Neural Network Verification

Part 3: *Unsound Methods*

Some false properties can be proved false

Neural Network Verification

Neural network f

Scalar output $z = f(\mathbf{x})$

Property: $f(\mathbf{x}) > 0$ for all $\mathbf{x} \in X$

Formally prove the property, or provide counter-example

Outline

- Projected Gradient Descent
- Robustness Measure

Projected Gradient Descent

Try to find a counter-example

$z^* = \min_{\mathbf{x} \in X} f(\mathbf{x})$ **NP-hard**

If $z^* < 0$ then a counter-example is found

Approximate optimization **May not succeed**

Projected Gradient Descent

$\min_{x \in X} f(x)$

Start at some parameter $\mathbf{x}_0$

For t = 0 to T // Number of iterations

$\mathbf{g}_t = \nabla f(\mathbf{x}_t)$ **Automatic differentiation**

$\mathbf{y}_t = \mathbf{w}_t - \eta_t \mathbf{g}_t$ **η_t = step-size**

$\mathbf{x}_{t+1} = \text{argmin}_{\mathbf{x} \in X} \|\mathbf{y}_t - \mathbf{x}\|$

Closed form solution for l_∞ ball

End

Projected Gradient Descent

Readily scales to very large neural networks

If it finds a counter-example, then property is false

If it doesn't find one, property may be true or false

https://adversarial-ml-tutorial.org/

Outline

- Projected Gradient Descent
- **Robustness Measure**

Webb, Rainforth, Teh and Kumar, 2019

Measuring Robustness

Two networks f_1 and f_2

Property is false for both networks

Are they equally bad in practice?

One may be more robust than the other

Outline

- Projected Gradient Descent
- Measuring Robustness
  - **Metric**
  - Adaptive Multi-Level Splitting
  - Results

Robustness Metric

Neural network f

Input model p($\mathbf{x}$), $\mathbf{x} \in X$

Property g($\mathbf{x}$) = -f($\mathbf{x}$) (-ve values are safe)

Metric I[p,g] = Probability of error wrt input model

Robustness Metric

Neural network f

Input model $p(\mathbf{x})$, $\mathbf{x} \in X$

Property $g(\mathbf{x}) = -f(\mathbf{x})$ (-ve values are safe)

Metric $I[p,g] = \int_X \delta_{\{g \geq 0\}}[\mathbf{x}]\, p(\mathbf{x})\, d\mathbf{x}$

1 if $g(\mathbf{x}) \geq 0$ and 0 if $g(\mathbf{x}) < 0$

Monte Carlo Estimate

Neural network f

Input model $p(\mathbf{x})$, $\mathbf{x} \in X$

Draw N samples from $p(\mathbf{x})$

Metric $I[p,g] = 1/N \sum_n \delta_{\{g \geq 0\}}[\mathbf{x}_n]$

$g(\mathbf{x}) \geq 0$ is typically a rare event

Outline

- Projected Gradient Descent

- Measuring Robustness
  - Metric
  - **Adaptive Multi-Level Splitting**
  - Results

Kahn and Harris, 1951; Guyader et al., 2011

Intuition

Define levels $-\infty = L_0 < L_1 < \ldots < L_K = 0$

P_k = probability that $g(\mathbf{x}) \geq L_k$

We want to compute P_K

The value of P_K is very small

Intuition

Define levels $-\infty = L_0 < L_1 < \ldots < L_K = 0$

P_k = probability that $g(\mathbf{x}) \geq L_k$ given $g(\mathbf{x}) \geq L_{k-1}$

We want to compute P_K

The value of P_K is very small

Intuition

Define levels $-\infty = L_0 < L_1 < \ldots < L_K = 0$

P_k = probability that $g(\mathbf{x}) \geq L_k$ given $g(\mathbf{x}) \geq L_{k-1}$

