THE UNIVERSITY of TENNESSEE KNOXVILLE

# Lecture 13: Physics-Informed Neural Networks (PINNs)

Sergei V. Kalinin

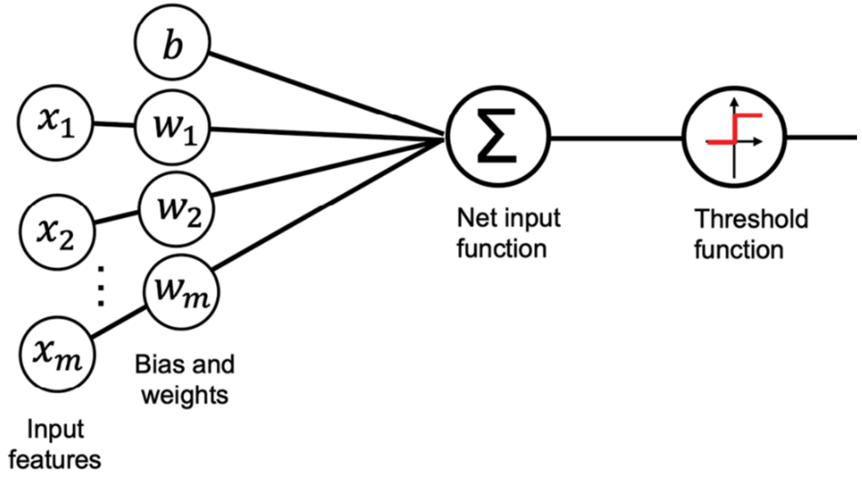
# Building Linear Neuron

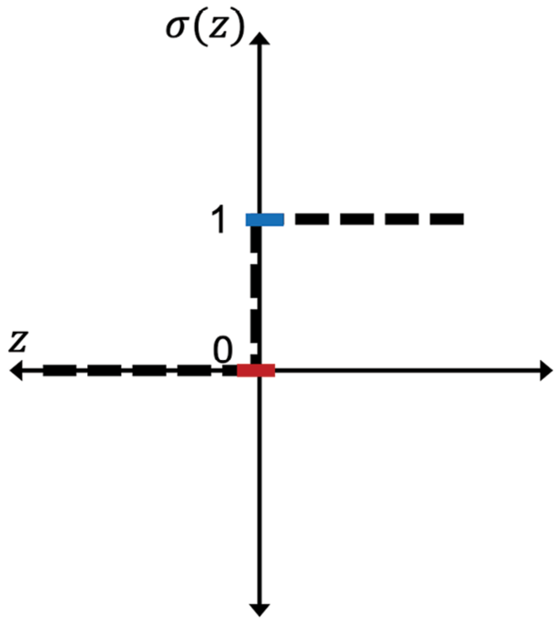
**Input:** $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$

**Weights:** $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

**Linear transform:** $z = \boldsymbol{w}_1\boldsymbol{x}_1 + \ldots + \boldsymbol{w}_m\boldsymbol{x}_m + \boldsymbol{b} = \boldsymbol{w}^T\boldsymbol{x} + \boldsymbol{b}$

**Output:** $\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$




where $z = \boldsymbol{w}^T\boldsymbol{x} + b$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

# Training Linear Neuron

- Initialize the weights and bias unit to 0 or small random numbers
- For each training example, ***x(i)***:
- Compute the output value, $\boldsymbol{y(i) = w^T x(i) + b}$
- Update the weights and bias unit: $w_j := w_j + \Delta w_j$ and $b := b + \Delta b$
- Where $\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$ and $\Delta b = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)$

