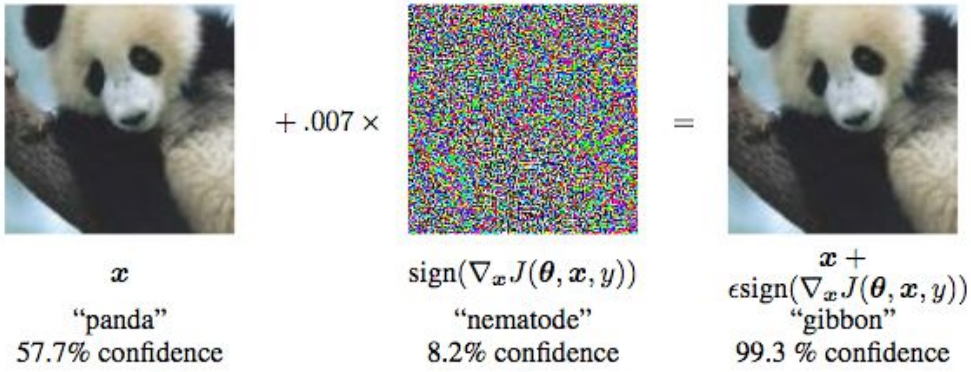
Explaining and Harnessing Adversarial Examples

Ian J. Goodfellow, Jonathon Shlens, & Christian Szegedy

Presented by - Kawin Ethayarajh and Abhishek Tiwari

Introduction

- **adversarial examples**: Inputs formed by applying small but worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence.



Motivation

- a wide variety of models with different architectures trained on different subsets of the training data misclassify the same adversarial example
- causes of adversarial examples a mystery: Extreme non-linearity of NNs? Insufficient model averaging? Insufficient regularization?
- suggests that classifiers based on most ML techniques are not learning the true underlying concepts that determine the correct output label
  - models do well on naturally occurring data, but fail for points **x** where P(**x**) is very low
- potential for use in adversarial training

Linear Explanation of Adv. Examples

- Let adversarial input x' = x + η for some input x.
- For a classifier **F**, we expect **F**(x) = **F**(x') if $||\eta||_\infty < \varepsilon$, for ε small enough to be discarded by the sensor or data storage.
- Dot product of weight w and an adversarial example x' is $w^\top x + w^\top \varepsilon$ (i.e., activation grows by $w^\top \varepsilon$).
  - Put another way, activation grows by εmn, where n is the dimensionality of w, and m is the average magnitude of a weight.
- **A simple linear model can have adversarial examples if its input has sufficient dimensionality.**

Linear Perturbation of Non-Linear Models

- LSTMs, ReLUS, and maxout networks are all designed to behave in highly linear ways, so that they are easier to optimize.
  - More nonlinear models such as sigmoid networks are tuned to spend most of their time in the non-saturating, more linear regime for the same reason.
- **Fast Gradient Sign Method (FGSM):** $\eta = \varepsilon \operatorname{sign}(\nabla_x J(\theta, \boldsymbol{x}, y))$
  - error rates on MNIST: 99.9% on shallow softmax with 79.3% avg. confidence, 89.4% on maxout with an avg. confidence of 97.6%
- High error rates support the theory that the effectiveness of adversarial examples can be ascribed to model linearity.

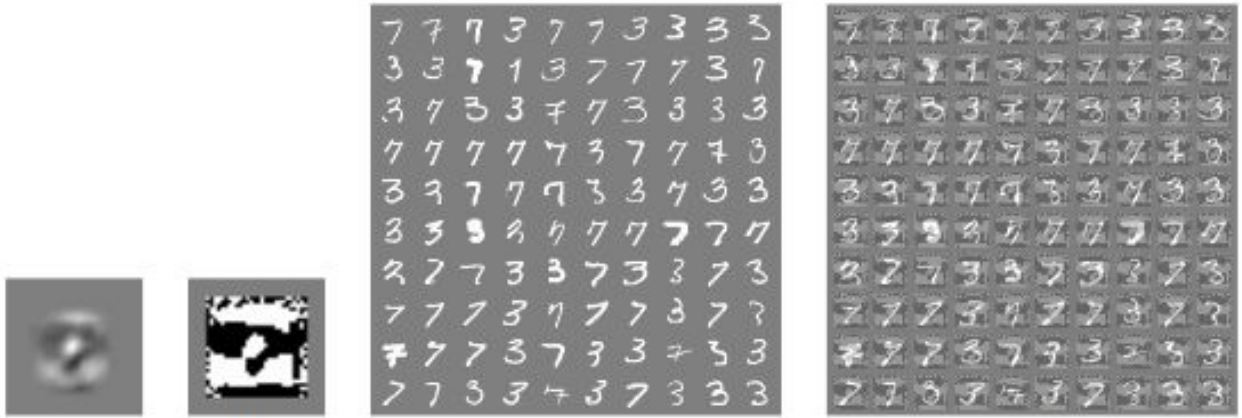
Adversarial Training of Linear Models

- For logistic regression, FGSM is the optimal perturbation method; exact, not just an approximation (increases error rate to 99% on MNIST).
- Adversarial training of logistic regression involves minimizing (where ζ is log(1 + exp(z))):

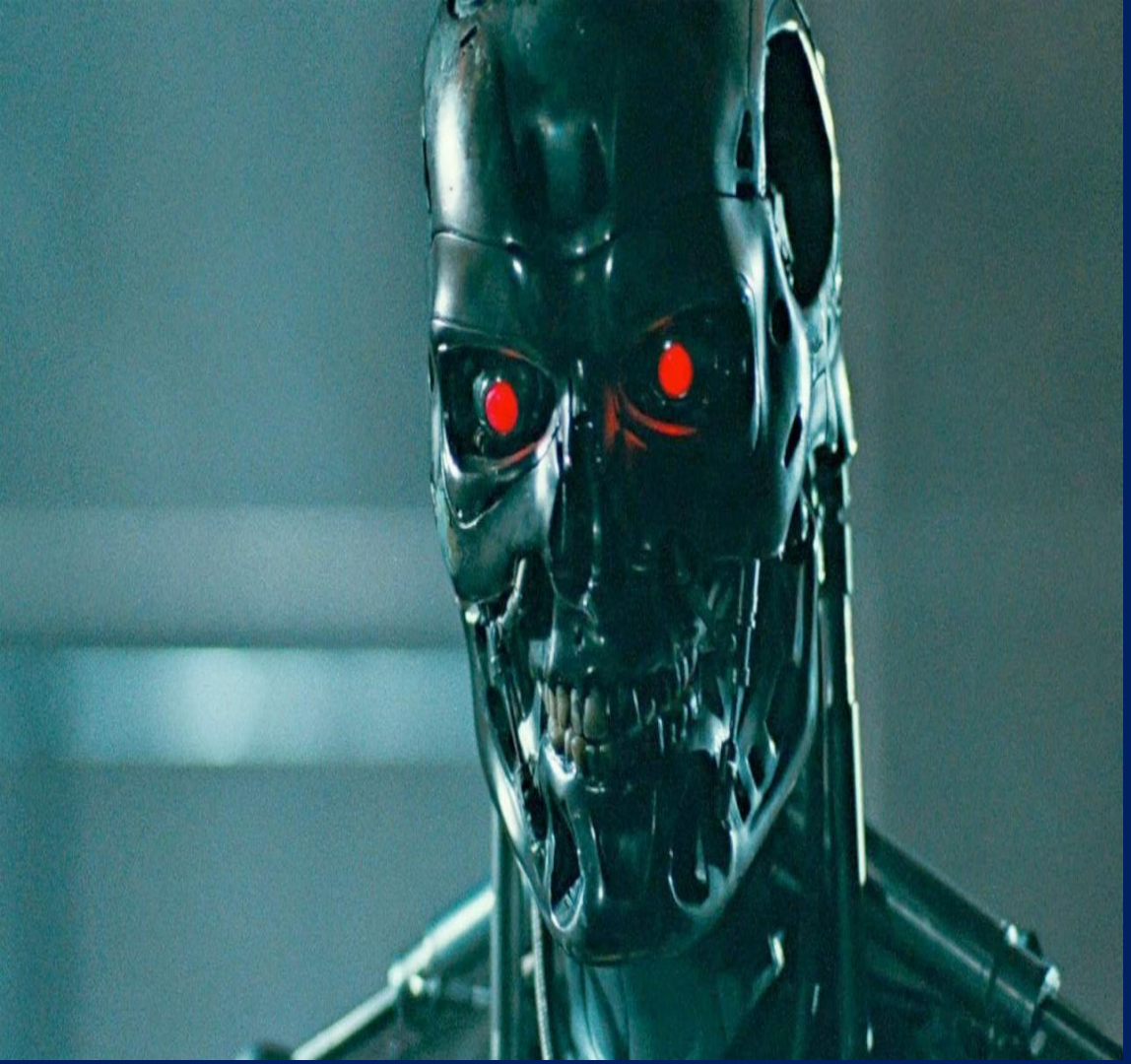
$$\mathrm{E}_{x,y}\, \zeta\left(y\left(\varepsilon \|w\|_1 - w^{\mathsf{T}}x - b\right)\right)$$

- Similar to L1 regularization, but less punitive; penalty effectively disappears when ζ is saturated.
  - When model underfits, adversarial training will simply worsen underfitting.

Adversarial Training of Linear Models (2)



(a) Weights of a logistic regression model trained on MNIST. (b) Sign of those weights (optimal perturbation). (c) MNIST 3's and 7's. (d) FGSM adversarial examples with $\varepsilon = 0.25 \rightarrow$ 99% error rate.



Adversarial Training of Deep Networks

- *Deep networks are vulnerable to adversarial examples*
- *Unlike shallow linear models, deep nets can ATLEAST represent functions robust to adversaries*
- *(Hornik et.al Multilayer feedforward Networks are universal approximators, 1989)*

3. DISCUSSION AND CONCLUDING REMARKS

The results of Section 2 establish that standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense. We have thus established that such "mapping" networks are universal approximators. This implies that any lack of success in applications must arise from inadequate learning, insufficient numbers of hidden units or the lack of a deterministic relationship between input and target.

IDEA 1 (we will see others as well)

- *An obvious choice : train the net with adversarial examples*
- *[Szegedy et.al 2014, Intriguing Properties of Neural Networks] Tried this on MNIST, ImageNet etc.*
- *And it's worth digressing to see their results*

We denote by $f : \mathbb{R}^m \longrightarrow \{1 \dots k\}$ a classifier mapping image pixel value vectors to a discrete label set. We also assume that f has an associated continuous loss function denoted by $\text{loss}_f : \mathbb{R}^m \times \{1 \dots k\} \longrightarrow \mathbb{R}^+$. For a given $x \in \mathbb{R}^m$ image and target label $l \in \{1 \dots k\}$, we aim to solve the following box-constrained optimization problem:

- Minimize $\|r\|_2$ subject to:
  1. $f(x+r) = l$
  2. $x + r \in [0,1]^m$

The minimizer r might not be unique, but we denote one such $x + r$ for an arbitrarily chosen minimizer by $D(x,l)$. Informally, $x + r$ is the closest image to x classified as l by f. Obviously, $D(x, f(x)) = f(x)$, so this task is non-trivial only if $f(x) \neq l$. In general, the exact computation of $D(x,l)$ is a hard problem, so we approximate it by using a box-constrained L-BFGS. Concretely,

Model Name	Description	Training error	Test error	Av. min. distortion
FC10(10^{-4})	Softmax with $\lambda = 10^{-4}$	6.7%	7.4%	0.062
FC10(10^{-2})	Softmax with $\lambda = 10^{-2}$	10%	9.4%	0.1
FC10(1)	Softmax with $\lambda = 1$	21.2%	20%	0.14
FC100-100-10	Sigmoid network $\lambda = 10^{-5}, 10^{-5}, 10^{-6}$	0%	1.64%	0.058
FC200-200-10	Sigmoid network $\lambda = 10^{-5}, 10^{-5}, 10^{-6}$	0%	1.54%	0.065
AE400-10	Autoencoder with Softmax $\lambda = 10^{-6}$	0.57%	1.9%	0.086

Table 1: Tests of the generalization of adversarial instances on MNIST.

	FC10(10^{-4})	FC10(10^{-2})	FC10(1)	FC100-100-10	FC200-200-10	AE400-10	Av. distortion
FC10(10^{-4})	100%	11.7%	22.7%	2%	3.9%	2.7%	0.062
FC10(10^{-2})	87.1%	100%	35.2%	35.9%	27.3%	9.8%	0.1
FC10(1)	71.9%	76.2%	100%	48.1%	47%	34.4%	0.14
FC100-100-10	28.9%	13.7%	21.1%	100%	6.6%	2%	0.058
FC200-200-10	38.2%	14%	23.8%	20.3%	100%	2.7%	0.065
AE400-10	23.4%	16%	24.8%	9.4%	6.6%	100%	0.086
Gaussian noise, stddev=0.1	5.0%	10.1%	18.3%	0%	0%	0.8%	0.1
Gaussian noise, stddev=0.3	15.6%	11.3%	22.7%	5%	4.3%	3.1%	0.3

Table 2: Cross-model generalization of adversarial examples. The columns of the Tables show the error induced by distorted examples fed to the given model. The last column shows average distortion wrt. original training set.

This time around the authors tried something different…

- *Improvement: Training with an adversarial objective function turns out to be an effective regularizer:*

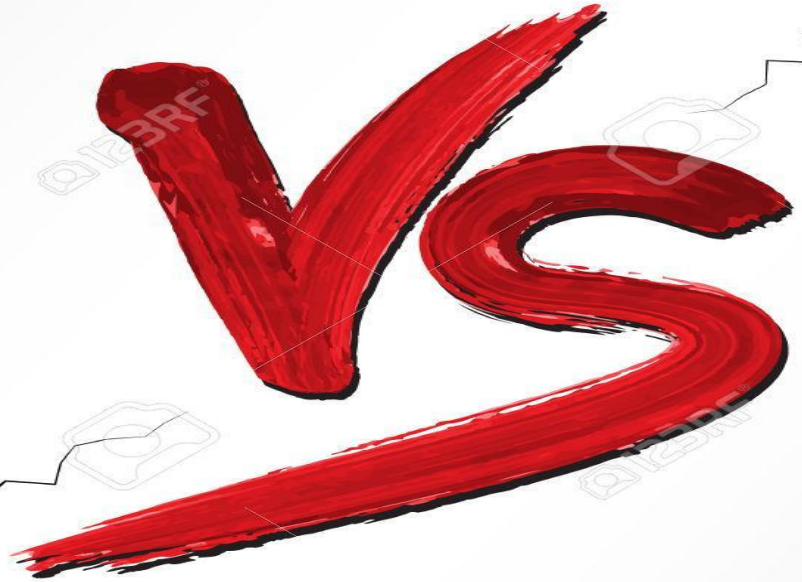
$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))) .$$

- *So we update the Adversarial Examples to make them resist the current model*

- *Result – Error rate went from 0.94% to 0.77% (maxout net with regularisation and early stopping on adversarial validation error set)*

But more Importantly…

- *Previously : misclassified 89.4% of adversarial examples with 97.6% avg confidence*



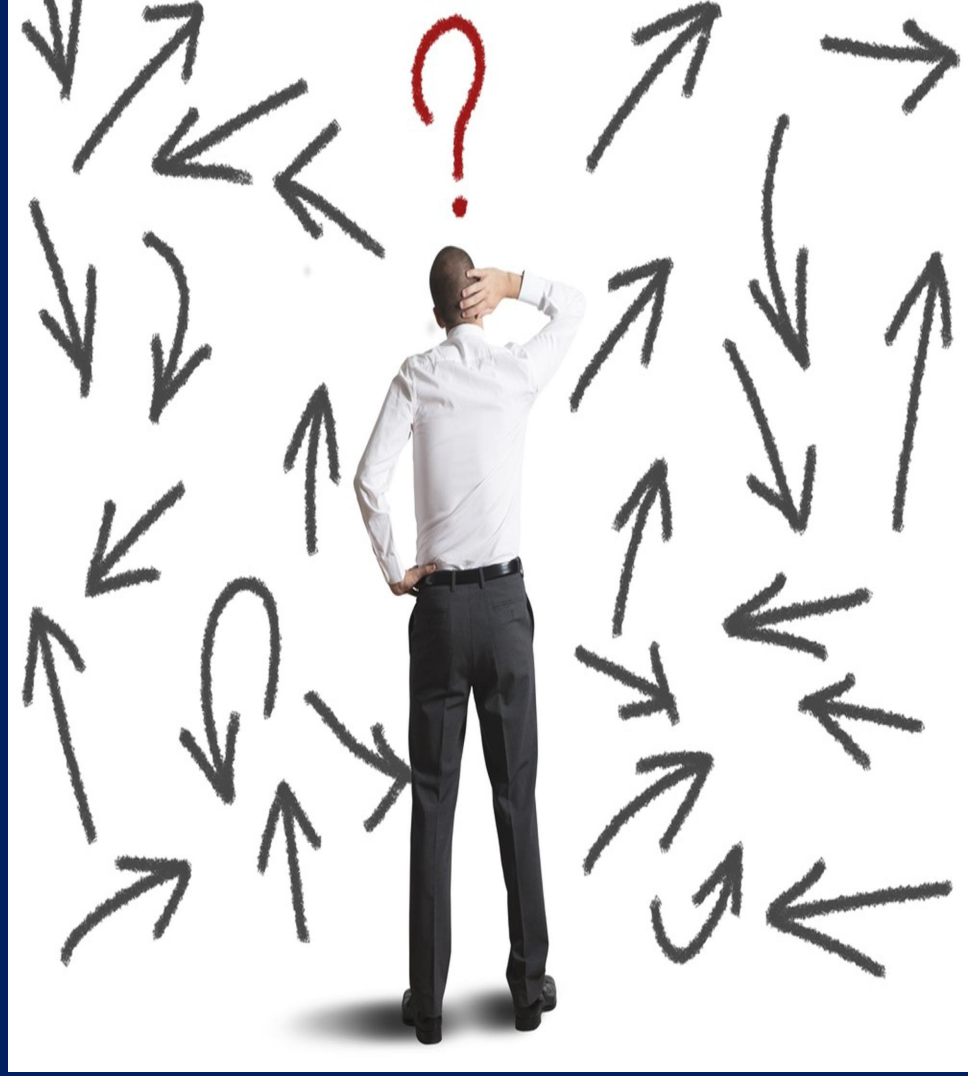
- *Now :*

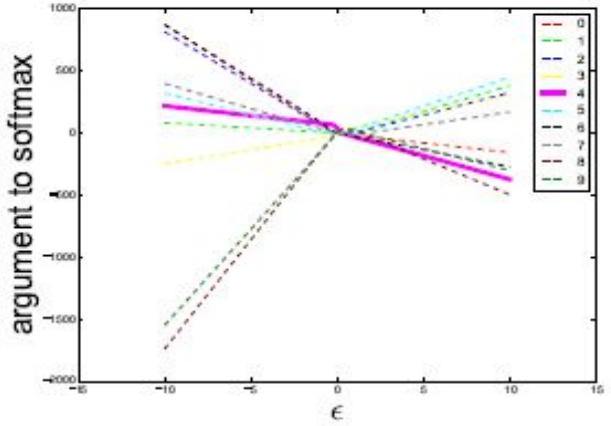
Error rate fell to 17.9%! (GREAT!)