We want to compute $P_K = \prod_k P_k$

The value of P_K is very small

Intuition

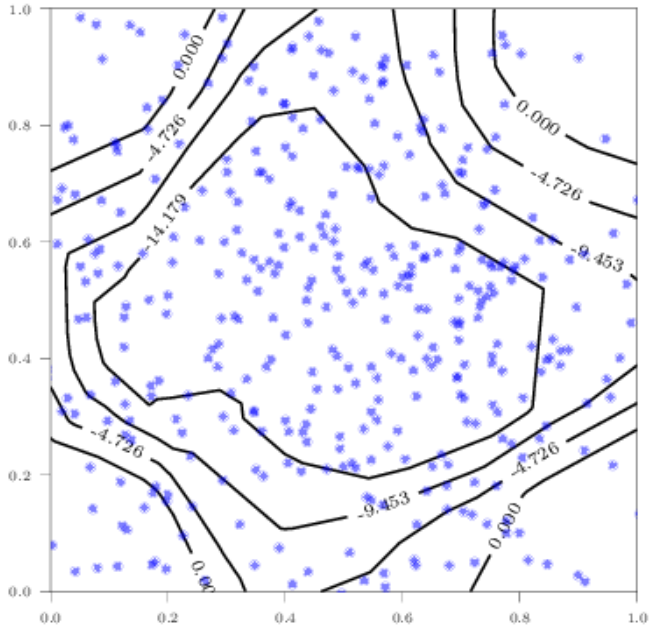
Define levels $-\infty = L_0 < L_1 < \ldots < L_K = 0$

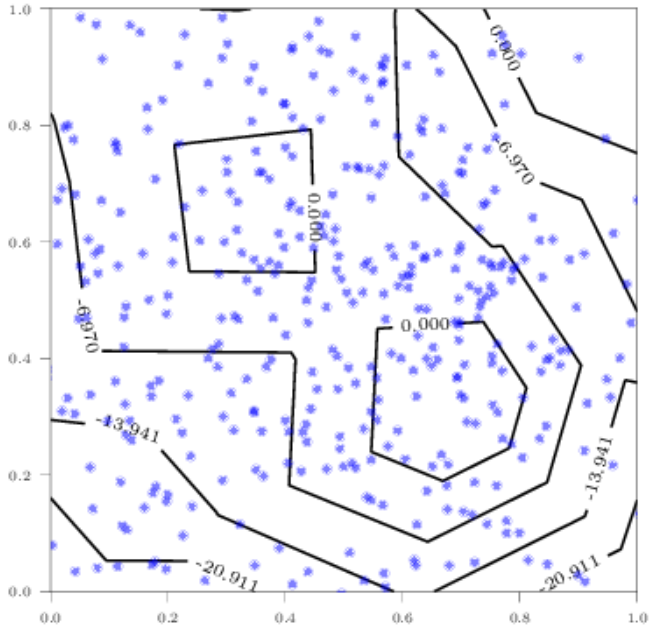
P_k = probability that $g(\mathbf{x}) \geq L_k$ given $g(\mathbf{x}) \geq L_{k-1}$