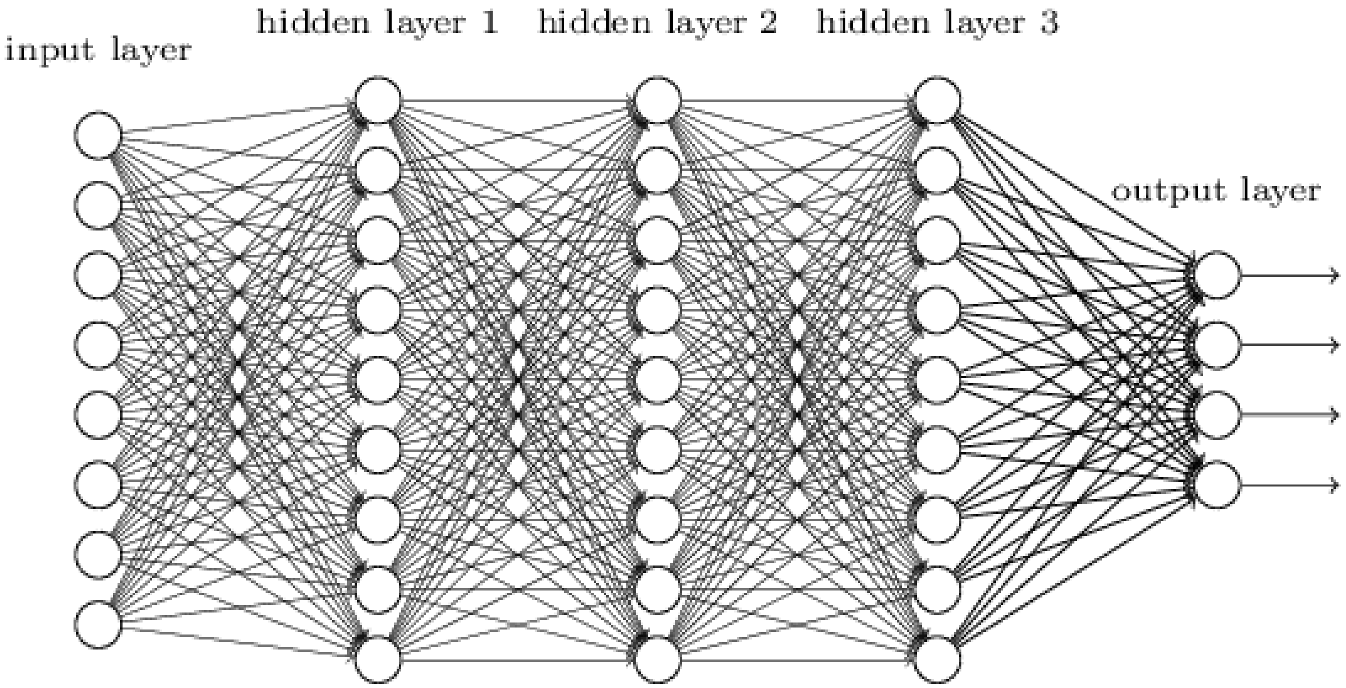
Each weight, $w_j$, corresponds to a feature, $x_j$, in the dataset,

