Robust neural ODEs — a second order adjoint sensitivity approach.

Tobias Wöhrer

in collaboration with Enrique Zuazua

FAU Erlangen-Nürnberg
Department of Data Science

Chair
DYNAMICS, CONTROL
AND NUMERICS
FAU

Goal: Train ODE based neural networks, such as ResNets, that are robust with respect to (certain) adversarial attacks.

Methods:

Optimize then discretize: Consider optimal control problem for continuous neural ODE. Then view neural network training as discretized approximation.

Part 1: Adversarial Attacks

Adversarial Attack Examples

Adversarial attacks are the ”viruses” of machine learning.

Figure 1: Manipulated stop signs lead to misclassification.

Eykholt, Kevin, et al. ”Robust physical-world attacks on deep learning visual classification.” ’18.

Adversarial Attack Examples

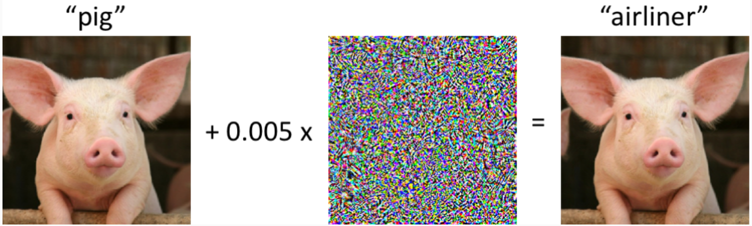


Figure 2: FGSM attack on GoogLeNet.

Goodfellow et al. "Explaining and Harnessing Adversarial Examples" '14.

Adversarial Attack Examples: NLP

Spam email filter:

“Adding these magic words to original spam emails is fairly successful in evading detection:

ferc, listbot, jhherbert, lokay, eyeforenergi, erisk, counterparti, ena, sitara, topica, kal, calger, beenladen, aggi, clickathom, cdnow, wassup, cera, enrononlin, pjm, kaminski ”

Wang, Chenran, et al. ”Crafting Adversarial Email Content against Machine Learning Based Spam Email Detection.” ’21

Adversarial Attacks

Discussion:

- Adversarial attacks are **input perturbations**
- They highlight vulnerabilities of neural networks
- Missing robustness poses security threat
- Limits applications
- Daily life more assisted by machine learning $\implies$ more potential for harm

Question: How can we defend against adversarial attacks?
I.e. how can we train robust neural networks?

Overview

Part 1: Robust Optimal Control of neural ODEs

- Robustness as saddle-point problem
- Augmented training
- Gradients via the adjoint method

Part 2: Numerical Aspects

- Memory cost
- Experiments

Part 2: Robust Optimization of Neural ODEs

neural ODE (nODE)

$$\begin{cases} \dot{x}(t) = g(u(t), x(t)) = w(t)\sigma(x(t)) + b(t), \quad t \in (0, T], \\ x(0) = x_0 \in \mathbb{R}^d. \end{cases}$$

- x_0 is the input data, e.g. an image
- $u(t) = [w(t), b(t)] \in L^2(0, T; \mathbb{R}^{d_u})$ controls
  - $w(t) \in \mathbb{R}^{d \times d}$ **weight function**
  - $b(t) \in \mathbb{R}^d$ **bias function**
- $\sigma(x) \in (0, 1)$ nonlinear **activation function** such as $\tanh(x)$
- $x_u(T)$

Time discretization of (nODE) generates neural network. Later more on that.

Optimization problem for loss function $\mathrm{J}(u, x_0, y)$:

$$\inf_u \mathbb{E}_{(x_0,y)\sim\mu}[\,\mathrm{J}(u, x_0, y)]$$

- u **control parameter**
- x_0 **input**, y classification **label** distributed according to
- (generally unknown) **data distribution** μ.
- $x_u(T)$ solution to (nODE) with initial datum x_0 parameters u at final time T.

Typical choices for J:

- **Square loss** (regression): $\mathrm{J}(u, x_0, y) = |x_u(T) - y|_2^2$, $y \in \mathbb{R}^d$.
- **Cross-entropy loss** (classification): $J_{\mathrm{CE}}(u, x_0, y) = -(x_u(T))_y + \log(\sum_{j=1}^m e^{x_j(T)})$ where $y \in \{0, \ldots, d\}$ and $(x_u(T))_y$ denotes the y-th coordinate of $x_u(T)$.