We want to compute $P_K = \prod_k P_k$

Set levels such that each P_k can be estimated reliably

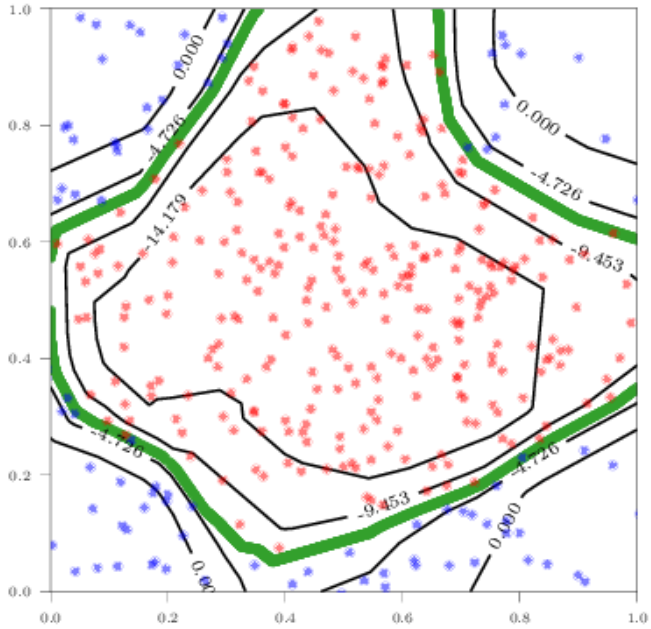
2D Examples

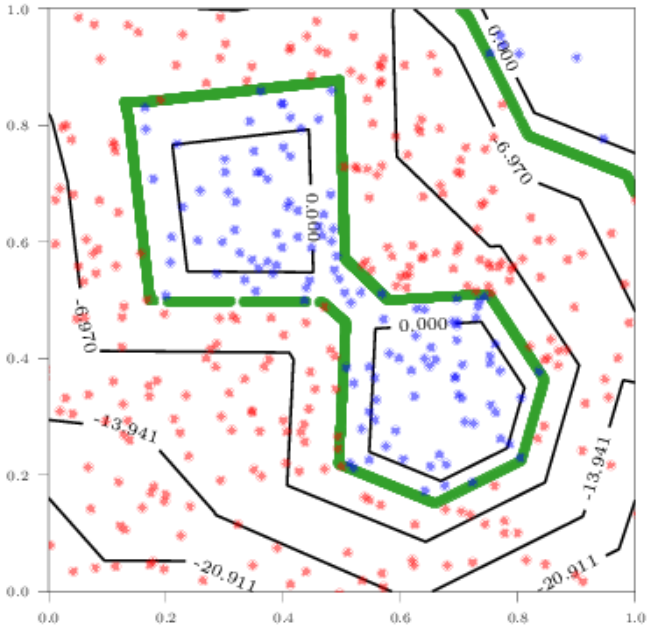




Draw N samples from p(**x**)

