Foothill College

# Neural ODE's - Theory and Application:

# From Residual Networks to Continuous-Depth Dynamics

by

**Sebastian Chegini**

Los Altos Hills, CA

December 2, 2025

**Abstract**

This paper explores Neural Ordinary Differential Equations (Neural ODE's) and then applies them to a real-world example of bacteria growth. We solve the population Initial Value Problem analytically using Bernoulli form and then compare it to the dynamics learned by a neural network. We demonstrate that Neural ODE's can accurately reconstruct continuous, dynamic systems from discrete data.

## Contents

## Illustrations

### Figures

# Introduction

## Residual Networks

In 2015, Residual Networks were introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.[1] ResNets allowed for the training neural networks using discrete layers, a process identical to Euler's Method for solving ODE's.

Then, in 2018, Chen et al. proposed Neural ODE's, which took the limit of ResNet layers as step size approached zero. Essentially, this work parameterized the derivative of a hidden state in the neural network, further allowing for continuous-time modeling.[2]

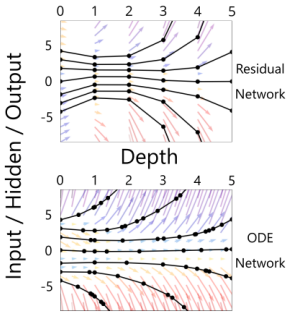


Figure 1. Residual Networks define a discrete sequence of transformations whereas Neural ODE networks learn the dynamics of a vector field to continuously transform its state (Chen *et al.*).

We will apply ODE techniques, like Bernoulli forms to solve initial value problems, to the field of **Machine Learning** and **Mathematical Biology**.

---

1. Kaiming He et al., *Deep Residual Learning for Image Recognition*, 2015, arXiv: 1512.03385 [cs.CV], https://arxiv.org/abs/1512.03385

2. Ricky T. Q. Chen et al., *Neural Ordinary Differential Equations*, 2019, arXiv: 1806.07366 [cs.LG], https://arxiv.org/abs/1806.07366

# Chapter 1

# Background Knowledge

## The Research Question

Now that we know Neural ODE's attempt to learn continuous dynamics, it would not be wrong to question the relevance of this. In comparison to ResNets, Neural Networks define the rate of change of output rather than just the output. This means that the Network is learning a *vector field* rather than a static map. Furthermore, when trying to make a prediction to calculate the next data point $y(t_1)$ from initial input $y(t_0)$, we must solve an Initial Value Problem given by

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(y(t), t, \theta)dt$$

where $f$ is a Neural Network parameterized by $\theta$.[1]

However, instead of solving this equation analytically, Neural Networks computationally solve the ODE using numerical integration methods. Specifically, they use methods like **Euler's Method** or **Runge Kutta** to approximate integration.[2]

This brings us to the research question: **How can we continuously predict successive data points with a Neural ODE, and how accurate will it be?**

---

1. Riccardo Valperga and Miltos Kofinas, "DS - Dynamical Systems and Neural ODEs," 2023, https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/DL2/Dynamical_systems/dynamical_systems_neural_odes.html

2. Michael Zeltkevic, "Runge-Kutta Methods," 1998, https://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node5.html

# Chapter 2

## Summary of ODE Techniques

### Bernoulli Differential Equations

A key to answering the focus question lies in the **Bernoulli form**. To recap, a non-linear, first-order ODE can be classified as a Bernoulli ODE if it is given in the form

$$\frac{dy}{dx} + P(t)y = Q(t)y^n.$$

The reason that this is so special is that Bernoulli ODE's can be solved relatively simply. The steps to solve Bernoulli ODE's are as follows:

1. Divide by $y^n$:

$$y^{-n}\frac{dy}{dx} + P(x)y^{1-n} = Q(x)$$

2. Define variable substitution, let

$$z = y^{1-n}$$

then

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

$$\Rightarrow \frac{dz}{dx} = (1-n)y^{-n} \cdot \frac{dy}{dx}$$

3. Multiply the ODE:

$$(1-n)y^{-n}\frac{dy}{dx} + (1-n)P(x)y^{1-n} = (1-n)Q(x)$$

4. Make substitution:

$$\frac{dz}{dx} + (1-n)P(x)z = (1-n)Q(x)$$

5. Solve for $z(x)$ using integrating factor $\mu(x) = e^{\int (1-nP(x))dx}$

6. Substitute back for $z = y^{1-n}$ and find implicit/explicit solutions

This is the algorithm for analytically solving Bernoulli ODE's. However, in the example next, we will show how Neural ODE's can forego this process using a training phase.

# Chapter 3

# Example and Model Derivation

## Bacteria Population

In Mathematical Biology, states of systems like population size are tracked over time. Here are some field concepts that will be helpful to know:

- **Vector Field**: A map that assigns some direction and magnitude at every point in the space
  - This is what the Neural ODE will learn
- **Carrying Capacity** ($K$): The maximum population size that an environment can sustain with limited resources
- **Intrinsic Growth Rate** ($r$): How fast the population grows given unlimited resources

In simple population models with unlimited resources, exponential growth occurs. However, since resources are finite in the real world, we must model population using a **logistic model** given by:

$$\frac{dy}{dt} = ry(1 - \frac{y}{K}).$$

When population $y$ is small, the term $(1 - \frac{y}{K}) \approx 1$ and the growth of $y$ is exponential.

However, as population $y$ approaches carrying capacity $K$, the term approaches 0, thus stopping growth of $y$.

## Problem Setup and Solution

Suppose we aim to find the population function $y(t)$ given the following parameters:

- $r = 0.5$ (growth rate per hour)
- $K = 10$ (carrying capacity of 10 bacteria units)
- $y_0 = 0.1$ (initial bacteria unit population)

Therefore, we can model the initial value problem as such

$$\frac{dy}{dt} = ry(1 - \frac{y}{K})$$

$$\Rightarrow \frac{dy}{dt} = ry - \frac{r}{K}y^2$$

Compared to the Bernoulli form, we can identify $P(t) = -r$, $Q(t) = -\frac{r}{K}$, and $n = 2$. Let's now follow the steps to solve Bernoulli ODE's:

Define variable substitution, let

$$z = y^{1-n} \Rightarrow z = y^{-1}$$

then

$$\frac{dz}{dt} = -y^{-2}\frac{dy}{dt}$$

Now multiply the ODE by $-y^{-2}$ and substitute:

$$-y^{-2}\frac{dy}{dt} + ry^{-1} = \frac{r}{K}$$

$$z' + rz = \frac{r}{K}$$

This is now a linear, first-order ODE! To solve this linear equation, we calculate the Integrating Factor, $\mu(t)$:

$$\mu(t) = e^{\int r\,dt} = e^{rt}$$

Now multiply the entire linear equation by $e^{rt}$:

$$e^{rt}z' + re^{rt}z = \frac{r}{K}e^{rt}$$

We recognize the left side as the derivative of the product $(e^{rt}z)$:

$$\frac{d}{dt}[e^{rt}z] = \frac{r}{K}e^{rt}$$

Now integrate both sides:

$$\int \frac{d}{dt}[e^{rt}z] = \int \frac{r}{K}e^{rt}\,dt$$

$$e^{rt}z = \frac{r}{K}\left(\frac{1}{r}e^{rt}\right) + Ce^{rt}z$$

$$\Rightarrow \frac{1}{K}e^{rt} + C$$

Solving for $z$:

$$z = \frac{1}{K} + Ce^{-rt}$$

Now we can substitute back for $y$:

$$y(t) = \frac{1}{\frac{1}{K} + Ce^{-rt}}$$

$$\Rightarrow y(t) = \frac{K}{1 + CKe^{-rt}}$$

Finally, let's apply our initial conditions $y(0) = 0.1$, knowing that $K = 10$ and $r = 0.5$:

$$y(0) = \frac{K}{1 + CK(1)} \Rightarrow y(0) = \frac{K}{1 + CK}$$

Solving for the arbitrary constant term $A = CK$:

$$A = \frac{K - y_0}{y_0} = \frac{10 - 0.1}{0.1} = 99$$

Substituting $A$ back into the general solution gives the specific solution:

$$y(t) = \frac{10}{1 + 99e^{-0.5t}}$$

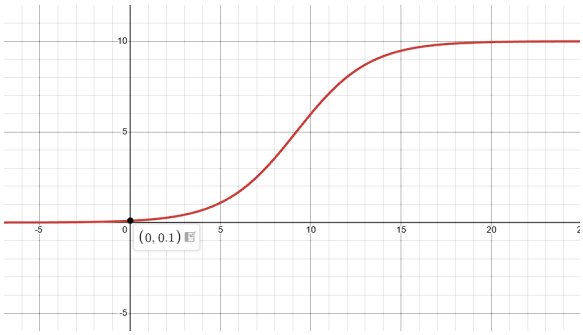


Figure 3.1. Sigmoid Function representing Bacteria Growth

And finally, we have constructed our logistic population growth model with a particular solution. This solution and graph will be the benchmark that the Neural ODE will be trained against.

# Chapter 4

# Predictions

## Neural ODE Construction and Limitations

Using our derived model $y(t) = \frac{10}{1+99e^{-0.5t}}$, we predict the population at time $t = 10$ hours:

$$y(10) = \frac{10}{1 + 99e^{-0.5(10)}} = \frac{10}{1 + 99e^{-5}}$$

$$y(10) \approx \frac{10}{1 + 99(0.0067)} \approx \frac{10}{1 + 0.663} \approx 6.01$$

After 10 hours, the bacteria population will be about **6.01 units**. **Note:** The model assumes that $r$ and $K$ are constant, but in reality, carrying capacity often fluctuates with the environment. We also assumed that the population is large enough to be treated as continuous, which would not hold for small populations (like discrete individuals). Additionally, Neural ODE's are often computationally expensive, so the limitations of using these networks include the capabilities of computer hardware.

With this in mind, let's find the end result after training a Neural ODE with 1000 iterations. See Appendix A for code files.

## Conclusion

As you can see, at $t = 10$ hours, the Neural ODE path runs through $(10, 6)$, which we predicted with our analytical Bernoulli solution.

