Solving differential equations using the deep learning library DeepXDE

Undergraduate End of Studies Project

Tareq Ait Hsain

Supervisor: Pr. Noureddine Alaa

Faculty of Sciences and Techniques Gueliz - Marrakech

June 18th 2020

Outline

Outline

Outline

Outline

Existence and uniqueness theory

Definitions

Ordinary Differential Equation(ODE)

Definition

An ordinary differential equation (ODE) of order n is a functional relation of the form

$$F(t, y, y', ..., y^{(n-1)}, y^{(n)}) = 0$$

Where:

F is a function of t, y, and derivatives of y **and** $F \subset C(U)$, U being an open subset of $\mathbb{R} \times \mathbb{R}^{m(n+1)}$ with m being the size of y

Ordinary Differential Equation(ODE)

Definition

An ordinary differential equation (ODE) of order n is a functional relation of the form

$$F(t, y, y', ..., y^{(n-1)}, y^{(n)}) = 0$$

Where:

F is a function of t, y, and derivatives of y **and** $F \subset C(U)$, U being an open subset of $\mathbb{R} \times \mathbb{R}^{m(n+1)}$ with m being the size of y

Initial Value Problem

Hypothesis

- $\Omega \subseteq \mathbb{R}^n$ a domain
- $I \subseteq \mathbb{R}$ an open interval.
- $f : I \times \Omega \to \mathbb{R}^n$ a continuous function defined by: $(t, y) \longmapsto f(t, y)$ where $y = (y_1, ...y_n)$.
- $(t_0, y_0) \subseteq I \times \Omega$ be an arbitrary point.

Initial Value Problem

Definition (Initial Value Problem)

An **Initial Value Problem** (also known as **Cauchy Problem**) for a first order system of n ordinary differential equations is given by:

$$y' = f(t, y) \tag{1}$$

$$y(t_0) = y_0 \tag{2}$$

Solution to IVP

Definition (IVP Solution)

An n-tuple of functions $y = (y_1, \ldots, y_n) \in C^1(I_0)$ where $I_0 \subset I$ is an open interval containing the point $t_0 \in I$ is called a solution of the IVP above if for every $t \in I_0$, the (n+1)-tuple $(t, y_1(t), y_2(t), \ldots, y_n(t)) \in I \times \Omega$,

$$y'(t) = f(t, y(t)) \quad \forall t \in I_0 \tag{3}$$

$$y(t_0) = y_0 \tag{4}$$

Local existence and uniqueness

Lemma 1

Let y be a continuous function defined on an interval I_0 containing the point t_0. The following statements are equivalent

1. The function y is a solution of the IVP
2. The function y satisfies the integral equation

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))ds \quad \forall t \in I_0$$

Existence and uniqueness theorem

Also known as also known as Picard–Lindelöf theorem, Picard's existence theorem or Cauchy–Lipschitz theorem.

Existence and uniqueness theorem

Consider the initial value problem

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0.$$

Suppose f is locally Lipschitz continuous in y and continuous in t. Then for some value $\varepsilon > 0$, there exists a unique solution $y(t)$ to the initial value problem on the interval $[t_0 - \varepsilon, t_0 + \varepsilon]$

Proof outline

- We construct $T[y(t)] = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds$
- We establish that:
  1. It maps a complete non-empty metric space into itself
  2. It is a contraction mapping.
- We apply the Banach fixed point theorem to conclude the existence of a unique solution.

Proof outline

- We construct $T[y(t)] = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds$
- We establish that:
  1. It maps a complete non-empty metric space into itself
  2. It is a contraction mapping.
- We apply the Banach fixed point theorem to conclude the existence of a unique solution.

Proof outline

- We construct $T[y(t)] = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds$
- We establish that:
  1. It maps a complete non-empty metric space into itself
  2. It is a contraction mapping.
- We apply the Banach fixed point theorem to conclude the existence of a unique solution.

Proof outline

- We construct $T[y(t)] = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds$
- We establish that:
  1. It maps a complete non-empty metric space into itself
  2. It is a contraction mapping.
- We apply the Banach fixed point theorem to conclude the existence of a unique solution.

Global existence

Let's consider the initial value problem

$$Y'(t) = f(t, Y(t)), \quad Y(0) = Y_0 \qquad IVP(1)$$

Definition

A solution Y of $IVP(1)$ is maximal if it cannot be extended to a larger time interval.

Let's consider the initial value problem

$$Y'(t) = f(t, Y(t)), \quad Y(0) = Y_0 \qquad IVP(1)$$

Definition

A solution Y of $IVP(1)$ is maximal if it cannot be extended to a larger time interval.

Theorem (Explosion alternative - Blow-up in finite time)

Let I be an open interval and $f : I \times \mathbb{R} \to \mathbb{R}^d$ a class C^1 function. Let α and β be in $\overline{\mathbb{R}}$ and $Y :]\alpha, \beta[\to \mathbb{R}$ a maximal solution of $Y'(t) = f(t, Y(t))$.

1. *If $\beta < \sup I$ then $\|Y(t)\| \to +\infty$ when $t \to \beta$.*
2. *If $\alpha > \inf I$ then $\|Y(t)\| \to +\infty$ when $t \to \alpha$.*

($\inf I$, $\sup I \in \overline{\mathbb{R}}$).

Corollary (Non-explosion criterion)

Let $\Omega = I \times \mathbb{R}^d$. Let Y be a maximal solution of IVP(1). Let $g : I \to \mathbb{R}$ be continuous. Let $t_0 \in [\alpha, \beta]$.
1) If $\|Y(t)\| \leq g(t)$ on $[t_0, \beta[$, then $\beta = \sup I$.
2) If $\|Y(t)\| \leq g(t)$ on $]\alpha, t_0]$, then $\alpha = \inf I$.
3) If $\|Y(t)\| \leq g(t)$ on $[\alpha, \beta]$, then Y is a global solution.

Invariant set of Cauchy-Problem

Definition (Invariant set)

A convex set $\mathcal{C}$ is invariant for the Cauchy problem $IVP(1)$ if for each $Y_0 \in \mathcal{C}$ the solution Y of $IVP(1)$ satisfies $Y(t) \in \mathcal{C}$ for all t $\in [0, T(Y_0)[$.

- $T(Y_0) =$ maximum time of existence of the solution with initial data Y_0

Proposition

A convex set $\mathcal{C}$ is invariant for $IVP(1)$ if :

$$\forall y \in \partial\mathcal{C}, p \in n(y) \quad < p, f(y) > \leq 0$$

where $n(y)$ is normal cone at $\mathcal{C}$ in y.