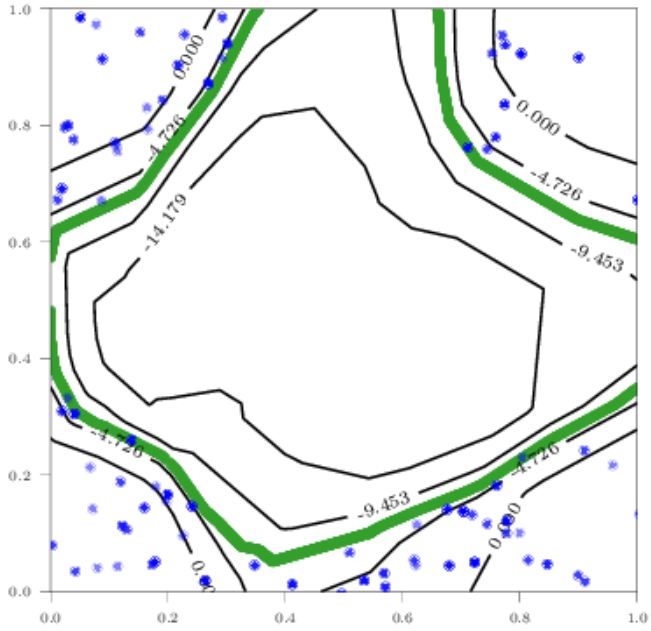
2D Examples

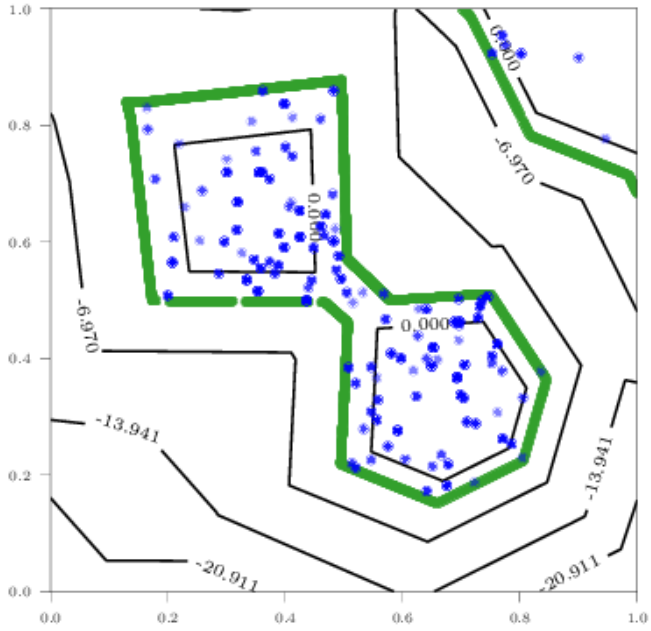




Set L_1 to retain ~10% of the samples

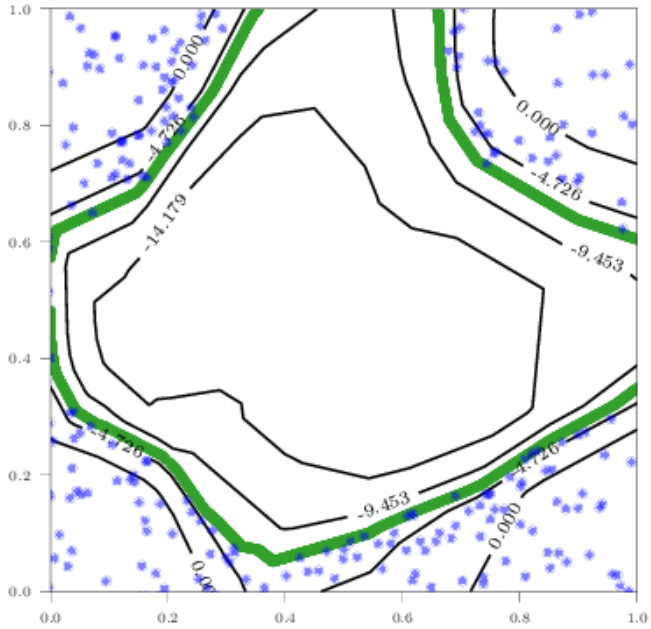
2D Examples

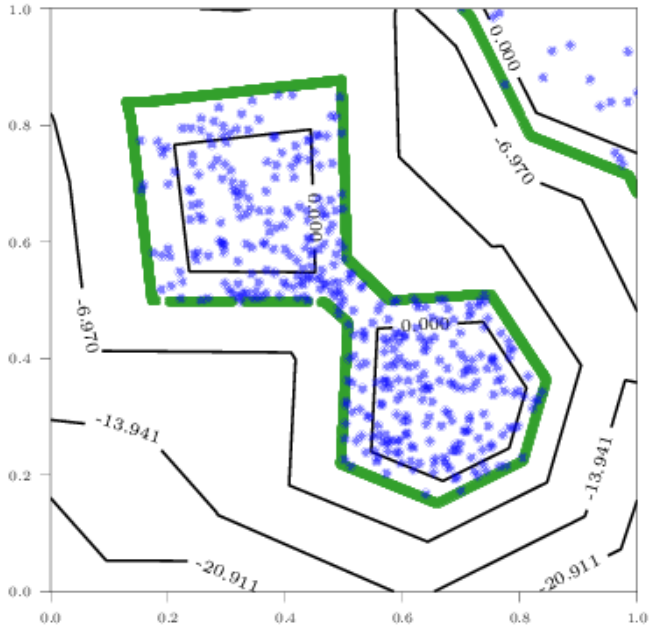




Uniformly resample with replacement to get N samples

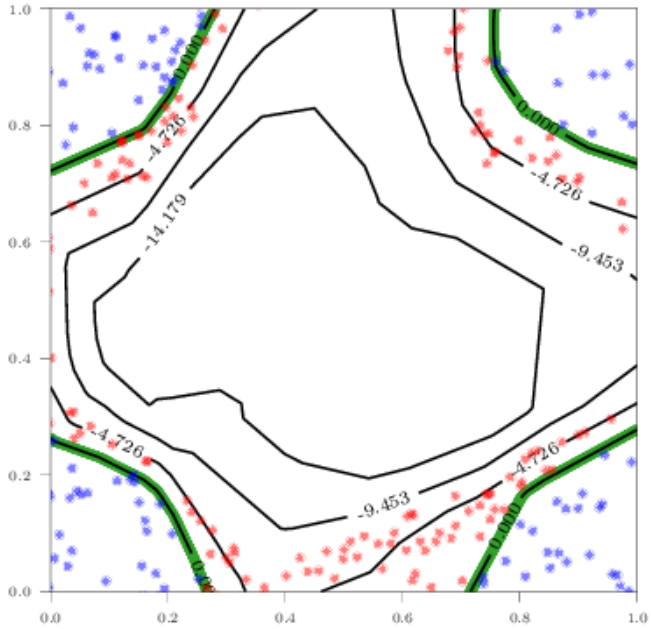
2D Examples

