

The security and interpretability of machine learning

Innopolis University

About me

- Fourth year PhD student
- Lab of machine learning and knowledge representation
- Adversarial attacks on neural networks
- Email: b.rasheed@innopolis.university

Why to take this course?

The era of AI

MI How are you today, ChatGPT? 

As a language model trained by OpenAI, I don't have the ability to experience emotions like a human would. I am designed to assist with providing information and answering questions to the best of my ability. Is there something specific you would like to know?  

MI What is the Capital of the Klingon homeworld, Qo'noS? 

In the fictional Star Trek universe, the Klingon homeworld is called Qo'noS and its capital is called First City. It is the political and cultural center of the Klingon Empire and the location of the Klingon High Council. It is a city of great size and importance in Klingon society.  

NLP

Computer
vision



Autonomous vehicles

**So, AI systems are very popular, but
are they robust?**

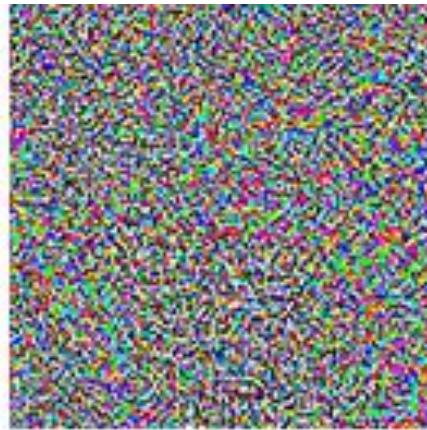
Adversarial attacks



“panda”

57.7% confidence

$+\epsilon$



=



“gibbon”

99.3% confidence

Image from: <https://openai.com/blog/adversarial-example-research/>

Why this might be important?

Adversarial attacks

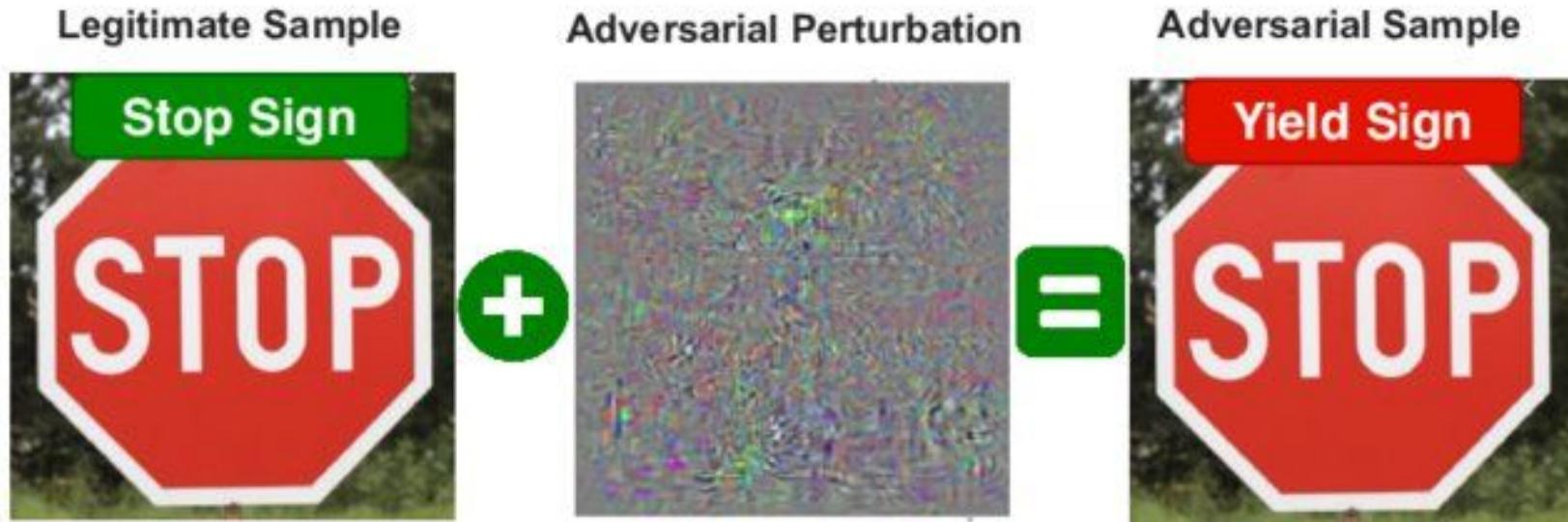


Image from:

https://www.researchgate.net/publication/347398615_Developing_Future_Human-Centered_Smart_Cities_Critical_Analysis_of_Smart_City_Security_Interpretability_and_Ethical_Challenges

Adversarial attacks

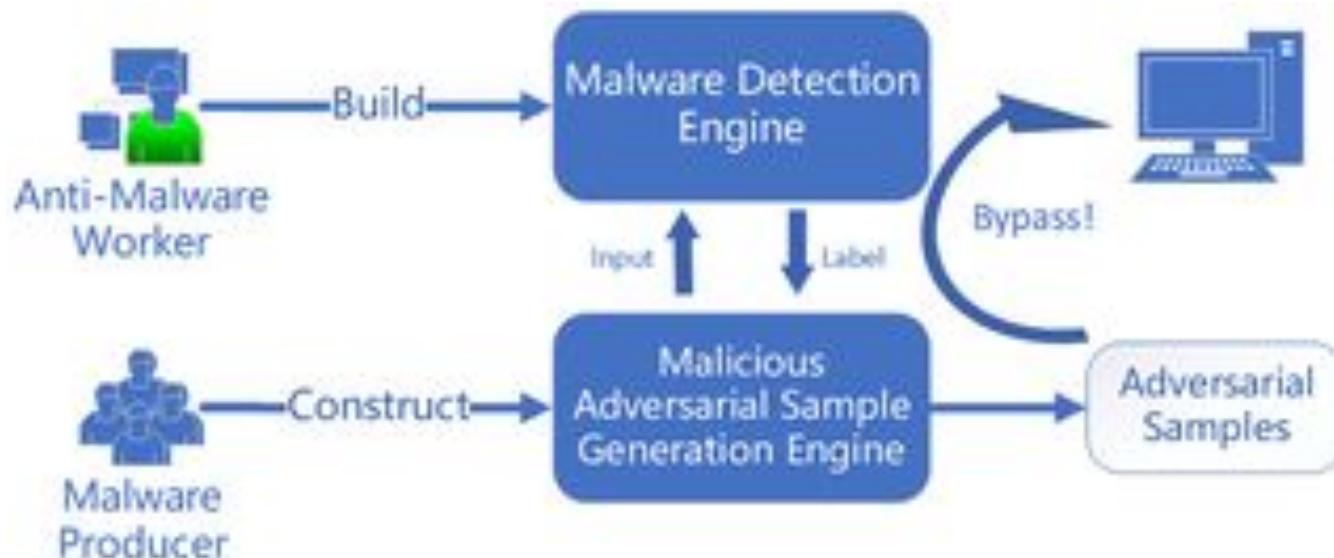


Image from: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0231626>

The goal of the course

The course will focus on that attacks and defenses on deep neural networks. Interactively, through code and exercises, we will try to build more robust and interpretable systems.