Invariant set of Cauchy-Problem

Definition (Invariant set)

A convex set $\mathcal{C}$ is invariant for the Cauchy problem $IVP(1)$ if for each $Y_0 \in \mathcal{C}$ the solution Y of $IVP(1)$ satisfies $Y(t) \in \mathcal{C}$ for all t $\in [0, T(Y_0)[$.

- $T(Y_0) =$ maximum time of existence of the solution with initial data Y_0

Proposition

A convex set $\mathcal{C}$ is invariant for $IVP(1)$ if :

$$\forall y \in \partial\mathcal{C}, p \in n(y) \quad < p, f(y) > \leq 0$$

where $n(y)$ is normal cone at $\mathcal{C}$ in y.

Theorem 3

Let $f : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ locally Lipscitzian. Let $\mathcal{C}$ be a convex and closed set of $\mathbb{R}^d$. Then we have the following equivalences:

1) $\forall y \in \partial\mathcal{C} \quad \forall p \in n(y) \quad < p, f(u_0) > \leq 0$

2) $\forall u_0 \in \mathcal{C} \quad \forall T > 0$ and there exists a solution u of

$$\begin{cases} u \in C^1([0, T], \mathbb{R}^d) \\ \dfrac{du}{dt}(t) = f(u(t)) \quad \forall t \in [0, T] \\ u(0) = u_0 \\ u(t) \in \mathcal{C} \quad t \in [0, T]. \end{cases}$$

Example to highlight the global existence

- Consider the system

$$\begin{cases} \dfrac{du}{dt}(t) = v^2 - u^2 \\ \dfrac{dv}{dt}(t) = u^2 - v^2 \\ u(0) = u_0, v(0) = v_0 \end{cases}$$

- We set up a convex set $\mathcal{C} = [0, M] \times [0, M]$ with $M > 0$
- $f(u, v) = (v^2 - u^2, u^2 - v^2)$ satisfies the local Lipschitz-continuity, we therefore have local existence.
- We establish global existence by verifying the non-explosion criterion:

Example to highlight the global existence

- Consider the system

$$\begin{cases} \dfrac{du}{dt}(t) = v^2 - u^2 \\ \dfrac{dv}{dt}(t) = u^2 - v^2 \\ u(0) = u_0, v(0) = v_0 \end{cases}$$

- We set up a convex set $\mathcal{C} = [0, M] \times [0, M]$ with $M > 0$
- $f(u, v) = (v^2 - u^2, u^2 - v^2)$ satisfies the local Lipschitz-continuity, we therefore have local existence.
- We establish global existence by verifying the non-explosion criterion:

Example to highlight the global existence

- Consider the system

$$\begin{cases} \dfrac{du}{dt}(t) = v^2 - u^2 \\ \dfrac{dv}{dt}(t) = u^2 - v^2 \\ u(0) = u_0, v(0) = v_0 \end{cases}$$

- We set up a convex set $\mathcal{C} = [0, M] \times [0, M]$ with $M > 0$
- $f(u, v) = (v^2 - u^2, u^2 - v^2)$ satisfies the local Lipschitz-continuity, we therefore have local existence.
- We establish global existence by verifying the non-explosion criterion:

Example to highlight the global existence

- Consider the system

$$\begin{cases} \dfrac{du}{dt}(t) = v^2 - u^2 \\ \dfrac{dv}{dt}(t) = u^2 - v^2 \\ u(0) = u_0, v(0) = v_0 \end{cases}$$

- We set up a convex set $\mathcal{C} = [0, M] \times [0, M]$ with $M > 0$
- $f(u, v) = (v^2 - u^2, u^2 - v^2)$ satisfies the local Lipschitz-continuity, we therefore have local existence.
- We establish global existence by verifying the non-explosion criterion:

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Example to highlight global existence

- $\mathcal{C}$ is invariant
  - We take the first border where we have y with the components u and $v = 0$
  - $p = (-1, 0) \in n(y) \Longrightarrow < p, f(u, v) >= -u^2 < 0$
  - Same process for other borders
- Using theorem 2 we get the existence of a solution
- We satisfy the non-explosion criterion, therefore we have the global existence

Solving DEs

Numerical methods

- Solving by quadrature (i.e. using elementary operations and primitivation) is only possible in a very limited number of cases.
- When a solution formula is available, it can involve integrals that can be calculated only by using a numerical quadrature formula.
- Most popular ones are Euler method (most basic) and Runge-Kutta 4 method (most effective)

Numerical methods

- Solving by quadrature (i.e. using elementary operations and primitivation) is only possible in a very limited number of cases.
- When a solution formula is available, it can involve integrals that can be calculated only by using a numerical quadrature formula.
- Most popular ones are Euler method (most basic) and Runge-Kutta 4 method (most effective)

Numerical methods

- Solving by quadrature (i.e. using elementary operations and primitivation) is only possible in a very limited number of cases.
- When a solution formula is available, it can involve integrals that can be calculated only by using a numerical quadrature formula.
- Most popular ones are Euler method (most basic) and Runge-Kutta 4 method (most effective)

Implementing Euler method in Python

```
def euler(f, a, y0, b, n):
        t, y = a, y0
        vt = [0] * (n + 1)
        vy = [0] * (n + 1)
        h = (b - a) / float(n)
        vt[0] = t = a
        vy[0] = y = y0
        for i in range(1, n + 1):
                t += h
                y += h * f(t, y)
                vt[i] = t
                vy[i] = y
        return vt, vy
vt, vye = euler(f, 0, 1, 10, 100)
```

Implementing RK4 method in Python

```
def rk4(f, x0, y0, x1, n):
        vx = [0] * (n + 1)
        vy = [0] * (n + 1)
        h = (x1 - x0) / float(n)
        vx[0] = x = x0
        vy[0] = y = y0
        for i in range(1, n + 1):
                k1 = h * f(x, y)
                k2 = h * f(x + 0.5 * h, y + 0.5 * k1)
                k3 = h * f(x + 0.5 * h, y + 0.5 * k2)
                k4 = h * f(x + h, y + k3)
                vx[i] = x = x0 + i * h
                vy[i] = y = y + (k1 + k2 + k2 + k3 + k3 +
                                                    k4) / 6
        return vx, vy
vx, vy = rk4(f, 0, 1, 10, 100)
```

Comparison

We use both methods to solve $y' + 2y = x^3 e^{-2x}, y(0) = 1$ for which the exact solution is $y(x) = \frac{1}{4} e^{-2x}(x^4 + 4)$,