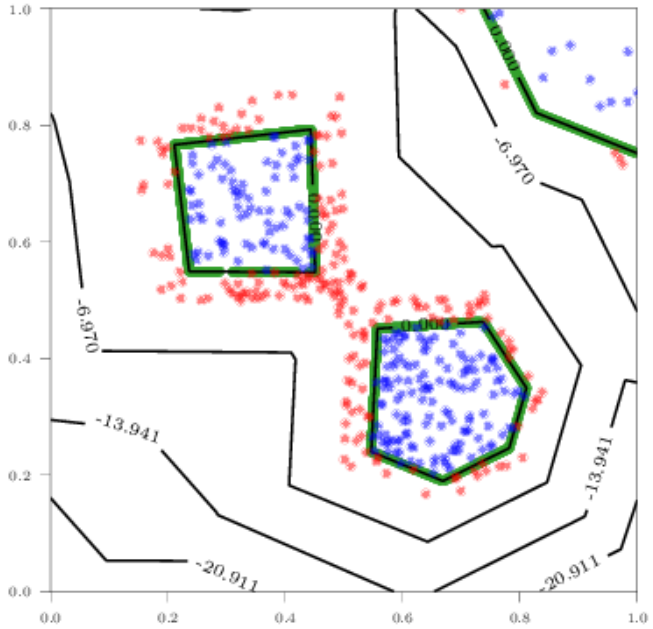




Apply Metropolis-Hastings transitions

2D Examples





Set L_2 to retain ~10% of the samples

Continue until we have an estimate of the metric

Outline

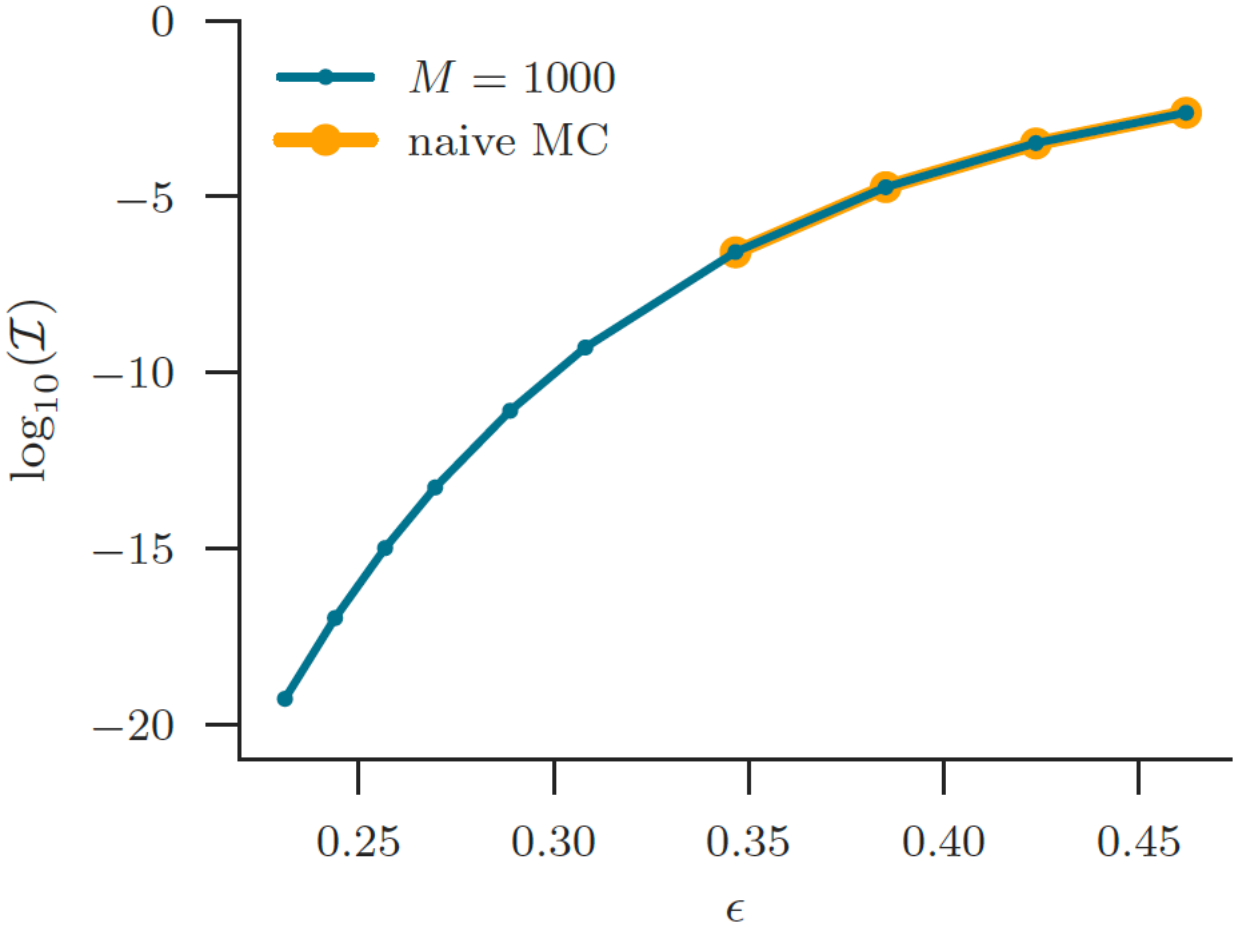
- Projected Gradient Descent
- Measuring Robustness
  - Metric
  - Adaptive Multi-Level Splitting
  - **Results**

