Controlling Neural Level Sets

Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, Yaron Lipman

Weizmann Institute of Science
Rehovot, Israel

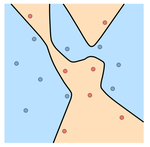
Introduction

The level sets of a Deep Neural Network (DNN) are known to capture important characteristics and properties of the network.

A popular example is when the network $F(x;\theta):\mathbb{R}^d\times\mathbb{R}^m\to\mathbb{R}^l$ represents a classifier, θ are its learnable parameters, $f_i(x;\theta)$ are its logits (the outputs of the final layer), and the level set

$$\mathcal{S}(\theta)=\left\{x\in\mathbb{R}^d \;\middle|\; f_j-\max_{i\neq j}\{f_i\}=0\right\} \qquad (1)$$

represents the *decision boundary* of the j-th class.

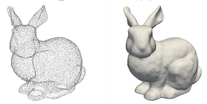


Decision boundary of a neural network trained using Cross-Entropy loss.

Another recent example is *modeling a manifold* (e.g., a curve or a surface in $\mathbb{R}^3$) implicitly using a level set of a neural network. That is,

$$\mathcal{S}(\theta)=\left\{x\in\mathbb{R}^d \;\middle|\; F(x;\theta)=0\right\} \qquad (2)$$

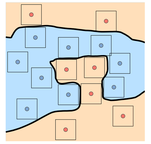
represents (generically) a manifold of dimension $d-l$ in $\mathbb{R}^d$.



0-Level set of neural network (left) reconstructed from a point cloud (right).

Motivation

Usually, decision boundary of a DNN is only controlled indirectly by loss functions that measure network output values of training examples (e.g., cross entropy loss). Increasing the distance between the training examples and the decision boundary (a.k.a *margin*), may lead to better generalization and adversarial robustness.



Decision boundary L_∞ manipulation.

In addition, previous works [3] learned surfaces implicit representation using regression, given a signed distance function or occupancy function computed from a ground truth surface. However, for raw point clouds, computing the signed distance function is a notoriously difficult task. Thus, introducing loss functions directly controlling (2) would enable learning from raw point clouds.

Goal

Controlling the level-sets of neural networks in a differential and a scalable way. Building level-set based loss functions suitable for tackling the following challenges:

- Improve adversarial robustness by directly encouraging the decision boundary to pass outside of an ϵ ball.
- Learn an implicit representation of a surface from noisy point cloud.
- Improve learning from small amount of data by generalizing the notion of SVM to deep neural networks.

Approach

We present a simple approach to directly control and manipulate neural level sets. Our method consists of two parts:

1. **Sampling of the neural level sets.**
2. **Relating the samples' positions to the network parameters.**

The latter is achieved by a *Sample Network* that is constructed by adding a single fixed linear layer to the original network. In turn, the sample network can be used to incorporate the level set samples into a loss function of interest as a proxy for $\mathcal{S}(\theta)$.

Sample Network

We denote by $D_xF(p;\theta)\in\mathbb{R}^{l\times d}$ the matrix of partial derivatives of F with respect to x.

Level set sampling

To sample $\mathcal{S}(\theta)$ at some $\theta=\theta_0$, we start with a set of n points $p_i,\ i\in[n]$ sampled from some probability measure in $\mathbb{R}^d$. Next, we perform *generalized Newton iterations* to move each point p towards $\mathcal{S}(\theta_0)$:

$$p^{\text{next}}=p-D_xF(p;\theta_0)^\dagger F(p;\theta_0) \qquad (3)$$

Where $D_xF(p,\theta_0)^\dagger$ is the Moore-Penrose pseudo-inverse of $D_xF(p,\theta_0)$.