Course structure

- Recap, deep neural networks, and statistical learning
- Adversarial attacks and their implementation
- Defenses against adversarial attacks and their implementation
- Robustness certifications
- Interpretability and the relation with robustness

Evaluation

- Project based:
 - 2 homeworks (60 points)
 - Quizzes (10 points)
 - midterm exam (20 points)
 - Demo day (project presentation) (10 points)
 - Project:
 - Design the model (First homework)
 - Design attacks and defenses (second homework)
 - Final: (Third homework)
 - Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples [Athalye et al 2018]
 - Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods [Carlini 2017]
 - On Adaptive Attacks to Adversarial Example Defenses [Tramer 2020]

Background

- L_p norm

$$l_p (1 \leq p \leq \infty)$$

$$\|\vec{u}\|_p = \sqrt[p]{\sum |u_i|^p}$$

Background

- L_p norm

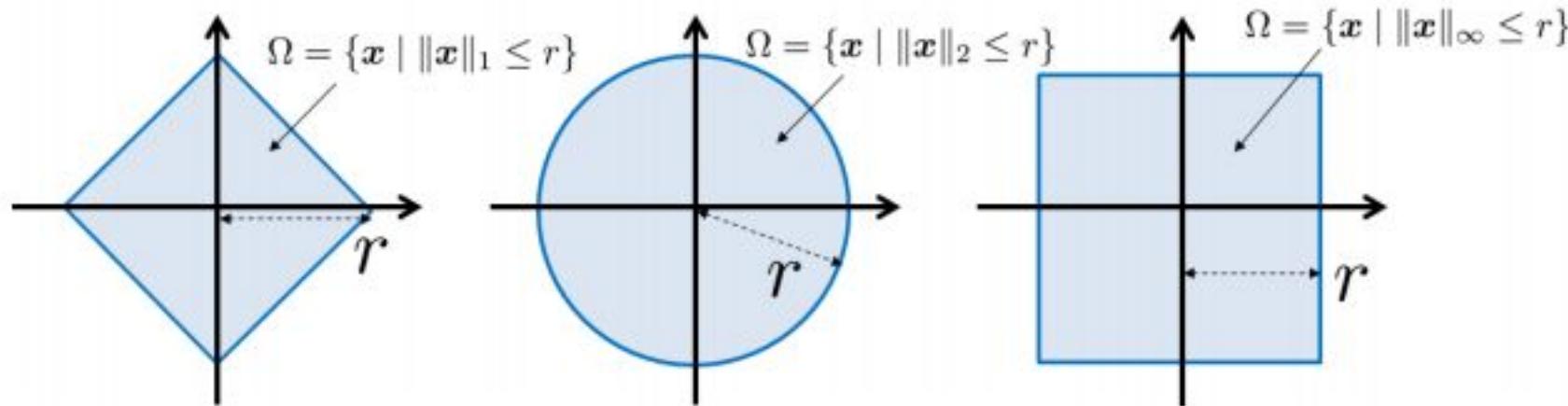
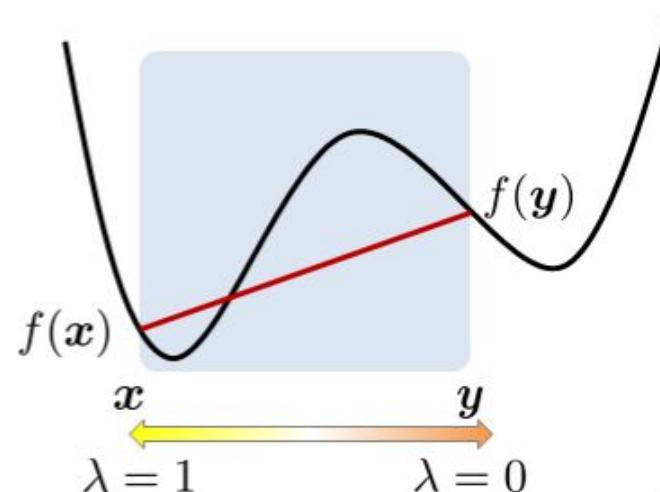
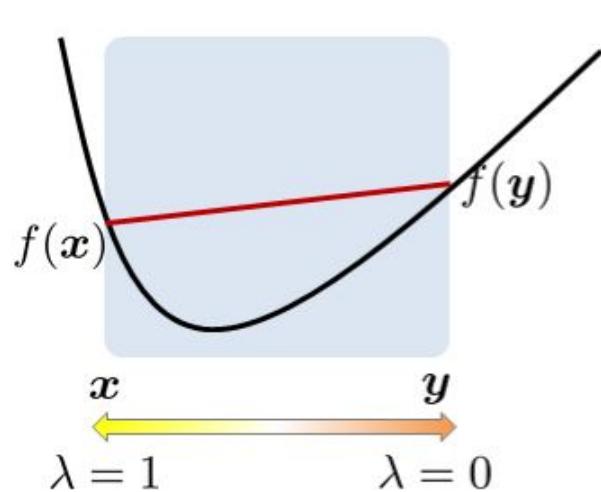


Image from: <https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter1.pdf>

Background

- Convexity

$$f(\lambda x + (1-\lambda)y) = \lambda f(x) + (1-\lambda)f(y)$$



Activate!

Image from: <https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter1.pdf>

Background

- Gradient Descent

$$x^{(t+1)} = x^{(t)} - \alpha \nabla(f(x^{(t)})) \quad t = 0, 1, 2, \dots$$

Background

- Gradient Descent

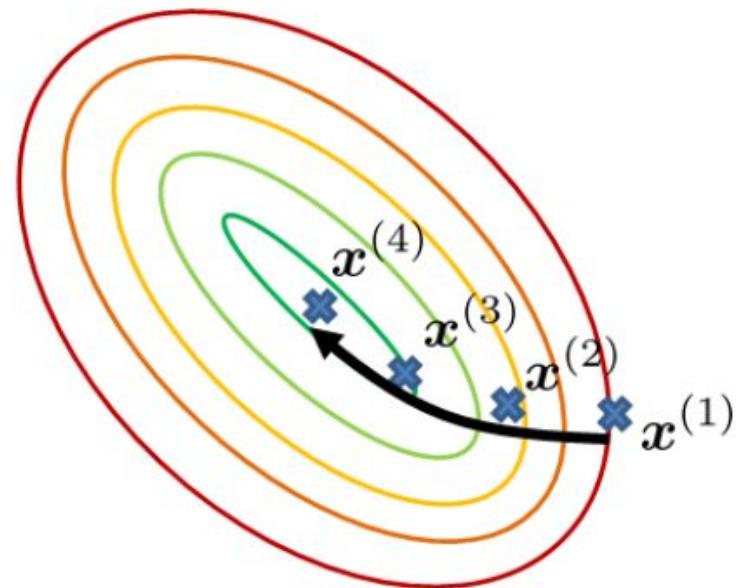
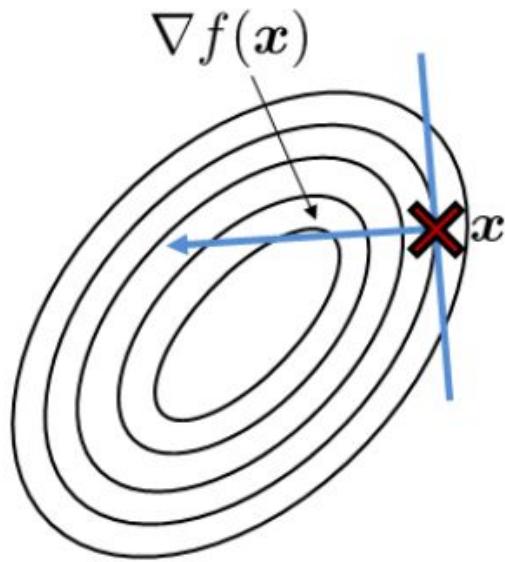


Image from: <https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter1.pdf>

Background

- Lagrange Multiplier

$$\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad f(\boldsymbol{x})$$

subject to $g_i(\boldsymbol{x}) \geq 0, \quad i = 1, \dots, m$
 $h_j(\boldsymbol{x}) = 0, \quad j = 1, \dots, k.$

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\nu}) \stackrel{\text{def}}{=} f(\boldsymbol{x}) - \sum_{i=1}^m \mu_i g_i(\boldsymbol{x}) - \sum_{j=1}^k \nu_j h_j(\boldsymbol{x}).$$

Background

input $x \in \mathbb{R}^d = X$.

output $y \in \{-1, +1\} = Y$.