$\eta$ is the **learning rate** (typically a constant between 0.0 and 1.0),

$y^{(i)}$ is the **true class label** of the *i*th training example,

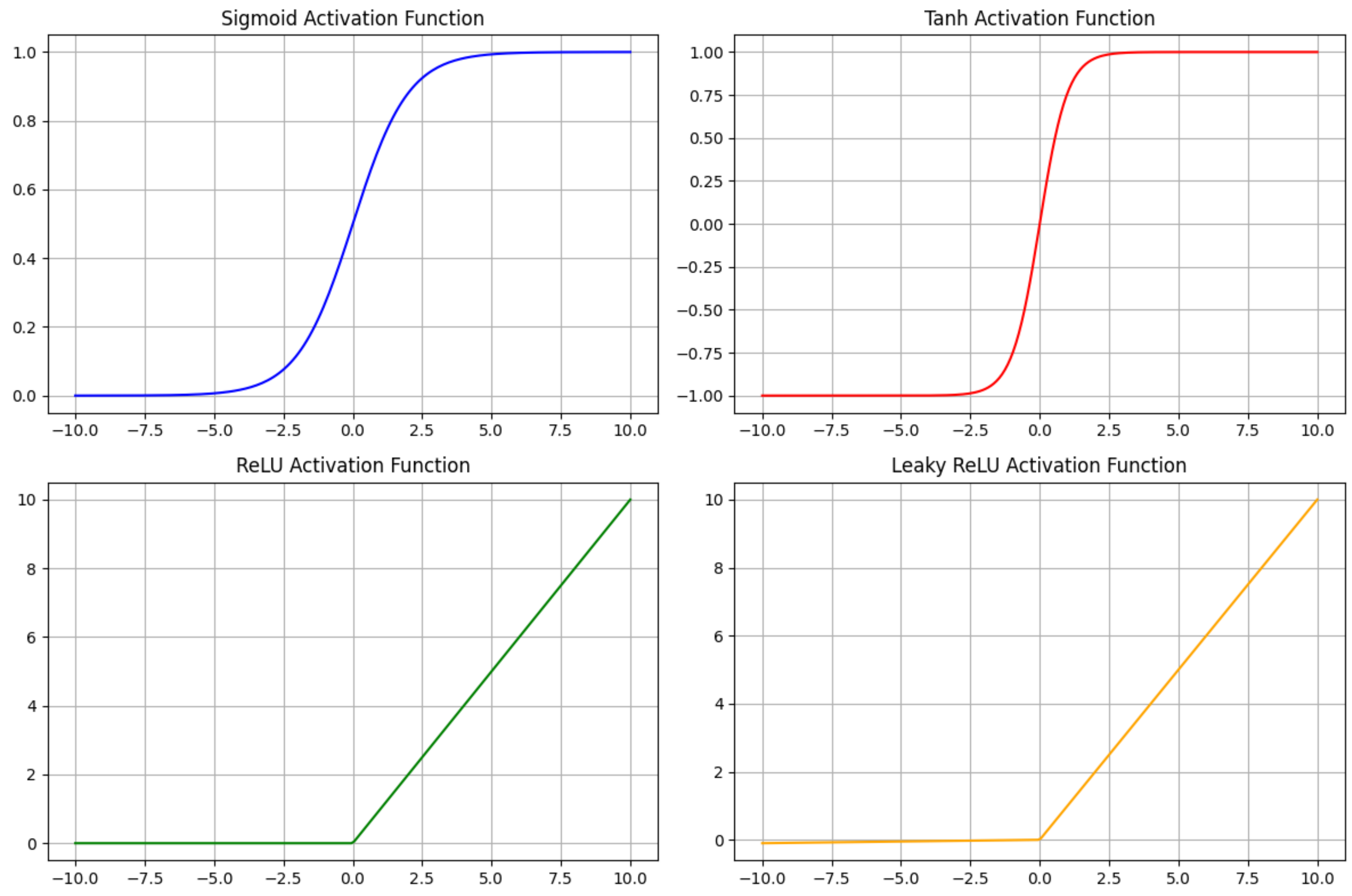
$\hat{y}^{(i)}$ is the **predicted class label**

# Putting Neurons Together




- Composed of multiple layers of artificial neurons.
- Each layer processes inputs received, applies a transformation (weights, biases, activation function), and passes the output to the next layer.
- Training a DNN involves adjusting weights and biases using backpropagation and a chosen optimization algorithm.
- The deep architecture enable the network to learn complex and abstract patterns in data.

# Activation functions



# Loss functions for supervised ML

A loss function, also known as a cost function, quantifies the difference between the predicted values and the actual target values. It guides the training of neural networks by providing a measure to minimize during optimization

**Mean Squared Error (MSE):** Used for regression problems.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Measures the average squared difference between actual and predicted values

**Cross-Entropy Loss:** Used for classification problems.

$$CE = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Measures the performance of a classification model whose output is a probability value between 0 and 1

- Loss functions provide the primary feedback signal for learning.
- The choice of loss function can significantly affect the model's performance and convergence

# Backpropagation

Backpropagation is a mechanism used to update the weights in a neural network efficiently, based on the error rate obtained in the previous epoch (i.e., iteration). It effectively distributes the error back through the network layers