However still with a confidence of 81.4! (Not so great)

Then there are these questions…

- *Why do adversarial examples generalise?*
- *Adversarial example generated for one model gets misclassified by other nets...*
- *Further, the models often agree on the misclassified class.*





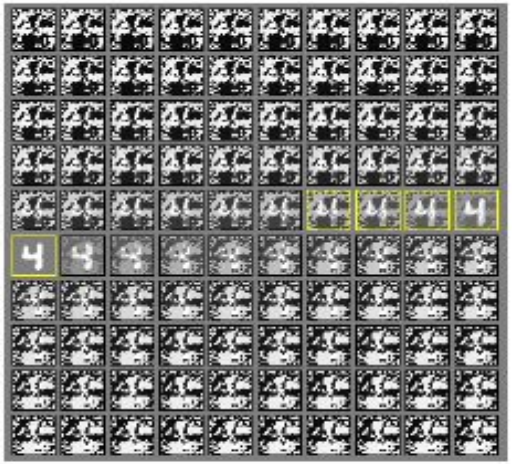


Figure 4: By tracing out different values of ϵ, we can see that adversarial examples occur reliably for almost any sufficiently large value of ϵ provided that we move in the correct direction. Correct classifications occur only on a thin manifold where x occurs in the data. Most of $\mathbb{R}^n$ consists of adversarial examples and *rubbish class examples* (see the appendix). This plot was made from a naively trained maxout network. Left) A plot showing the argument to the softmax layer for each of the 10 MNIST classes as we vary ϵ on a single input example. The correct class is 4. We see that the unnormalized log probabilities for each class are conspicuously piecewise linear with ϵ and that the wrong classifications are stable across a wide region of ϵ values. Moreover, the predictions become very extreme as we increase ϵ enough to move into the regime of rubbish inputs. Right) The inputs used to generate the curve (upper left = negative ϵ, lower right = positive ϵ, yellow boxes indicate correctly classified inputs).

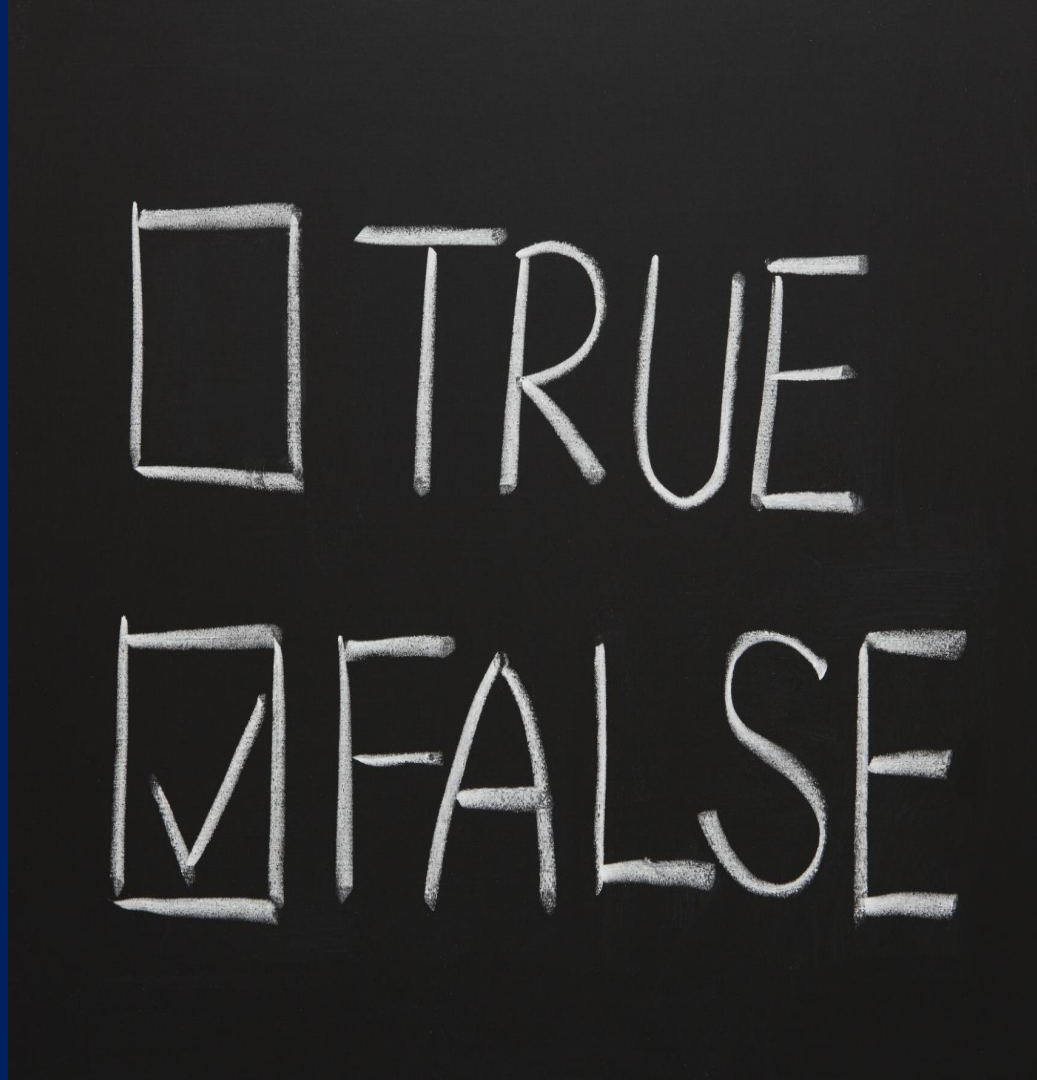
Two Viewpoints

- *1 - Adversarial Examples finely tile the space like Rational Numbers among Real Numbers*
- *2 – (we just saw) Adversarial examples occur in contiguous regions of space (here, space defined by the fast gradient method)*

- *This would explain why adv examples are abundant and why an adv example has fairly high chances of getting misclassified by another classifier*

- *To explain why different classifiers agree on the misclassified class: authors hypothesise that the nets trained with current techniques resemble the same linear classifier learned on the same set*

Next the authors debunk some other hypotheses



- *“Generative models can be more robust to adversarial examples than discriminative models” : Took MP-DBM model showed 95% error rate on adversarial examples, debunking this theory*

- *“Ensembles can ‘wash out’ Adversarial Examples” : Trained 12 maxout nets on MNIST still got 91% error rate on adversarial examples (87% when examples were generated for just one net)*

- *The generalization of adversarial results from adversarial distortion being highly aligned with the weight vectors of a model, different models learning similar functions.*

- *The fact that these simple, cheap algorithms are able to generate adv examples serves as evidence that they are a result of linearity rather than non linearity.*

That’s all, the next group will now take it forward from here!

Summary

- Adversarial examples can be explained as a result of high-dimensional dot products; result of too much linearity, not non-linearity.
  - Direction of perturbation, not specific point in space, matters most.
- Generalization of adversarial attacks due to different models learning similar functions for a given task.
- FGSM is a fast and effective way of generating adversarial examples.
- Adversarial training result in regularization (even more than dropout).
- Linear models lack capacity to resist adversarial perturbation; only structured with a hidden layer can be trained to do so.

Adversarial Examples for Generative Models

Jernej Kos, Ian Fischer, Dawn Song
Presenters: Atef Chaudhury, Brandon Zhao, Kevin Shen

Overview

We have already seen from past papers that discriminative models suffer from adversarial examples

This paper looks at how generative models are also susceptible to adversarial examples

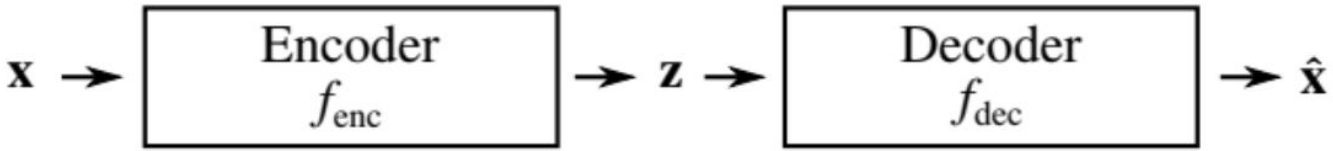
Rest of Presentation

1. Quick review of VAEs
2. Motivating scenario for an adversarial attack on generative models
3. The three attack methods described by the paper, and their underlying mechanism
4. The results of these attacks
5. Areas for future work

Quick Review of Variational Autoencoders

VAEs sample a latent space to generate examples from a distribution of interest

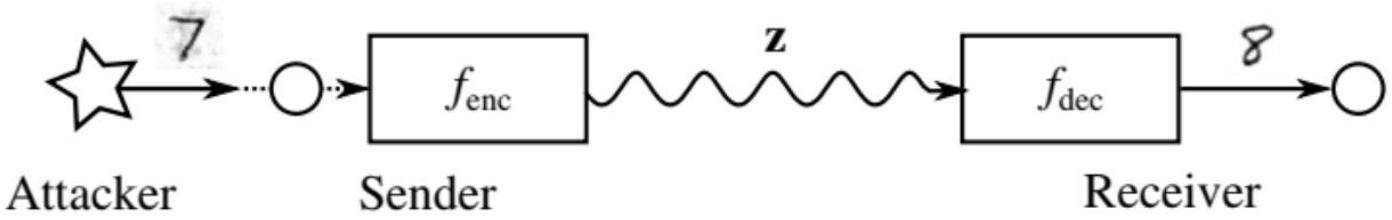
- Learn an encoder function (typically an NN) to map high-dimensional input *x* to low-dimensional latent space *z*
- Learn a decoder function (also an NN) to map back from latent space to high-dimensional output



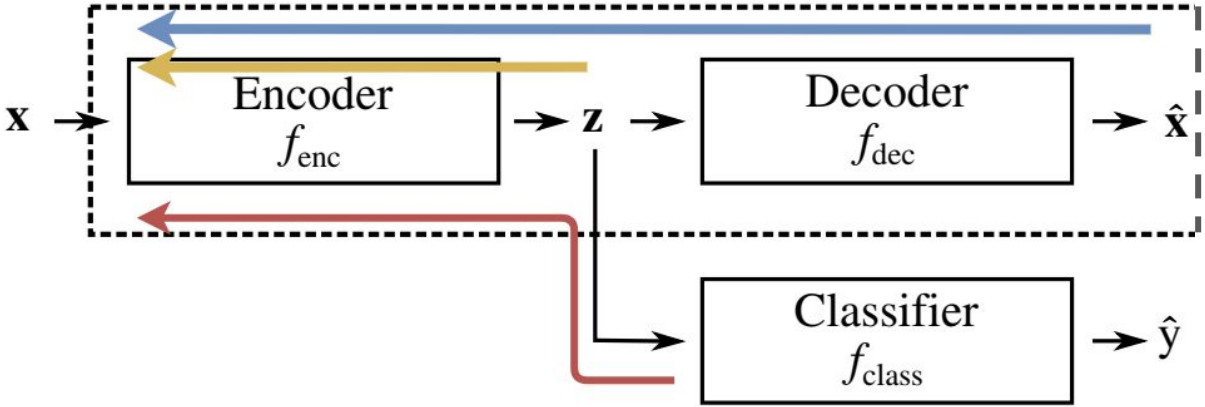
Motivating Scenario

VAEs can be used as compressed communication channels

What if an adversary tricks the sender into transmitting an input that resembles something entirely different once it is reconstructed

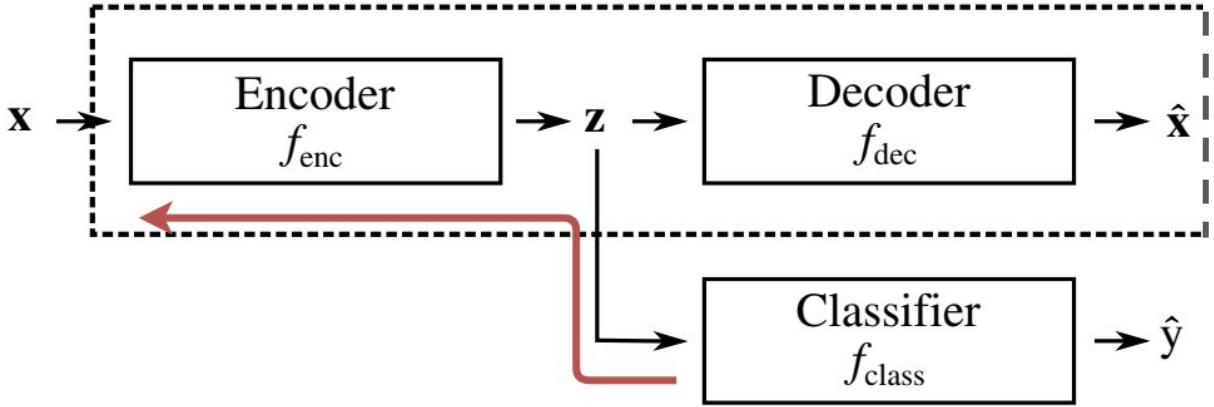


Attacks



Red: Classifier Attack
Yellow: Latent Attack
Blue: VAE Attack

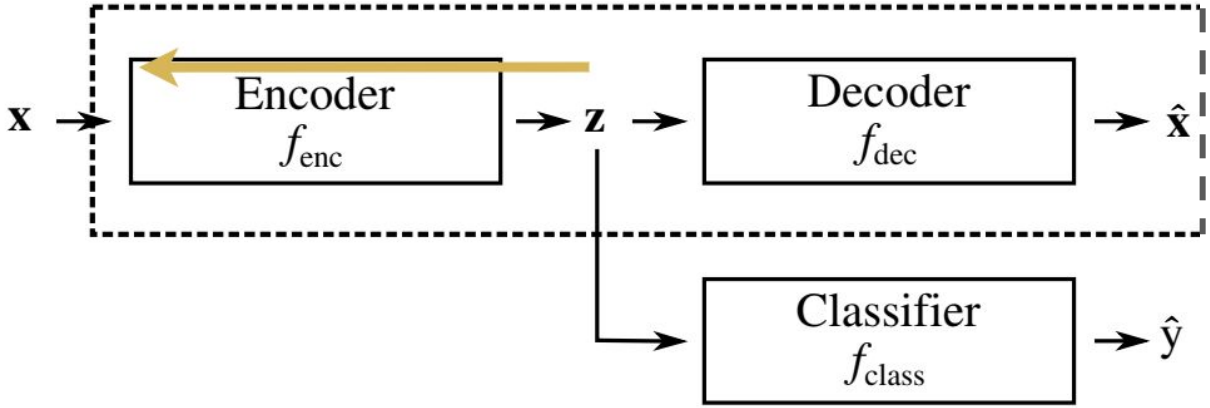
Classifier Attack



$$\arg\min_{x^*} \; L(x^*, y_t) + \lambda \, \|x - x^*\|$$

$$L(x^*, y_t) = \text{CrossEntropy}(f_{\text{class}}(z^*), y_t)$$

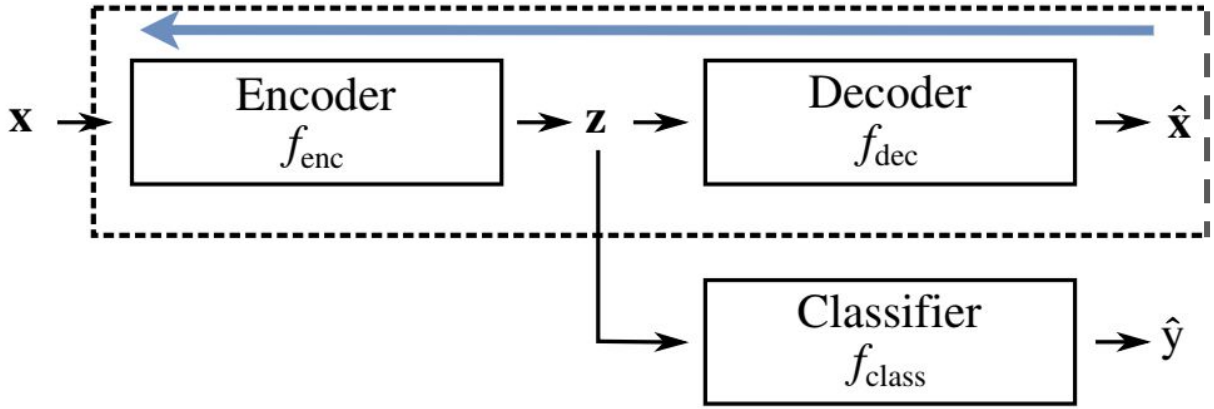
Latent Attack



$$\arg\min_{x^*} \; L(x^*, z_t) + \lambda \|x - x^*\|$$

$$L(x^*, z_t) = \|z_t - z^*\|^2$$

VAE Attack



$$\arg\min_{x^*} \; L(x^*, x_t) + \lambda \left\| x - x^* \right\|$$

$$L(x^*, x_t) = L_{\text{vae}}(x^*, \tilde{x}_t) = -D_{\text{KL}}[q(z|x^*)||p(z)] + E_q[\log p(\tilde{x}_t|z)]$$