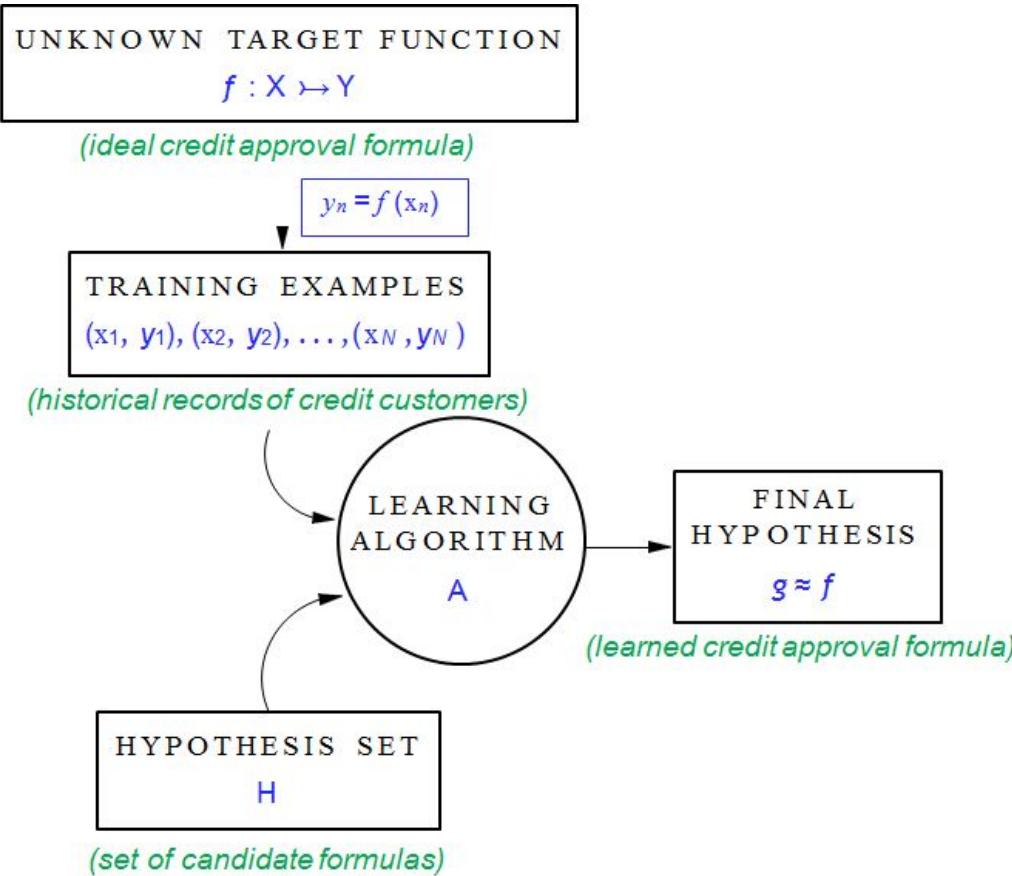
target function $f : X \rightarrow Y$.

(The target f is unknown.)

data set $D = (x_1, y_1), \dots, (x_N, y_N)$.

($y_n = f(x_n)$.)

Background



Supervised and Unsupervised Learning

- Supervised (classify labeled data)
- Unsupervised (find patterns in unlabeled data)



Deep Learning Theory

- “Learns” with big data (lots of input examples) and trial-and-error guesses to adjust weights and bias and establish key features
- Creates a predictive system to identify new examples

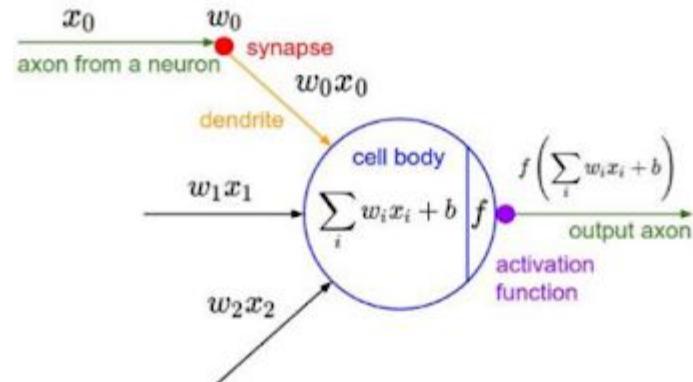
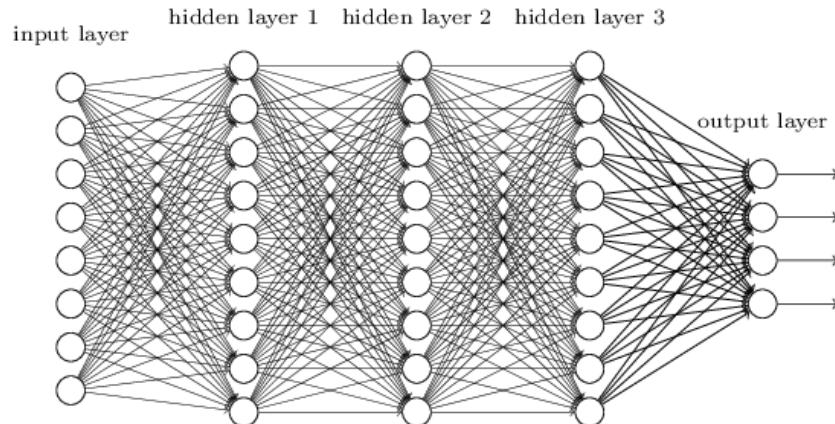
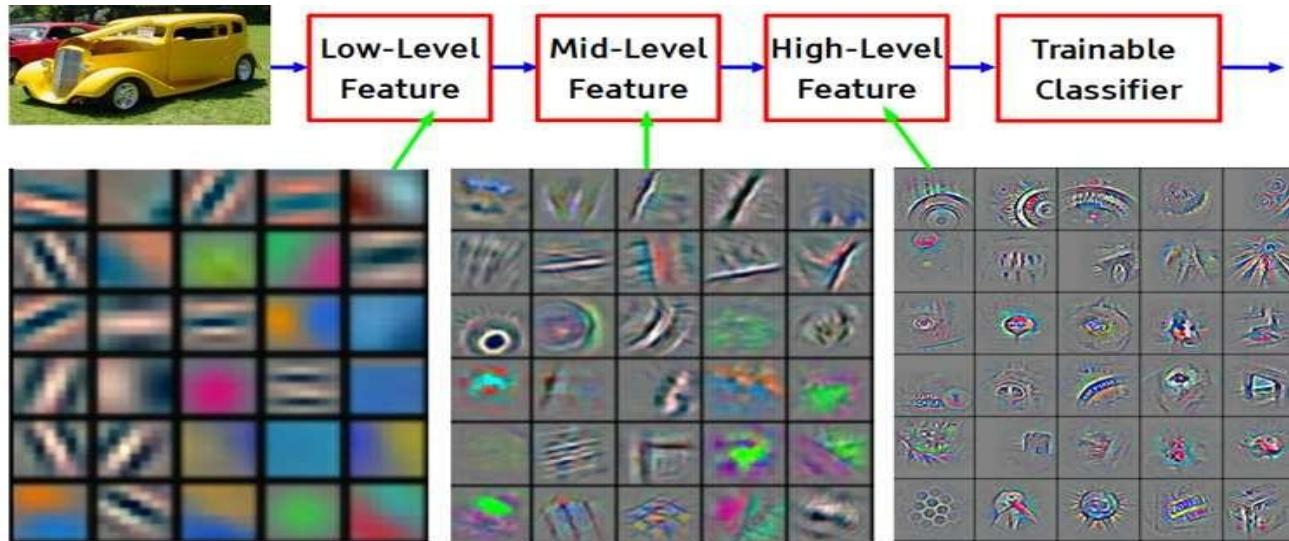


Image from: <https://pytech-solution.blogspot.com/2019/01/activation-functions.html>

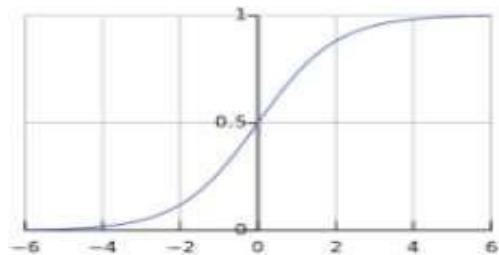
Deep Learning Theory



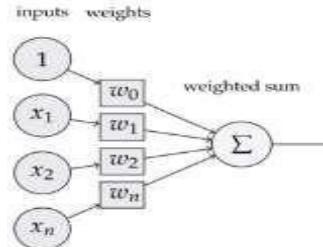
Source: Jann LeCun,
<http://www.pamitc.org/cvpr15/files/lecun-20150610-cvpr-keynote.pdf>