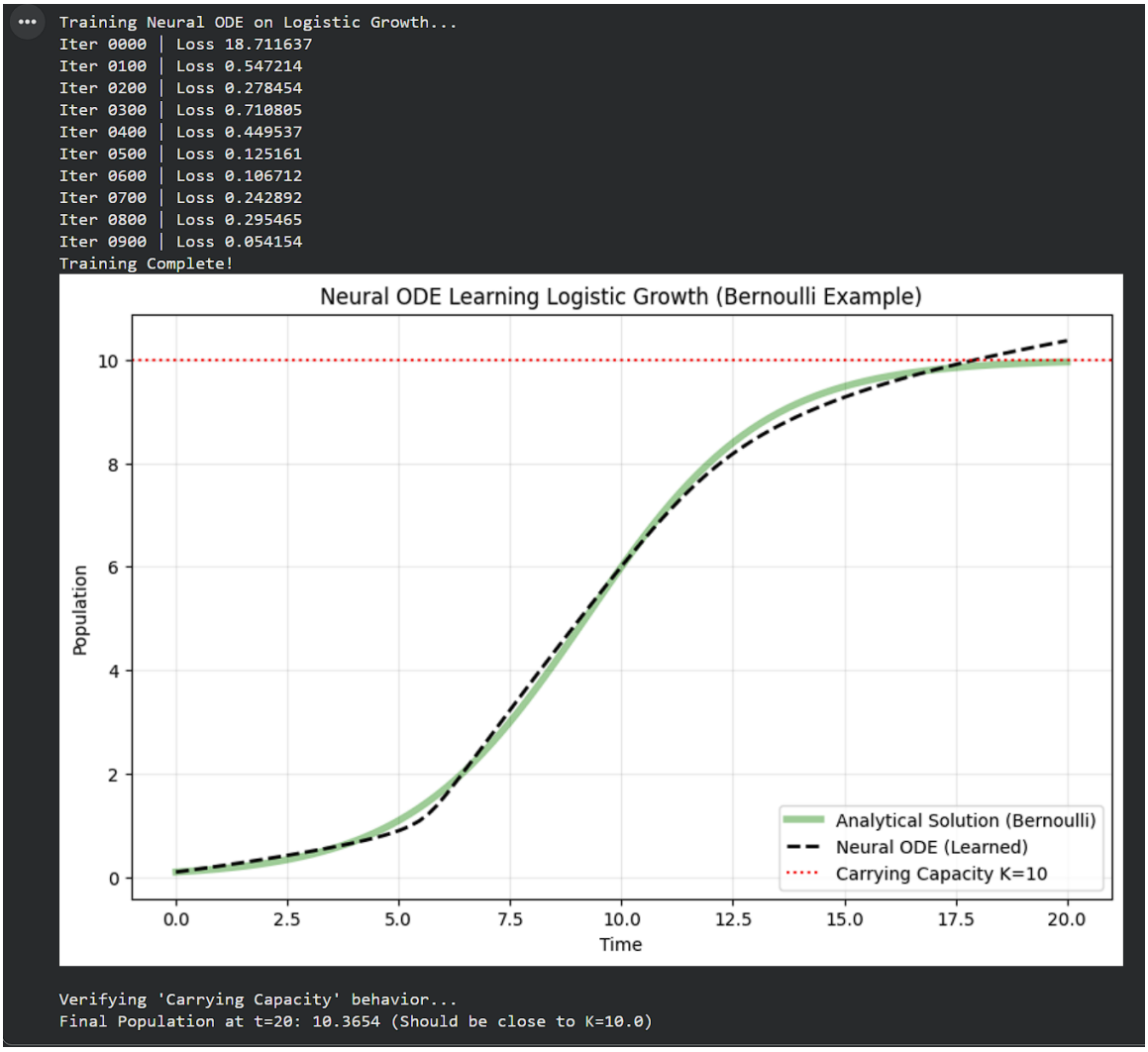


Figure 2. Output of neuralode.py

By treating the Neural ODE as a learned vector field of an ODE, we successfully modeled the Logistic Growth of a bacteria population with limited resources. Both the Neural ODE and the Bernoulli Differential Equation techniques yielded a sigmoid curve that stabilized once carrying capacity was hit.

## An Invitation for Further Exploration

Suppose the carrying capacity $K$ was not constant, but a periodic function that changed with the seasons. Let

$$K(t) = 10 + 2\sin(t).$$

How would the solution to the Bernoulli Equation change, and would Neural ODE's be a superior method to solving this new equation?

