

Robust training of neural ordinary differential equations

Daniel Fernández Martínez
Benasque, 26 August, 2024

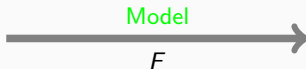
Universidad Autónoma de Madrid

Supervised Learning setting

- **Setting:** Data $(x, y) \sim \gamma$
- γ is (in general) an unknown probability distribution
- **Goal:** Given a sample data $x \in \mathbb{R}^d$ predict $y \in \mathbb{R}^D$
- Choose a model F

Feature (input)

$x \in \mathbb{R}^d$



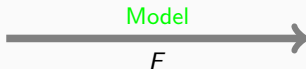
Label (output)

$F(x) \in \mathbb{R}^D$

Supervised Learning setting

- **Setting:** Data $(x, y) \sim \gamma$
- γ is (in general) an unknown probability distribution
- **Goal:** Given a sample data $x \in \mathbb{R}^d$ predict $y \in \mathbb{R}^D$
- Choose a model F

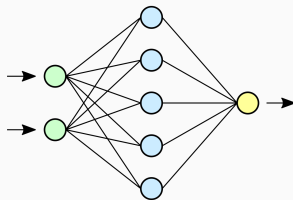
Feature (input)
 $x \in \mathbb{R}^d$



Label (output)
 $F(x) \in \mathbb{R}^D$

Some typical model choices:

- Linear regression
- Neural network
- **Neural ODE**



Neural ordinary differential equations

A **neural ODE** (NODE) [Chen, 2018] in its most general form, where $x_0 \in \mathbb{R}^d$ is the **input** (features), $u = [w, b] \in \mathbb{R}^{d_u}$ is the **control** (parameters) and f some **neural network architecture**, is given by

$$\begin{cases} \dot{x}(t) = f(t, x(t), u), & t \in (0, T] \\ x(0) = x_0 \end{cases} \quad (1)$$

Neural ordinary differential equations

A **neural ODE** (NODE) [Chen, 2018] in its most general form, where $x_0 \in \mathbb{R}^d$ is the **input** (features), $u = [w, b] \in \mathbb{R}^{d_u}$ is the **control** (parameters) and f some **neural network architecture**, is given by

$$\begin{cases} \dot{x}(t) = f(t, x(t), u), & t \in (0, T] \\ x(0) = x_0 \end{cases} \quad (1)$$



$$\begin{cases} x^{(t+1)} = x^{(t)} + hf(t, x^{(t)}, u) \\ x^{(0)} = x_0 \end{cases}$$

Neural ordinary differential equations

A **neural ODE** (NODE) [Chen, 2018] in its most general form, where $x_0 \in \mathbb{R}^d$ is the **input** (features), $u = [w, b] \in \mathbb{R}^{d_u}$ is the **control** (parameters) and f some **neural network architecture**, is given by

$$\begin{cases} \dot{x}(t) = f(t, x(t), u), & t \in (0, T] \\ x(0) = x_0 \end{cases} \quad (1)$$



$$\begin{cases} x^{(t+1)} = x^{(t)} + hf(t, x^{(t)}, u) \\ x^{(0)} = x_0 \end{cases}$$

The “N” in NODEs:

- **Inside:** $f(x, u) = \sigma(w \cdot x + b)$
- **Outside:** $f(x, u) = w\sigma(x) + b$
- **Bottleneck:**
 $f(x, u) = w_2\sigma(w_1 \cdot x + b)$

Output:

$$F(x_0) := P \circ \Phi_T(x_0)$$

- $\Phi_t(x_0) = x(t; x_0)$ flow map
- $P = Mx + N$, M, N linear

Optimising the model

- **Loss** function $J(u; x, y)$ as a measure of error between predicted and actual values for each control/parameter u .
- **Goal:** Find

$$\min_u [\mathbb{E}_{(x,y) \sim \gamma} J(u; x, y)]$$

But γ is unknown... find instead

$$\min_u \frac{1}{N} \sum_{i=1}^N J(u; x_i, y_i)$$

Optimising the model

- **Loss function** $J(u; x, y)$ as a measure of error between predicted and actual values for each control/parameter u .
- **Goal:** Find

$$\min_u [\mathbb{E}_{(x,y) \sim \gamma} J(u; x, y)]$$

But γ is unknown... find instead

$$\min_u \frac{1}{N} \sum_{i=1}^N J(u; x_i, y_i)$$

Training data
 $\{(x_i, y_i)\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}^D$