3 Key Technical Principles of Deep Learning

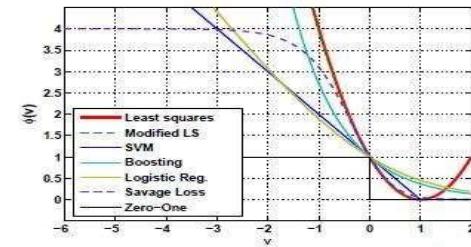
Sigmoid Function



Perceptron Structure



Loss Function



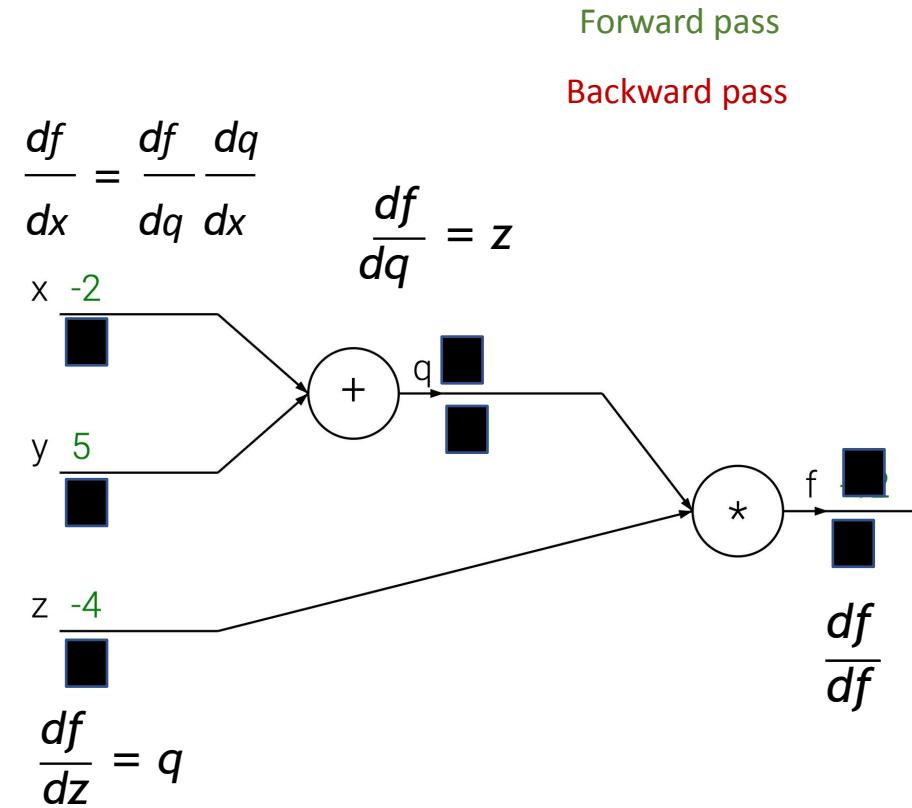
Backpropagation illustrated

$$f(x, y, z) = (x + y)z$$

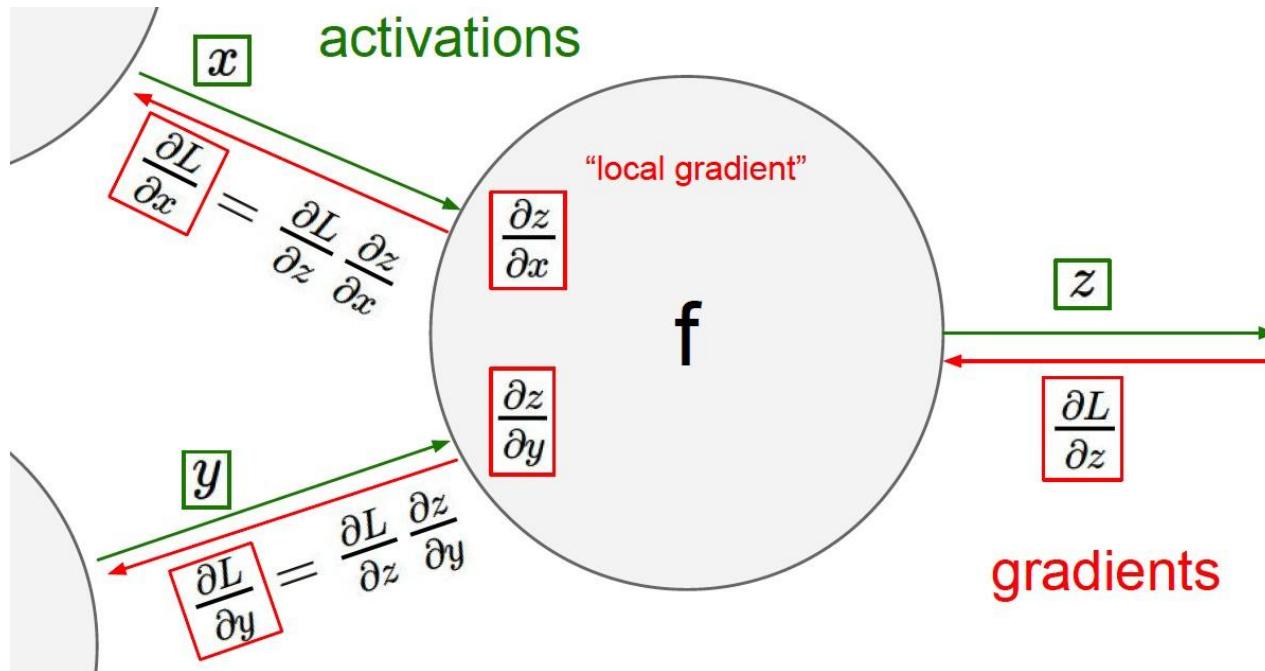
$$q = x + y \quad \frac{dq}{dx} = 1 \quad \frac{dq}{dy} = 1$$

$$f = qz \quad \frac{df}{dq} = z \quad \frac{df}{dz} = q$$

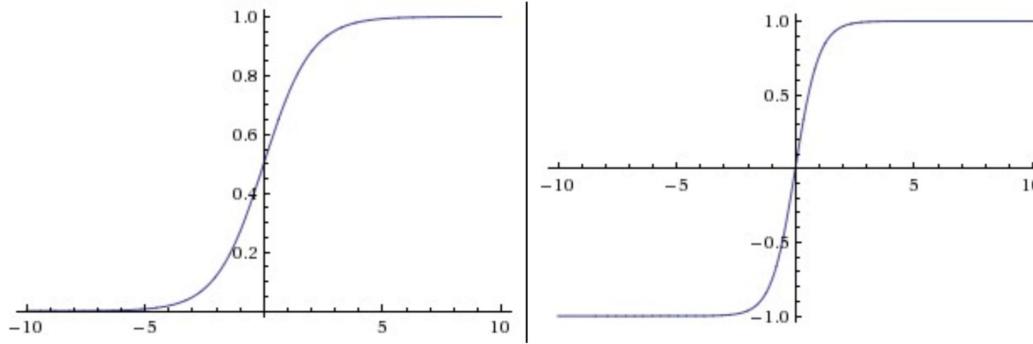
Compute : $\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}$



Backpropagation: key local step



Commonly used activation functions

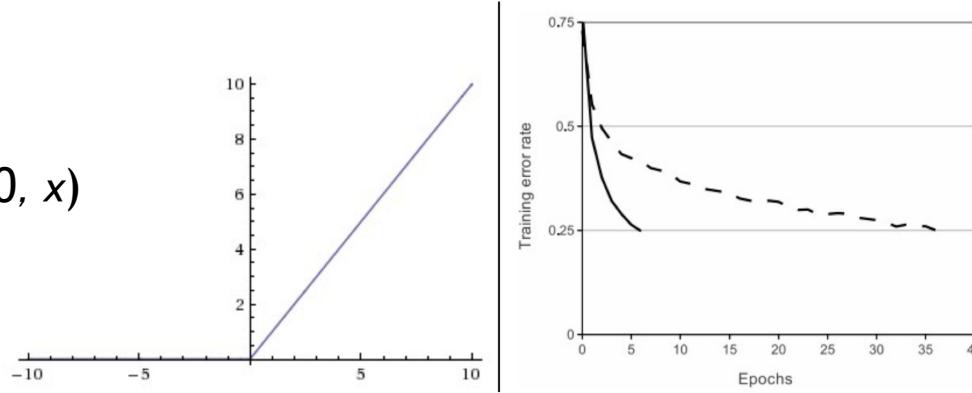


Left: Sigmoid non-linearity squashes real numbers to range between [0,1] Right: The tanh non-linearity squashes real numbers to range between [-1,1].

- Sigmoid weaknesses:
 - saturate and kill gradients
 - outputs not zero-centered
- Tanh outputs are zero centered

Commonly used activation functions

$$f(x) = \max(0, x)$$



Left: Rectified Linear Unit (ReLU) activation function, which is zero when $x < 0$ and then linear with slope 1 when $x > 0$. Right: A plot from Krizhevsky et al. (pdf) paper indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.