# Appendix A

# Code Files



```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# We generate synthetic data using the Logistic Growth Equation.
# dy/dt = r*y * (1 - y/K)

class LogisticGrowth:
    def __init__(self, r=0.5, K=10.0):
        self.r = r      # Growth rate
        self.K = K      # Carrying capacity

    def __call__(self, t, y):
        # Logistic Equation
        return self.r * y * (1.0 - y / self.K)

    # We also know the Analytical Solution (from Bernoulli method)
    # y(t) = K / (1 + A * exp(-rt))
    def analytical_solution(self, y0, t_tensor):
        A = (self.K - y0) / y0
        return self.K / (1 + A * torch.exp(-self.r * t_tensor))

# This network takes state y (population) and outputs derivative dy/dt.
# It has no idea what "r" or "K" are; it has to learn the curve shape.
class ODEFunc(nn.Module):
    def __init__(self):
        super(ODEFunc, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(1, 50),
            nn.Tanh(),
            nn.Linear(50, 50),
            nn.Tanh(),
            nn.Linear(50, 1)
        )

    def forward(self, t, y):
        return self.net(y)

# We use RK4 instead of Euler for better stability and accuracy.
def rk4_solve(func, y0, t):
    dt = t[1] - t[0]
    y_trajectory = [y0]
    curr_y = y0

    for i in range(len(t) - 1):
        t_curr = t[i]

        # k1 = f(t, y)
        k1 = func(t_curr, curr_y)

        # k2 = f(t + dt/2, y + dt*k1/2)
        k2 = func(t_curr + dt/2, curr_y + dt * k1 / 2)

        # k3 = f(t + dt/2, y + dt*k2/2)
        k3 = func(t_curr + dt/2, curr_y + dt * k2 / 2)

        # k4 = f(t + dt, y + dt*k3)
        k4 = func(t_curr + dt, curr_y + dt * k3)

        # Update rule: y_{n+1} = y_n + (dt/6) * (k1 + 2k2 + 2k3 + k4)
        curr_y = curr_y + (dt / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)

        y_trajectory.append(curr_y)

    return torch.stack(y_trajectory)

# Generate Data
t_size = 100
t_max = 20
t = torch.linspace(0, t_max, t_size).view(-1, 1) # Time points

# Initial Population: 0.1 (Small starting colony)
y0 = torch.tensor([0.1])
```



Figure A.1. Neural ODE Code File Page 1

```
# Generate "True" training data
true_model = LogisticGrowth(r=0.5, K=10.0)
# We compute true data using the analytical solution (Bernoulli result)
true_y = true_model.analytical_solution(y0, t)

# Now train with 1000 iterations
func = ODEFunc()
optimizer = optim.Adam(func.parameters(), lr=0.01)
n_iters = 1000

print("Training Neural ODE on Logistic Growth...")

for i in range(n_iters):
    optimizer.zero_grad()

    # Forward Pass: Integrate NN using RK4
    pred_y = rk4_solve(func, y0, t)

    # Compute Loss (MSE)
    loss = torch.mean((pred_y - true_y)**2)

    loss.backward()
    optimizer.step()

    if i % 100 == 0:
        print(f"Iter {i:04d} | Loss {loss.item():.6f}")

print("Training Complete!")

# Visualize the data
t_np = t.detach().numpy().flatten()
true_y_np = true_y.detach().numpy().flatten()
pred_y_np = pred_y.detach().numpy().flatten()

plt.figure(figsize=(10, 6))
plt.title("Neural ODE Learning Logistic Growth (Bernoulli Example)")

# Plot True Data (The Analytical Solution)
plt.plot(t_np, true_y_np, 'g-', label='Analytical Solution (Bernoulli)', linewidth=4, alpha=0.4)

# Plot Neural ODE Prediction
plt.plot(t_np, pred_y_np, 'k--', label='Neural ODE (Learned)', linewidth=2)

# Plot Carrying Capacity line
plt.axhline(y=10.0, color='r', linestyle=':', label='Carrying Capacity K=10')

plt.xlabel("Time")
plt.ylabel("Population")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Verification: Predict outside training range (Extrapolation)
print("\nVerifying 'Carrying Capacity' behavior...")
last_val = pred_y_np[-1]
print(f"Final Population at t={t_max}: {last_val:.4f} (Should be close to K=10.0)")
```



Figure A.2. Neural ODE Code File Page 2

## Bibliography


Chen, Ricky T. Q., Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. *Neural Ordinary Differential Equations*, 2019. arXiv: 1806.07366 [cs.LG]. https://arxiv.org/abs/1806.07366.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*, 2015. arXiv: 1512.03385 [cs.CV]. https://arxiv.org/abs/1512.03385.

Valperga, Riccardo, and Miltos Kofinas. "DS - Dynamical Systems and Neural ODEs," 2023. https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/DL2/Dynamical_systems/dynamical_systems_neural_odes.html.

Zeltkevic, Michael. "Runge-Kutta Methods," 1998. https://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node5.html.