Training (optimisation)

→ **“Optimal”
parameters (control)**
 $u \in \mathbb{R}^{d_u}$

Optimising the model

- **Loss function** $J(u; x, y)$ as a measure of error between predicted and actual values for each control/parameter u .
- **Goal:** Find

$$\min_u [\mathbb{E}_{(x,y) \sim \gamma} J(u; x, y)]$$

But γ is unknown... find instead

$$\min_u \frac{1}{N} \sum_{i=1}^N J(u; x_i, y_i)$$

Training data
 $\{(x_i, y_i)\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}^D$

Training (optimisation)



"Optimal"
parameters (control)
 $u \in \mathbb{R}^{d_u}$

- **Optimisation through Gradient Descent (GD):**

$$u^{k+1} = u^k - \eta \nabla_u J [u^k]$$

- **Variants** of GD used in practice: SGD, Adam, ...

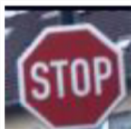
- **Robustness:** the ability to withstand or overcome adverse conditions or rigorous testing.
- Data (x, y) is supposed to follow a probability distribution γ
- **Classical training** might not give good results for perturbed input data

- **Robustness:** the ability to withstand or overcome adverse conditions or rigorous testing.
- Data (x, y) is supposed to follow a probability distribution γ
- **Classical training** might not give good results for perturbed input data



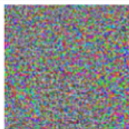
Robustness

- **Robustness:** the ability to withstand or overcome adverse conditions or rigorous testing.
- Data (x, y) is supposed to follow a probability distribution γ
- **Classical training** might not give good results for perturbed input data

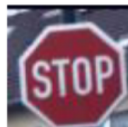


stop

+



interference



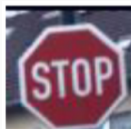
limit 70

- **Robustness:** the ability to withstand or overcome adverse conditions or rigorous testing.
- Data (x, y) is supposed to follow a probability distribution γ
- **Classical training** might not give good results for perturbed input data



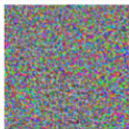
Applications in

- Autonomous driving
- Malware/spam detection
- Malfunctions (aeronautics, medicine)



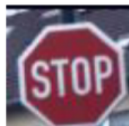
stop

+



interference

→



limit 70

Input perturbation in classification problems:

- Budget/force $\epsilon > 0$, perturbation $s(\epsilon) \in \mathbb{R}^d$
- Perturbed input $x + s(\epsilon)$

Input perturbation in classification problems:

- Budget/force $\epsilon > 0$, perturbation $s(\epsilon) \in \mathbb{R}^d$
- Perturbed input $x + s(\epsilon)$
- Random perturbation $s \sim \epsilon \cdot N(0, Id)$
- Adversarial attack $s \in B_\epsilon(0) \subseteq \mathbb{R}^d$

Input perturbation in classification problems:

- Budget/force $\epsilon > 0$, perturbation $s(\epsilon) \in \mathbb{R}^d$
- Perturbed input $x + s(\epsilon)$
- Random perturbation $s \sim \epsilon \cdot N(0, Id)$
- Adversarial attack $s \in B_\epsilon(0) \subseteq \mathbb{R}^d$

Solution: For a given norm l in \mathbb{R}^d ,
deal with the **robust training** problem

$$\min_u \mathbb{E}_{(x,y) \sim \gamma} \left[\max_{l(s) \leq \epsilon} J(u; x + s, y) \right]$$



$$\min_u \mathbb{E}_{(x,y) \sim \gamma} \left[\max_{l(v) \leq 1} J(u; x + \epsilon v, y) \right]$$