Differentiable sample position

Next, we would like to relate each sample p, belonging to some level set $F(p;\theta_0)=c$, to the network parameters θ. Namely, $p=p(\theta)$, that holds:

$$p(\theta_0)=p \quad ; \quad F(p(\theta);\theta)=c. \qquad (4)$$

Taking derivative on the second condition, yields (using the chain rule):

$$\frac{\partial}{\partial\theta}\Big|_{\theta=\theta_0}F(p(\theta);\theta)=0 \quad\Rightarrow\quad D_\theta p(\theta_0)=-D_xF(p,\theta_0)^\dagger D_\theta F(p,\theta_0) \qquad (5)$$

Hence, an efficient solution is given by:

$$p(\theta)=p-D_xF(p;\theta_0)^\dagger\left[F(p;\theta)-c\right] \qquad (6)$$

This allows an simple implementation with a single network $G(p,D_xF(p;\theta_0)^\dagger;\theta):=p(\theta)$. Note that for scalar networks, $l=1$, $D_xF\in\mathbb{R}^{1\times d}$, a direct computation shows that:

$$D_xF(p;\theta_0)^\dagger=\frac{D_xF(p;\theta_0)^T}{\|D_xF(p;\theta_0)\|^2}. \qquad (7)$$

Robustness to adversarial perturbations

The goal of robust training is to prevent a change in a model's classification result when small perturbations are applied to the input. Let (x_j,y_j) denote training examples and labels, and let $\mathcal{S}^j(\theta)=\{x\in\mathbb{R}^d\mid F^j(x;\theta)=0\}$, where $F^j(x;\theta)=f_j-\max_{i\neq j}f_i$, the decision boundary of label j. We define the loss

$$\text{loss}(\theta)=\frac{1}{N}\sum_{j=1}^{N}\lambda_j\max\left\{0,\varepsilon_j-\text{sign}(F^{y_j}(x_j;\theta))\text{d}(x_j,\mathcal{S}^{y_j}(\theta))\right\}, \qquad (8)$$

where $\text{d}(x,\mathcal{S}^j)$ is some notion of a distance between x and $\mathcal{S}^j$. the idea of this loss is: (i) if x_j is incorrectly classified, pull the decision boundary $\mathcal{S}^{y_j}$ toward x_j; (ii) if x_j is classified correctly, push the decision boundary $\mathcal{S}^{y_j}$ to be within a distance of at-least ε_j from x_j.

In our implementation we have used $\text{d}(x,\mathcal{S}^j)=\rho(x,p(\theta))$, where $p(\theta_0)\in\mathcal{S}^j$ is a sample of this level set; $\rho(x,p)=|x^{i_*}-p^{i_*}|$, and x^{i_*},p^{i_*} denote the i_*-th coordinates of x,p (resp.), $i_*=\text{argmax}_{i\in[d]}|D_xF(p;\theta_0)|$.

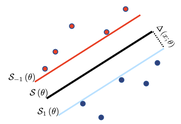
Method	Dataset	Attack	Test Acc.	Rob. Acc. Xent	Rob. Acc. Margin
Standard	MNIST	PGD$^{40}(\varepsilon=0.3)$	99.34%	13.59%	0.00%
Madry et al. [2]	MNIST	PGD$^{40}(\varepsilon=0.3)$	99.35%	96.04%	96.11%
TRADES [4]	MNIST	PGD$^{40}(\varepsilon=0.3)$	98.97%	96.75%	96.74%
Ours	MNIST	PGD$^{40}(\varepsilon=0.3)$	99.35%	99.23%	97.35%
Standard	CIFAR10	PGD^{20} $(\varepsilon=0.031)$	83.67%	0.00%	0.00%
Madry et al. [2]	CIFAR10	PGD^{20} $(\varepsilon=0.031)$	71.86%	39.84%	38.18%
TRADES [4]	CIFAR10	PGD^{20} $(\varepsilon=0.031)$	71.24%	41.89%	38.4%
Ours	CIFAR10	PGD^{20} $(\varepsilon=0.031)$	71.96%	38.45%	38.54%

Results of L_∞-bounded attacks compared to other methods.