- ReLU is often used in modern deep networks
 - Linear, non-saturating form speeds up convergence of stochastic gradient descent; efficient to compute (threshold operation)
 - If learning rate is high, then ReLU units can die i.e., never activate during subsequent training

Commonly used activation functions

- Leaky ReLU
 - Function has small negative slope when $x < 0$ to avoid dying

$$f(x) = I(x < 0)(\alpha * x) + I(x \geq 0)(x)$$

- Maxout
 - Generalizes ReLU and Leaky ReLU; advantages of both but more parameters

$$h(x) = \max(Z_1, Z_2, \dots, Z_n)$$

$$h(x) = \max(W_1 \cdot x + b_1, W_2 \cdot x + b_2, \dots, Wn \cdot x + bn)$$

Representational power

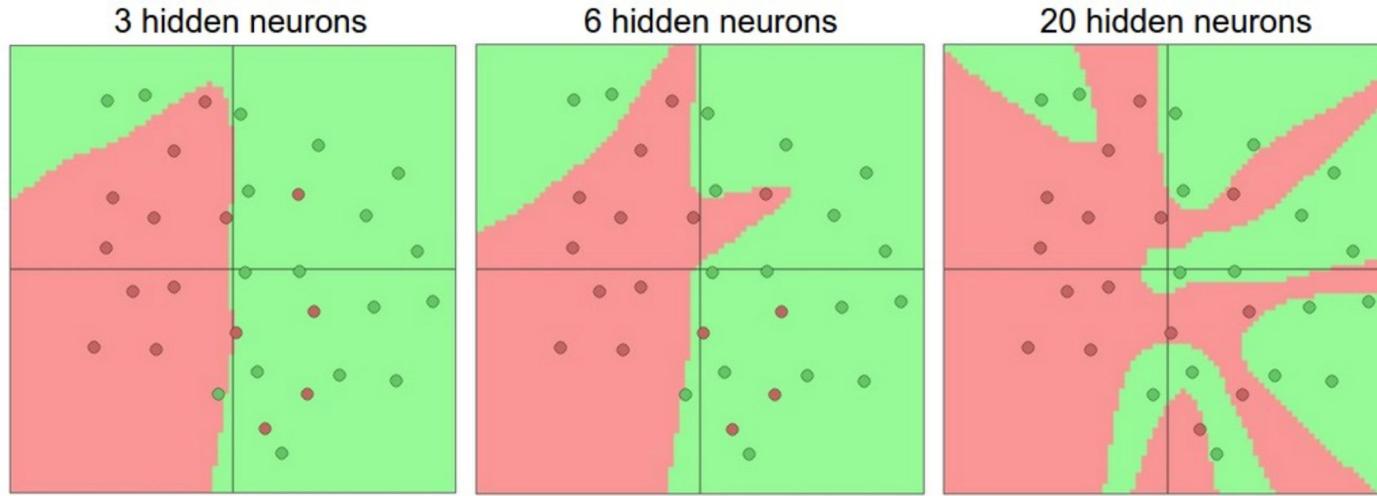
Neural Networks with at least one hidden layer are
universal approximators:

Given any continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x)$ with one hidden layer (with a reasonable choice of non-linearity, e.g. sigmoid) such that $\forall x, |f(x) - g(x)| < \epsilon$

Representational power

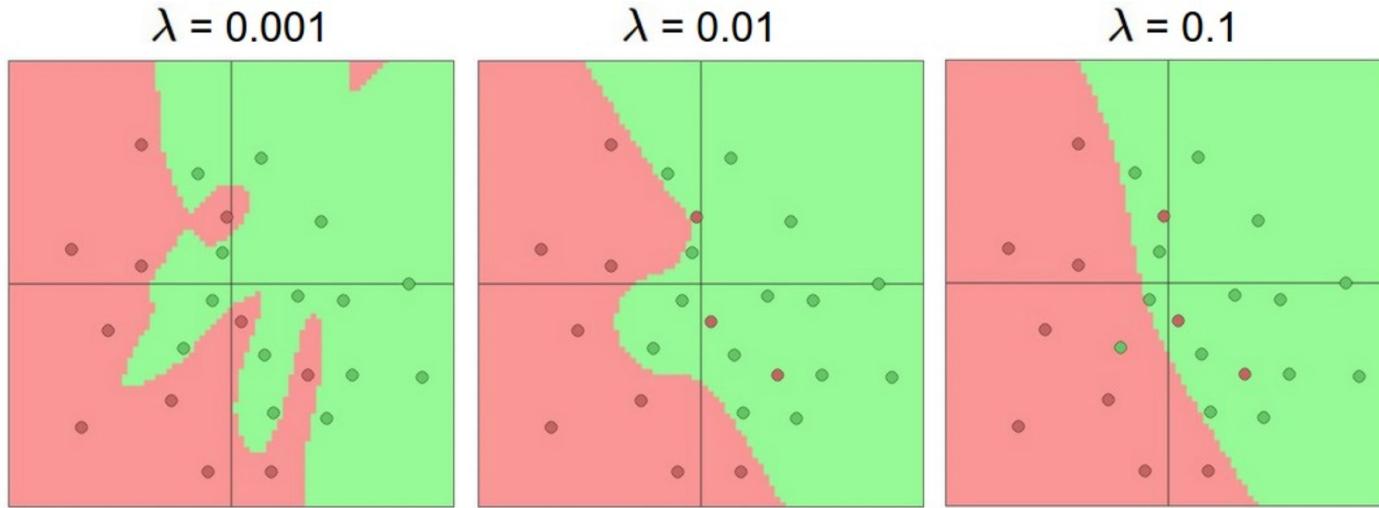
- Neural Networks work well in practice because they compactly express nice, smooth functions that fit well with the statistical properties of data we encounter in practice, and are also easy to learn using our optimization algorithms (e.g. gradient descent).
- The fact that deeper networks (with multiple hidden layers) can work better than a single-hidden-layer networks is an empirical observation, despite the fact that their representational power is equal.

Setting number of layers and their sizes



With more neurons, we have greater representation power but possibly more overfitting

Setting number of layers and their sizes



Train large network; control overfitting with regularization

[The Loss Surfaces of Multilayer Networks](#)

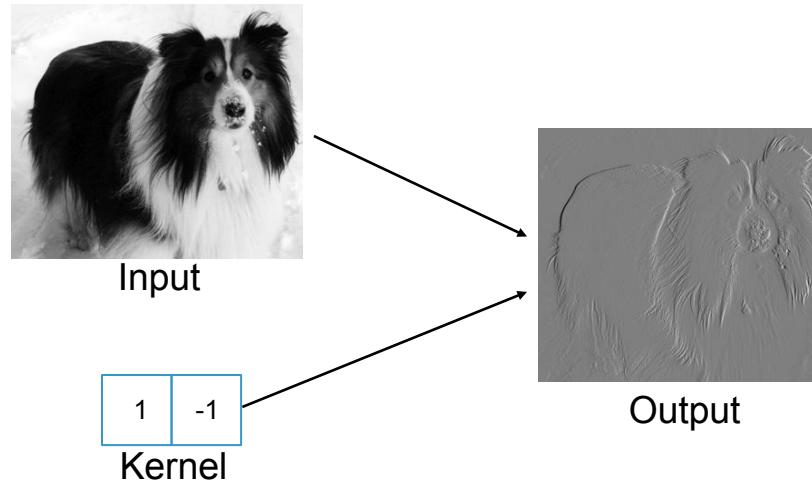
Convolutional Neural Networks [LeCun 1989]

- Specialized to the case where inputs are images (more generally, data with a grid-like topology)
- Sparse connections, parameter sharing
 - Efficient to train
 - Avoid overfitting
- Generalize across spatial translations of input
 - By sliding “filters” that learn distinct patterns (edges, blobs of color etc.)