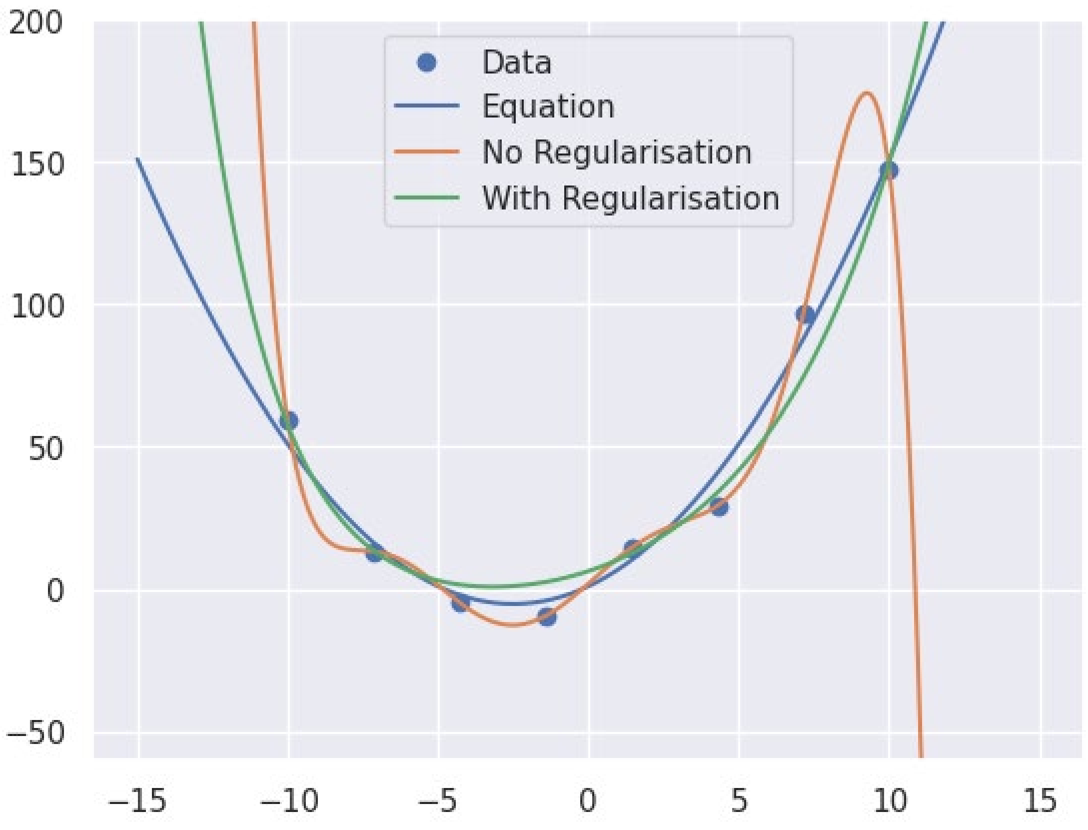
- **Forward Pass:** Calculating the predicted output, moving the input data through the network layers
- **Loss Function:** Determining the error by comparing the predicted output to the actual output
- **Backward Pass:** Computing the gradient of the loss function with respect to each weight by the chain rule
- **Weight Update:** Adjusting the weights of the network in a direction that minimally reduces the loss (gradient descent)

**Input Data → Forward Pass → Calculate Loss → Backward Pass → Update Weights**

https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

# Neural Network Based Regression

$$Loss_{reg} = \frac{1}{N}\sum_{i}^{N}(f(x_i|\theta) - y_i)^2 + \lambda||\theta||_2^2$$




https://medium.com/@theo.wolf/physics-informed-neural-networks-a-simple-tutorial-with-pytorch-f28a890b874a

# Physics-Informed Neural Networks

We have:

- a differential equation $g(x, y) = 0$,
- some data $\{x_j, y_j\}$ and
- a neural network $f(x \mid \theta)$ that approximates $y$.

For a PINN, we would get a loss function that looks like the following,

$$Loss_{PINN} = \underbrace{\frac{1}{N}\sum_{j}^{N} ||f(x_j|\theta) - y_j||_2^2}_{\text{Data loss}} + \lambda \underbrace{\frac{1}{M}\sum_{i}^{M} ||g(x_i, f(x_i, |\theta))||_2^2}_{\text{Physics loss}}$$

- Here $x_i$ are *collocation* points. These can be any value we want them to be, usually you would want them to be in the range of values we are interested in.
- The $x_j$ and $y_j$ are our data.
- We can also add a parameter controlling the relative strength of the data loss function and the physics loss function, here we use $\lambda$.
- And then just train as you would any other neural network.

https://medium.com/@theo.wolf/physics-informed-neural-networks-a-simple-tutorial-with-pytorch-f28a890b874a

