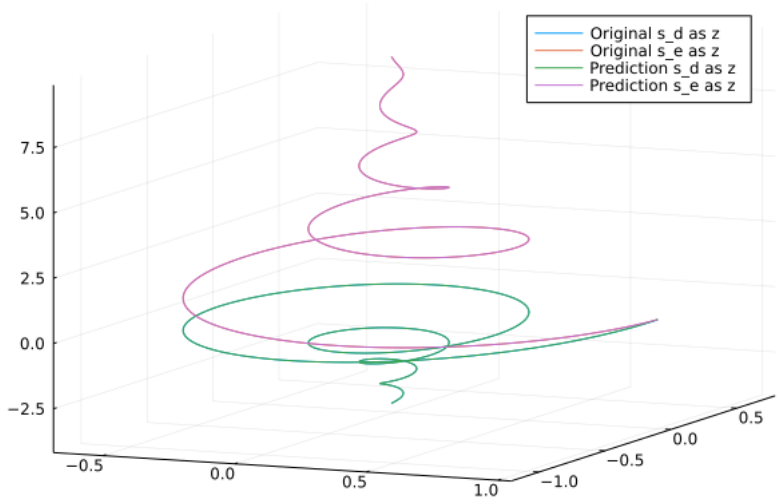


Figure 1.11: Neural network prediction and origin of nonisothermal damped harmonic oscillator

1.8.4 Summary of the section

During the model training, a package called BenchmarkTools is used to print the elapsed time.

Table 1.1: Elapsed time for models training.

System	Elapsed time
Undamped Harmonic Oscillator	359.386s
Isothermal Damped Harmonic Oscillator	533.470s
Nonisothermal Damped Harmonic Oscillator	826.878s

1.9 Structured Neural ODE

In the preceding section, the baseline model learned the state of the system $(\dot{q}, \dot{p})$. The whole of the RHS was seen as a neural network. However, this method requires batches of training data and very long training time. Moreover, the baseline model lacks interpretability since it is a pure black box model. Humans cannot intuitively understand how machine obtains these models. With these considerations some works use hamiltonian-based models rather than black box models.
Hamiltonian-based model also known as Hamiltonian Neural Networks(HNNs)[Sam]. The main purpose of HNNs is to endow neural networks with physics priors. In fact, if the system is not conserve, it has another upper-level name called physics-informed neural network[Maz], which is a method to solve ODEs with neural networks, regardless of energy conservation.
In this section, the learning object is the hamiltonian. To compute the loss function, the key is to compare the truth and the model's estimation of the gradient of the neural network $(\frac{\partial H}{\partial p}, \frac{\partial H}{\partial q})$.

1.9.1 physical priors

1.10 Parameters estimation

It is assumed that the ODEs of the system are completely known, except for some scalar parameters. In this case, the gradient descent algorithms can be applied in parameters estimation.