Input perturbation in classification problems:

- Budget/force $\epsilon > 0$, perturbation $s(\epsilon) \in \mathbb{R}^d$
- Perturbed input $x + s(\epsilon)$
- Random perturbation $s \sim \epsilon \cdot N(0, Id)$
- Adversarial attack $s \in B_\epsilon(0) \subseteq \mathbb{R}^d$

Solution: For a given norm l in \mathbb{R}^d , deal with the **robust training** problem

$$\min_u \mathbb{E}_{(x,y) \sim \gamma} \left[\max_{l(s) \leq \epsilon} J(u; x + s, y) \right]$$



$$\min_u \mathbb{E}_{(x,y) \sim \gamma} \left[\max_{l(v) \leq 1} J(u; x + \epsilon v, y) \right]$$

- 1) Solve the inner **maximization** problem

$$H(u; x, y) := \max_{l(v) \leq 1} J(u; x + \epsilon v, y).$$

- 2) Solve the outer **minimization** problem

$$\min_u \mathbb{E}_{(x,y) \sim \gamma} H(u; x, y).$$

Inner maximization problem

Taylor expansion of J at x results in

$$\max_{\|v\| \leq 1} J(u; x + \epsilon v, y) = J(u; x, y) + \epsilon \max_{\|v\| \leq 1} \langle \nabla_x J(u; x, y), v \rangle + O(\epsilon^2)$$

Inner maximization problem

Taylor expansion of J at x results in

$$\max_{l(v) \leq 1} J(u; x + \epsilon v, y) = J(u; x, y) + \epsilon \max_{l(v) \leq 1} \langle \nabla_x J(u; x, y), v \rangle + O(\epsilon^2)$$

Modified robust training problem

$$\min_u \mathbb{E}_{(x,y) \sim \gamma} \left[J(u; x, y) + \epsilon \max_{l(v) \leq 1} \langle \nabla_x J(u; x, y), v \rangle \right]$$

Inner maximization problem

Taylor expansion of J at x results in

$$\max_{\|v\| \leq 1} J(u; x + \epsilon v, y) = J(u; x, y) + \epsilon \max_{\|v\| \leq 1} \langle \nabla_x J(u; x, y), v \rangle + O(\epsilon^2)$$

Modified robust training problem

$$\min_u \mathbb{E}_{(x,y) \sim \gamma} \left[J(u; x, y) + \epsilon \max_{\|v\| \leq 1} \langle \nabla_x J(u; x, y), v \rangle \right]$$

ℓ^∞ norm (Fast Gradient Sign Method [Goodfellow, 2014]):

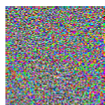
- $\|\nabla_x J(u; x, y)\|_1 = \max_{\|v\|_\infty \leq 1} \langle \nabla_x J(u; x, y), v \rangle$
- $v = \text{sign}(\nabla_x J(u; x, y))$ maximizes $\langle \nabla_x J(u; x, y), v \rangle$



x

$y = \text{"panda"}$
w/ 57.7%
confidence

+ .007 ×



$\text{sign}(\nabla_x J(\theta, x, y))$

"nematode"
w/ 8.2%
confidence

=



$x +$

$\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

"gibbon"
w/ 99.3 %
confidence

Remember gradient descent $u^{k+1} = u^k - \delta \nabla_u J [u^k]$.