# PINNs are Very Recent

Google Scholar

## George Em Karniadakis

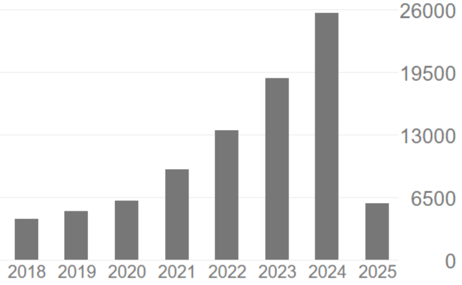
FOLLOW

The Charles Pitts Robinson and John Palmer Barstow Professor of Applied Mathematics and Engineering
Verified email at brown.edu - Homepage
Math+Machine Learning Probabilistic Scientific Com… Stochastic Multiscale Mode…

| TITLE | CITED BY | YEAR |
|---|---|---|
| Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations<br>M Raissi, P Perdikaris, GE Karniadakis<br>Journal of Computational physics 378, 686-707 | 13291 | 2019 |
| The Wiener--Askey polynomial chaos for stochastic differential equations<br>D Xiu, GE Karniadakis<br>SIAM journal on scientific computing 24 (2), 619-644 | 6034 | 2002 |
| Physics-informed machine learning<br>GE Karniadakis, IG Kevrekidis, L Lu, P Perdikaris, S Wang, L Yang<br>Nature Reviews Physics 3 (6), 422-440 | 5210 | 2021 |
| Microflows and nanoflows: fundamentals and simulation<br>G Karniadakis, A Beskok, N Aluru<br>Springer Science & Business Media | 4160 * | 2006 |
| Spectral/hp element methods for computational fluid dynamics<br>G Karniadakis, SJ Sherwin<br>Oxford University Press | 3645 | 2005 |
| Discontinuous Galerkin methods: theory, computation and applications<br>B Cockburn, GE Karniadakis, CW Shu<br>Springer Science & Business Media | 3100 * | 2012 |
| Learning nonlinear operators via DeepONet based on the universal approximation theorem of | 2248 | 2021 |

Cited by VIEW ALL

| | All | Since 2020 |
|---|---|---|
| Citations | 125349 | 79944 |
| h-index | 149 | 106 |
| i10-index | 646 | 509 |




Public access VIEW ALL

24 articles 312 articles
not available available

Based on funding mandates

Co-authors

Ali Beskok

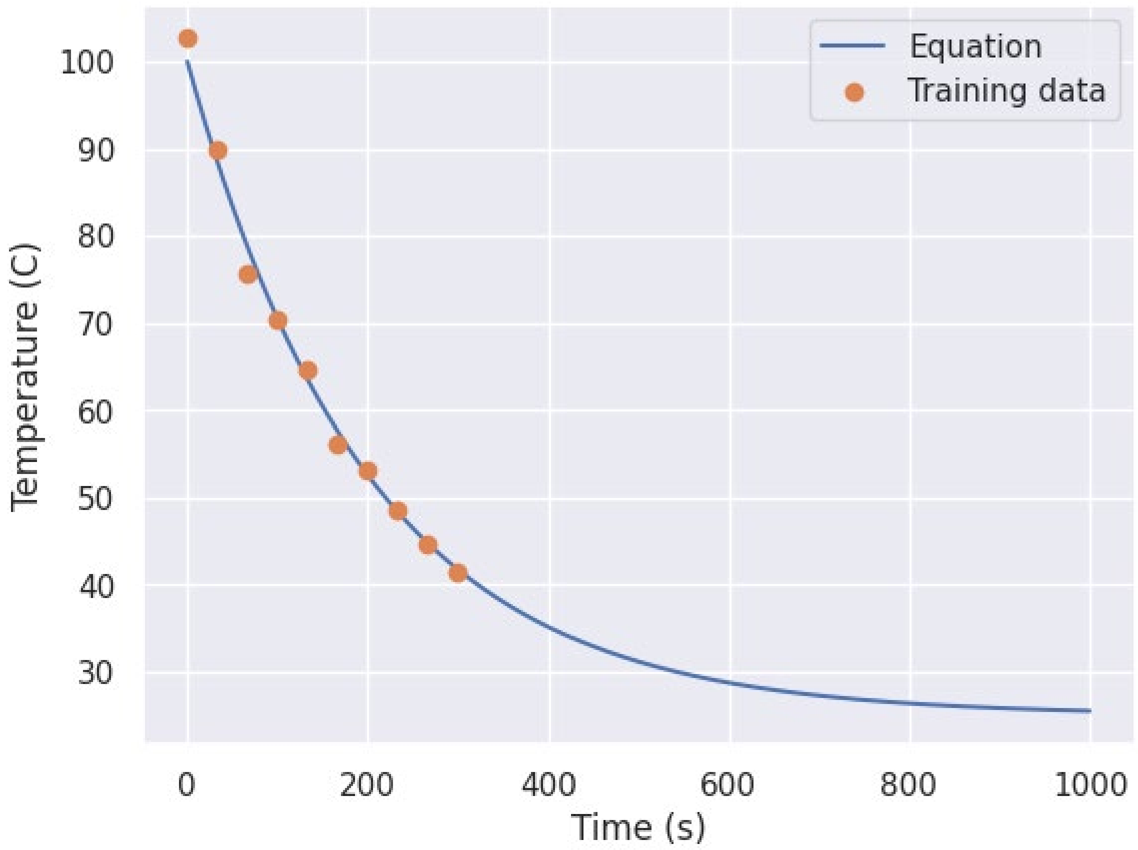
# NNs and PINNs for a simple cooling problem