Robust optimization

Defending against adversarial attacks:

Solve robust optimization problem

$$\inf_{u} \mathbb{E}_{(x_0,y)\sim\mu}[\sup_{\ell(\xi)\le 1} \mathrm{J}(u, x_0 + \varepsilon\xi, y)] \tag{1}$$

- Attack specific norm $\ell(\xi)$, e.g. $\ell(\xi) = |\xi|_\infty$
- $\varepsilon > 0$ perturbation budget

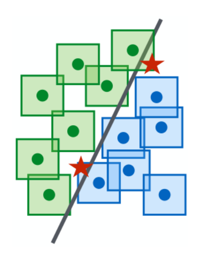


Figure 3: from [Madry et al. ’19.]

Such saddle-point problems are challenging! No convex structure.

[Shaham et al. ’18, Madry et al. ’19]

Augmented Training

Proposition 1 (Gradient regularization)

*For fixed control $u \in L^2(0, T; \mathbb{R}^{d_u})$, the **augmented loss function** ansatz*

$$\mathcal{J}_\ell(x_0) := \mathrm{J}(x_0) + \varepsilon \max_{v \in \mathbb{R}^d, \ell(v) \leq 1} \langle \nabla_{x_0} \mathrm{J}(x_0), v \rangle$$

approximates the robust optimization problem (1) *in linear order.*
For $\ell(v) = |v|_p$ follows

$$\mathcal{J}_q(x_0) = \mathrm{J}(x_0) + \varepsilon |\nabla_{x_0} J(x_0)|_{q^*},$$

where $q \in [1, \infty]$ and q^* is the Hölder conjugate satisfying $\frac{1}{q} + \frac{1}{q^*} = 1$.

[LeCun, Drucker '92; Trillos, Trillos '21]

Proof:

Taylor expansion of robust optimization problem (1) leads to

$$
\begin{aligned}
&\inf_u \mathbb{E}_{(x_0,y)\sim\mu}[\sup_{\ell(v)\leq 1} \mathrm{J}(u, x_0 + \varepsilon v)] \\
&= \inf_u \mathbb{E}_{(x_0,y)\sim\mu}[\mathrm{J}(x_0) + \varepsilon \sup_{\ell(v)\leq 1} \langle \nabla_{x_0} \mathrm{J}(x_0), v\rangle + R(x_0, u, v)\varepsilon^2] \\
&\approx \inf_u \mathbb{E}_{(x_0,y)\sim\mu}[\mathrm{J}(x_0) + \varepsilon \sup_{\ell(v)\leq 1} \langle \nabla_{x_0} \mathrm{J}(x_0), v\rangle].
\end{aligned}
$$

E.g. $\ell(v) = |v|_\infty \leq 1$: $v^* = \mathrm{sign}(\nabla_{x_0} \mathrm{J}(x_0))$ yields

$$
\sup_{\ell(v)\leq 1} \langle \nabla_{x_0} \mathrm{J}(x_0), v\rangle = \langle \nabla_{x_0} \mathrm{J}(x_0), v^*\rangle = |\nabla_{x_0} \mathrm{J}(x_0)|_1.
$$

Lemma 1 (Adjoint equation of J)

For $v \in \mathbb{R}^d, \eta \in L^2(0,T;\mathbb{R}^{d_u})$ holds

$$\langle \nabla_{x_0} \mathrm{J}(x_0), v \rangle = \langle p(0), v \rangle,$$

$$\langle \nabla_u \mathrm{J}(u), \eta \rangle_{L^2} = \int_0^T \langle D_u g(u(t), x(t))^T p(t), \eta(t) \rangle dt,$$

where $p(t) \in \mathbb{R}^d$ solves the linear adjoint equation

$$\begin{cases} \dot{p}(t) = -D_x g(u(t), x(t))^T p(t), & t \in [0,T), \\ p(T) = \nabla_{x(T)} \mathrm{J}(x_0). \end{cases} \tag{2}$$

Augmented loss function expressed as:

$$\mathcal{J}_q(x_0) = \mathrm{J}(x_0) + |\nabla_{x_0} J(x_0)|_{q^*} = \mathrm{J}(x_0) + \varepsilon |p(0)|_{q^*}, \qquad \frac{1}{q} + \frac{1}{q^*} = 1.$$

Parameter Gradients

$$\mathcal{J}_q(x_0) = \mathrm{J}(x_0) + |\nabla_{x_0} J(x_0)|_{q^*} = \mathrm{J}(x_0) + \varepsilon |p(0)|_{q^*}, \qquad \frac{1}{q} + \frac{1}{q^*} = 1.$$

The optimization of the model parameters u is usually realized via (stochastic) gradient descent of the loss function.