Gradient computation:

1) Discretize $x(t)$

$$x^{(l)} = \text{ODESolver}(t, x^{(0)}, \dots, x^{(l-1)}, u).$$

2) Apply chain rule (backpropagation)

$$\nabla_{u^{(l)}} J = \frac{\partial J}{\partial x^{(L)}} \frac{\partial x^{(L)}}{\partial x^{(L-1)}} \cdots \frac{\partial x^{(l)}}{\partial u^{(l)}}.$$

Classical training: Optimize \implies Discretize

From Pontryagin's Maximum Principle we can directly compute $\nabla_u J$.
Suppose J can be written as

$$J(u; x_0, y) = \int_0^T L(x(t), u) dt + \Psi(\Phi_T(x_0), y) \quad (2)$$

Classical training: Optimize \implies Discretize

From Pontryagin's Maximum Principle we can directly compute $\nabla_u J$.
Suppose J can be written as

$$J(u; x_0, y) = \int_0^T L(x(t), u) dt + \Psi(\Phi_T(x_0), y) \quad (2)$$

Gradient computation through the adjoint [Massaroli, 2020]

$$\nabla_u J = \int_0^T \langle p(t), D_u f(t, x(t), u) \rangle dt,$$

where p is the solution to

$$\begin{cases} \dot{p}(t) = -D_x f(t, x(t), u)^\top p(t) - \nabla_x L(t, x(t), u), \\ p(T) = \nabla_{x(T)} \Psi(x(T), y). \end{cases}$$

Adjoint method IS backpropagation!

Classical training: Optimize \implies Discretize

From Pontryagin's Maximum Principle we can directly compute $\nabla_u J$.
Suppose J can be written as

$$J(u; x_0, y) = \int_0^T L(x(t), u) dt + \Psi(\Phi_T(x_0), y) \quad (2)$$

Gradient computation through the adjoint [Massaroli, 2020]

$$\nabla_u J = \int_0^T \langle p(t), D_u f(t, x(t), u) \rangle dt,$$

where p is the solution to

$$\begin{cases} \dot{p}(t) = -D_x f(t, x(t), u)^\top p(t) - \nabla_x L(t, x(t), u), \\ p(T) = \nabla_{x(T)} \Psi(x(T), y). \end{cases}$$

Adjoint method IS backpropagation!

Case of time-dependent controls: Gateaux derivative

$$d_u J(u)\eta = \int_0^T \langle p(t), D_u f(u(t), x(t))\eta(t) \rangle dt.$$

Linear sensitivity - initial data

For $u \in L^2((0, T), \mathbb{R}^{d_u})$ and $y \in \mathbb{R}^d$ fixed, linear sensitivity of $x_0 \rightarrow J(u; x_0, y)$ in the direction $v \in \mathbb{R}^d$ can be expressed as

$$\nabla_{x_0} J(x_0)v := \lim_{\epsilon \rightarrow 0} \frac{J(u; x_0 + \epsilon v) - J(x_0)}{\epsilon} = p(0) \cdot v$$

where $p(t)$ is the solution to the adjoint equation.

$$\begin{cases} \dot{p}(t) = -D_x f(x(t), u(t))^T p(t) - \nabla_x L(t, x(t), u), & t \in [0, T] \\ p(T) = D_{\Phi_T(x_0)} J(u; x_0, y) \end{cases} \quad (3)$$

This result is derived from the linear sensitivity equation of $x(t)$ in the direction $v \in \mathbb{R}^d$:

$$\begin{cases} \frac{d}{dt} \delta_v x(t) = D_x f(x(t), u(t)) \delta_v x(t), & t \in (0, T] \\ \delta_v x(0) = v \end{cases} \quad (4)$$

Then penalty term then becomes

$$\epsilon \langle \nabla_{x_0} J(x_0), v \rangle = \epsilon \max_{\|v\| \leq 1} \langle p(0), v \rangle$$

Augmented loss function

Let the control $u \in L^2((0, T); \mathbb{R}^{d_u})$ be fixed and let $x(t)$ y $p(t)$ be the solutions of the state and adjoint equation respectively. For fixed $\epsilon > 0$ the augmented loss function

$$J_l[u; x_0; \epsilon] := J[u; x_0] + \epsilon \max_{l(v) \leq 1} \langle p(0), v \rangle$$

approximates the minimization problem with linear precision in ϵ .