$L(x^*, z_t)$	$L(x^*, y_t)$	$L(x^*, x_t)$
Image or class level	Class level	Image level
Most effective attacks	Reconstructions are bad	Computationally expensive
	Adversary needs labels for images	

Evaluation Setup

- A separate classifier is used to evaluate the accuracy of the reconstructions.
- A reconstruction feedback mechanism (i.e. pass the reconstructed image back through the encoder) is used to improve the accuracy of this classifier.

$$\hat{y} = f_{\text{class}}(f_{\text{enc}}(\hat{\mathbf{x}}^*)).$$

Evaluation Metrics

Based on the classifier output, two metrics were computed

1) attack success rate ignoring targeting

$$AS_{ignore-target} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\hat{y}^i \neq y^i}$$

2) attack success rate including targeting

$$AS_{target} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\hat{y}^i = y_t^i}.$$

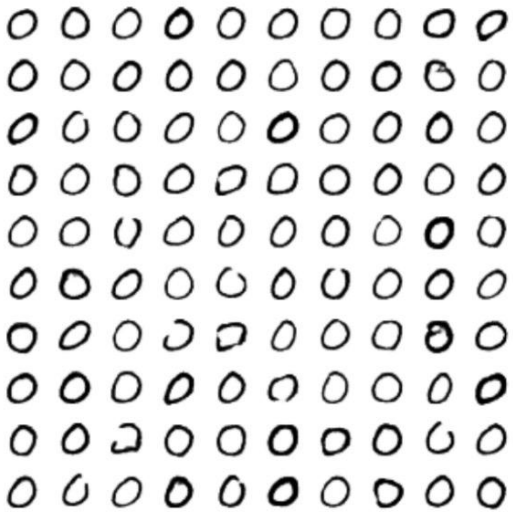
Metrics Evaluation for MNIST Classifier attack

Source	Target 0	Target 1	Target 2	Target 3	Target 4	Target 5	Target 6	Target 7	Target 8	Target 9
0	-	40.96% (1.20%)	6.02% (4.82%)	10.84% (7.23%)	75.90% (0.00%)	6.02% (3.61%)	28.92% (28.92%)	37.35% (20.48%)	6.02% (1.20%)	10.84% (3.61%)
1	99.20% (77.60%)	-	7.20% (5.60%)	1.60% (1.60%)	85.60% (0.00%)	8.00% (5.60%)	28.80% (28.00%)	8.80% (7.20%)	3.20% (1.60%)	69.60% (0.80%)
2	85.96% (80.70%)	3.51% (2.63%)	-	29.82% (23.68%)	78.95% (0.00%)	72.81% (20.18%)	72.81% (46.49%)	35.09% (8.77%)	41.23% (12.28%)	68.42% (2.63%)
3	93.46% (83.18%)	26.17% (12.15%)	27.10% (16.82%)	-	67.29% (0.00%)	66.36% (62.62%)	87.85% (22.43%)	50.47% (27.10%)	23.36% (8.41%)	33.64% (8.41%)
4	100.00% (82.73%)	70.00% (48.18%)	28.18% (10.91%)	84.55% (17.27%)	-	66.36% (31.82%)	95.45% (71.82%)	62.73% (37.27%)	20.91% (0.91%)	51.82% (44.55%)
5	93.10% (89.66%)	21.84% (1.15%)	68.97% (11.49%)	28.74% (18.39%)	3.45% (0.00%)	-	20.69% (19.54%)	80.46% (41.38%)	22.99% (2.30%)	44.83% (12.64%)
6	29.89% (28.74%)	44.83% (1.15%)	24.14% (3.45%)	59.77% (11.49%)	77.01% (0.00%)	10.34% (8.05%)	-	62.07% (8.05%)	8.05% (0.00%)	75.86% (4.60%)
7	79.80% (65.66%)	77.78% (26.26%)	20.20% (8.08%)	8.08% (4.04%)	100.00% (0.00%)	56.57% (23.23%)	97.98% (17.17%)	-	38.38% (1.01%)	17.17% (10.10%)
8	94.32% (84.09%)	96.59% (18.18%)	60.23% (42.05%)	57.95% (43.18%)	100.00% (0.00%)	93.18% (80.68%)	100.00% (57.95%)	100.00% (34.09%)	-	87.50% (26.14%)
9	98.91% (79.35%)	97.83% (33.70%)	26.09% (1.09%)	17.39% (2.17%)	100.00% (0.00%)	22.83% (21.74%)	100.00% (30.43%)	47.83% (43.48%)	31.52% (4.35%)	-

MNIST: Successful Latent Attack

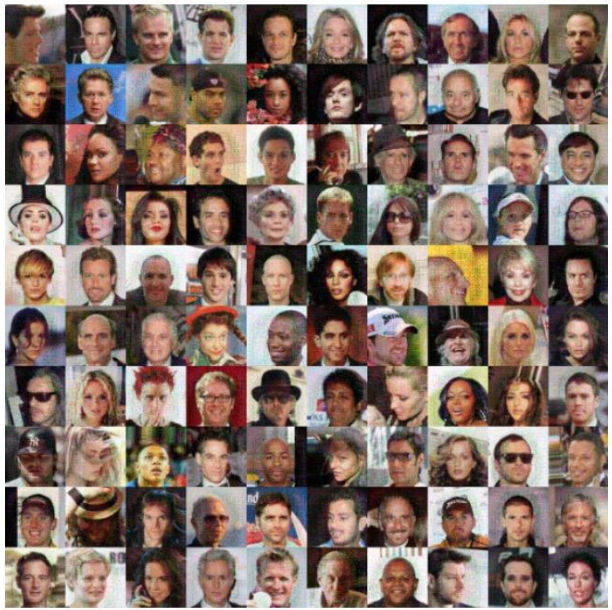
Adversarial Examples

Adversarial Reconstructions

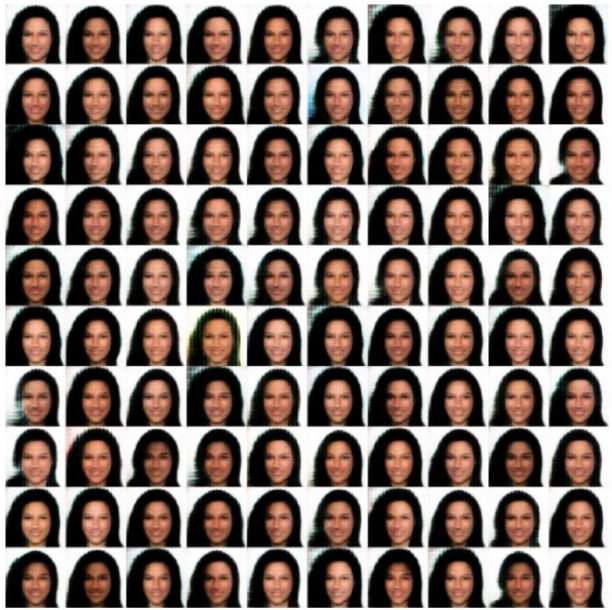


CelebA: Successful Latent Attack

Adversarial Examples

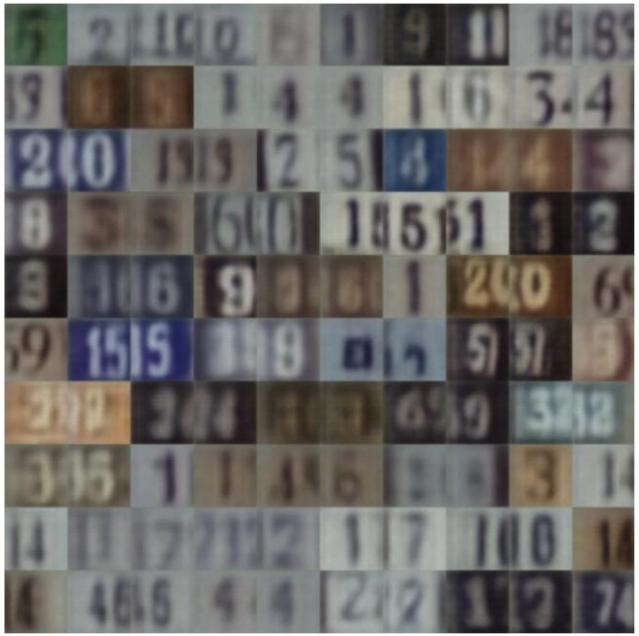


Adversarial Reconstruction



SVHN: Failed L_{vae} Attack

Reconstructions from L_{vae} Attack



Reconstructions from Latent Attack

Future works and Relevant Papers

1. Attacks on natural image dataset such as CIFAR-10 or Imagenet
2. Defence and robustification against these attacks

MagNet: a Two-Pronged Defense against Adversarial Examples

- They use VAEs to detect and fix adversarial examples for a classifier (which may not work if you know how to attack the VAEs in the first place)

Adversarial Images for Variational Autoencoders

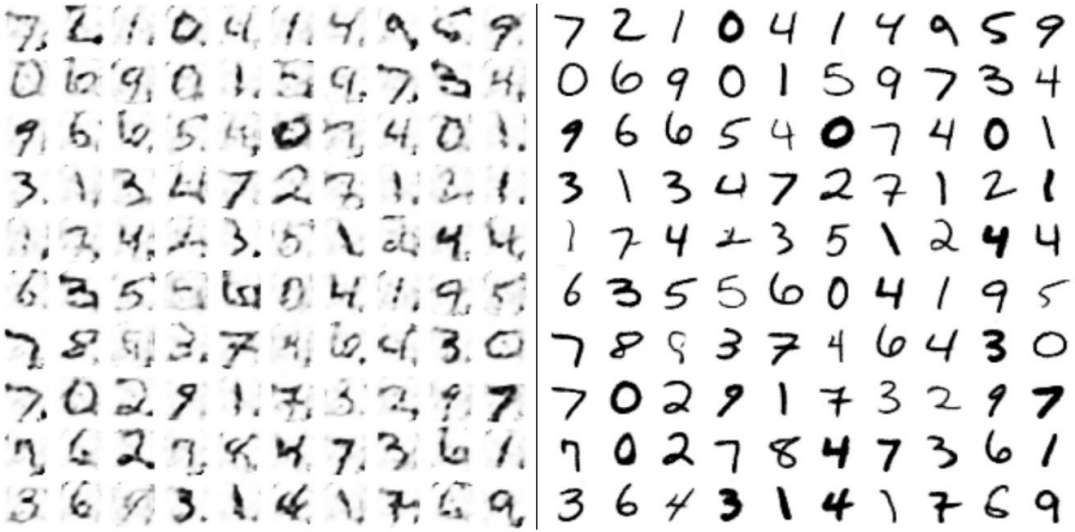
- Original VAE attack paper

Appendix

MNIST: Failed FGS optimization

VAE Reconstructions

VAE-GAN Reconstructions



Possible hypothesis for why adversarial attacks work

- (although this paper won't explore them, good to keep in mind)
- Posteriors of training examples tend to clump together, why do adversarial examples work?
  - Insufficient posterior: gaps that are not being filled by the posteriors of different data points, q is a poor approximation to posterior
  - Interpolation between the means of the posteriors of two datapoints is adversarial
  - Adversary exploits the architecture of the neural network (i.e. it's the sample problem as the classifier)

Additional details

- In all attacks, they train with mean latent z from encoder, they do not sample
- they blame bad reconstruction for y-attack on classifier inaccuracy but it’s probably because adversarial z’s in classifier’s input space do not correspond to actual images

Evaluation Criteria

1. Loss Type: **Classifier** versus $\mathbf{L_{VAE}}$ versus **Latent**

2. Optimization type: $\mathbf{L_2}$ **Optimization** versus **FGS**

Potentially relevant papers

Cited

- Adversarial Images for Variational Autoencoders
  - They did a subset of what this paper did, results are not very important

Cited by

- MagNet: a Two-Pronged Defense against Adversarial Examples
  - They use VAEs to detect and fix adversarial examples for a classifier (which may not work if you know how to attack the VAEs in the first place)

Limitations of Deep Learning in Adversarial Settings

Paper by: Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami

Presented by: Ramin Hamedi and Matthew MacKay

Presentation Summary

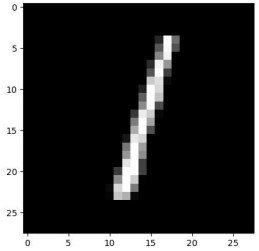
1. Threat model taxonomy
2. Generic algorithm to construct adversarial examples
3. Application of algorithm to MNIST
4. Metrics to evaluate attack’s effectiveness

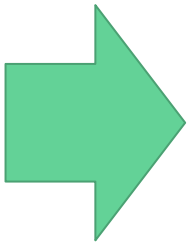
Threat Model Taxonomy

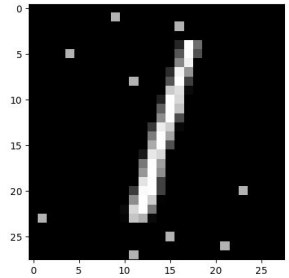
- Adversary seeks to provide an input to a deep learning classifier causing undesired behavior
- Adversarial Goals:
  - What behavior is adversary trying to elicit?
- Adversarial Capabilities:
  - What information can adversary use to attack our system?

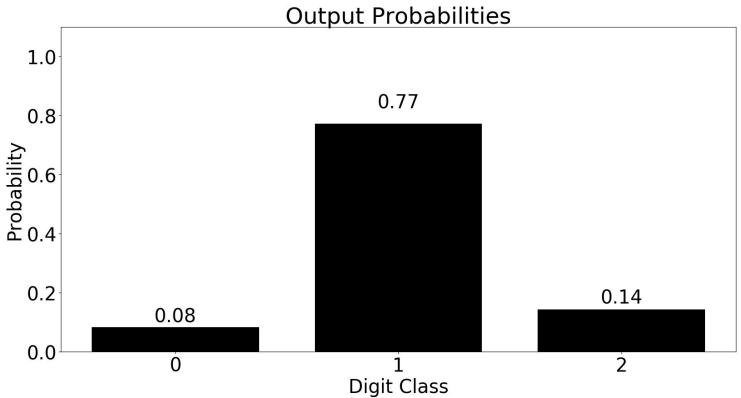
Adversarial Goals

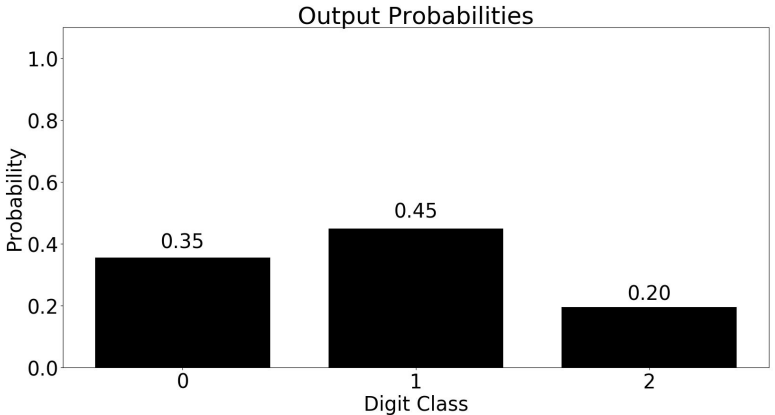
1. Confidence Reduction: reduce output confidence classification





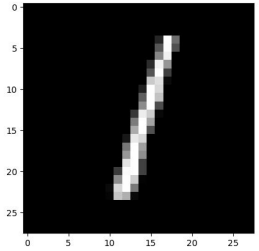


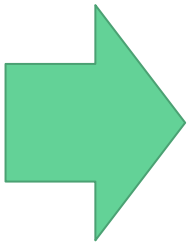


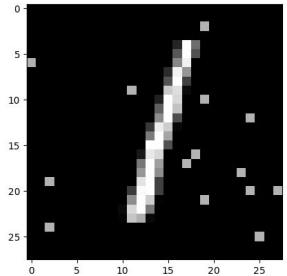


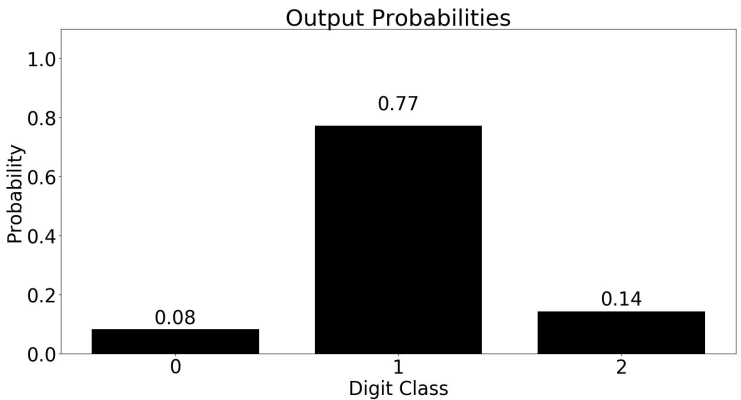
Adversarial Goals

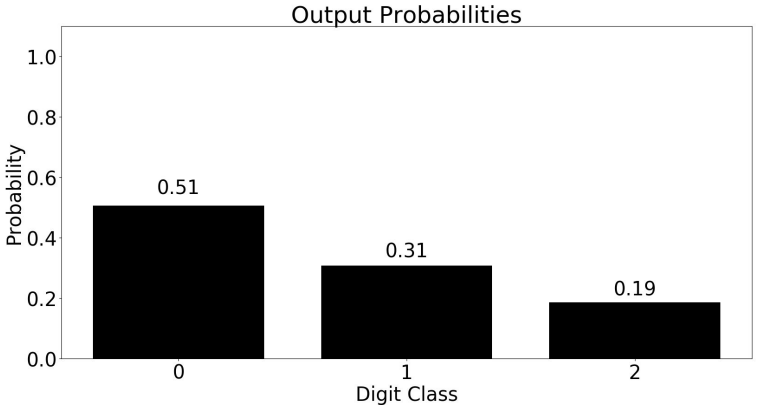
2. Misclassification: perturb existing image to classify as any incorrect class