$$\frac{dT(t)}{dt} = r(T_{env} - T(t))$$

$$T(t) : \text{temperature}$$

$$T_{env} : \text{temperature of the environment}$$

$$r : \text{cooling rate}$$



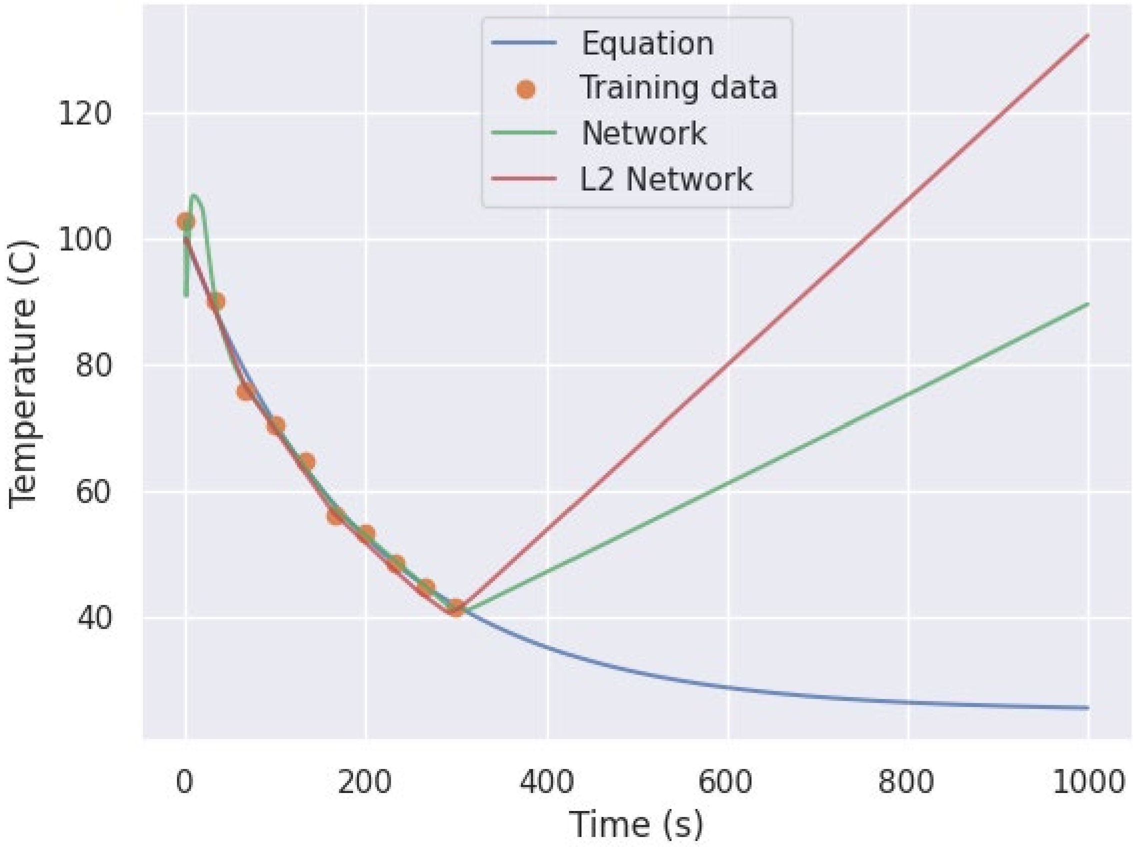
# NNs Solution

$$\frac{dT(t)}{dt} = r(T_{env} - T(t))$$