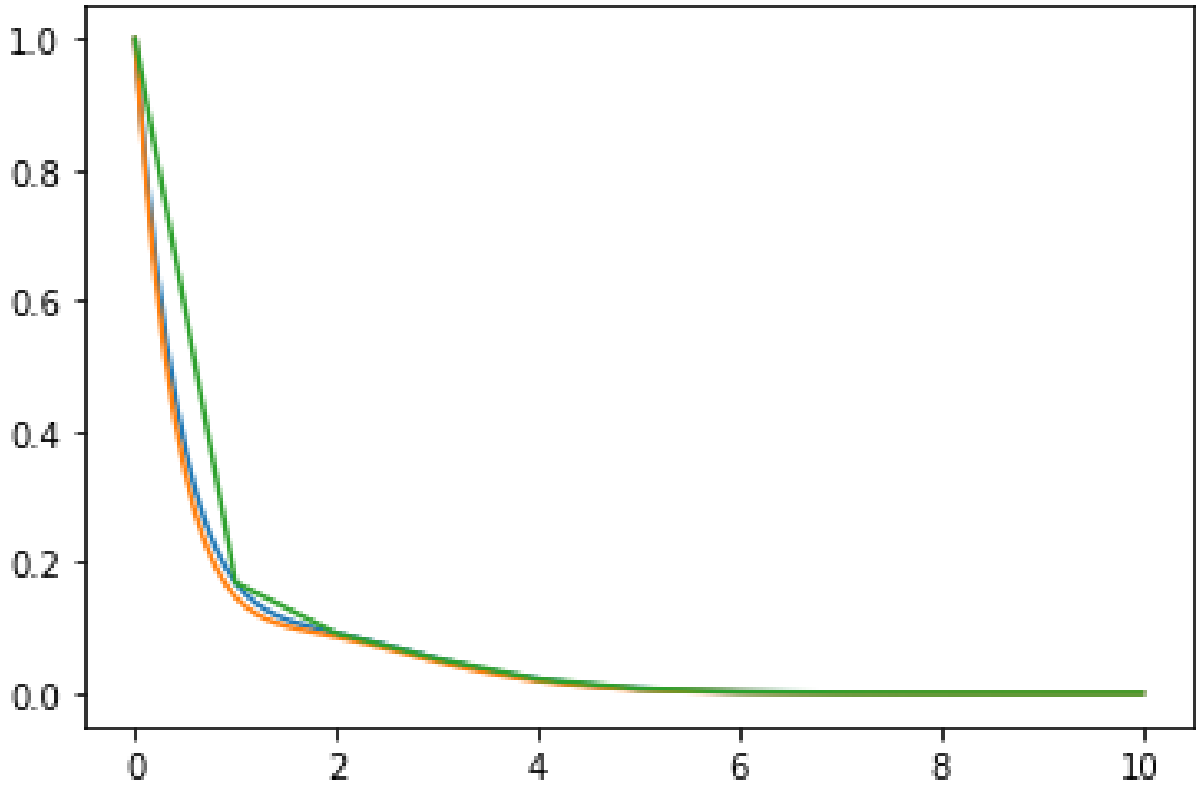


Figure: Comparison that shows accuracy of RK4 (blue), but usefulness of Euler (green) to describe variations of the exact solution (yellow)

Constraints for numerical methods

- Using step by step methods can turn out to be obsolete for large time domains
- Necessity to do additional work to make the problem suit the method's functionality
- Solving by Deep Learning absolves these constraints

Constraints for numerical methods

- Using step by step methods can turn out to be obsolete for large time domains
- Necessity to do additional work to make the problem suit the method's functionality
- Solving by Deep Learning absolves these constraints

Constraints for numerical methods

- Using step by step methods can turn out to be obsolete for large time domains
- Necessity to do additional work to make the problem suit the method's functionality
- Solving by Deep Learning absolves these constraints

Constraints for numerical methods

- Using step by step methods can turn out to be obsolete for large time domains
- Necessity to do additional work to make the problem suit the method's functionality
- Solving by Deep Learning absolves these constraints

Neural Networks

Artificial neuron

- Also known as units, nodes or cells, are simple processing elements.
- Connected through communication links associated to weights.
- Receive input, combine the input with their internal state using an activation function, and produce output using an output function.

Artificial neuron

- Also known as units, nodes or cells, are simple processing elements.
- Connected through communication links associated to weights.
- Receive input, combine the input with their internal state using an activation function, and produce output using an output function.