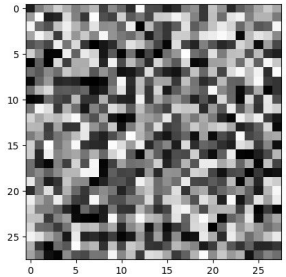


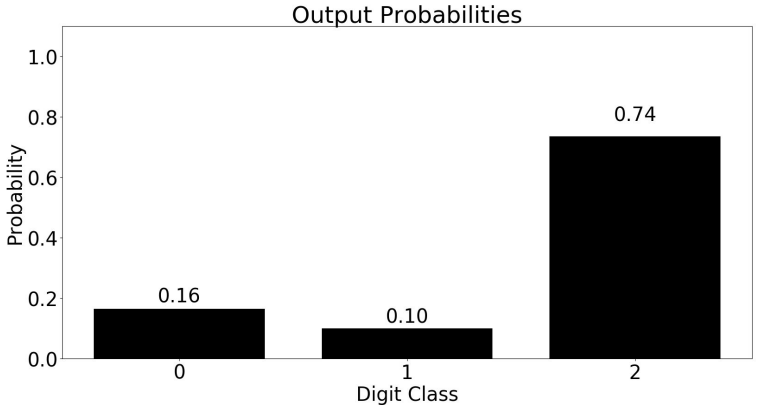




Adversarial Goals

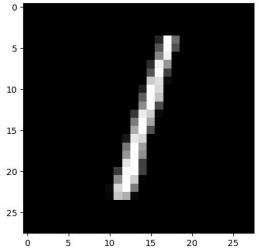
3. Targeted misclassification: produce inputs classified as target class

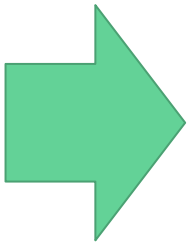


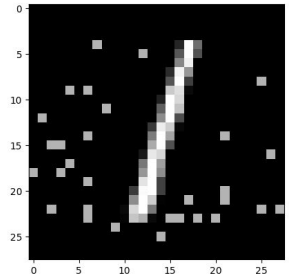


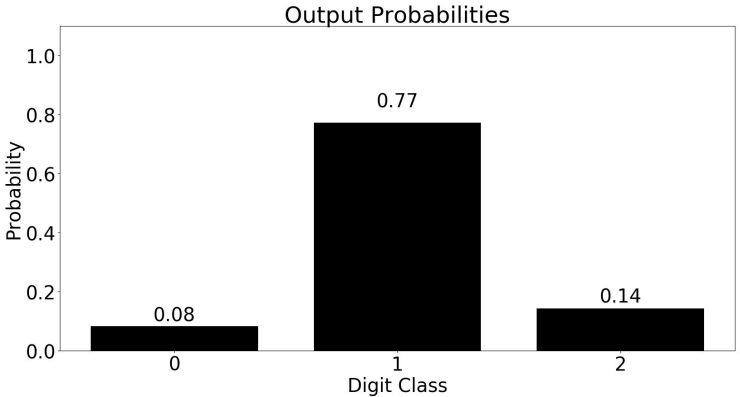
Adversarial Goals

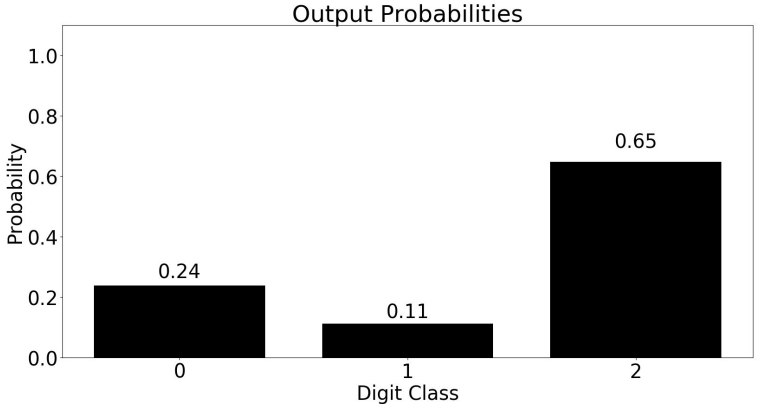
4. Source/target misclassification: perturb existing image to classify as target class











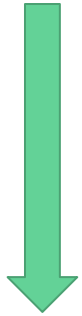
Adversarial Goals (Summary)

Increasing complexity

1. Confidence Reduction: reduce output confidence classification
2. Misclassification: perturb existing image to classify as any incorrect class
3. Targeted misclassification: produce inputs classified as target class
4. Source/target misclassification: perturb existing image to classify as target class

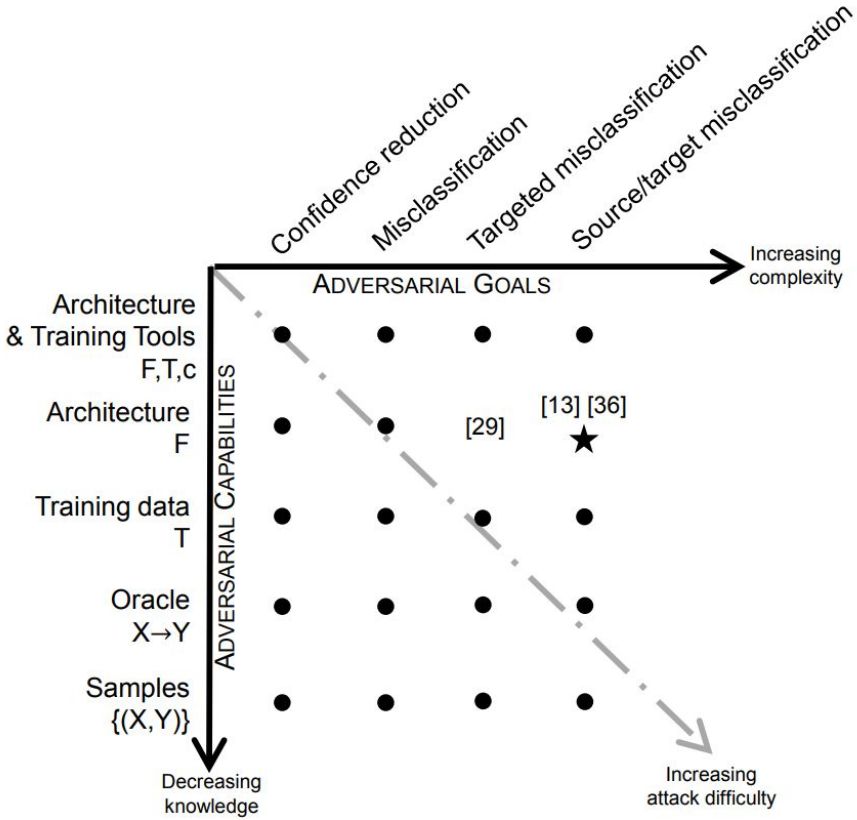
Adversarial Capabilities (Summary)

- What information can adversary use to attack our system?
  1. Training data and network architecture
  2. Network architecture
  3. Training data
  4. Oracle (can see outputs from supplied inputs)
  5. Samples (have inputs and outputs from network but cannot choose inputs)



Decreasing knowledge

Threat Model Taxonomy (Summary)



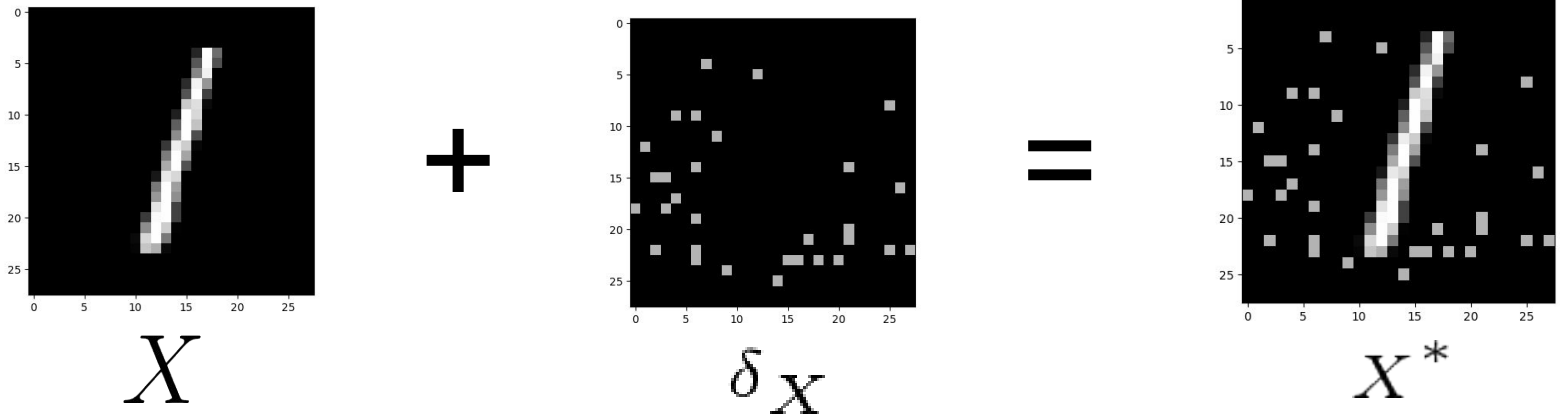
- Adversarial Goals:
  - What behavior is adversary trying to elicit?
- Adversarial Capabilities:
  - What information can adversary use to attack our system?
- In this paper:
  - Goal: Source/target misclassification
  - Capability: Architecture

Formal Problem Definition

- Given a trained neural network $F : \mathbb{R}^D \rightarrow \mathbb{R}^C$ such that $\mathrm{softmax}(F(X))_j = \Pr(class(X) = j)$
- Let $\tilde{F}(X) = \mathrm{argmax}_j \mathrm{softmax}(F(X))_j$

Formal Problem Definition

- Also given: training example X and a target label Y^*
- Goal: Find X^* s.t. $\tilde{F}(X^*) = Y^*$ and X^* similar to X
- More formally: find δ_X satisfying $\delta_X = \text{argmin}_\delta ||\delta|| \text{ s.t. } \tilde{F}(X + \delta) = Y^*$
- Then: set $X^* = X + \delta_X$



Summary of Basic Algorithm

1. Compute the Jacobian matrix of F evaluated at input X
2. Use Jacobian to find which features of input should be perturbed
3. Modify X by perturbing the features found in step 2
4. Repeat while X not misclassified and perturbation still small

Step 1: Compute Jacobian

- Recall $F : \mathbb{R}^D \rightarrow \mathbb{R}^C$
- The Jacobian $\frac{\partial F}{\partial X}(X)$ is defined to be a $C \times D$ matrix such that:

$$\frac{\partial F}{\partial X}(X)[i,j] = \frac{\partial F_i}{\partial X_j}(X)$$

- Note: this is not equivalent to the derivative of the loss function!
- For explicit computation, see paper. Otherwise, just use auto-diff software

Step 2: Construct Adversarial Saliency Maps

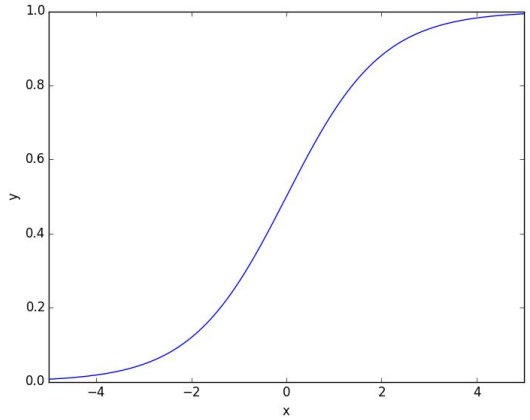
- Set $t = Y^*$. Define an adversarial saliency map $S(X, t)$ by:

$$S(X,t)[i] = \begin{cases} 0 \text{ if } \frac{\partial F_t(X)}{\partial X_i} < 0 \text{ or } \sum_{c \neq t} \frac{\partial F_c(X)}{\partial X_i} > 0 \\ (\frac{\partial F_t(X)}{\partial X_i}) \left| \sum_{c \neq t} \frac{\partial F_c(X)}{\partial X_i} \right| \text{ otherwise} \end{cases}$$

- High value of saliency map correspond to input features that, if increased, will:
  - Increase probability of target class
  - Decrease probability of other classes

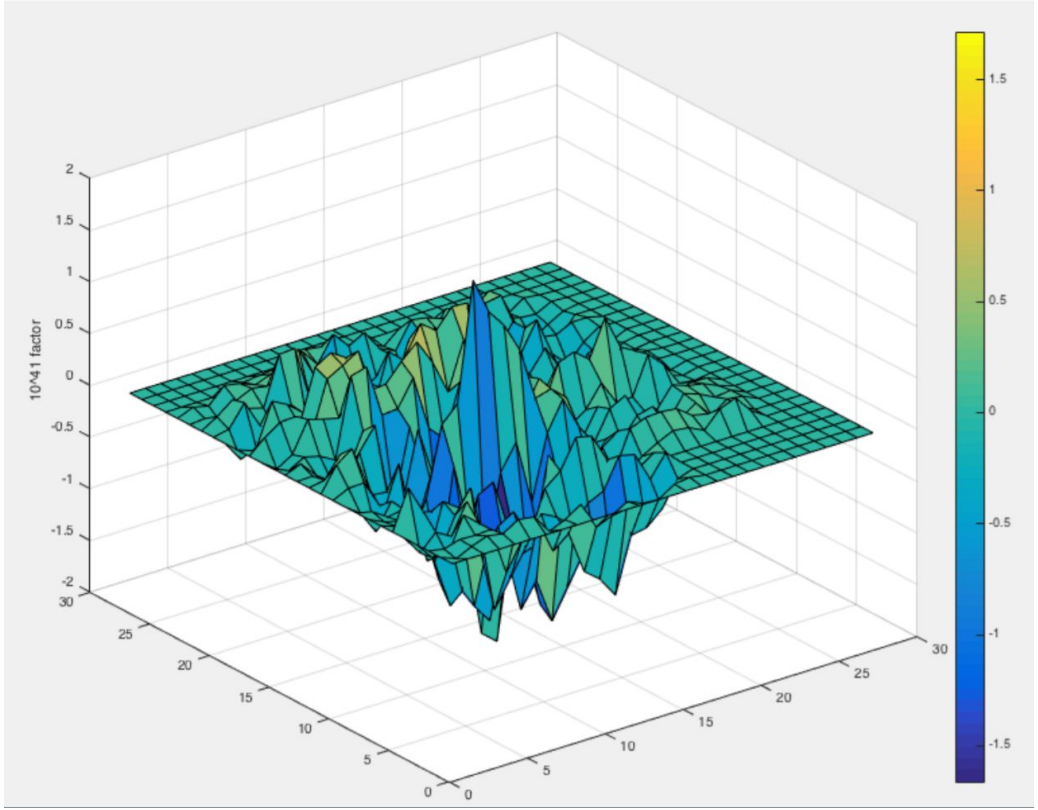
Question: Why not probabilities?

- We could have defined F to be output *after* softmax, not before
- However, doing so leads to extreme derivative values due to squashing needed to ensure probabilities add to 1
- This reduces quality of information about how inputs influence network behavior



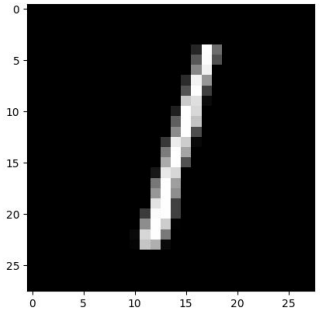
- Binary classification example: sigmoid derivatives vanish in the tails

Saliency Map Example

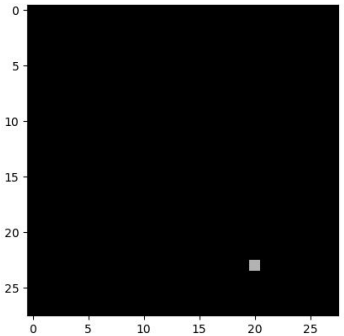


Step 3: Modify input

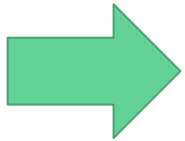
- Choose $i^* = \text{argmax}_i S(X, Y^*)[i]$
- Change current input by setting $X^*[i] = X^*[i] + \theta$
- θ is problem specific perturbation amount (later will discuss how to set)

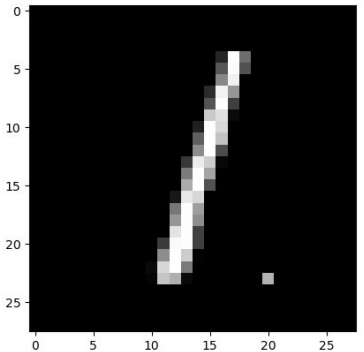


BEFORE



i^*





AFTER

Application of Approach to MNIST

- Assume attacker has access to trained model
- In this case: LeNet architecture trained on 60000 MNIST samples
- Objective: Change a limited number of pixels on input X, originally correctly classified so network misclassifies as target class Y^*