Geometric SVM

Given training data $(x_j,y_j)\in\mathbb{R}^d\times\{-1,1\},\ j\in[n]$, one can train a linear classifier $w\in\mathbb{R}^d,\ b\in\mathbb{R}$ using the soft SVM loss:

$$\frac{1}{N}\sum_{j=1}^{N}\max\left\{0,1-y_j(w^Tx_j+b)\right\}+\lambda\|w\|_2^2 \qquad (9)$$



This formulation aims to maximize the *margin*, which in the linear case is given by:

$$\|w\|_2^{-1}=\text{d}(\mathcal{S}(\theta),\mathcal{S}_1(\theta))=\text{d}(\mathcal{S}(\theta),\mathcal{S}_{-1}(\theta))$$

Generalization to deep neural networks

The *margin* for $F(x;\theta)$, and a data sample x can be defined by:

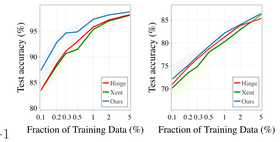
$$\Delta(x;\theta)=\min\left\{\text{d}(p(\theta),\mathcal{S}_1(\theta)),\text{d}(p(\theta),\mathcal{S}_{-1}(\theta))\right\}$$

where $p(\theta)$ is the *sample network* of the projection of x on $\mathcal{S}(\theta_0)$. Note that in Eq. 9:

$$y_j(w^Tx_j+b)=\text{sign}(y_j(w^Tx_j+b))\left|\frac{w^Tx_j+b}{\|w\|}\right|\|w\|$$

Yielding the following geometrical general formulation:

$$\frac{1}{N}\sum_{j=1}^{N}\max\left\{0,1-\text{sign}(y_jF(x_j;\theta))\frac{\text{d}(x_j,p_j)}{\Delta(x_j;\theta)}\right\}+\frac{\lambda}{N}\sum_{j=1}^{N}\Delta(x_j;\theta)^{-1}$$



Our geometrical SVM soft-margin.

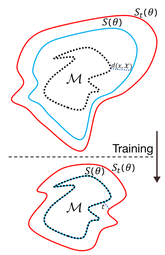
Generalization from small fractions of the data on MNIST and CIFAR10

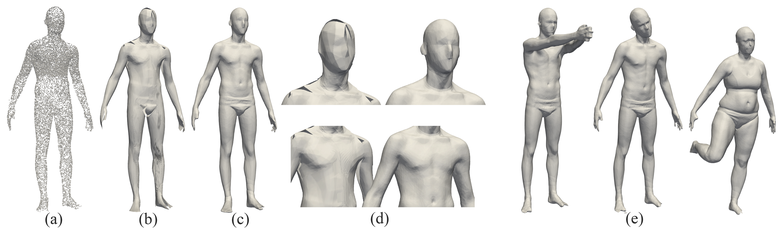
Manifold reconstruction

Given a point cloud $\mathcal{X}=\{x_j\}_{j=1}^N\subset\mathbb{R}^d$ that samples some surface $\mathcal{M}\subset\mathbb{R}^3$, our goal is to find parameters θ of $F:\mathbb{R}^3\times\mathbb{R}^m\to\mathbb{R}$, so that the level set $\mathcal{S}(\theta)$ approximates $\mathcal{M}$. We suggest the following loss:

$$\sum_{t\in\mathcal{T}}\left[\int_{\mathcal{S}_t(\theta)}\Big|\text{d}(x,\mathcal{X})-|t|\Big|^p dv(x)\right]^{\frac{1}{p}}+\frac{\lambda}{N}\sum_{j=1}^{N}|F(x_j;\theta)| \qquad (10)$$

where $\text{d}(x,\mathcal{X})=\min_{j\in[N]}\|x-x_j\|_2$ be the distance between x and $\mathcal{X}$. The loss encourages the neural level set $\mathcal{S}_t(\theta),\ t\in\mathcal{T}$ to be of distance $|t|$ to $\mathcal{X}$.





(a) input point cloud; (b) AtlasNet [1]; (c) our result; (d) blow-ups; (e) more examples of our reconstruction.

References

[1] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Atlasnet: A papier-m\˙ ach\'e approach to learning 3d surface generation. *arXiv preprint arXiv:1802.05384*, 2018.

[2] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[3] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019.

[4] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.