$$T(t) : \text{temperature}$$

$$T_{env} : \text{temperature of the environment}$$

$$r : \text{cooling rate}$$



# Setting up PINN

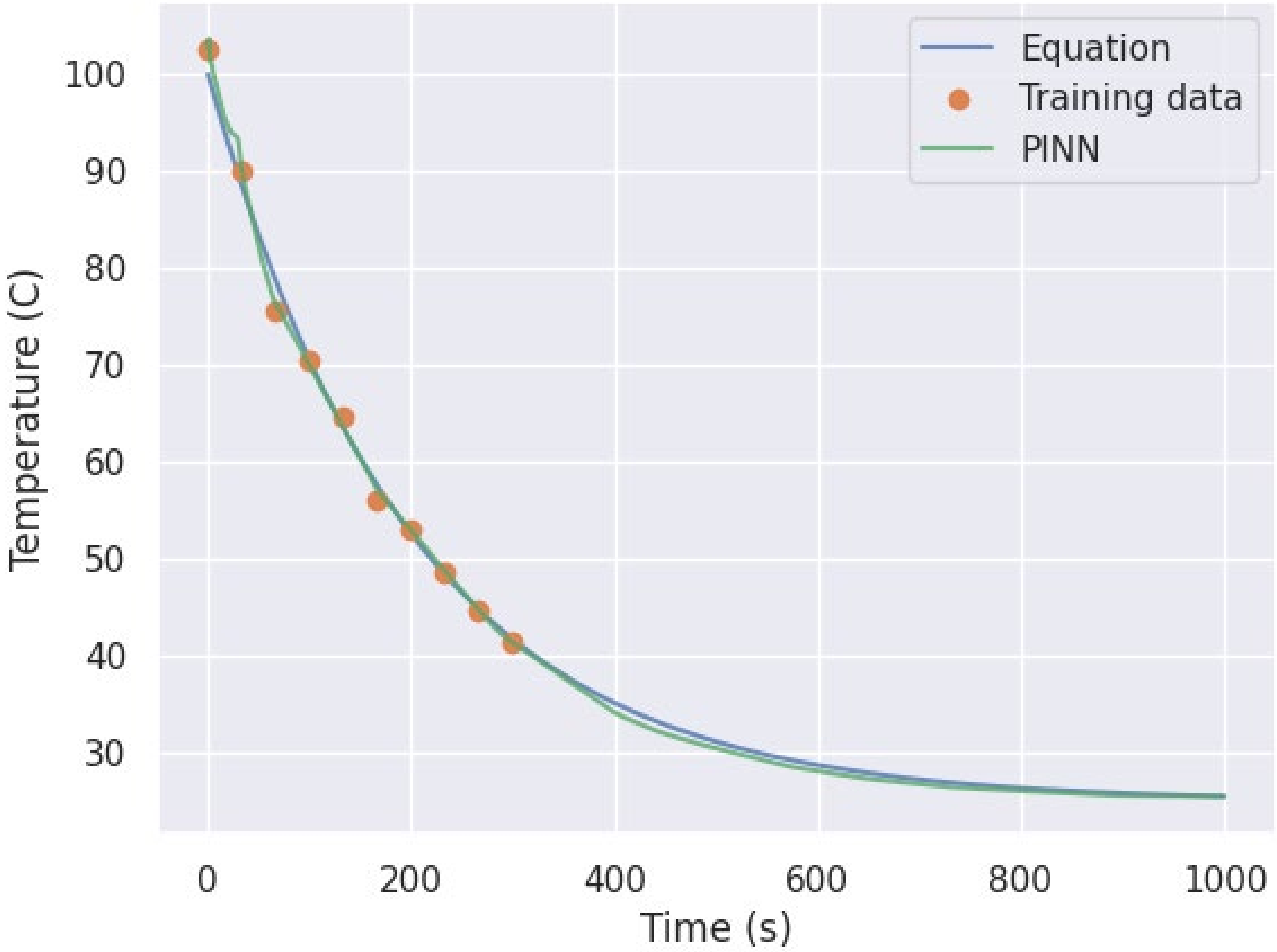
$$g(t,T) = \frac{dT(t)}{dt} - r(T_{env} - T(t)) = 0$$