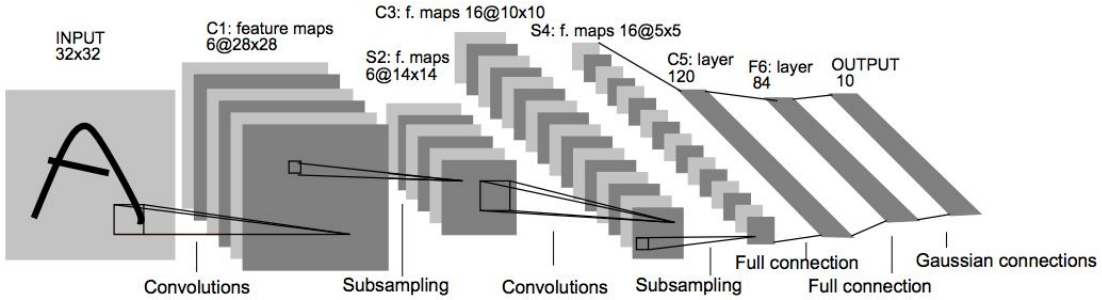


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Practical Considerations

- Set perturbation amount θ to 1 (turning pixel completely on) or -1 (turning completely off)
  - If θ an intermediate value, more pixels need to be changed to misclassify
- Once a pixel reaches zero or one, we need to stop changing them
  - Keep track of candidate set of pixels Γ to perturb on each iteration
- Very few individual pixels have saliency map value greater than 0
  - Instead consider two pixels at a time (see paper for changed saliency map)

Practical Considerations (continued)

- Quantify maximum distortion Υ by allowable percentage of modified pixels (e.g. $\Upsilon = 5$)
- The maximum number of iterations will be:

$$\texttt{max\_iter} = \left\lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \right\rfloor$$

- Note: two is in denominator because we are tweaking two pixels per iteration

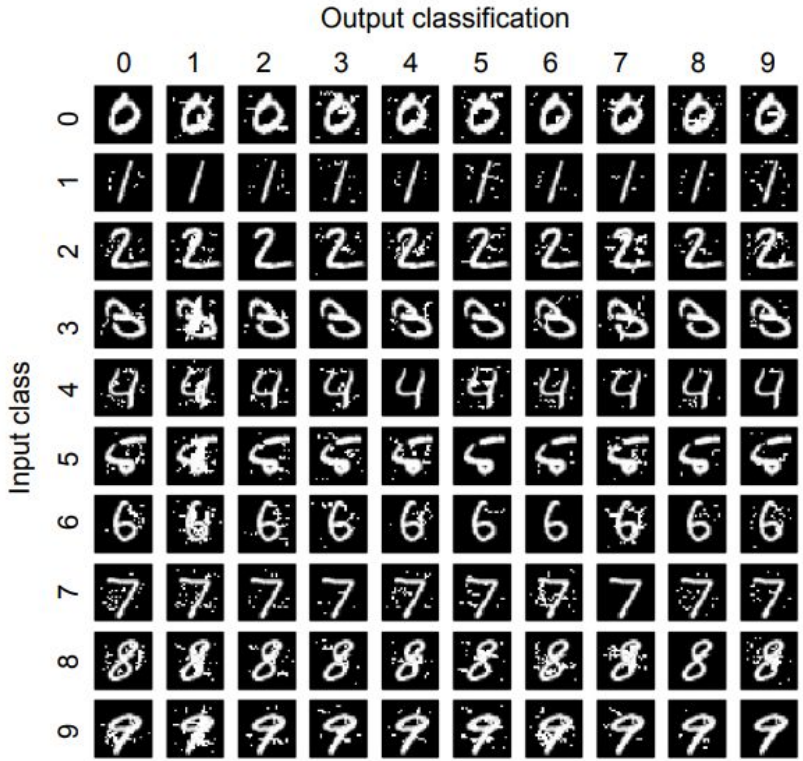
Formal Algorithm for MNIST

Input: $X, Y^*, F, \Upsilon, \theta$

1. Set $X^* = X$, $\texttt{max\_iter} = \left\lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \right\rfloor$, $\Gamma = \{1, \ldots, 784\}$
2. while $\tilde{F}(X^*) \neq Y^*$ and $\texttt{iter} < \texttt{max\_iter}$ and $\Gamma \neq \emptyset$:
3. Compute Jacobian matrix $\frac{\partial F}{\partial X}(X^*)$
4. Compute modified saliency map $S(X^*, Y^*, \Gamma)$ for two pixels
5. Find two “best” pixels p_1, p_2 and remove them from Γ
6. Set $X^*[p_i] = X^*[p_i] + \theta$
7. Increment $\texttt{iter}$
8. Return

Results for Empty Input

Samples created by increasing intensity



Success Rate and Distortion

- Success rate: percentage of adversarial samples that were successfully classified by the DNN as the adversarial target class
- Distortion: percentage of pixels modified in the legitimate sample to obtain the adversarial sample
- Two distortion values computed: one taking into account all samples and a second one only taking into account successful samples

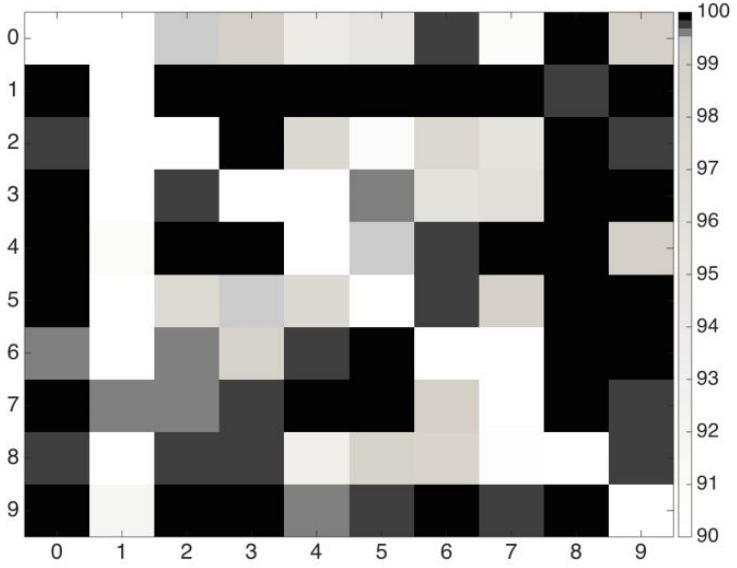
Results

Source set of 10,000 original samples	Adversarial samples successfully misclassified	Average distortion	
		All adversarial samples	Successful adversarial samples
Training	97.05%	4.45%	4.03%
Validation	97.19%	4.41%	4.01%
Test	97.05%	4.45%	4.03%

- Table shows results for increasing pixel features

Source-Target Pair Metrics

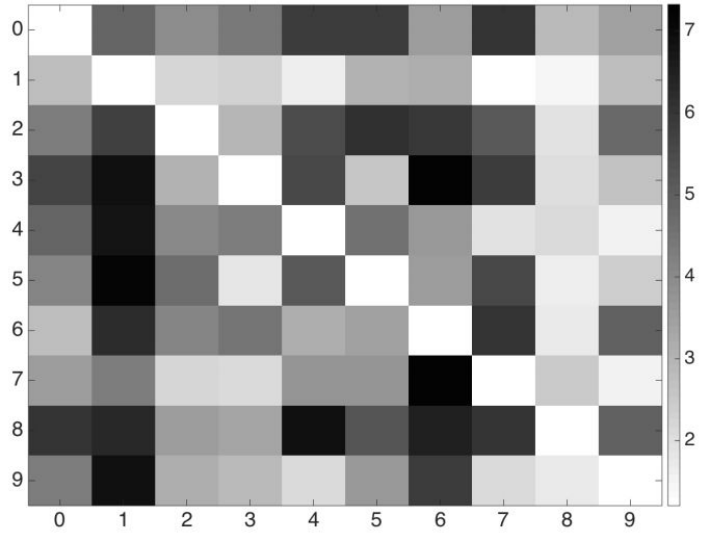
Target



Source

Success rate per source-target class pair.

Target



Source

Average distortion ε of successful samples

Hardness Matrix

- Can we quantify how hard it is to convert different source-target class pairs?
- Define:
  - τ : success rate
  - $\varepsilon(s,t,\tau)$: average distortion required to convert class s to class t with success rate τ

$$H(s,t) = \int_{\tau} \varepsilon(s,t,\tau)\mathrm{d}\tau$$

- In practice: obtain (ϵ_k, τ_k) pairs for specific maximum distortions Υ_k (average over 9000 adversarial samples)
- Then estimate as:

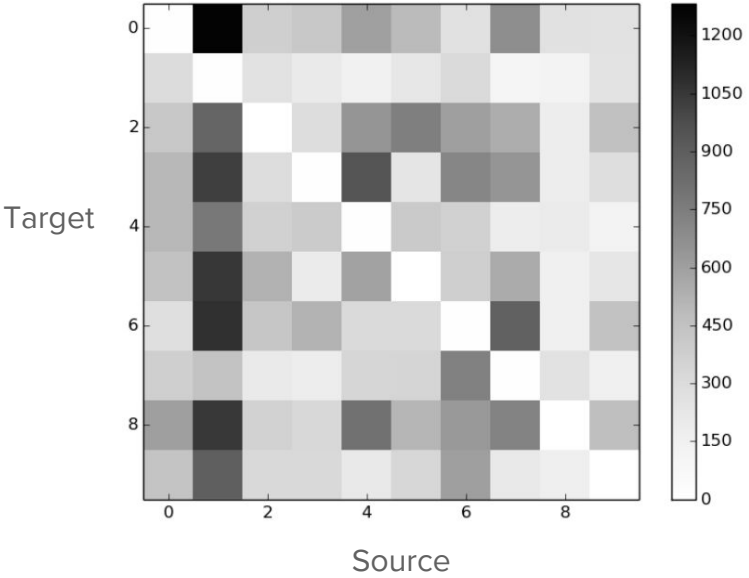
$$H(s,t) \approx \sum_{k=1}^{K-1} (\tau_{k+1} - \tau_k) \frac{\varepsilon(s,t,\tau_{k+1}) + \varepsilon(s,t,\tau_k)}{2}$$

Adversarial Distance

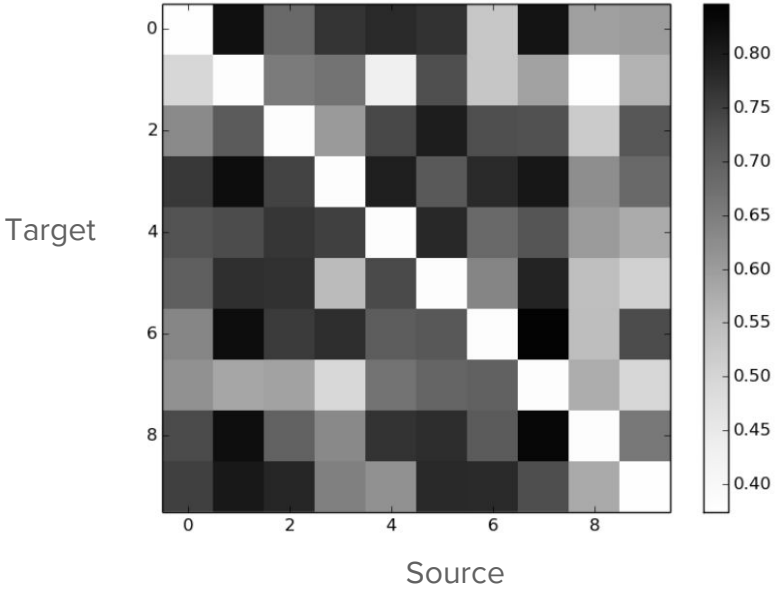
- Define $A(\mathbf{X}, t)$: the average number of zero elements in the adversarial saliency map of X computed during the first crafting iteration
- Closer adversarial distance is to 1, more likely input will be harder to misclassify
- Metric of robustness for the network:

$$R(\mathbf{F}) = \min_{(\mathbf{X},t)} A(\mathbf{X}, t)$$

Adversarial distance



Hardness matrix of source-target class pairs.



Adversarial distance averaged per source-destination

- Adversarial distance is a good proxy for difficult-to-evaluate hardness

Takeaways

Adversary Taxonomy

1. Can model multiple levels of adversarial capabilities/knowledge
2. Adversaries can have different goals- what unintended behavior does adversary want to elicit?

Algorithm for Adversarial Examples

1. Small input variations can lead to extreme output variations
2. Not all regions of input are conducive to adversarial examples
3. Use of Jacobian can help find these regions

Results

1. Some inputs are easier to corrupt than others
2. Some source-target classes are easier to corrupt than others
3. Saliency maps can help identify how vulnerable network is

Thanks!

Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks

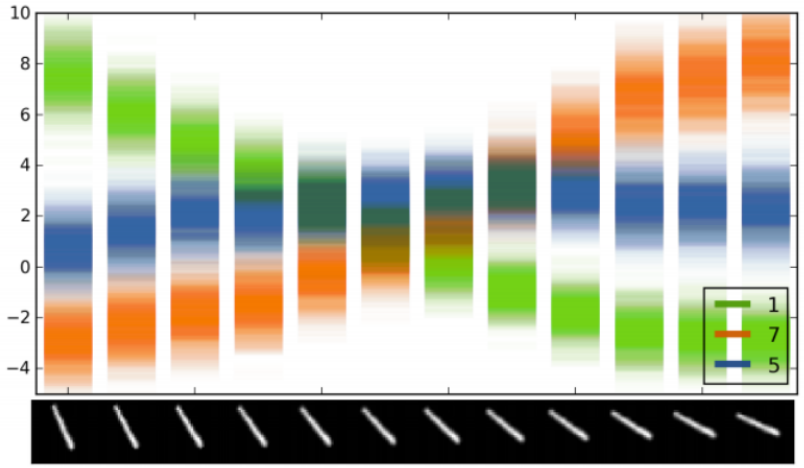
John Bradshaw, Alexander G. de G. Matthews, Zoubin Ghahramani

Presented by:

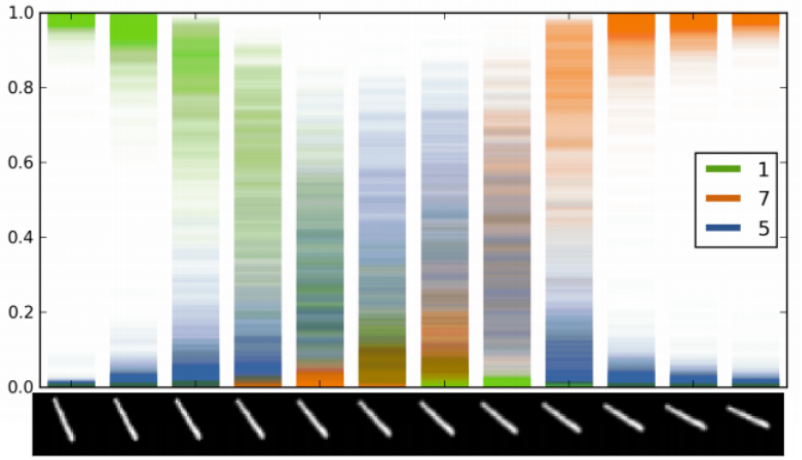
Pashootan Vaezipoor and **Sylvester Chiang**

Introduction

- Some issues with plain DNNs:
  - Do not capture their own *uncertainties*
    - Important in *Bayesian Optimization*, *Active Learning*, …
  - Vulnerable to *adversarial examples*
    - Important in security sensitive and safety regimes
- Models with good uncertainty may be able to prevent some Adversarial examples.
- So let's make DNNs Bayesian and account for uncertainty in the weights.
- Bayesian non-parametrics such as *Gaussian Process* (GP) can offer good probability estimates
- In this paper they use GP hybrid Deep Model *GPDNN*



(a) Softmax *input* scatter



(b) Softmax *output* scatter

Pictures from Yarin Gal et al. "**Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning**"

Outline of the paper

- Background
- Model architecture
- Results
  - Classification Accuracy
  - Adversarial Robustness
    - Fast Gradient Sign Method (FGSM)
    - L2 Optimization Attack of Carlini and Wagner
  - Transfer Testing

Background

- GPs express the distribution over latent variables with respect to the inputs x as a Gaussian distribution:

$$f_{\boldsymbol{x}} \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'})\right)$$

- And the learning of the parameters of k amounts to optimization of the following log marginal likelihood:

$$\log p(\mathbf{y}|X) \;=\; -\tfrac{1}{2}\mathbf{y}^\top (K + \sigma_n^2 I)^{-1}\mathbf{y} - \tfrac{1}{2}\log|K + \sigma_n^2 I| - \tfrac{n}{2}\log 2\pi$$

Problems with GP

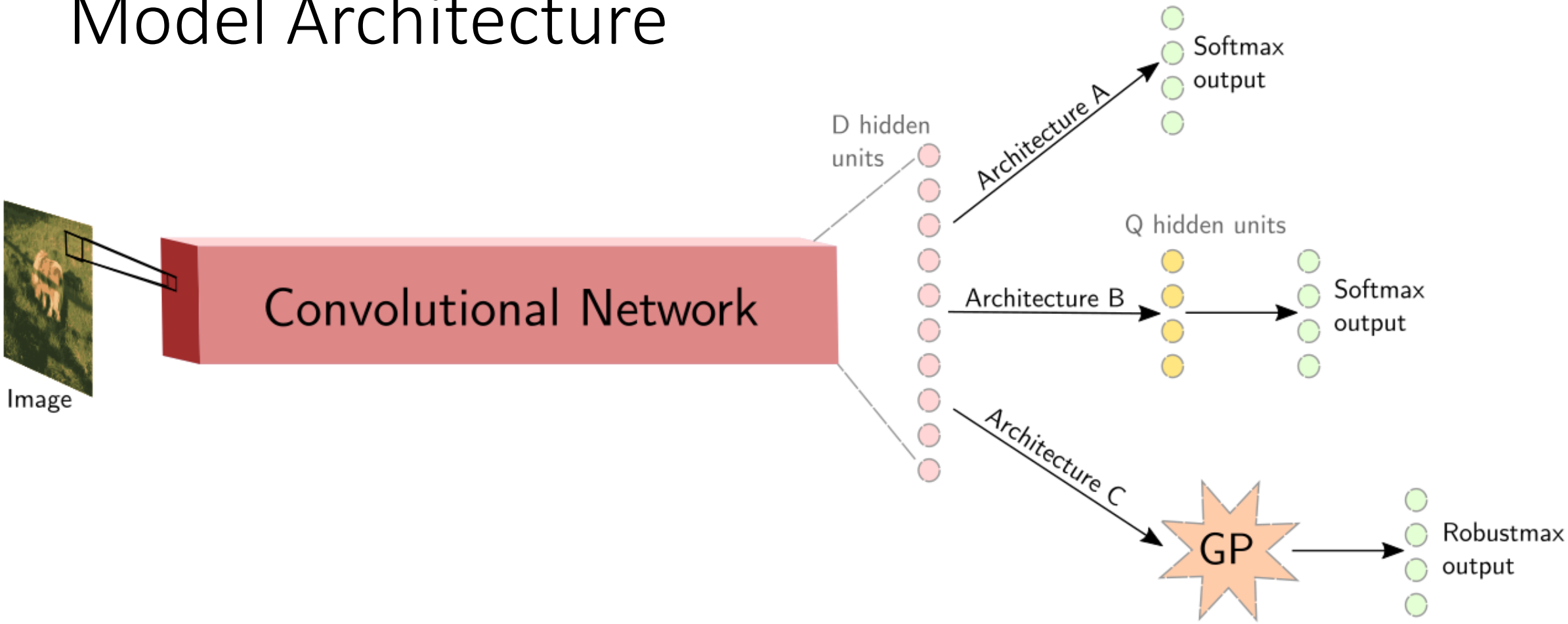
- Scalability:
  - Matrix inversion using *Cholesky Decomposition* is an $O(n^3)$ operation
  - They use *inducing points* to reduce the complexity to $O(nm^2)$
  - And they use a stochastic variant of Titsias' variational method to pick the points
  - They use an extension so that they can use non-conjugate likelihoods (for classification)

$$\log p(Y) \geq \sum_{y,\boldsymbol{x} \in Y,\boldsymbol{X}} \mathbb{E}_{q(f_{\boldsymbol{x}})}[\log p(y|f_{\boldsymbol{x}})] - \mathcal{KL}\left(q(f_{\boldsymbol{Z}})||p(f_{\boldsymbol{Z}})\right)$$