Experimental Setup

Input model is ε-perturbation of an image

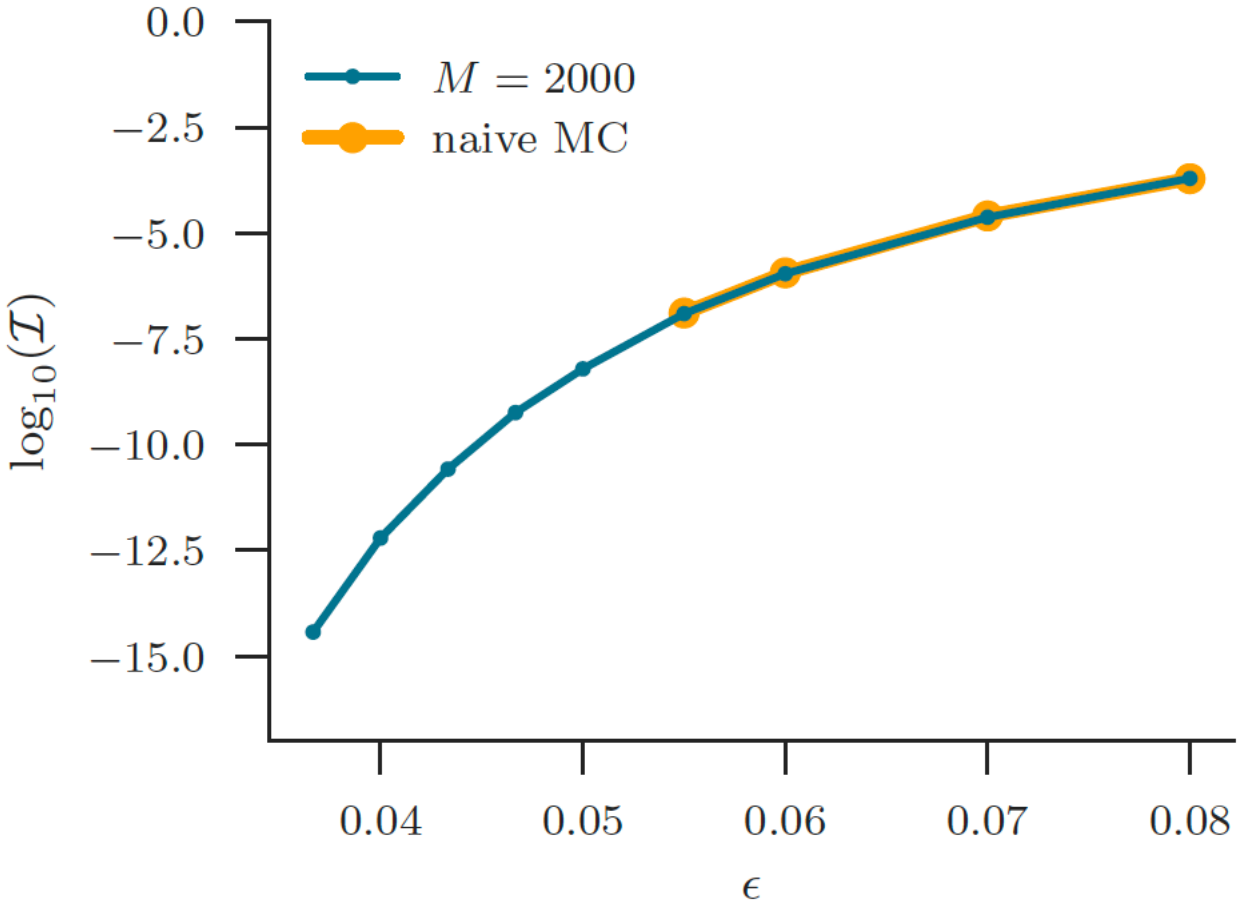
Property corresponds to adversarial examples