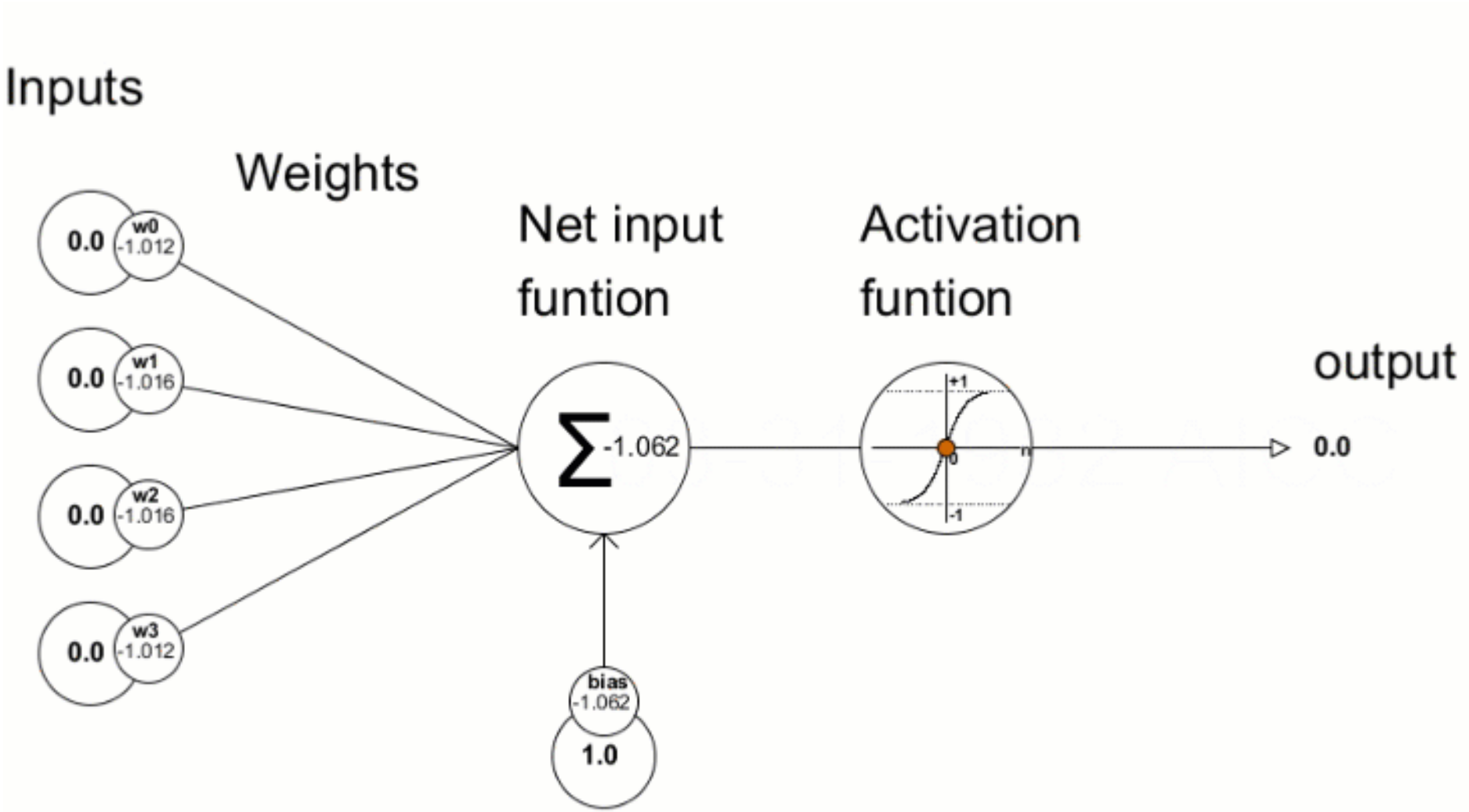
Artificial neuron

- Also known as units, nodes or cells, are simple processing elements.
- Connected through communication links associated to weights.
- Receive input, combine the input with their internal state using an activation function, and produce output using an output function.

Artificial neuron

- Also known as units, nodes or cells, are simple processing elements.
- Connected through communication links associated to weights.
- Receive input, combine the input with their internal state using an activation function, and produce output using an output function.

Artificial Neuron



Multilayered Neural Network

- The neurons are organised in the form of layers
- At least two layers: an input and an output layer
- The layers between the input and the output layer (if any) are called hidden layers

Multilayered Neural Network

- The neurons are organised in the form of layers
- At least two layers: an input and an output layer
- The layers between the input and the output layer (if any) are called hidden layers

Multilayered Neural Network

- The neurons are organised in the form of layers
- At least two layers: an input and an output layer
- The layers between the input and the output layer (if any) are called hidden layers

Multilayered Neural Network

- The neurons are organised in the form of layers
- At least two layers: an input and an output layer
- The layers between the input and the output layer (if any) are called hidden layers

Multilayered Neural Network

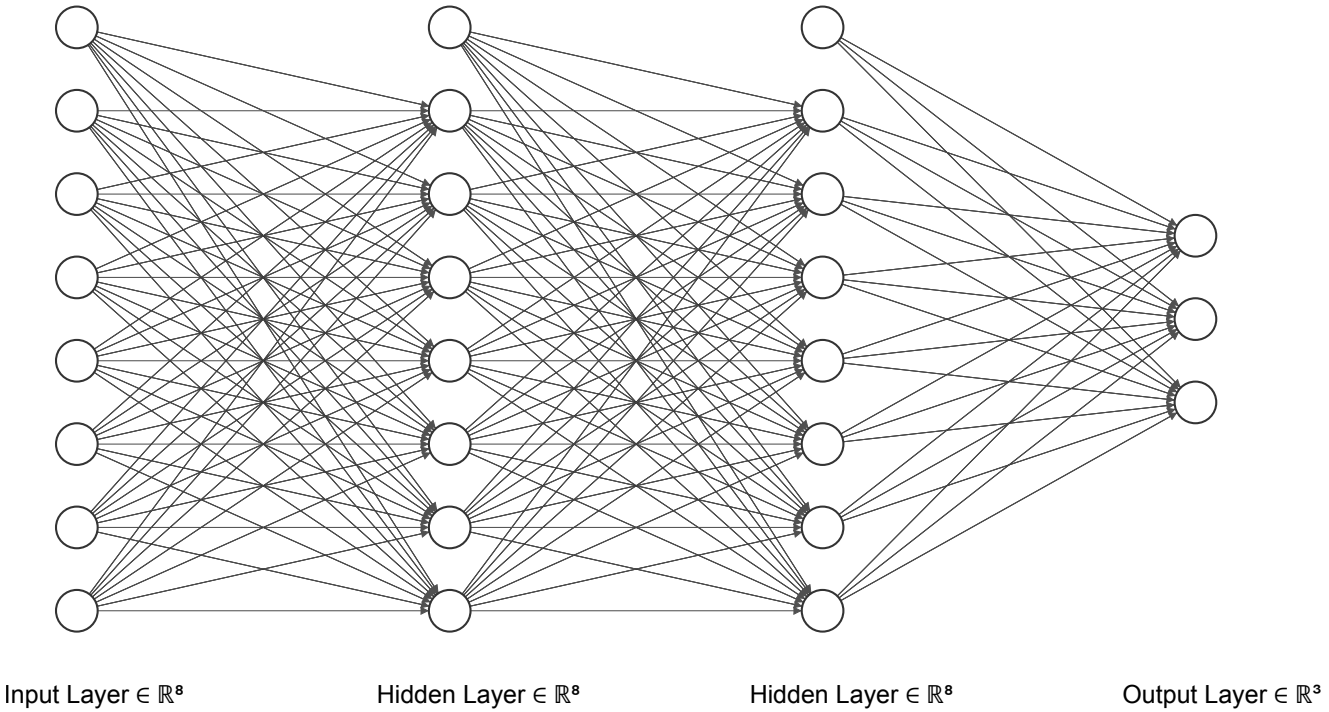


Figure: Feedforward Neural Network

Learning process

Training FNNs

FNNs used to approximate functions for scientific computing use 2 techniques:

- Backpropagation
- Gradient descent optimization

Training FNNs

FNNs used to approximate functions for scientific computing use 2 techniques:

- Backpropagation
- Gradient descent optimization

Training FNNs

FNNs used to approximate functions for scientific computing use 2 techniques:

- Backpropagation
- Gradient descent optimization

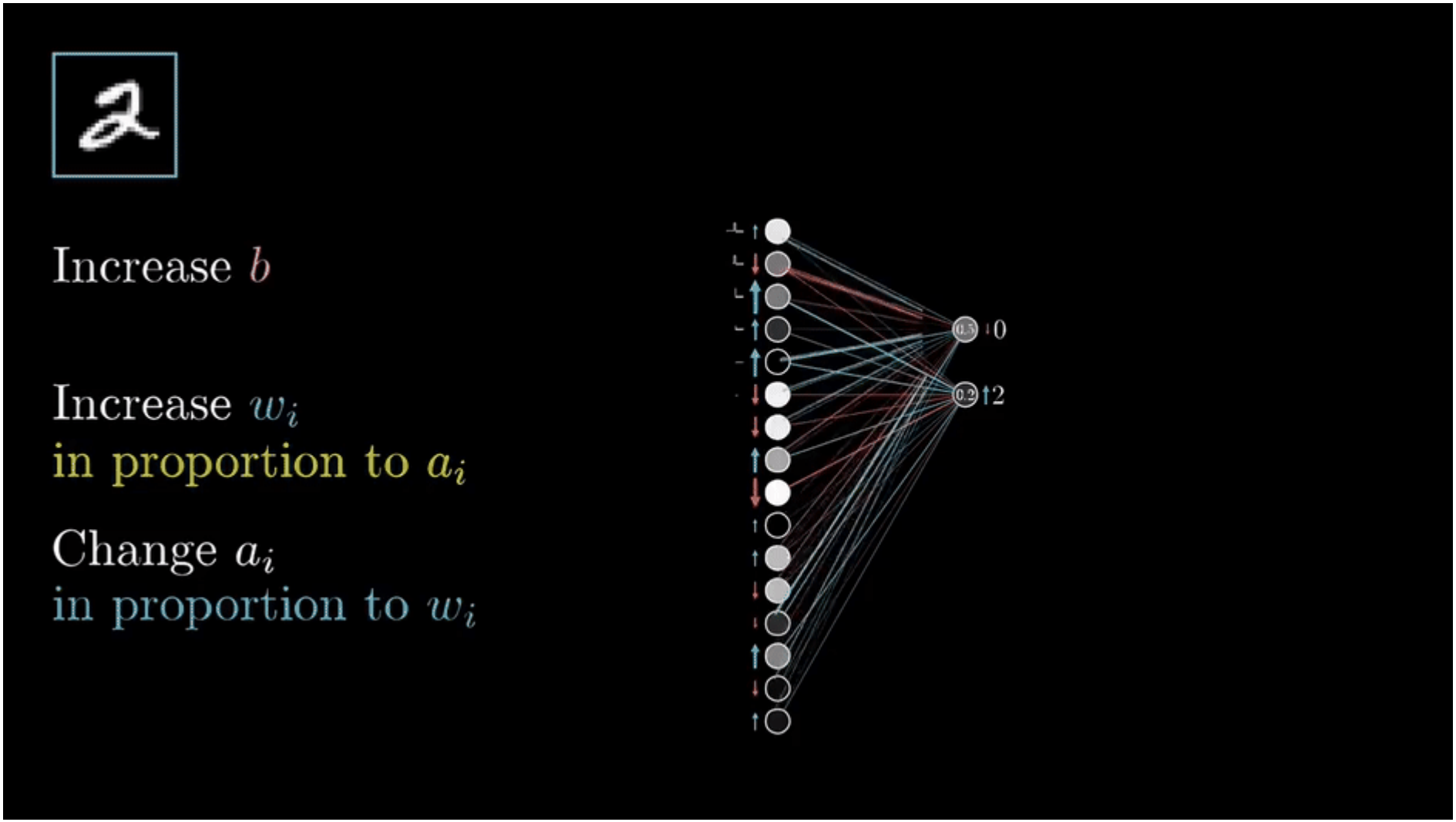
Backpropagation

Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network.

Backpropagation

Using this information, the algorithm adjusts (read : optimizes) the weights of each connection in order to reduce the value of the error function by some small amount

Backpropagation



Backpropagation

• After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small.

• In this case, one would say that the network has learned a certain target function.

Backpropagation

• After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small.
• In this case, one would say that the network has learned a certain target function.

Optimization

To adjust weights properly, one applies a general method for non-linear optimization that is called gradient descent.

Optimization

For this, the network calculates the derivative of the error function with respect to the network weights, and changes the weights such that the error decreases (thus going downhill on the surface of the error function).

DeepXDE

Physics-Informed Neural Network

DeepXDE is an application of Physics-Informed Neural Networks

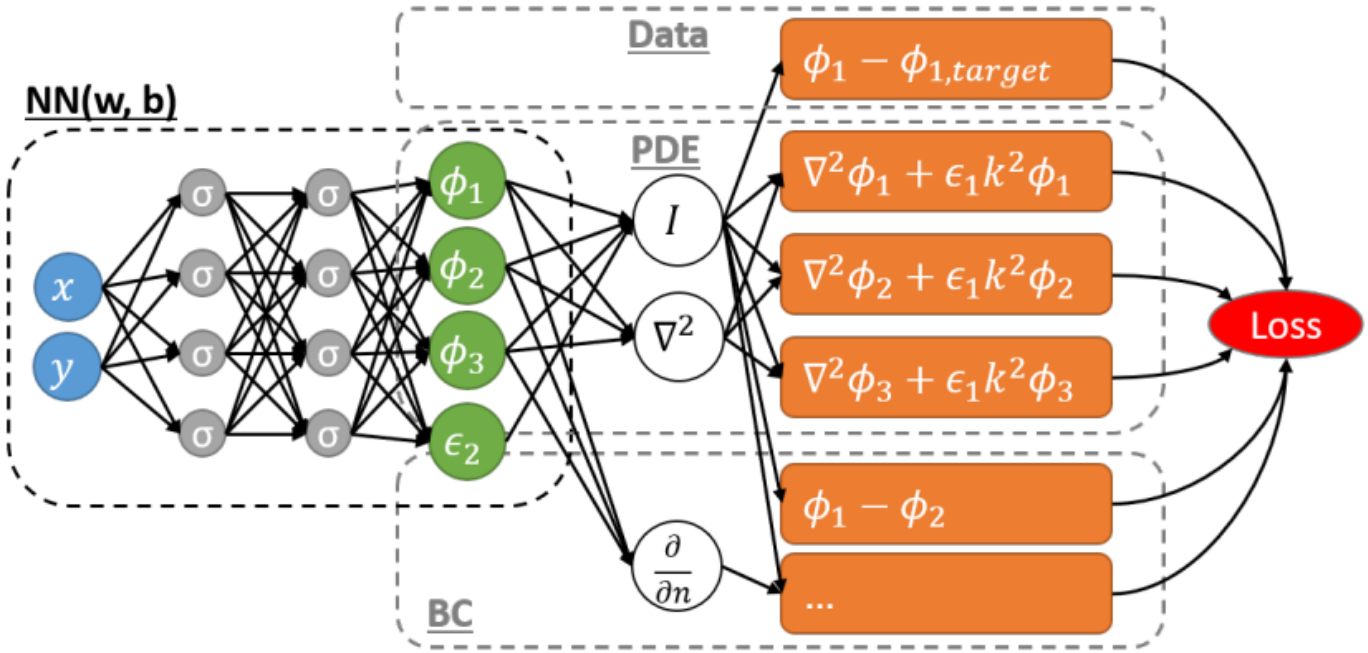


Figure:

Usage

Solving differential equations in DeepXDE is no more than specifying the problem using the built-in modules

Flowchart

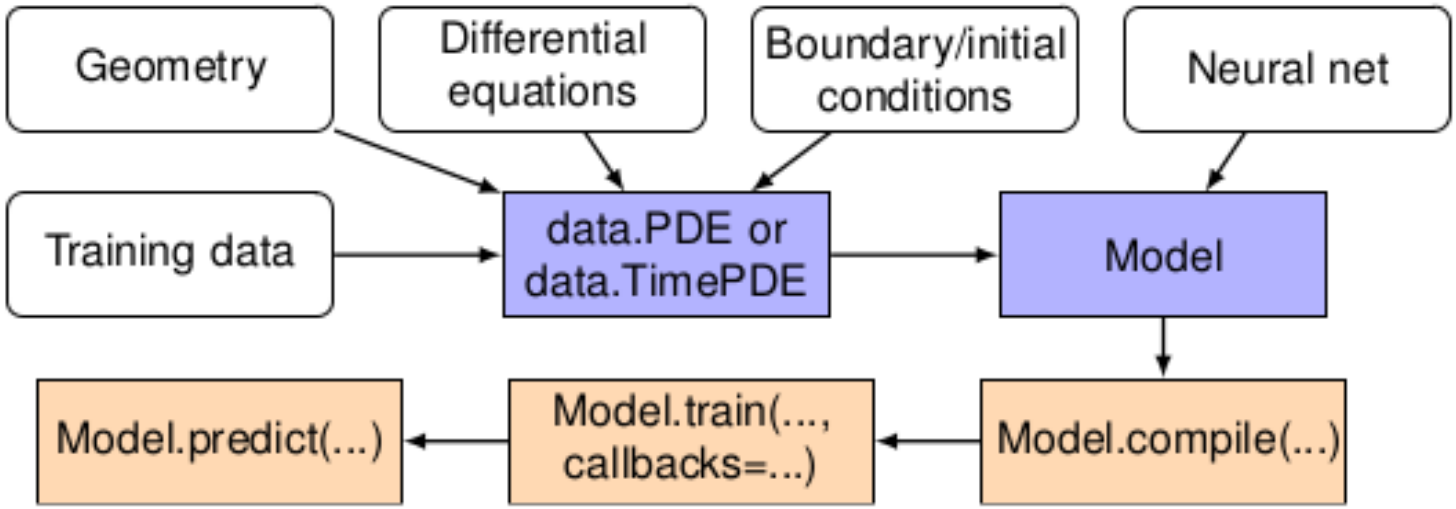


Figure: DeepXDE usage flowchart

Application example

Problem

We consider the following non-linear problem:

$$\begin{cases} -\dfrac{d^2y}{dt^2} = 1 - y^3 \\ y(0) = y(1) = 0 \end{cases}$$

System

Defining the system, in its implicit form.

```
def equation(t, y):
    dy_t = tf.gradients(y, t)[0]
    dy_tt = tf.gradients(dy_t,t)[0]
    return dy_tt + 1 - y**3
```

Geometry

Defining the interval

```
geom = dde.geometry.Interval(0, 1)
```

Boundary conditions

Defining $u(0) = 0$

```
def boundary_l(t, on_boundary):
        return on_boundary and np.isclose(t[0], 0)
bc1 = dde.DirichletBC(geom, lambda t:
            np.full_like(t,0), boundary_l)
```

`np.full_like(t,0)` $\equiv$ array of $0 \equiv 0$

Boundary conditions

Same for $u(1) = 0$,

```
def boundary_r(t, on_boundary):
        return on_boundary and np.isclose(t[0], 1)
bc2 = dde.DirichletBC(geom, lambda t:
np.full_like(t,0), boundary_r)
```

Data module

It compiles the data of the problem and the number of test points

```
data = dde.data.PDE(geom, equation, [bc1, bc2],
     50, 2 , num_test=100)
```

Neural Net and hyperparameters

```
layer_size = [1] + [50] * 3 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)
```

Training the model

```
model = dde.Model(data, net)
model.compile("adam", lr=0.001)
losshistory, train_state = model.train(epochs=20000)
dde.saveplot(losshistory, train_state,
                    issave=True, isplot=True)
```