1.10.1 Undamped Harmonic Oscillator

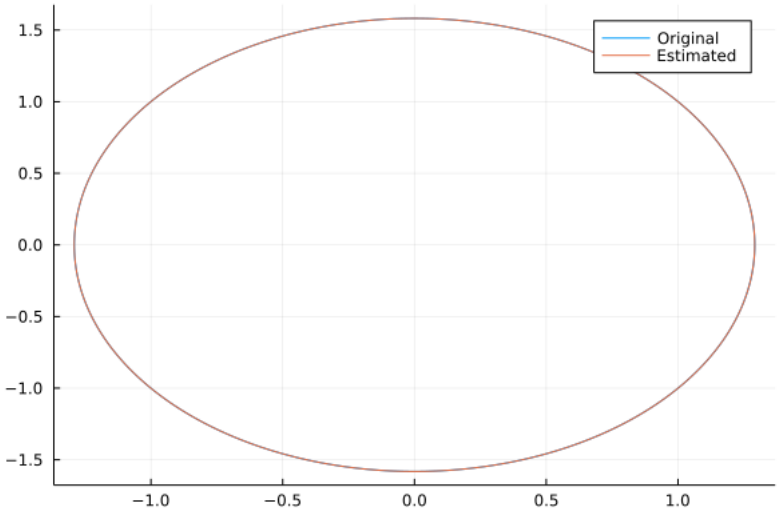


Figure 1.12: Parameters estimation result of undamped damped harmonic oscillator

1.10.2 Isothermal Damped Harmonic Oscillator

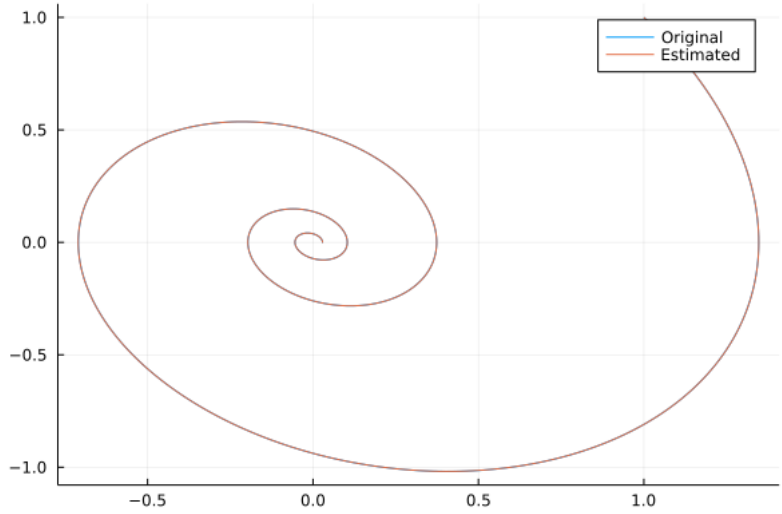


Figure 1.13: Parameters estimation result of isothermal damped harmonic oscillator

1.10.3 Nonisothermal Damped Harmonic Oscillator

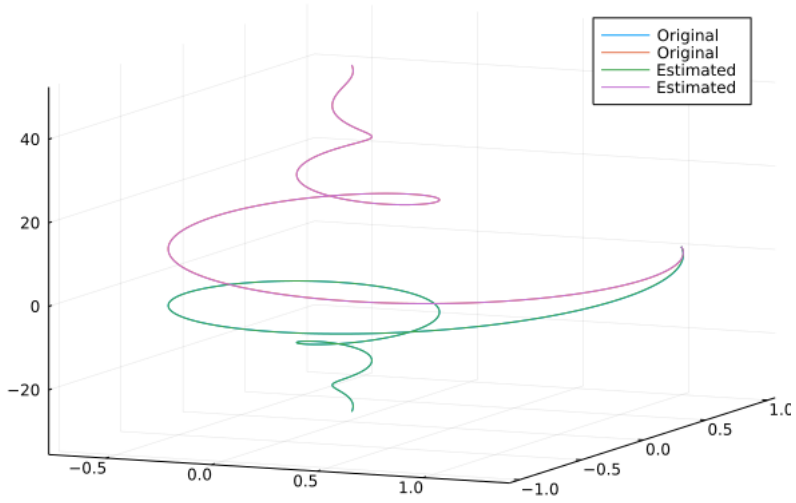


Figure 1.14: Parameters estimation result of nonisothermal damped harmonic oscillator

1.10.4 Summary of the section

Table 1.2: Result of parameters estimation.

System 1	**Undamped Harmonic Oscillator**
Ground truth	[1.5, 1.0]
Estimated parameters	[1.5003, 0.99971]
Error	0.0232%
System 2	**Isothermal Damped Harmonic Oscillator**
Ground truth	[1.0, 0.4, 1.0, 1.0]
Estimated parameters	[1.0035, 0.40059, 0.99973, 0.99689]
Error	0.264%
System 3	**Nonisothermal Damped Harmonic Oscillator**
Ground truth	[1.0, 0.4, 1.0, 2.0, 3.0, 5.0]
Estimated parameters	[1.004, 0.40149, 0.9967, 2.0003, 3.0003, 5.0008]
Error	0.0867%

Error in the table 1.2 is computed with the formula

$$Error = \sqrt{\frac{RSS}{\sum_{i=1}^{m} (y_i)^2}}, \tag{1.39}$$

where $RSS = \sum_{i=1}^{m} \left(y_i - \hat{f}(x_i) \right)^2$ is the residual sum of squares in statistics.

1.11 Compositional systems

elevator system(Vincent, Appendix A)

References

[Maz] Maziar Raissi, Paris Perdikaris, George Em Karniadakis. Physics informed deep learning (part i) data-driven solutions of nonlinear partial differential equations.

[Sam] Sam Greydanus, Misko Dzamba, Jason Yosinski. Hamiltonian neural networks.

includeappendix/code