$\Longrightarrow$ **We want to compute** $\nabla_u \mathcal{J}(x_0)$.

We can express $\nabla_u \mathcal{J}(u)$ with adjoint equations of second order.

Theorem 2 (Second order adjoint)

Let $u \in L^2(0,T;\mathbb{R}^{d_u})$ be fixed. We consider augmented loss

$$\mathcal{J}(u) = |x_u(T) - y|_2^2 + |p_u(0)|_2^2$$

where p_u is solution to the adjoint equation (2). Then,

$$\langle \nabla_u |p_u(0)|^2, \eta \rangle = -\int_0^T \langle s(t), D_{xx} g(u(t), x(t))[\delta_\eta x(t), p(t)]$$
$$D_{ux} g(u(t), x(t))^T [\eta(t), p(t)] \rangle dt,$$

- *$D_{xx} g(u,x)$ and $D_{ux} g(u,x)$ denote the second order derivatives of $(\overline{u}, \overline{x}) \mapsto g(\overline{u}, \overline{x}) \in \mathbb{R}^d$*
- *s and $\delta_\eta x(t)$ solve the ODE*

$$\begin{cases} \frac{d}{dt}\phi(t) = D_x g(u(t), x(t))\phi(t), \quad t \in (0,T], \end{cases} \tag{3}$$

with initial data $s(0) = -p_u(0)$ and $\delta_\eta x(0) = 0$.

Part 2: Numerical Aspects

ODE based neural networks

Time discretization of (nODE) generates neural network.

E.g.: Euler discretization (with step size 1) yields
residual neural network:

$$
\begin{cases} x^{k+1} = x^k + w^k \sigma(x^k) + b^k, \quad k \in \{0, \dots, N_{\text{layers}} - 1\}, \\ x^0 = x_0 \in \mathbb{R}^d. \end{cases} \tag{4}
$$

Very specific discretization:
#layers $\leftrightarrow$ #function evaluation $\leftrightarrow$ #parameters

ODE based neural networks: Also higher order and variable step-size discretizations of (nODE) can be considered. Better adapted to data and dynamics.

Advantage: ODE based neural networks can increase "depth" without increasing parameter load.

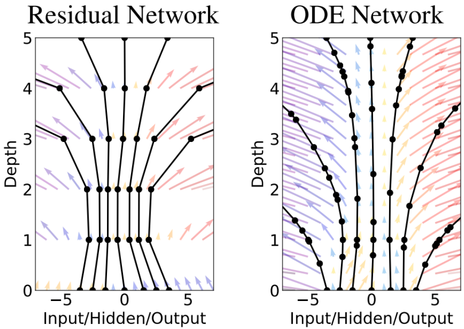


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

[Chen et al. ’18]

Memory cost

Problem 1: ODE Network with many evaluation points: $\implies$ training process with naive backpropagation (chain rule) has **very high memory cost.**

Problem 2: To evaluate our augmented loss function

$$\mathcal{J}(u) = \mathrm{J}(x_0) + \varepsilon |p_u(0)|,$$

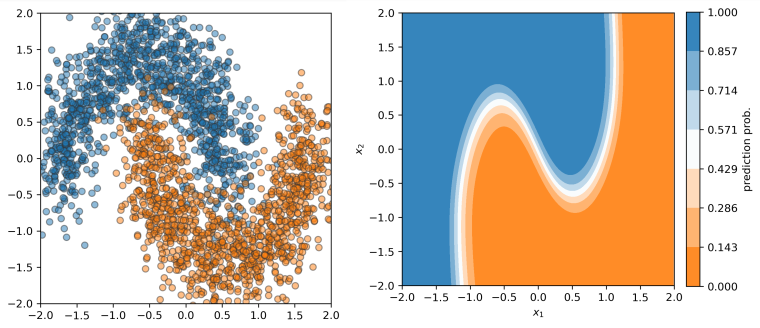
we need to

1. Solve nODE: $x_0 \mapsto x(T)$.
2. Solve adjoint equation backwards: $p(T) \mapsto p(0)$.

“Depth” essentially doubles
$\implies$ Memory cost of naive chain-rule based “Double Backpropagation”.