Result

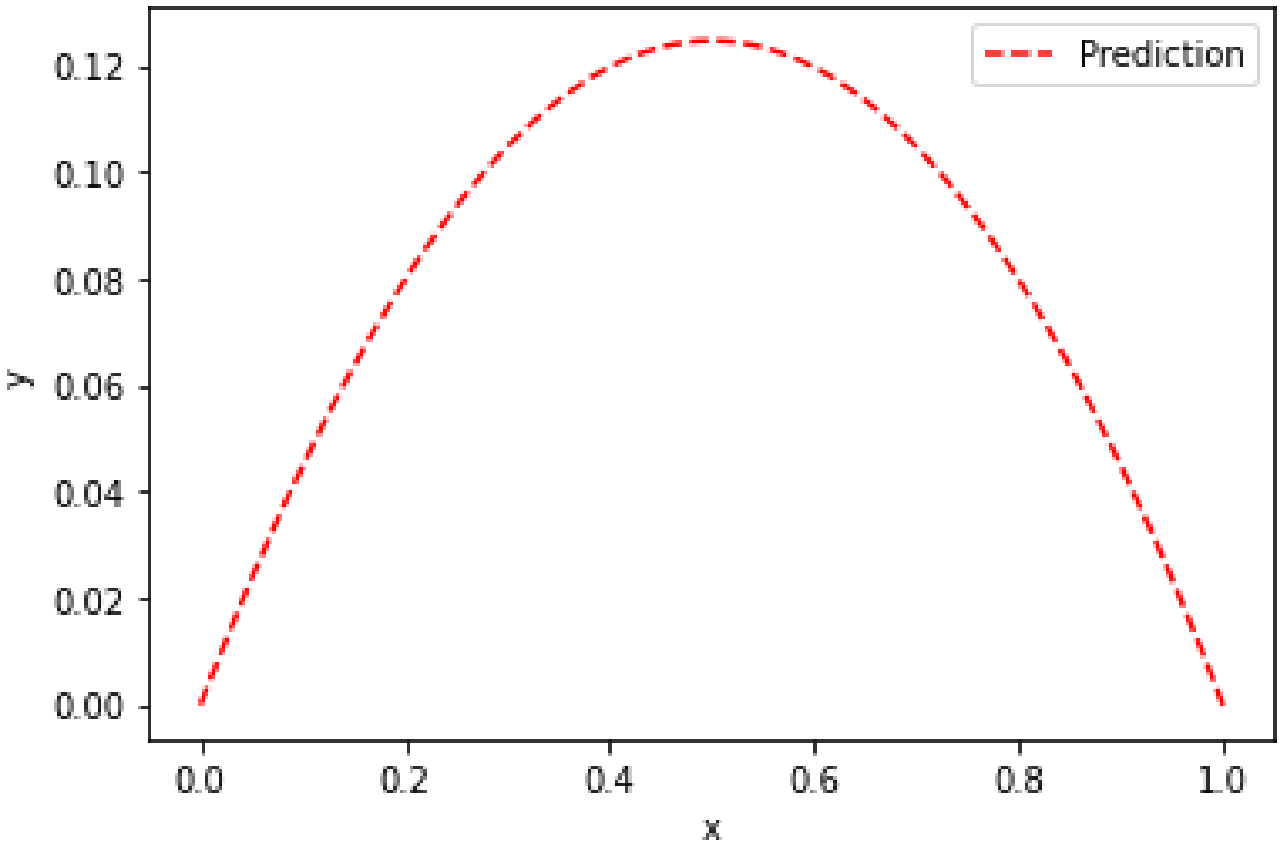


Figure: Predicted solution

Covid-19 model

Reservoir-People Model

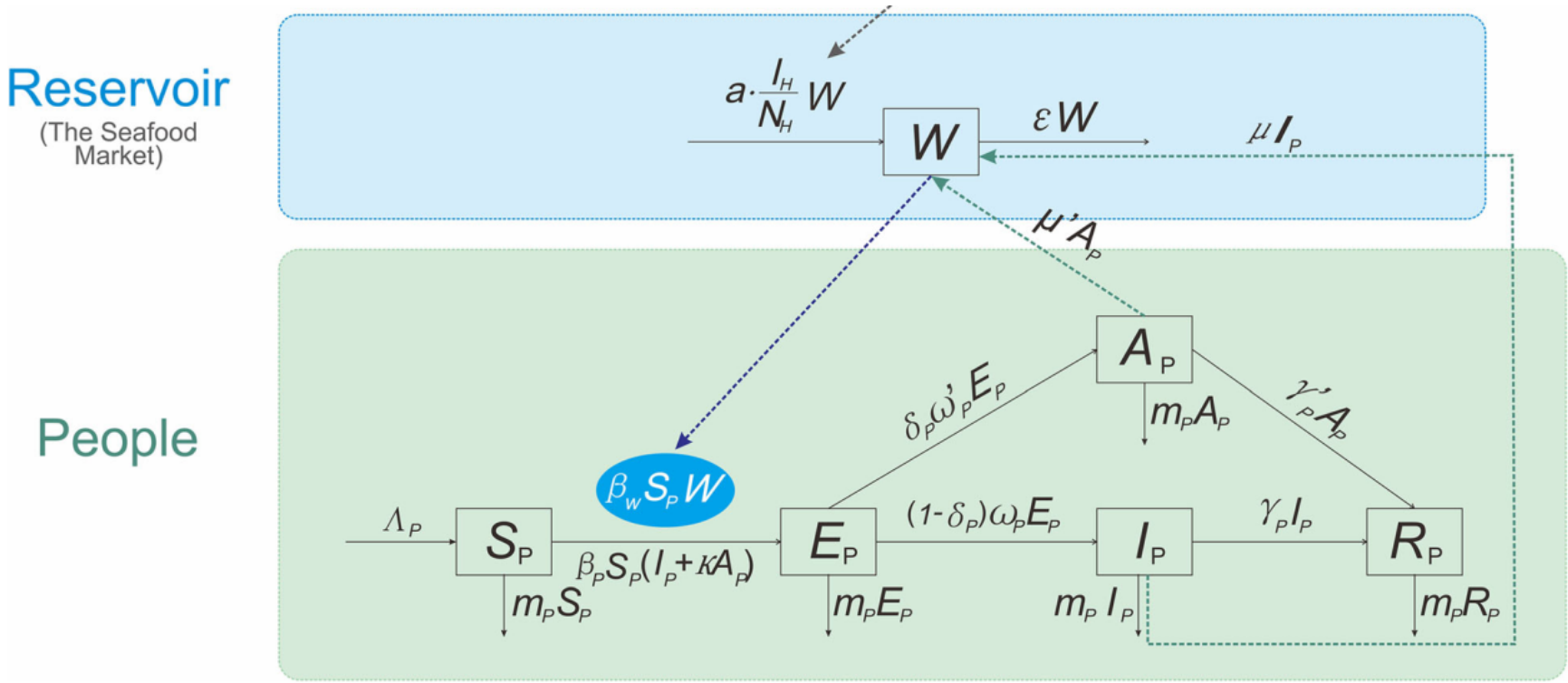


Figure: Flowchart of Reservoir-People model[1]

[1] Chen et al. (2020). "A mathematical model for simulating the phase-based transmissibility of a novel coronavirus". In: Infectious diseases of poverty.

Reservoir-People model

$$\textbf{(RP)}: \begin{cases} \frac{ds_p}{dt} = n_p - m_p s_p - b_p s_p (i_p + \kappa a_p) - b_W s_p w \\ \frac{de_p}{dt} = b_p s_p (i_p + \kappa a_p) + b_W s_p w - (1-\delta_p)\omega_p e_p - \delta_p \omega'_p e_p - m_p e_p \\ \frac{di_p}{dt} = (1-\delta_p)\omega_p e_p - (\gamma_p + m_p) i_p \\ \frac{da_p}{dt} = \delta_p \omega'_p e_p - (\gamma'_p + m_p) a_p \\ \frac{dr_p}{dt} = \gamma_p i_p + \gamma'_p a_p - m_p r_p \\ \frac{dw}{dt} = \epsilon (i_p + c a_p - w) \end{cases}$$

Defining the system

```
def ode_system(t, y):
sp, ep, ip, ap, rp, w = y[:, 0:1],
y[:, 1:2], y[:, 2:3], y[:, 3:4],
y[:, 4:5], y[:, 5:]
dsp_t = tf.gradients(sp, t)[0]
dep_t = tf.gradients(ep, t)[0]
dip_t = tf.gradients(ip, t)[0]
dap_t = tf.gradients(ap, t)[0]
drp_t = tf.gradients(rp, t)[0]
dw_t = tf.gradients(w, t)[0]
```

Defining the system

```
return [
dsp_t - n + mp * sp + bp * sp * ( ip + kappa * ap )
              + bw * sp * w,
dep_t - bp * sp * ( ip + kappa * ap ) - bw * sp * w
               + ( 1 - deltap )
* omegap * ep + deltap * omegap * ep + mp * ep,
dip_t - ( 1 - deltap ) * omegap * ep +
              ( gammap + mp ) * ip,
dap_t - deltap * omegap * ep + ( gammap + mp ) * ap,
drp_t - gammap * ip - gammap * ap + mp * rp,
dw_t - eps * ( ip + c * ap - w )
]
```