2 Layer MLP with MNIST



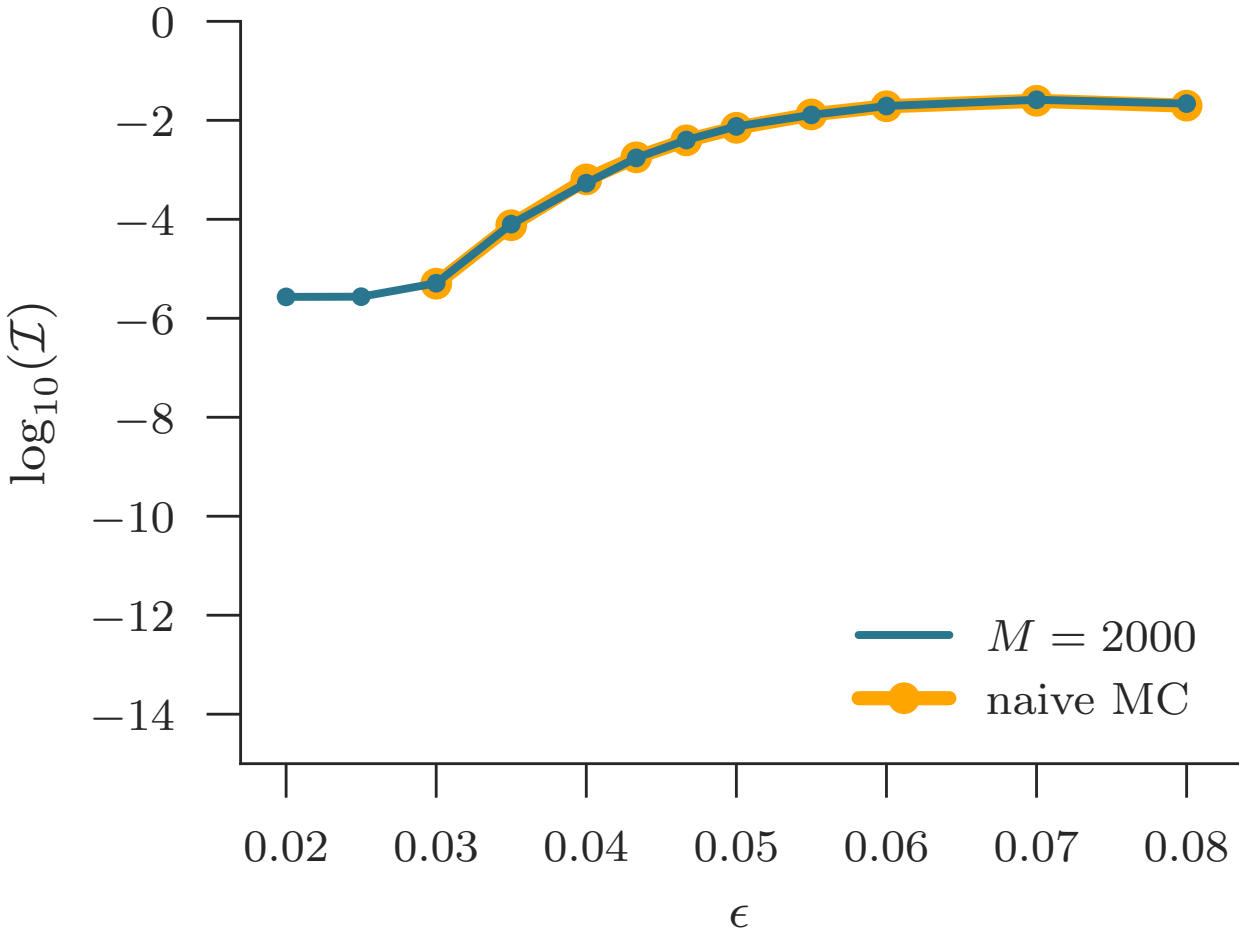
Naïve Monte-Carlo uses 10^{10} samples

2 Layer MLP with CIFAR-10



Naïve Monte-Carlo uses 10^{10} samples

DenseNet with CIFAR-100



Naïve Monte-Carlo uses 10^{10} samples

Questions?

References for other unsound methods on tutorial webpage