  - $q(f_x)$ is the variational approx. to distribution of f_x and Z are the inducing point locations

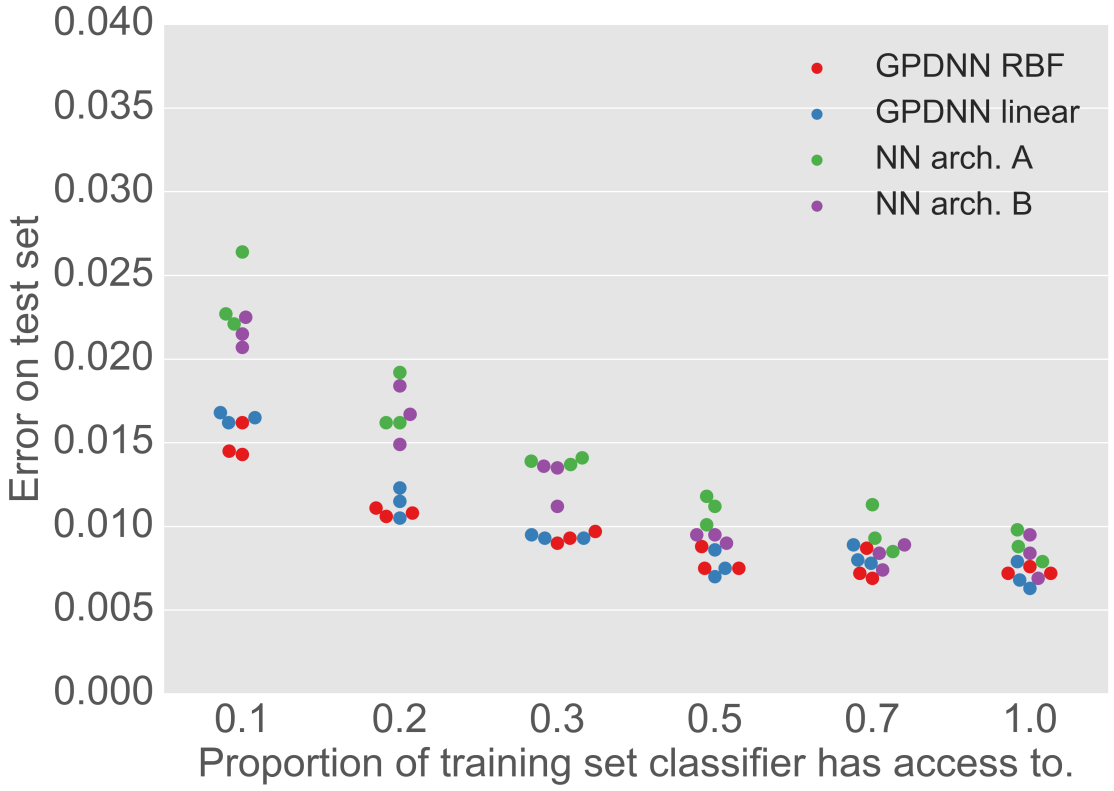
- Kernel Expressiveness:
  - No good representational power to model relationship between complex high dimentional data (e.g. images)

Model Architecture

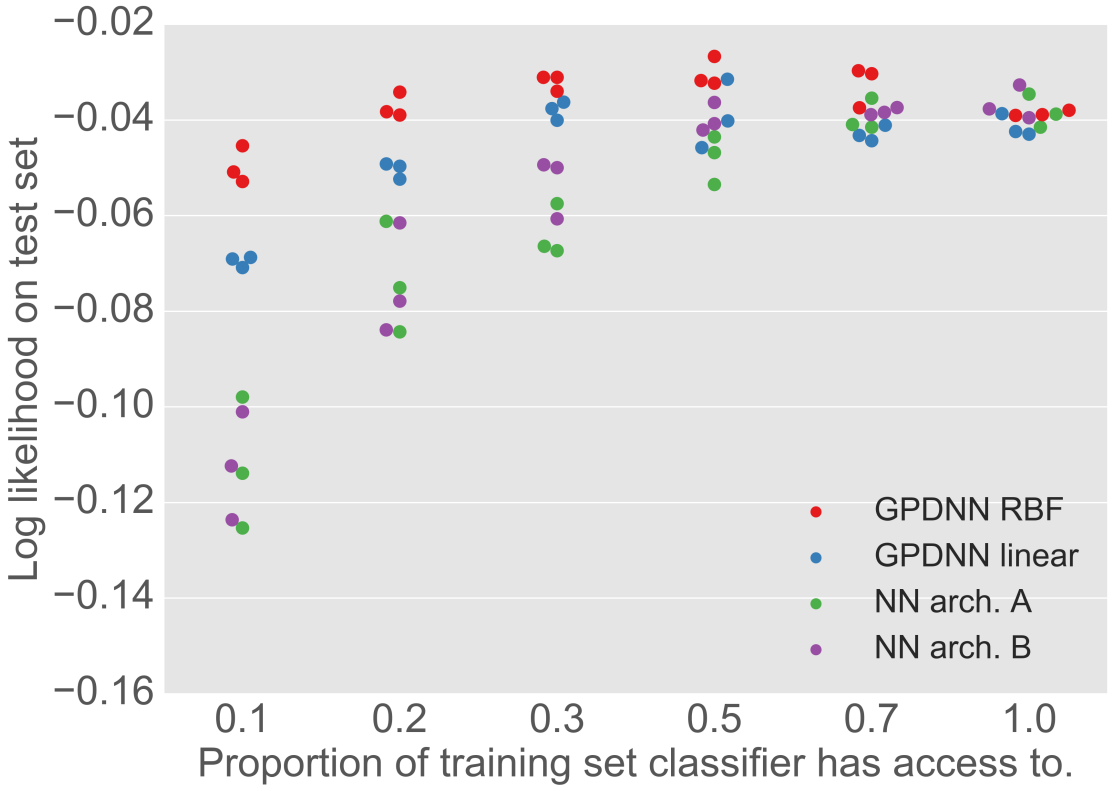


$$p(y_{\boldsymbol{x}}|\boldsymbol{f}_{\boldsymbol{x}}) = \begin{cases} 1-\beta, & \text{if } y_{\boldsymbol{x}} = \operatorname{argmax}\boldsymbol{f}_{\boldsymbol{x}} \\ \beta/(\text{number of classes -1}), & \text{otherwise} \end{cases}$$

Classification (MNIST)

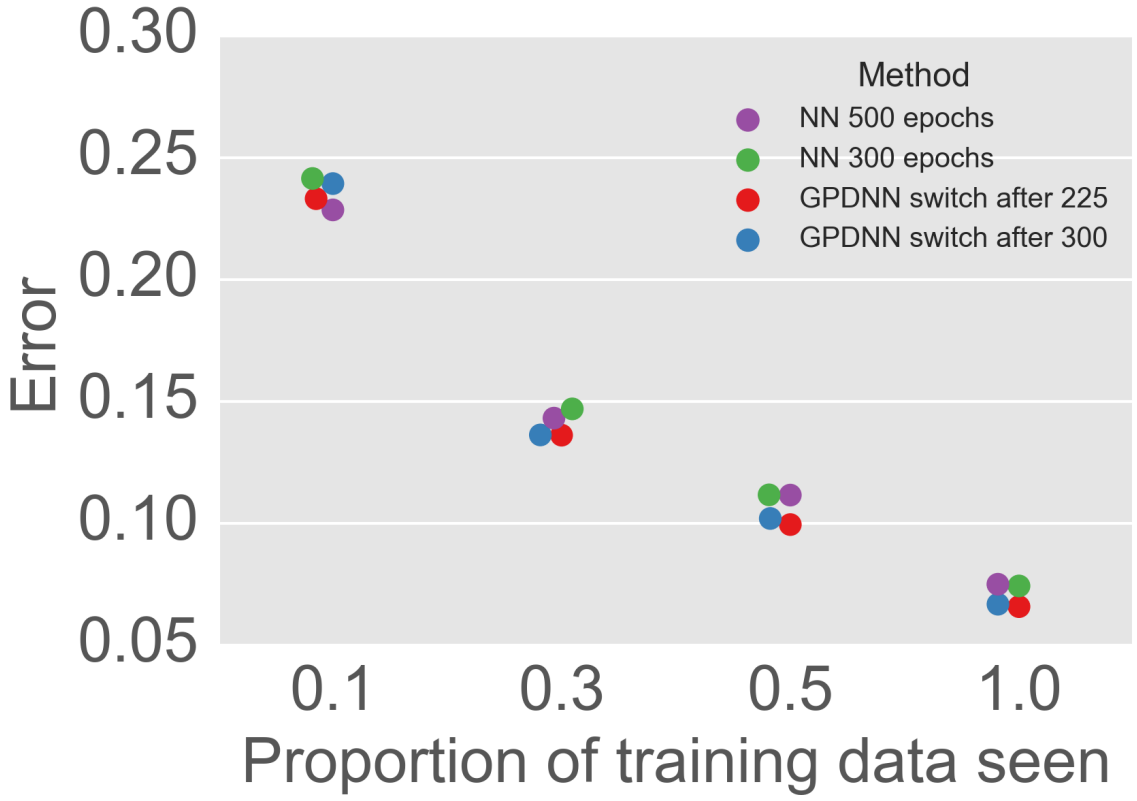


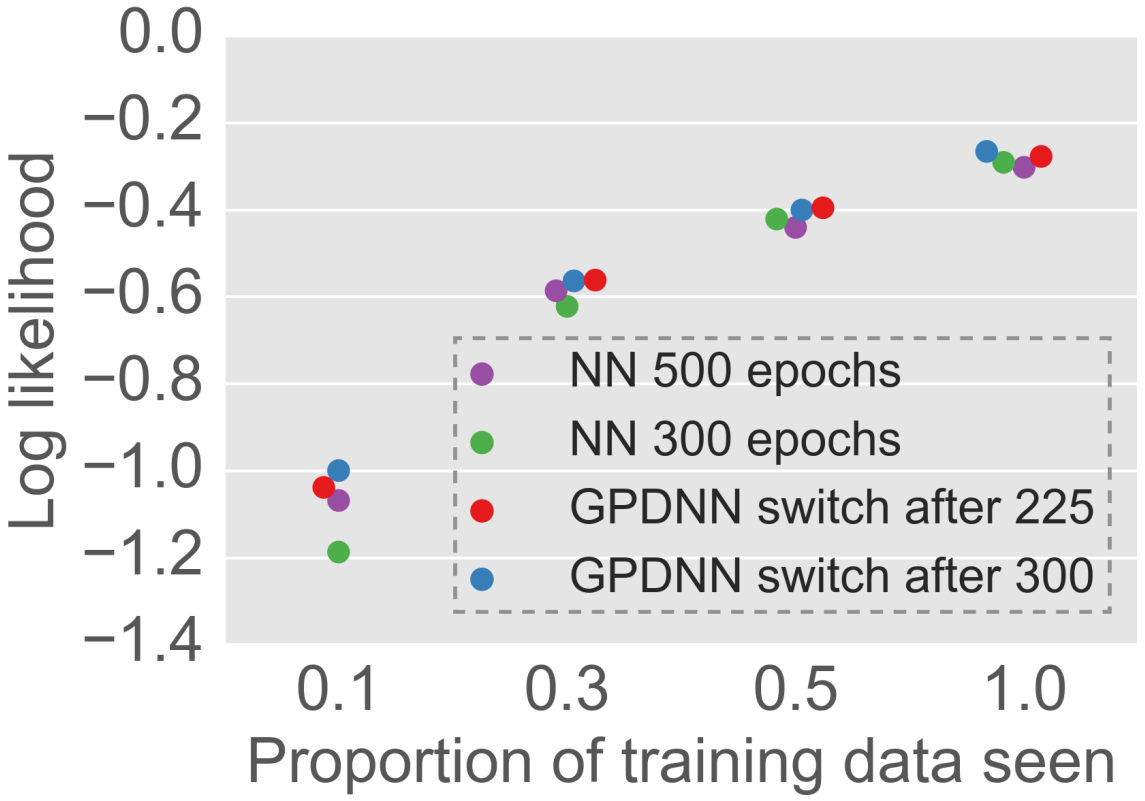
(a) Errors



(b) Log likelihoods

Classification (CIFAR10)



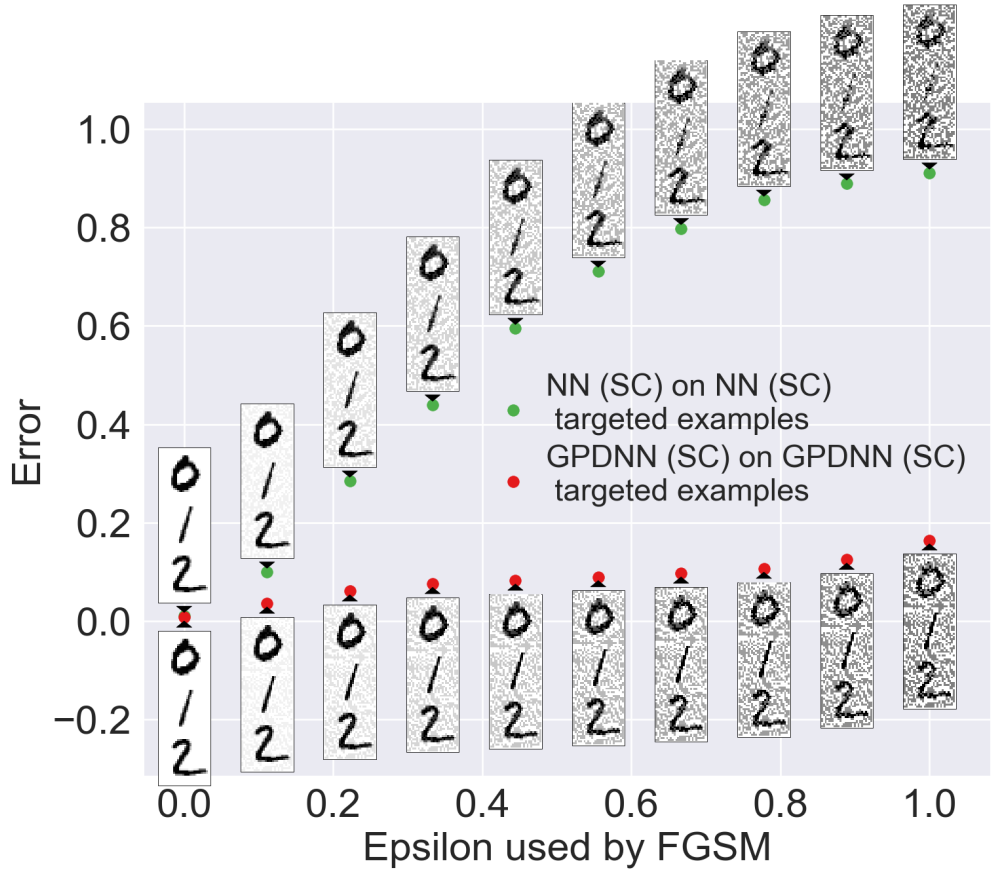


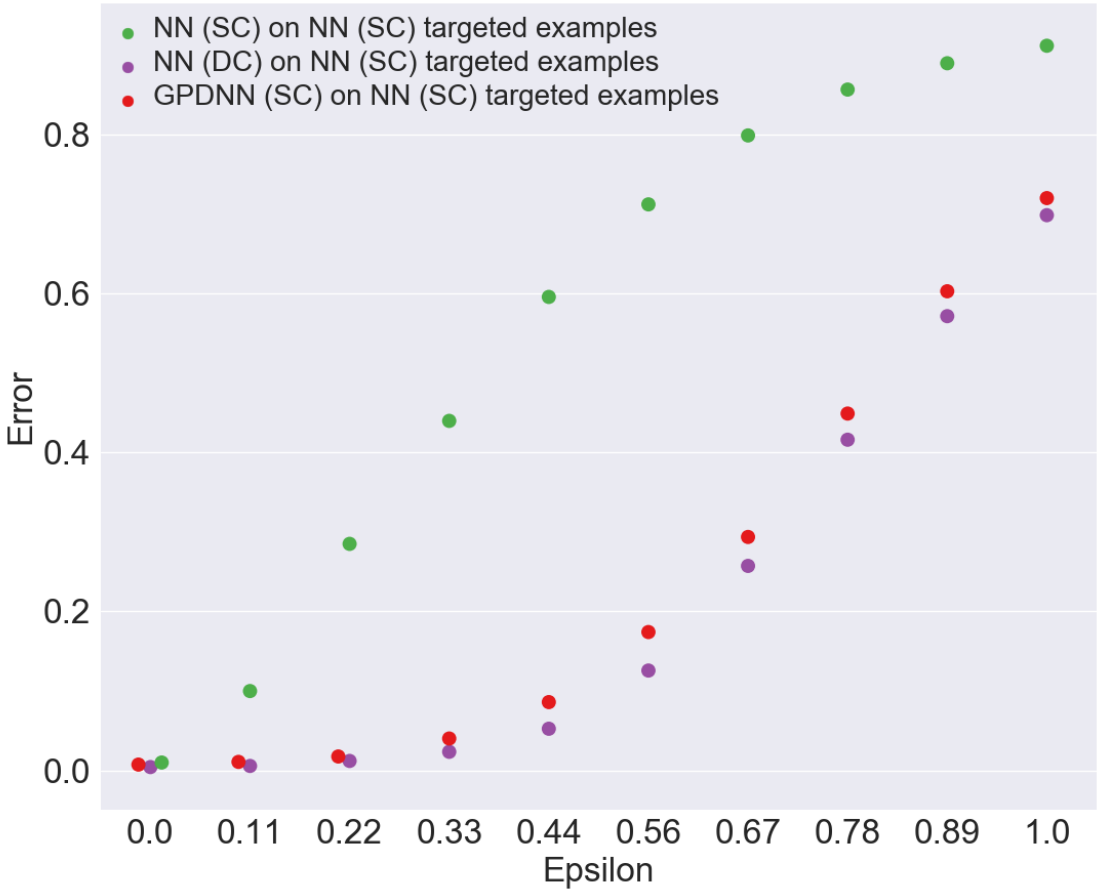
Adversarial Robustness

- Attacks are often transferable between different architectures and different machine learning methods
- Given a classification model $M_\theta(x)$ and purturbation μ attacks can be divided to:
  - Targeted: $M_\theta(x+\mu) = l'$
  - Non-targeted: $M_\theta(x+\mu) \neq M_\theta(x)$