If l is the norm ℓ^r for $r \in [1, \infty]$, the augmented loss function can be written as

$$J_r[u; x_0; \epsilon] := J[u; x_0] + \epsilon \|p(0)\|_{r'}$$

where r and r' are Hölder conjugates, i.e. $1/r + 1/r' = 1$.

Proof is based on Hölder inequality.

Gradient of quadratic penalty term [Wöhler, Zuazua]

Let the control u be fixed and consider a loss function J having only terminal cost Ψ . We consider the quadratic penalty term with

$$S[u] := \|p_u(0)\|_2^2 \quad (5)$$

where p is solution of (3) with controls u and q solution of (6).

$$\begin{aligned} d_u S(u)\eta &:= - \int_0^T q(t) \cdot D_{xx} f(x(t), u(t))^T [\delta_\eta x(t), p(t)] \\ &\quad - \int_0^T q(t) \cdot D_{ux} f(x(t), u(t))^T [\eta(t), p(t)] dt \\ &\begin{cases} \dot{q}(t) = D_x f(x(t), u(t))q(t), & t \in (0, T) \\ q(0) = -p_u(0) \end{cases} \end{aligned} \quad (6)$$

Piecewise constant controls

$$u(t) = u_i, \quad t \in [t_i, t_{i+1}] \quad (7)$$

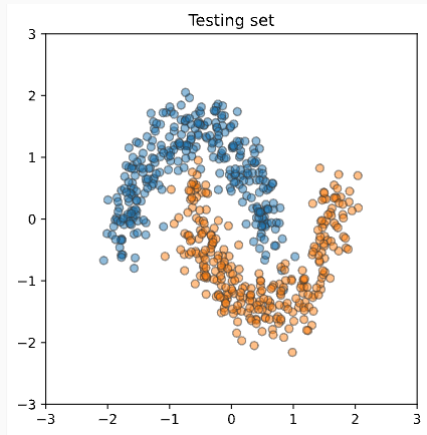
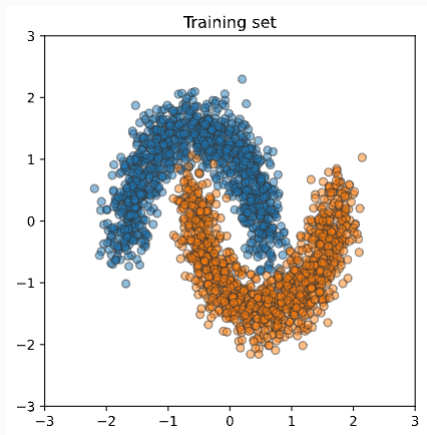
with $t_0 := 0$ y $t_m := T$, then

$$\frac{d}{du_i} J(u) = \int_{t_i}^{t_{i+1}} p(t) D_{u_i} f(x(t), u(t)) dt \quad (8)$$

where p is the solution to the adjoint equation.

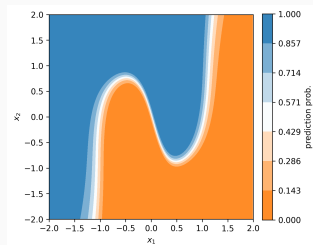
- In the numerical experiments we use piecewise constant controls consisting of 10 pieces (stacked NODEs)
- Code developed from https://github.com/twoehrer/robust_nODE