$$g(t, f(t|\theta)) = \frac{df(t|\theta)}{dt} - r(T_{env} - f(t|\theta))$$

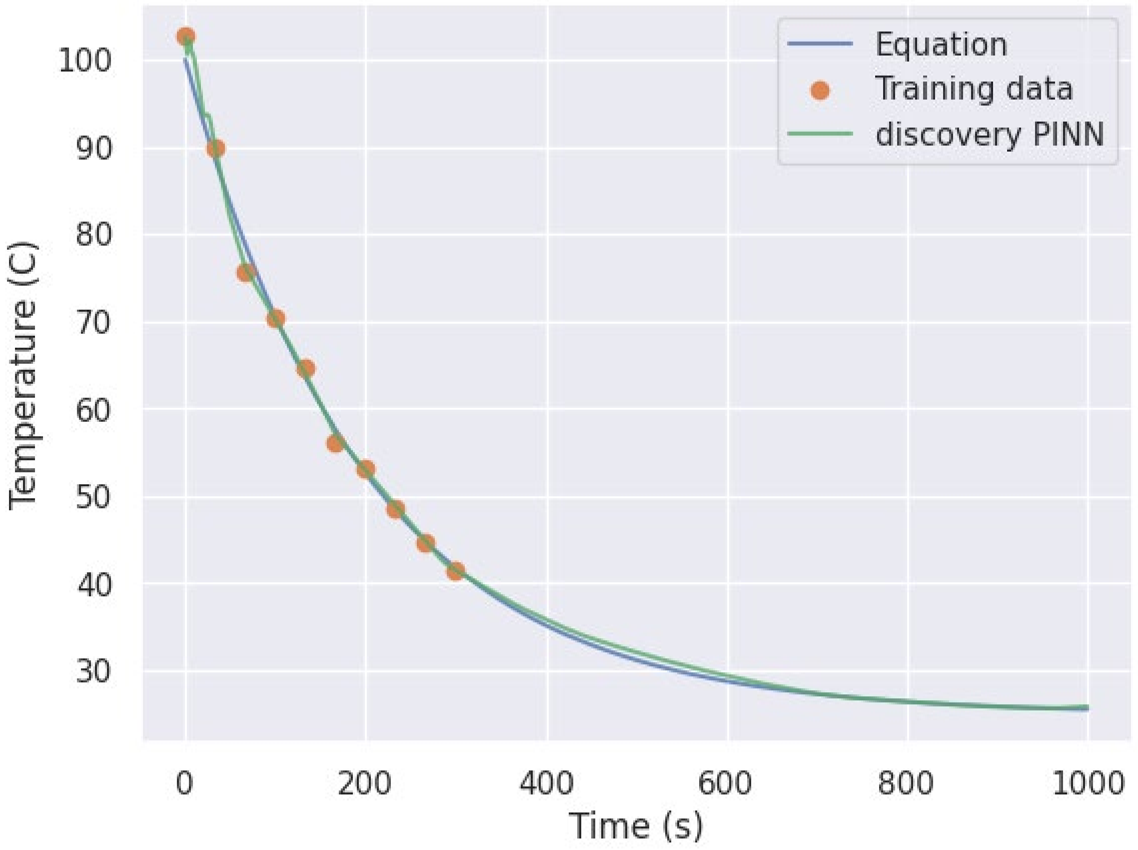
$$Loss_{PINN} = \underbrace{\frac{1}{10}\sum_{j}^{10}(f(t_j|\theta) - T_j)^2}_{\text{data loss}} + \lambda \underbrace{\frac{1}{M}\sum_{i}^{M}\left(\frac{df(t_i|\theta)}{dt_i} - r(T_{env} - f(t_i|\theta))\right)^2}_{\text{physics loss}}$$

To take the derivative of your neural network, *torch.autograd* module has a function called *grad()* which does exactly that (you can even take higher order derivatives). Just ensure that *create_graph* is set to True

# PINN for known cooling rate



# But what if the cooling rate is unknown?




Our differential equation is then g(t, T | r) = 0 where r is unknown. Thanks to PyTorch, all we need to do is just one small change: add r as a differentiable parameter.