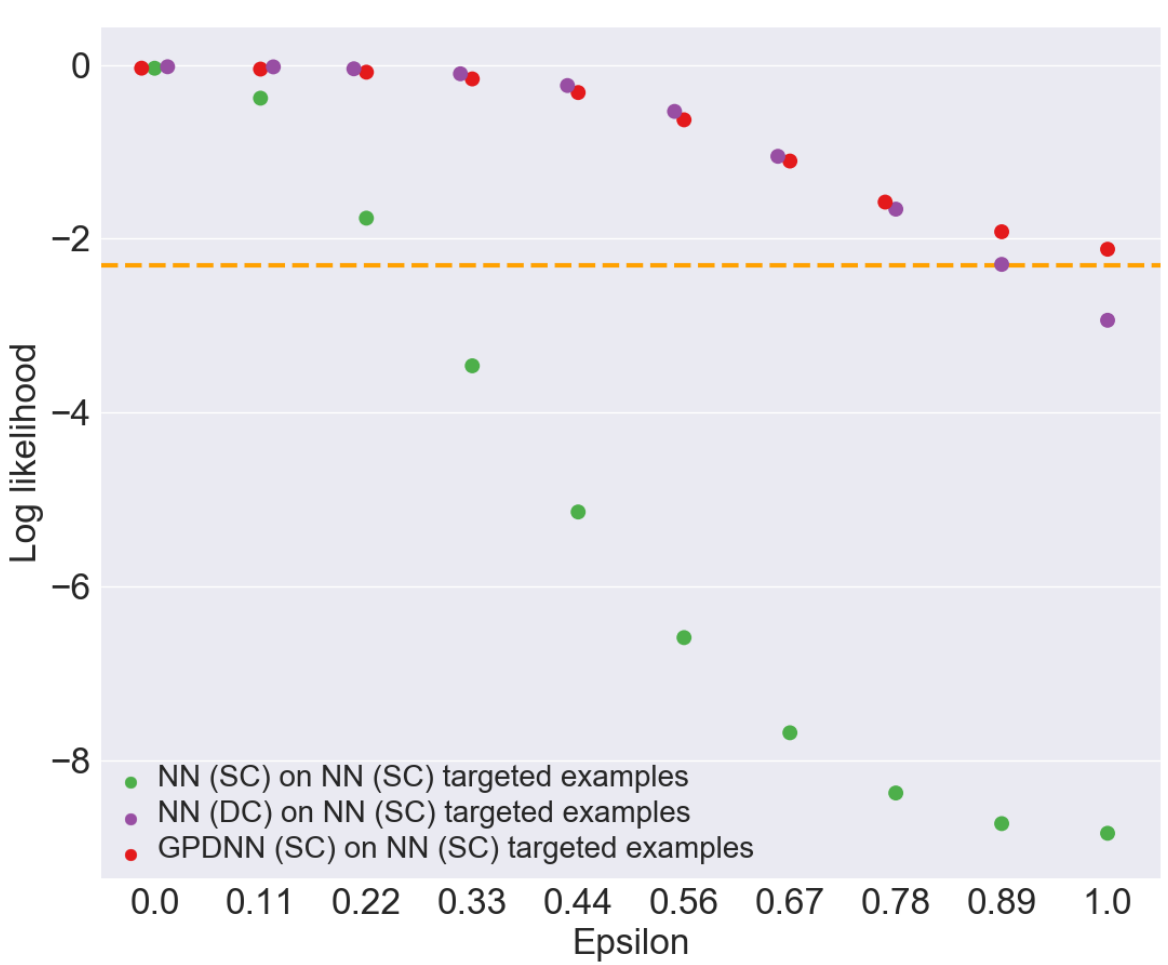
The fast gradient sign method (FGSM)

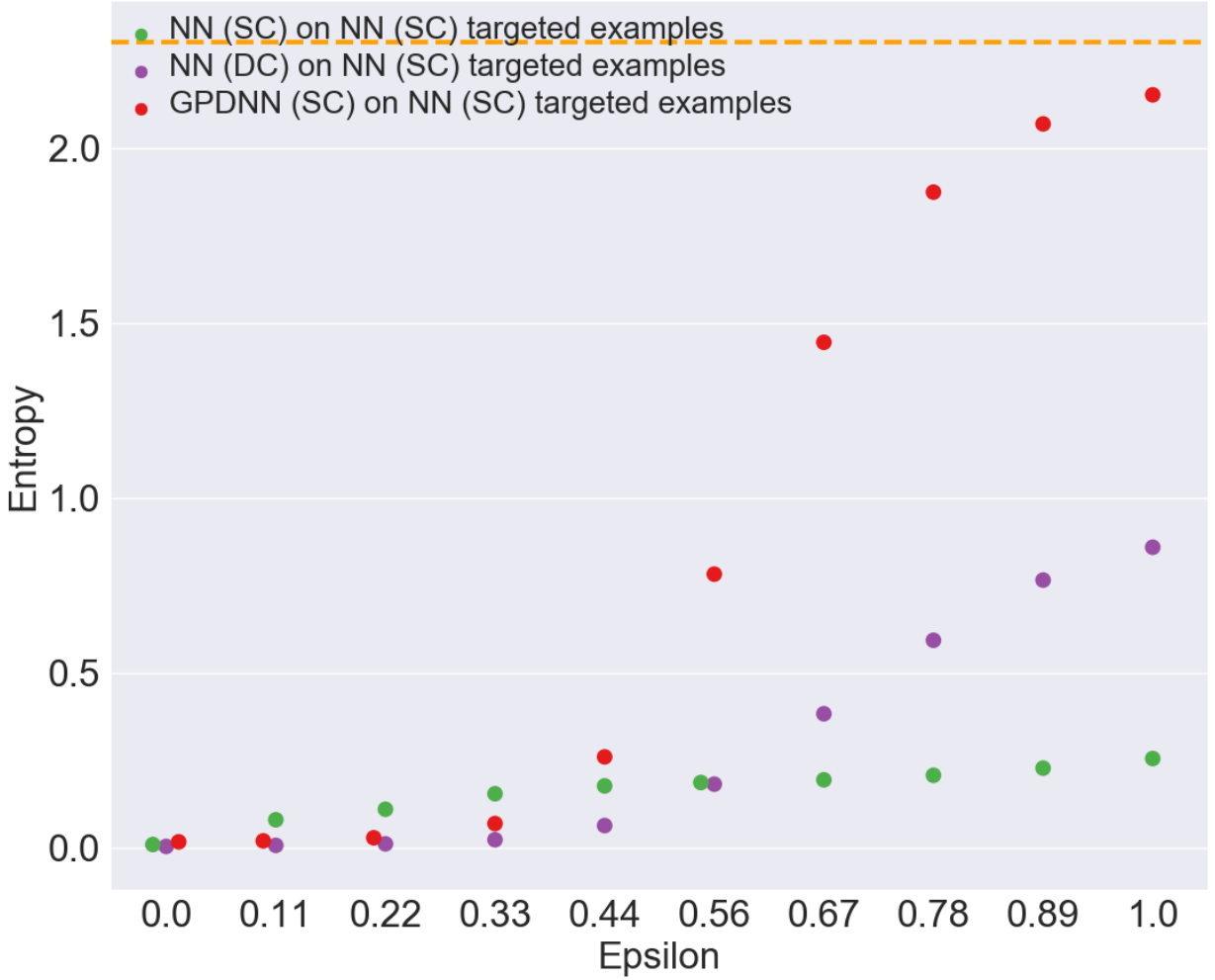
- It perturbs the image by: $\mu = \epsilon \, sign(\nabla_{\boldsymbol{x}} J(\theta, \boldsymbol{x}, y))$



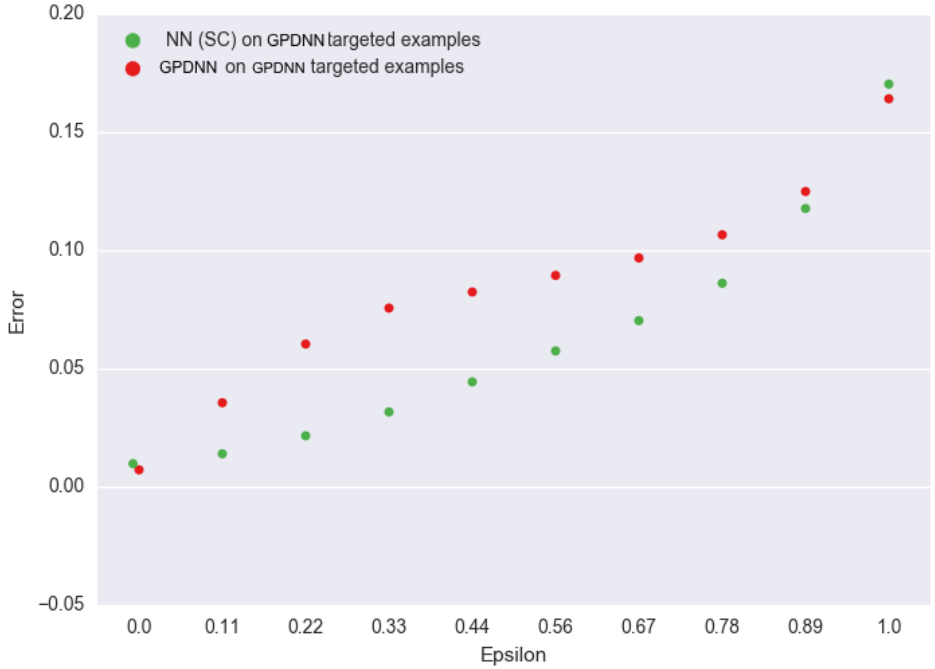


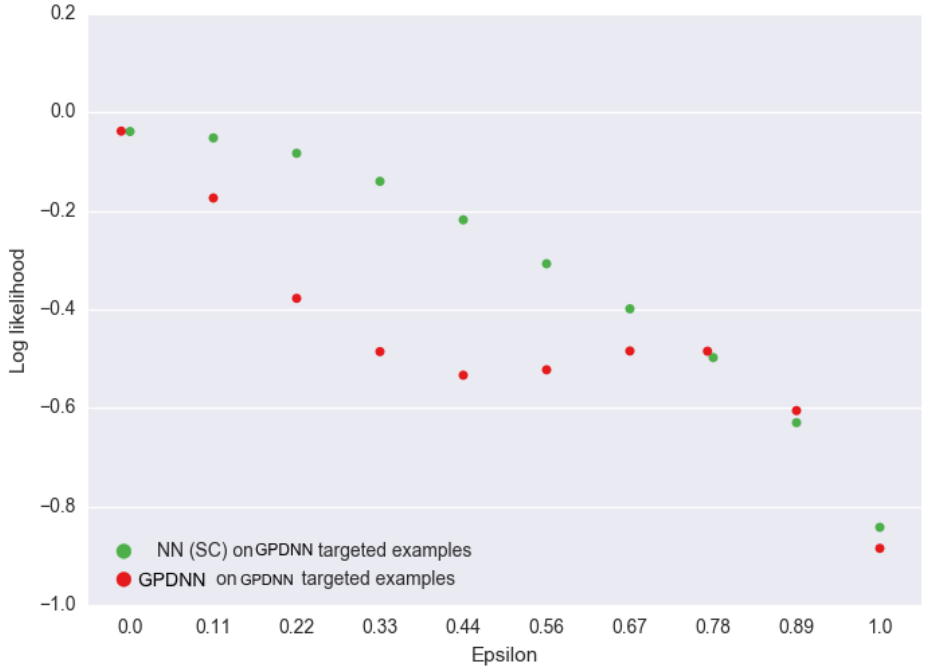
FGSM (MNIST)

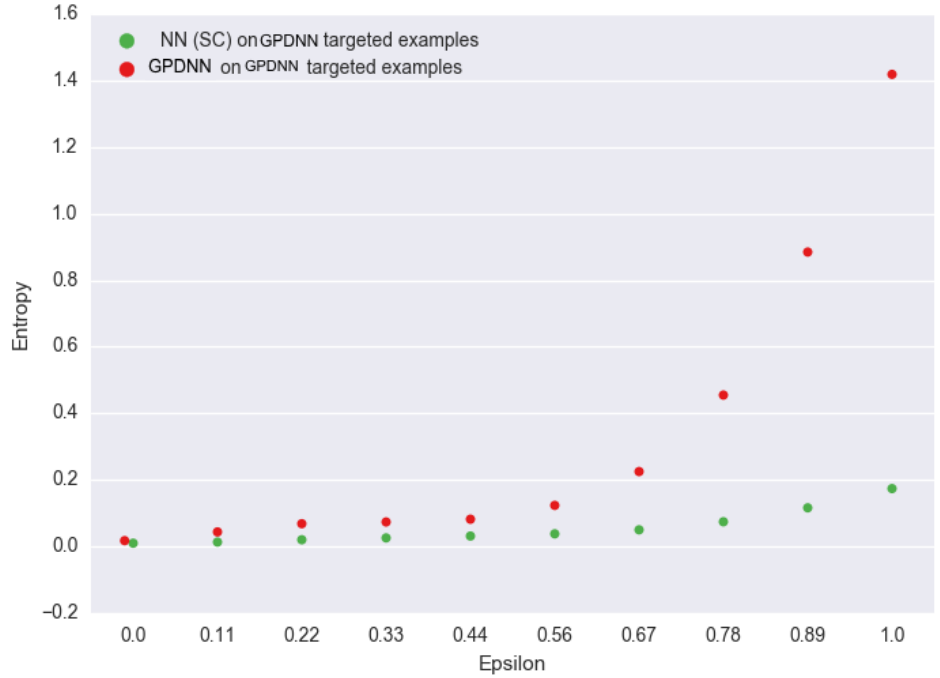

0
−2
−4
−6
−8
Log likelihood
NN (SC) on NN (SC) targeted examples
NN (DC) on NN (SC) targeted examples
GPDNN (SC) on NN (SC) targeted examples
0.0
0.11
0.22
0.33
0.44
0.56
0.67
0.78
0.89
1.0
Epsilon


NN (SC) on NN (SC) targeted examples
NN (DC) on NN (SC) targeted examples
GPDNN (SC) on NN (SC) targeted examples
2.0
1.5
1.0
0.5
0.0
Entropy
0.0
0.11
0.22
0.33
0.44
0.56
0.67
0.78
0.89
1.0
Epsilon

FGSM (MNIST) – Attacking GPDNN







Uncertainty

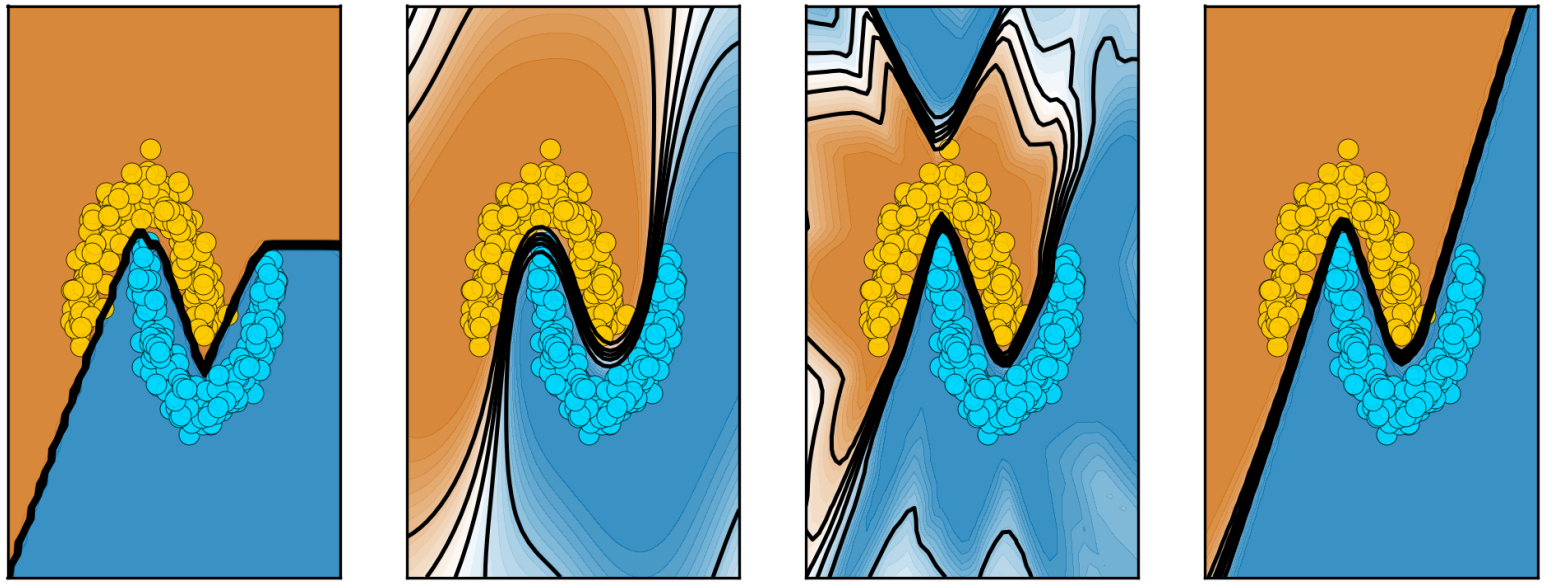
Zoomed in

NN

GP

GPDNN (RBF)