Key idea

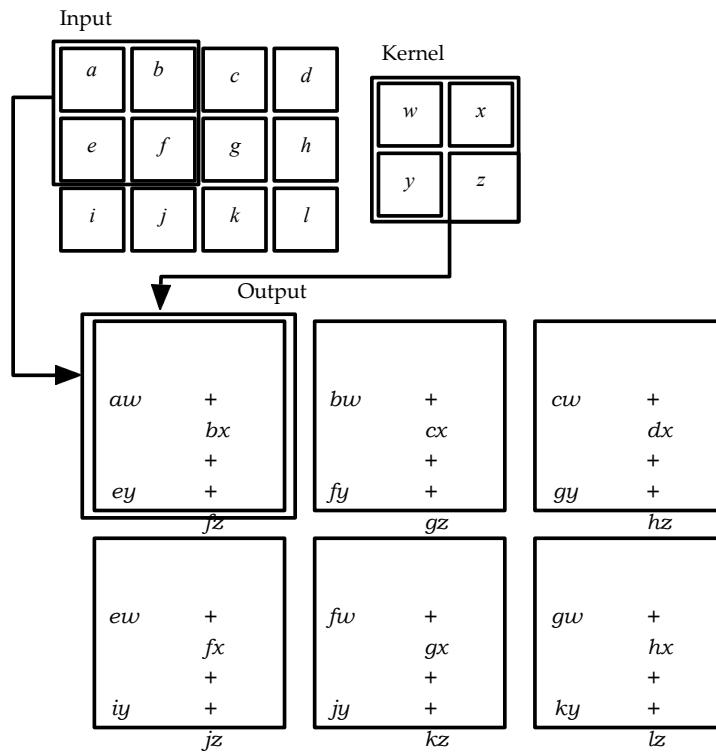
- Replace matrix multiplication in neural networks with convolution
- Everything else remains the same

Edge detection by convolution



(Goodfellow
2016)

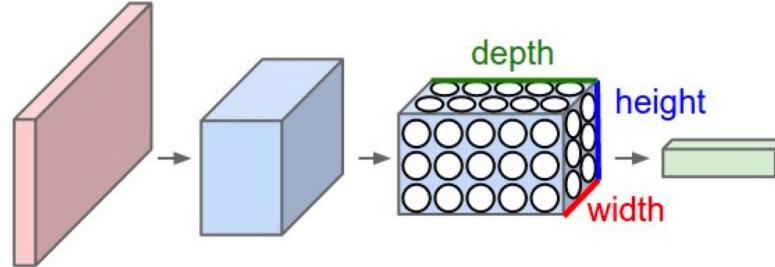
2D Convolution



Sliding filters (kernels)

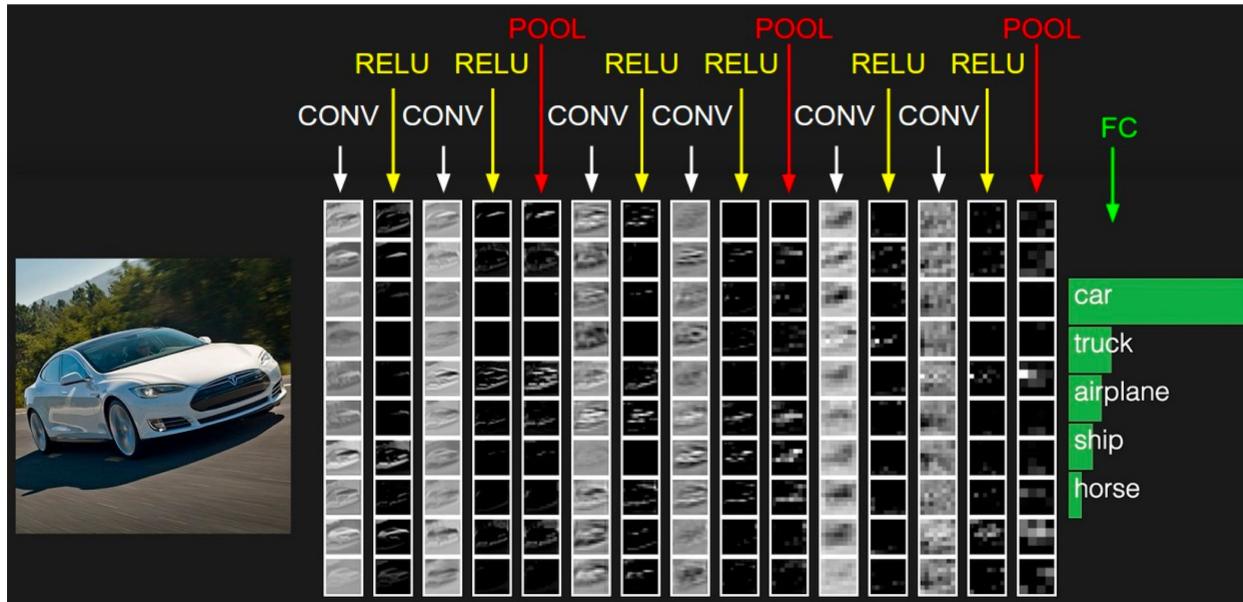
Fig. Goodfellow et al.
2016

Convolutional Neural Networks



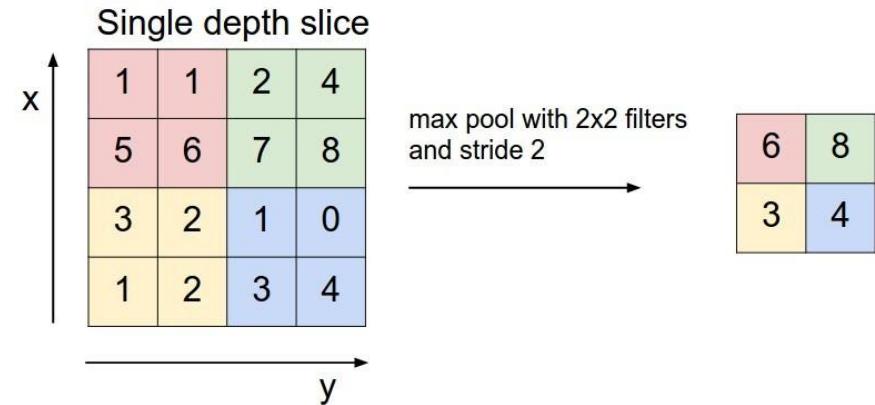
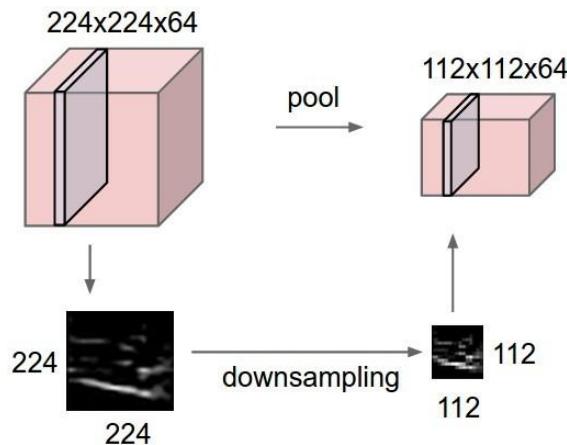
- A ConvNet is made up of Layers
- Every Layer transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters
- Neurons in a layer will only be connected to a small region of the layer before it

Example ConvNet architecture



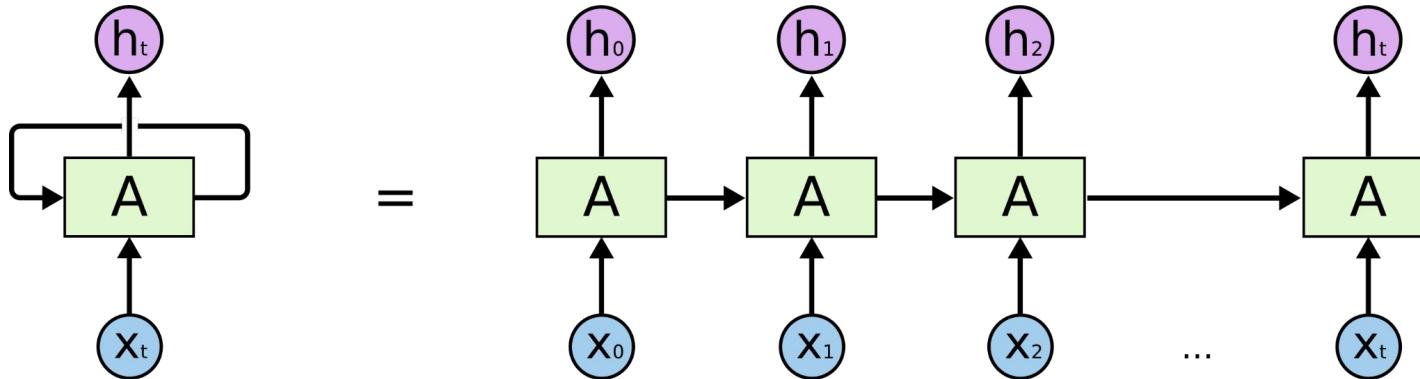
Layers: CONV, RELU, POOL,
FC

Max pooling



Reduce the amount of parameters and computation in the network, and hence to also control overfitting