Defining the geometry: [0,60] interval

```
geom = dde.geometry.TimeDomain(0, 60)
```

Defining the boundary conditions

```
def boundary(_, on_initial):
    return on_initial

ic1 = dde.IC(geom, lambda X: 0 * np.ones(X.shape),
                   boundary, component=0)
ic2 = dde.IC(geom, lambda X: 0 * np.ones(X.shape),
                   boundary, component=1)
ic3 = dde.IC(geom, lambda X: 0 * np.ones(X.shape),
                   boundary, component=2)
ic4 = dde.IC(geom, lambda X: 0 * np.ones(X.shape),
                   boundary, component=3)
ic5 = dde.IC(geom, lambda X: 0 * np.ones(X.shape),
                   boundary, component=4)
ic6 = dde.IC(geom, lambda X: 1 * np.ones(X.shape),
                   boundary, component=5)
```

Compiling the previous data

```
data = dde.data.PDE(
geom, ode_system, [ic1, ic2, ic3, ic4, ic5, ic6],
                                700, 12, num_test=250
)
```

Defining the neural network and hyperparameters

```
layer_size = [1] + [50] * 3 + [6]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)
```

Defining, compiling and training the model

```
model = dde.Model(data, net)
model.compile("adam", lr=0.001)
losshistory, train_state = model.train(epochs=40000)
```

Plotting the result and the loss history

```
dde.saveplot(losshistory, train_state,
                          issave=True, isplot=True)
```

Results

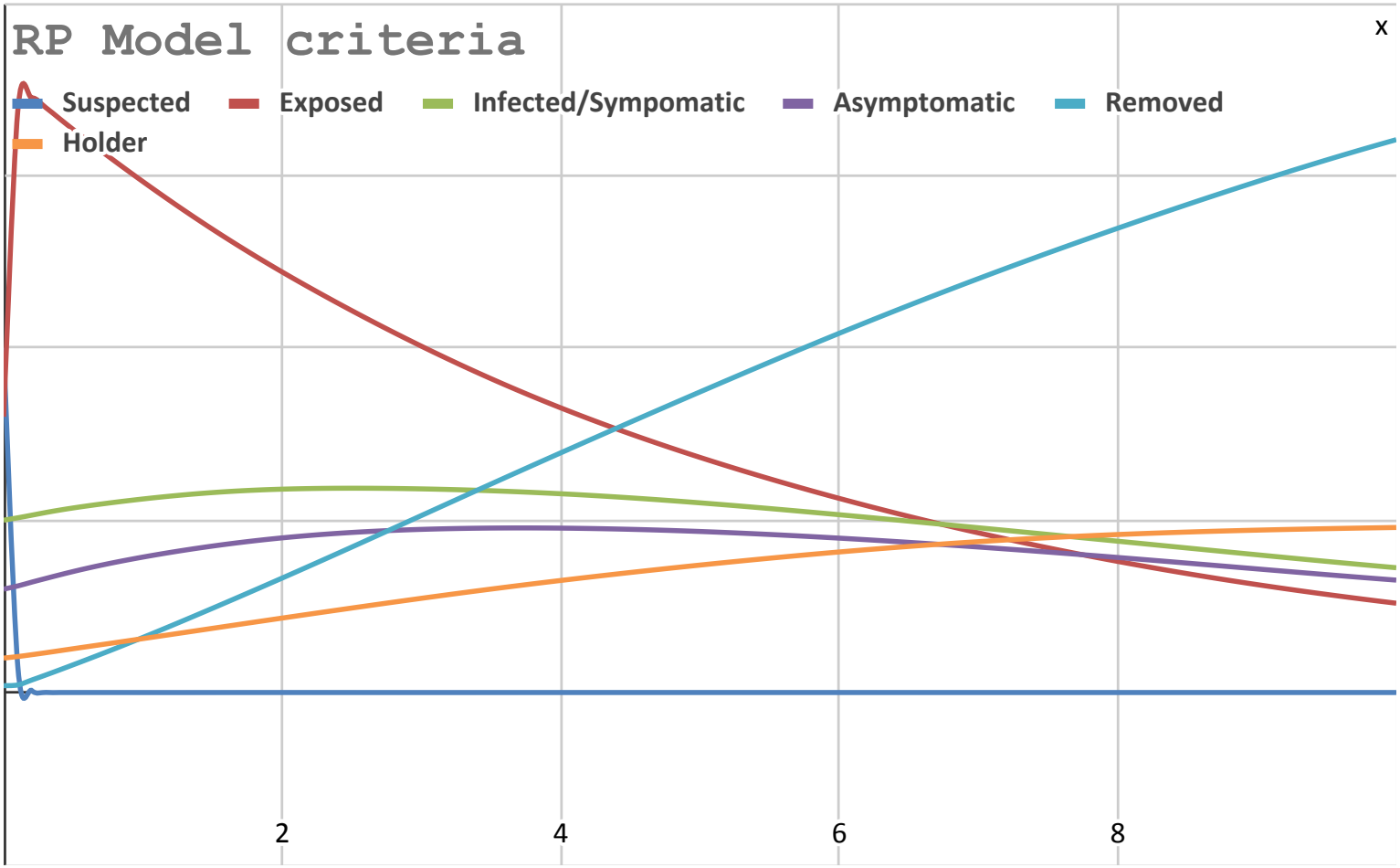


Figure: Result

Conclusion and perspectives

Conclusion

In this work we focus on solving differential equations, as its such an important and ever developing field of mathematics. We established the conditions for which we have local and global existence, and then we took interest in the scientific tool in our hand: DeepXDE.

Conclusion

The use of neural proved to be as simple as it is effective. It is basically a mesh-free method that can avoid the problems cause by noisy training data, which is a huge upside from classical and more established methods.

- A another asset is the immense customizability and diverse applications, and many other extensions for multi-physics and multi-scale problems are possible across different scientific disciplines by creatively designing the loss function and introducing suitable solution spaces.

Conclusion

The use of neural proved to be as simple as it is effective. It is basically a mesh-free method that can avoid the problems cause by noisy training data, which is a huge upside from classical and more established methods.

- A another asset is the immense customizability and diverse applications, and many other extensions for multi-physics and multi-scale problems are possible across different scientific disciplines by creatively designing the loss function and introducing suitable solution spaces.

Perspectives

We hope to further explore and improve the already available deep learning methods, as both an educational and a scientific tool.

THANK YOU