GPDNN (Lin)



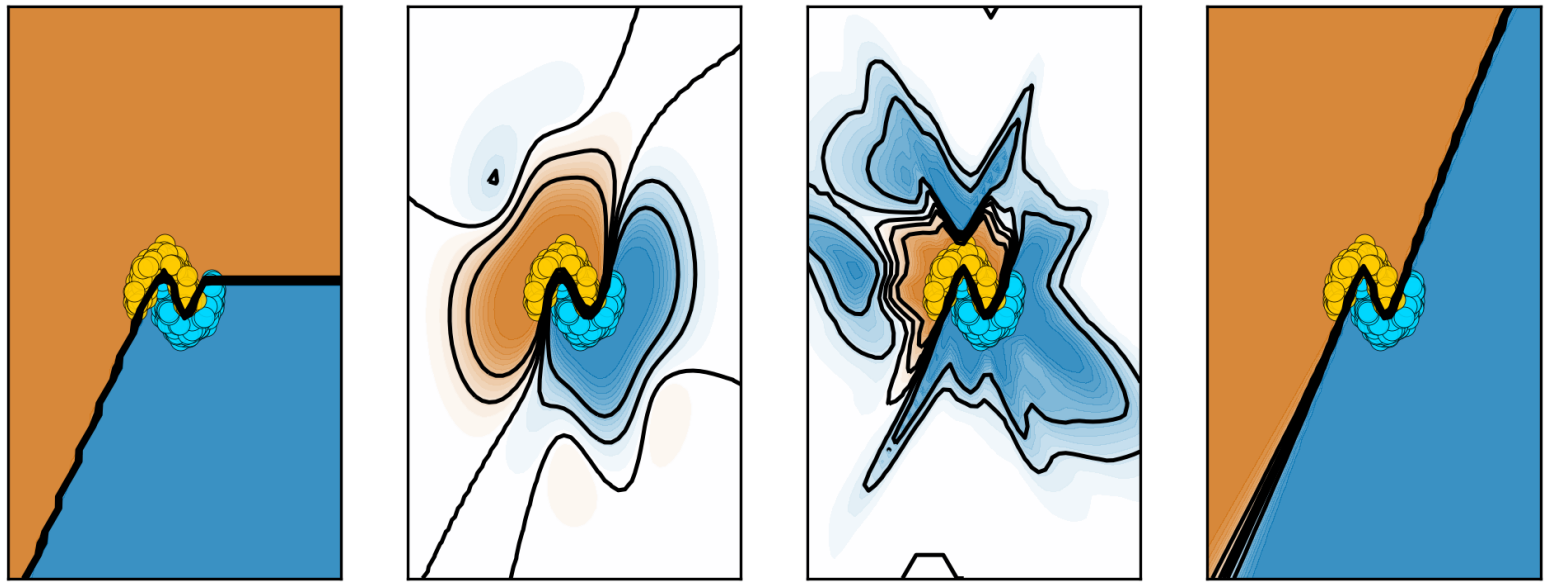
Zoomed out

NN

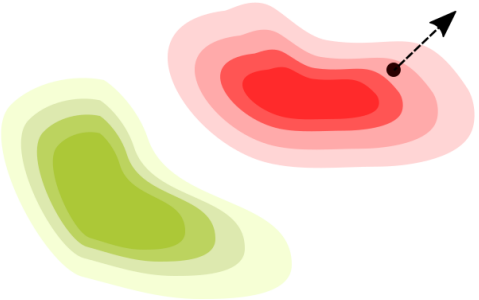
GP

GPDNN (RBF)

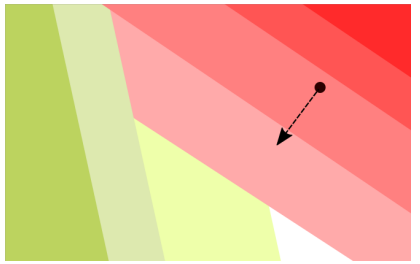
GPDNN (Lin)



Intuition behind
Adversarial Robustness



Nonlinear



Linear

L2 Optimization Attack

$$\begin{aligned} \textbf{minimize} \quad & \mathcal{D}(x, x+\delta) \\ \textbf{such that} \quad & C(x+\delta) = t \\ & x+\delta \in [0,1]^n \end{aligned}$$

Where D is a distance metric, and delta is a small noise change

L2 Optimization Attack

$$\begin{aligned} &\text{minimize} \quad \mathcal{D}(x, x+\delta) + c \cdot f(x+\delta) \\ &\text{such that} \quad x + \delta \in [0,1]^n \end{aligned}$$

Where f can be equal to:

$$f_1(x') = -\text{loss}_{F,t}(x') + 1$$

$$f_2(x') = (\max_{i \neq t}(F(x')_i) - F(x')_t)^+$$

$$f_3(x') = \text{softplus}(\max_{i \neq t}(F(x')_i) - F(x')_t) - \log(2)$$

$$f_4(x') = (0.5 - F(x')_t)^+$$

$$f_5(x') = -\log(2F(x')_t - 2)$$

$$f_6(x') = (\max_{i \neq t}(Z(x')_i) - Z(x')_t)^+$$

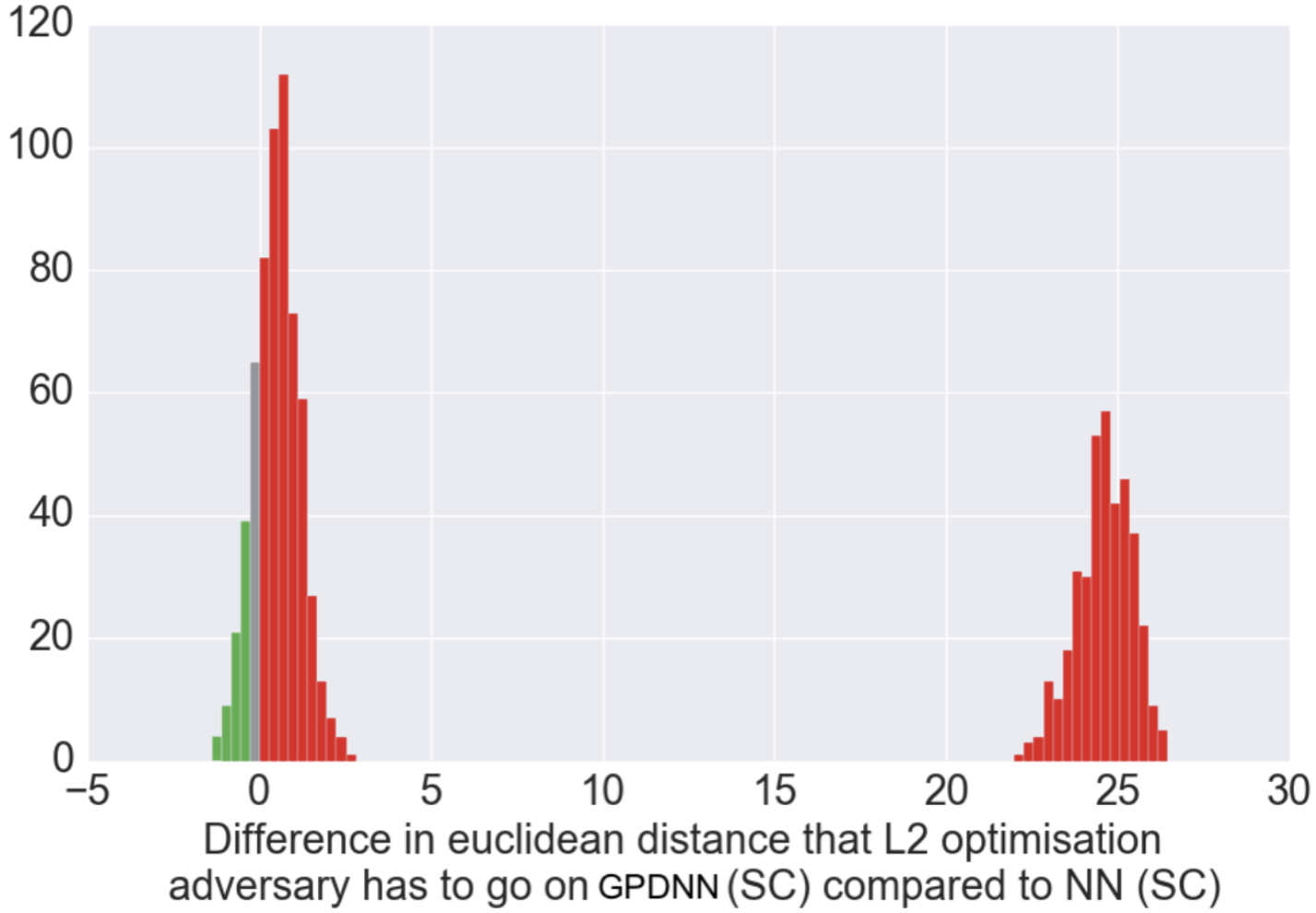
$$f_7(x') = \text{softplus}(\max_{i \neq t}(Z(x')_i) - Z(x')_t) - \log(2)$$

Derivations taken from Carlini et al. "**Towards Evaluating the Robustness of Neural Networks**"

Attacking GPDNN

On 1000 MNIST Images:

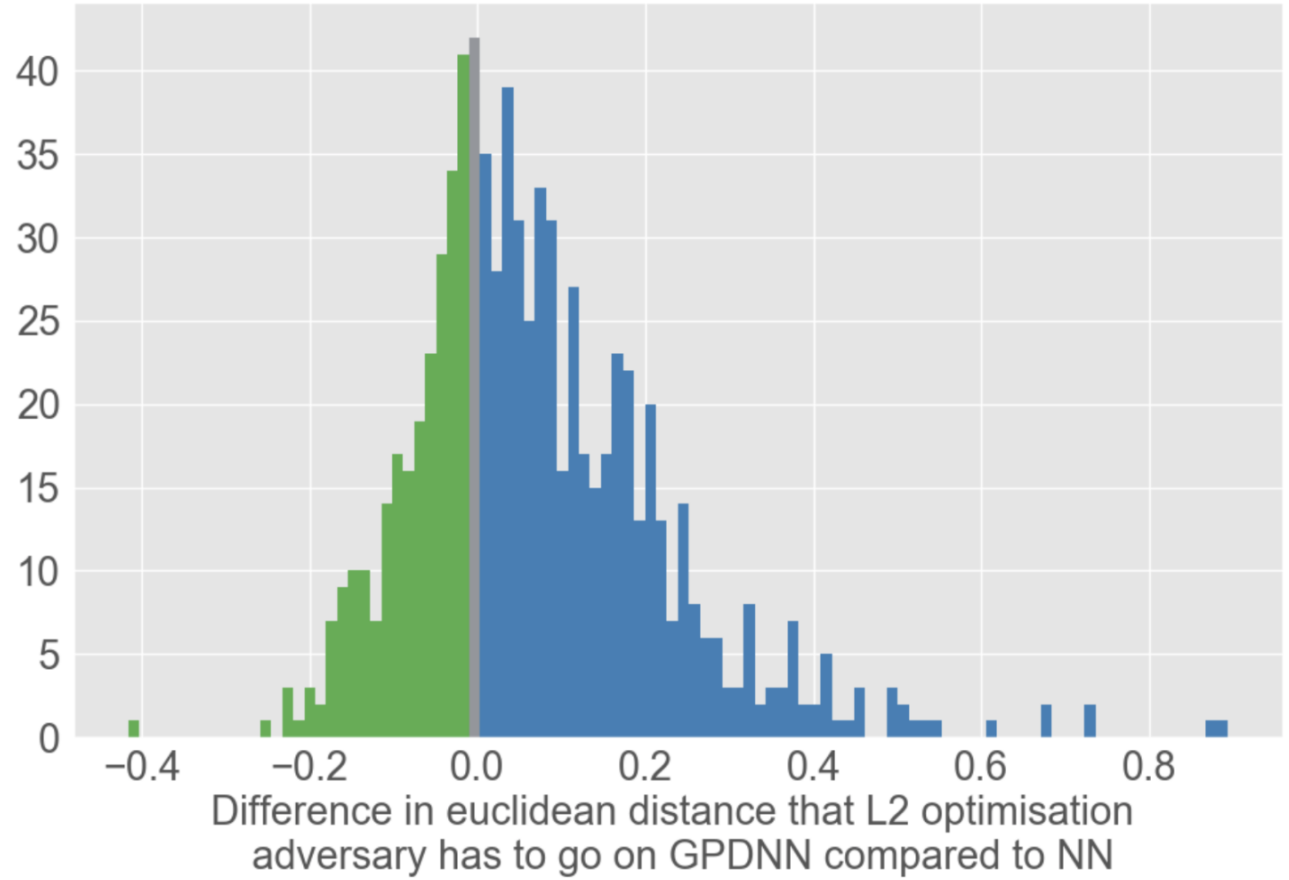
- 381 attacks failed
- Successful attacks have a 0.529 greater perturbatic
- GPDNN more robust to adversarial attacks



Attacking GPDNN

On 1000 CIFAR10 Images:

- 207 attacks failed
- Greater perturbation needed to generate adversarial examples



Attack Transferability

MNIST

Model targeted	NN (SC) Error	NN (SC) LL	GPDNN (SC) Error	GPDNN (SC) LL	NN (DC) Error	NN (DC) LL
Plain MNIST †	0.0100	-0.039	0.007	-0.038	0.004	-0.021
NN (SC, arch. A)	1.000	-0.718	0.009	-0.056	0.003	-0.014
GPDNN (SC, arch C)	0.031	-0.094	1.000	-0.828	0.011	-0.056
NN (DC, arch B)	0.014	-0.047	0.013	-0.067	1.000	-0.725

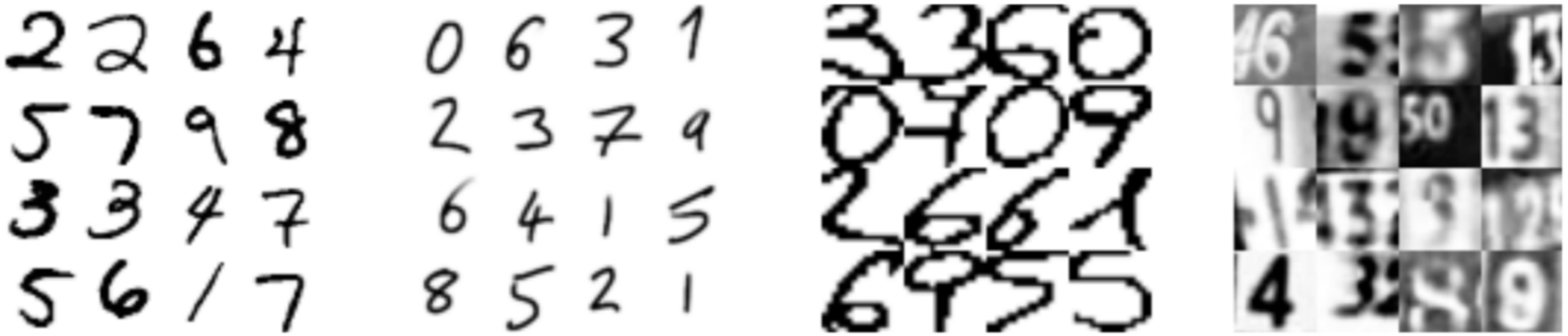
CIFAR

Model targeted	NN (DC) Error	NN (DC) LL	GPDNN (DC) Error	GPDNN (DC) LL
NN (DC, arch B)	1	-0.788	0.012	-0.08
GPDNN (DC, arch C)	0.016	-0.066	1	-0.935

Transfer Testing

How well GPDNN models notice **domain shifts**?

MNIST ANOMNIST Semeion SVHN



Transfer Testing Results

	NN (SC) arch A.		GPDNN (SC) arch C.	
	Acc.	LL	Acc.	LL
MNIST	0.990	-0.039	0.993	-0.038
ANOMNIST	0.781	-1.125	0.831	-0.818
Semeion	0.249	-9.609	0.320	-2.841
SVHN	0.304	-4.687	0.276	-2.151

Transfer Testing Results

	NN (SC) arch A.		GPDNN (SC) arch C.	
	Acc.	LL	Acc.	LL
MNIST	0.990	-0.039	0.993	-0.038
ANOMNIST	0.781	-1.125	0.831	-0.818
Semeion	0.249	-9.609	0.320	-2.841
SVHN	0.304	-4.687	0.276	-2.151

Conclusion

- Explored GPDNN’s robustness in classification
- These hybrid models are competitive to other NN’s
- They have better calibrated uncertainties
- Better at knowing “when they don’t know”