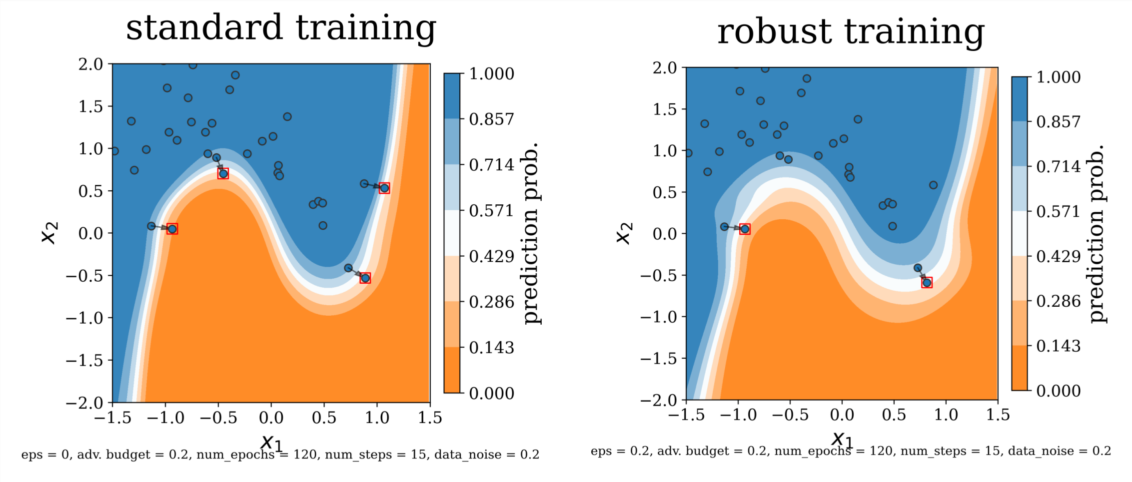
Solution: Use adjoint method layer-by-layer to compute gradients of $\mathcal{J}(u)$.

Classification example: Two-dimensional point clouds

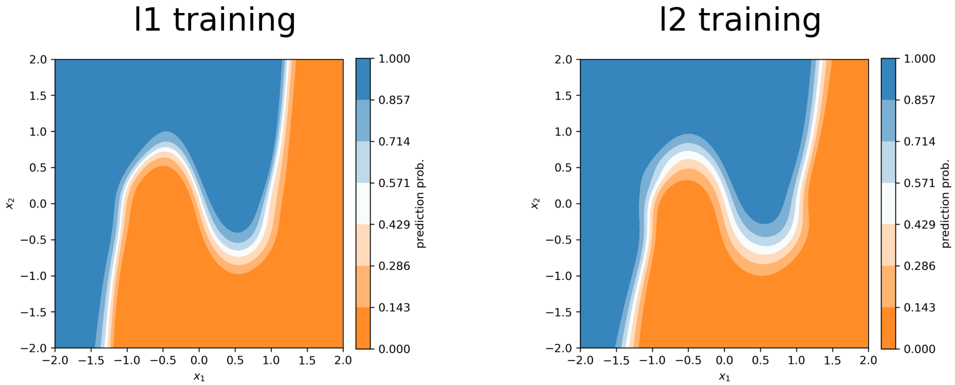


Classification of orange and blue data points.

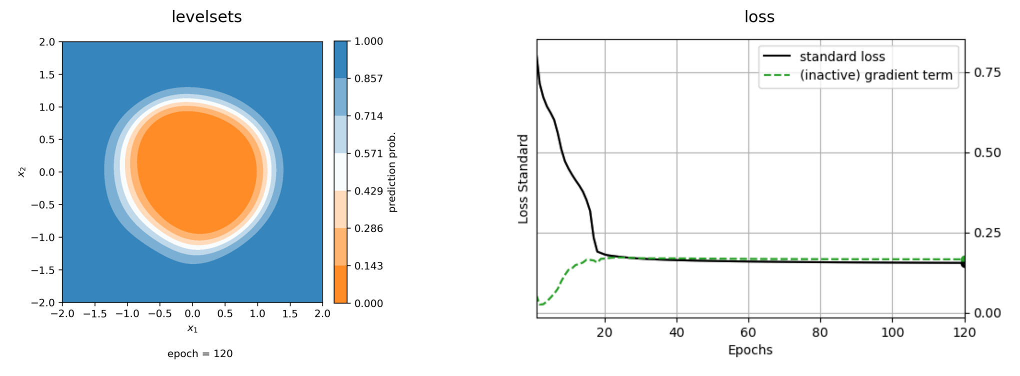
Left: Training set. **Right:** Model prediction level sets