Experiments

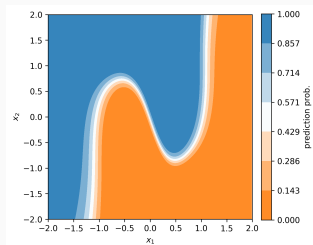


Experiments

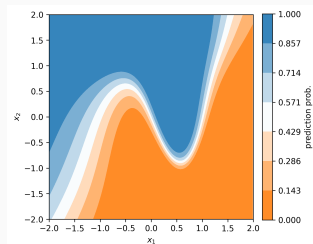
$\epsilon = 0$



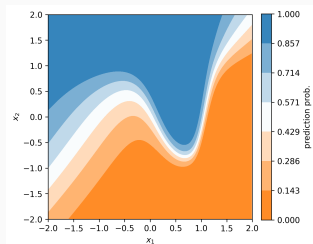
$\epsilon = 0.1$



$\epsilon = 0.2$



$\epsilon = 0.3$

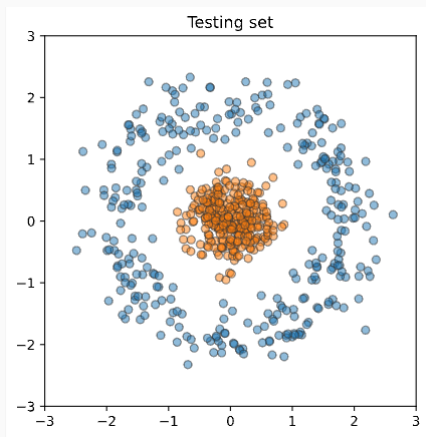
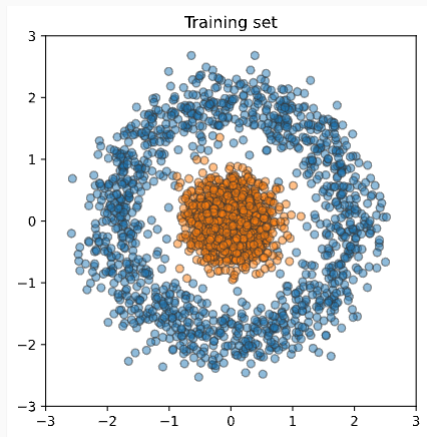


Experiments

- We perform robust trainings with different values ϵ given to penalty term
- We compare the performance in perturbed testing set

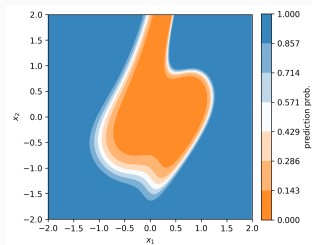
Evaluation set	Classical training	0.1-robust	0.2-robust	0.3-robust
Test	0.042488	0.041761	0.075317	0.092077
0.1-FGSM-attack test	0.064908	0.063728	0.085838	0.207678
0.1-perturbed test	0.046271	0.041761	0.080335	0.182127
0.2-perturbed test	0.075131	0.087367	0.080335	0.190990
0.3-perturbed test	0.0105135	0.0102906	0.092077	0.199580

Experiments

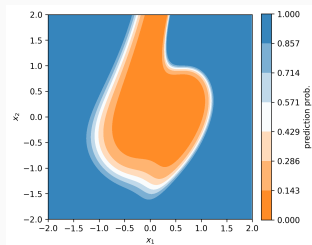


Experiments

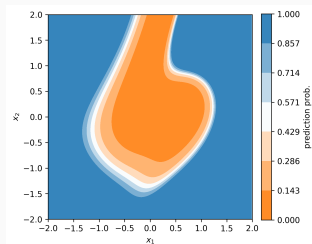
$\epsilon = 0$



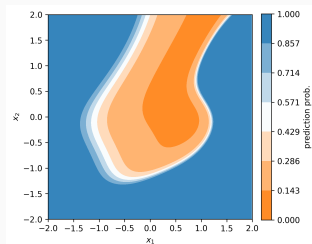
$\epsilon = 0.1$



$\epsilon = 0.2$



$\epsilon = 0.3$



Conclusions:

- Adversarial attacks are a threat to Machine Learning models
- NODEs let us treat problems about neural networks from a continuous point of view
- **Main takeaway:** memory efficient methods for computing gradient of loss function

Conclusions:

- Adversarial attacks are a threat to Machine Learning models
- NODEs let us treat problems about neural networks from a continuous point of view
- **Main takeaway:** memory efficient methods for computing gradient of loss function

Future directions:

- Generalize gradient computation of the penalty term to a more general norm
- Choice of $\epsilon > 0$ for robust training
- Consider other types of perturbation
- Enable robust training only in some part of the training process and then switch to classical training

Thank you for your attention!