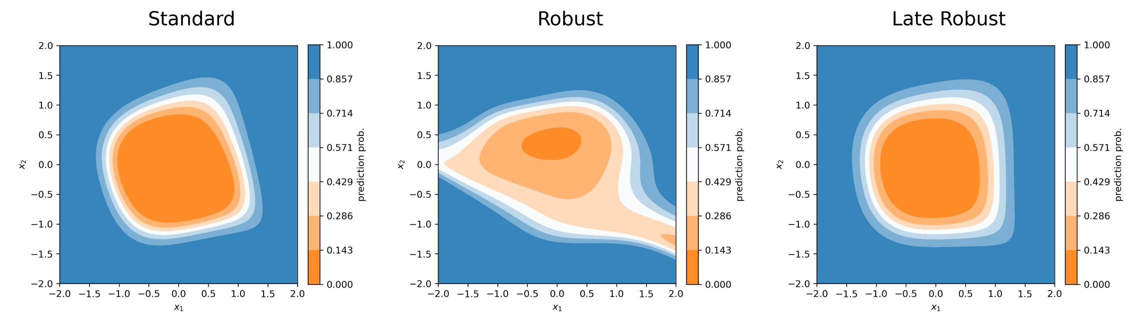
Attacks given via perturbation $x_0 + \varepsilon \nabla_{x_0} J(x_0)$


l1 training
l2 training
prediction prob.
1.000
0.857
0.714
0.571
0.429
0.286
0.143
0.000
x_1
x_2
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0

Topological considerations: movie



Topological considerations



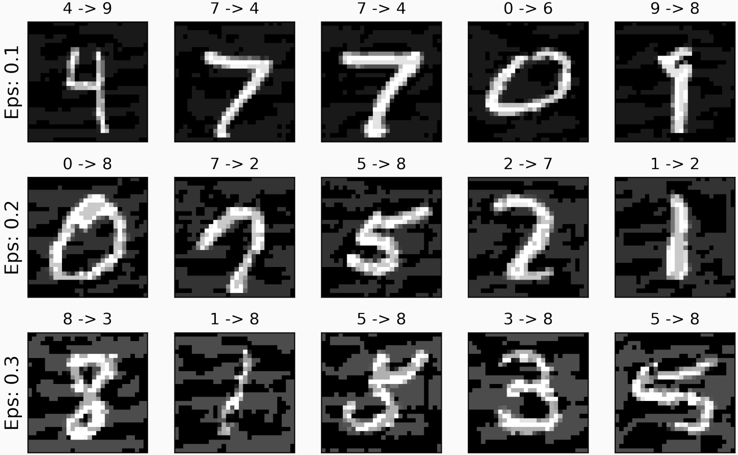


Figure 4: $|\cdot|_\infty$ Adversarial attacks: $\tilde{x_0} = x_0 + \varepsilon\,\mathrm{sign}(\nabla_{x_0} J(x_0))$.

Image Classification: MNIST

ε	standard	**robust training**
0.1	39.5 %	96.86 %
0.2	8.2 %	91.98 %
0.3	3.19 %	59.53 %

Architecture:

- 3 convolutional layers
- **6 ResNet layers: width = 64**
- 1 Pooling, 1 Linear layer: From 64 channels to 10 channels for the 10 numbers to be detected.
- Activation function tanh.

Training: Stochastic gradient descent with adjoint method with **torchdiffeq** package.

Summary: Robust ODE based neural networks

Adversarial attacks

- Lead to misclassification.
- Robustness can be formulated as **robust optimal control problem**.

Adversarial defence

- Augmented cost function of neural ODE with first order term.
- Compute gradients via adjoint method $\implies$ continuous backpropagation.
- Careful implementation with regards to data topology.

Thank you for your attention.