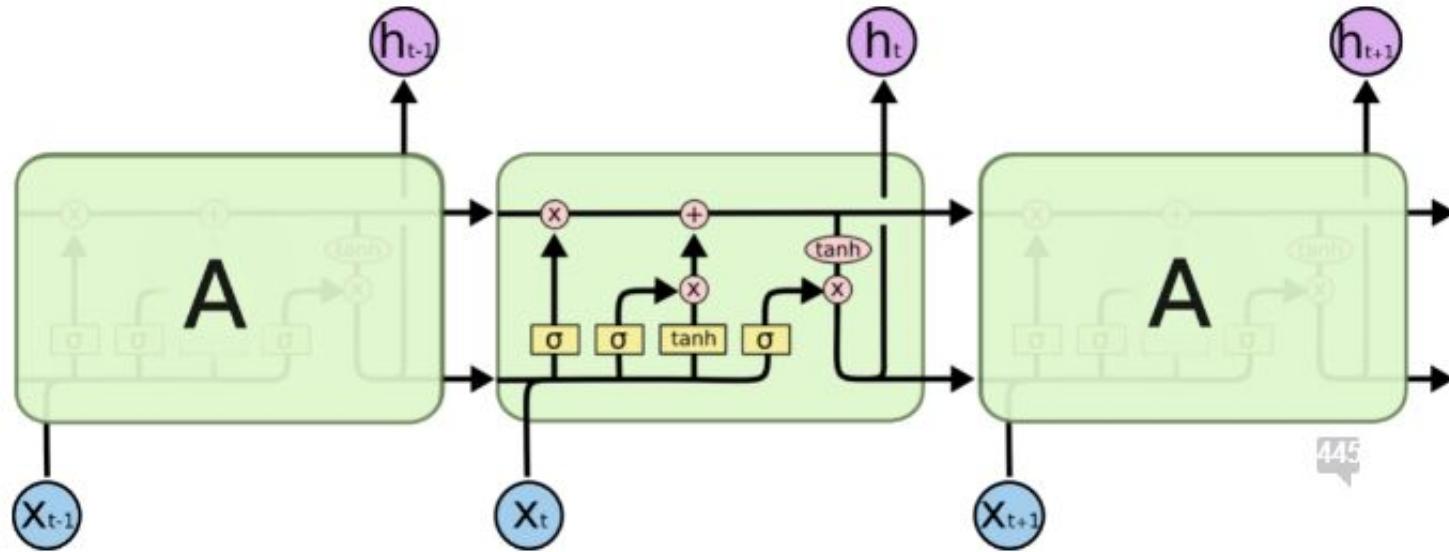
Recurrent Neural networks



- Use reasoning about previous events to predict future upcoming events
- They are networks with loops in them, allowing information to persist.
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

Unfortunately, It's entirely possible for the gap between the relevant information and the point where it is needed to become very large

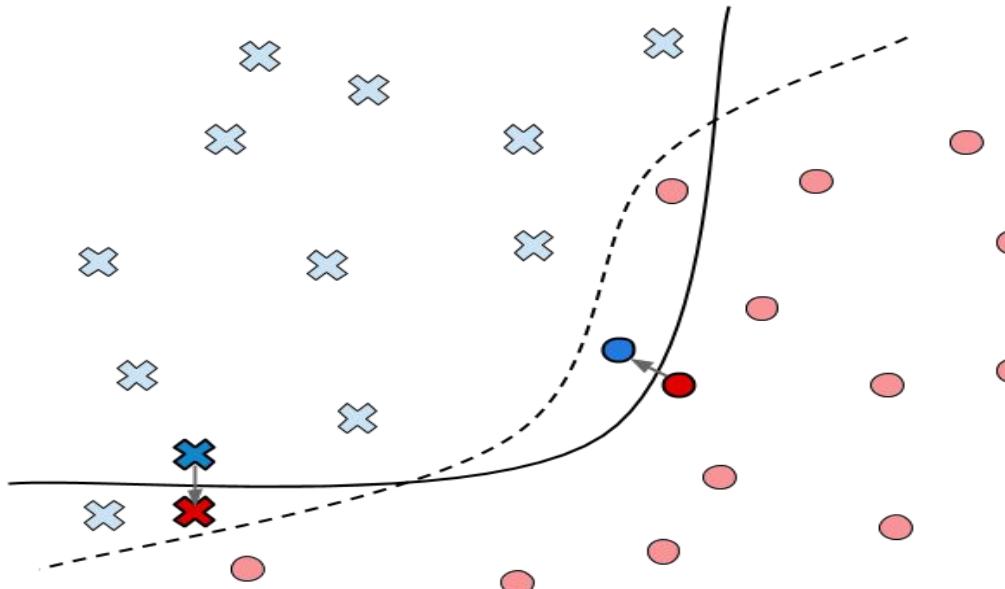
Long Short Term Memory networks



- Special kind of RNN, capable of learning long-term dependencies.
- They are networks with loops in them, allowing information to persist.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

Discussion

What are Adversarial Attacks?



--- Task decision boundary

— Model decision boundary

✖ Test point for class 1

✗ Adversarial example for class 1

✖ Training points for class 1

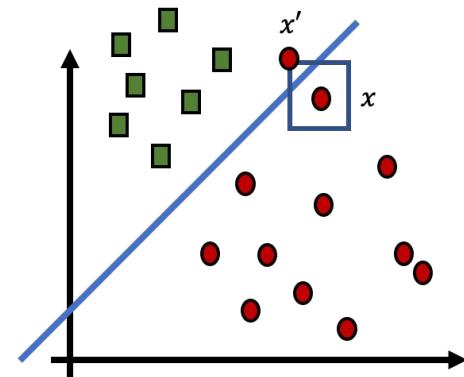
● Training points for class 2

● Test point for class 2

● Adversarial example for class 2

White-Box Adversarial Attack: FGSM and PGD

- Attacker's knowledge:
- Suppose model F with parameter θ is given to attacker;
- Goal:
 - For a test sample x with label y (true)
 - Find δ such that $F(x + \delta) \neq y$



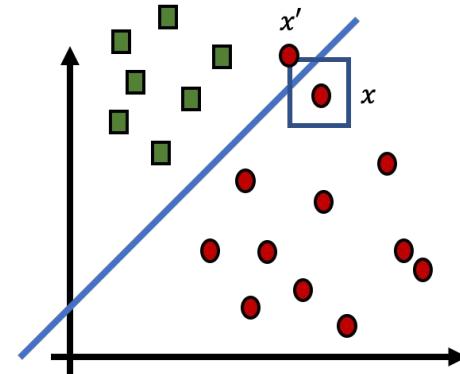
White-Box Adversarial Attack: FGSM and PGD

- Attacker's knowledge:
- Suppose model F with parameter θ is given to attacker;

- Goal:

- For a test sample x with label y (true)
- Find δ such that $F(x + \delta) \neq y$

- δ is under the constraints (similarity):
 - 1. l_p -norm: $\|\delta\|_\infty \leq \epsilon$
 - 2. spatial transformation
 - 3. more similarity measurements



White-Box Adversarial Attack: FGSM and PGD

- Goal:

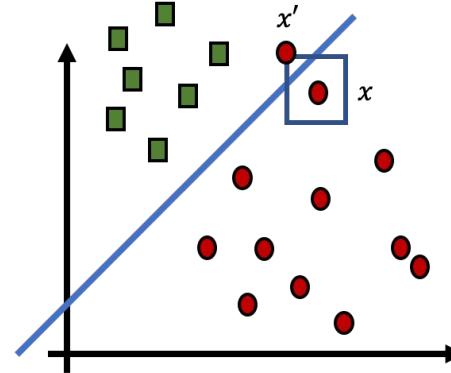
Find δ such that $F(x + \delta) \neq y$

Subject to $\|\delta\|_\infty \leq \epsilon$

White-Box Adversarial Attack: FGSM and PGD

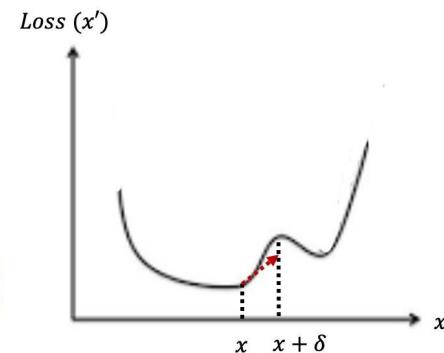
- Goal:
 - Find δ such that $F(x + \delta) \neq y$
 - Subject to $\|\delta\|_\infty \leq \epsilon$

- Recall: Goal of Training:
 - $\min Loss(F(x; \theta), y)$
- Now: Goal of Attack:
 - $\max_{\delta} Loss(F(x + \delta; \theta), y)$
 - Subject to $\|\delta\|_\infty \leq \epsilon$



White-Box Adversarial Attack: FGSM and PGD

- Now: Goal of Attack:
- $\max_{\delta} \text{Loss}(F(x + \delta; \theta), y)$
- Subject to $\|\delta\|_{\infty} \leq \epsilon$
- Fast Gradient Sign Method (FGSM)
- $x + \delta = x + \epsilon \cdot \text{sign}(\nabla_x \text{Loss}(F(x; \theta), y))$



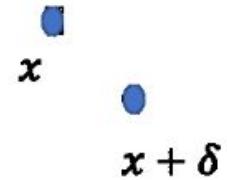
One Step Gradient Ascent to find bad δ

White-Box Adversarial Attack: FGSM and PGD

- Fast Gradient Sign Method (FGSM)
- $\delta = \epsilon \cdot \text{sign}(\nabla_x \text{Loss}(F(x; \theta), y))$

- Projected Gradient Descent (PGD)
- $x + \delta = \text{clip}_{(x, \epsilon)}\{\bar{x} + \alpha \cdot \text{sign}(\nabla_x \text{Loss}(F(x; \theta), y))\}$

PGD attack



$x + \delta$: most dangerous example

White-Box Adversarial Attack: FGSM and PGD

- Goal of Attack:
- $\max_{\delta} \text{Loss}(F(x + \delta; \theta), y) = L_{adv}(x; \theta)$
- Subject to $\|\delta\|_{\infty} \leq \epsilon$



Adversarial Risk of F on x

White-Box Adversarial Attack: C & W's Attack (CW)

- Optimizes for having the minimal distance from the original example, under the constraint of having the example be misclassified by the original model

- Goal of Attack:

- $\min_{\delta} \|\delta\|_{\infty}$

- Subject to $F(x + \delta; \theta) \neq y$

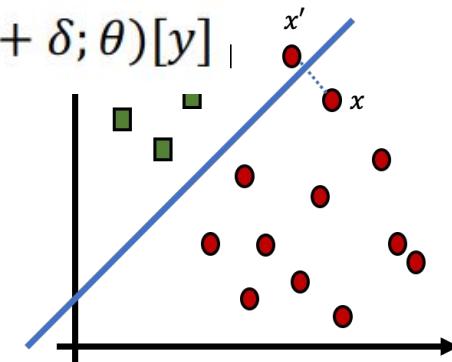
- Margin Loss:

- $margin_loss(x + \delta) = \max_{i \neq y} Z(x + \delta; \theta)[i] - Z(x + \delta; \theta)[y]$

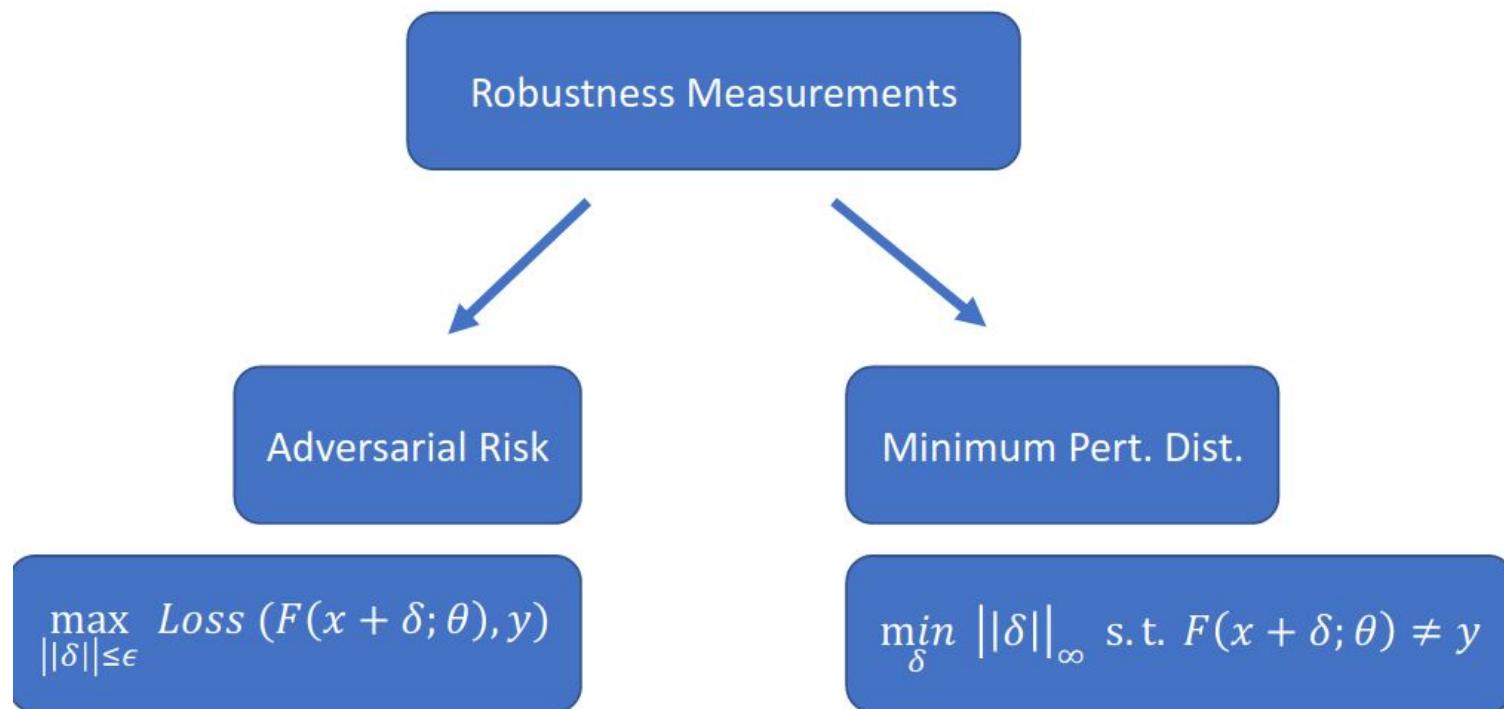
- Goal of Attack (new):

- $\min_{\delta} \|\delta\|_{\infty}$

- Subject to $margin_loss(x) < 0$



Robustness Measurement



Winning attack at NIPS 2017

Boosting Adversarial attacks with Momentum (MI-FGSM)

Input: A classifier f with loss function J ; a real example \mathbf{x} and ground-truth label y ;

Input: The size of perturbation ϵ ; iterations T and decay factor μ .

Output: An adversarial example \mathbf{x}^* with $\|\mathbf{x}^* - \mathbf{x}\|_\infty \leq \epsilon$.

- 1: $\alpha = \epsilon/T$;
- 2: $\mathbf{g}_0 = 0$; $\mathbf{x}_0^* = \mathbf{x}$;
- 3: **for** $t = 0$ to $T - 1$ **do**
- 4: Input \mathbf{x}_t^* to f and obtain the gradient $\nabla_{\mathbf{x}} J(\mathbf{x}_t^*, y)$;
- 5: Update \mathbf{g}_{t+1} by accumulating the velocity vector in the gradient direction as

$$\mathbf{g}_{t+1} = \mu \cdot \mathbf{g}_t + \frac{\nabla_{\mathbf{x}} J(\mathbf{x}_t^*, y)}{\|\nabla_{\mathbf{x}} J(\mathbf{x}_t^*, y)\|_1}; \quad (6)$$

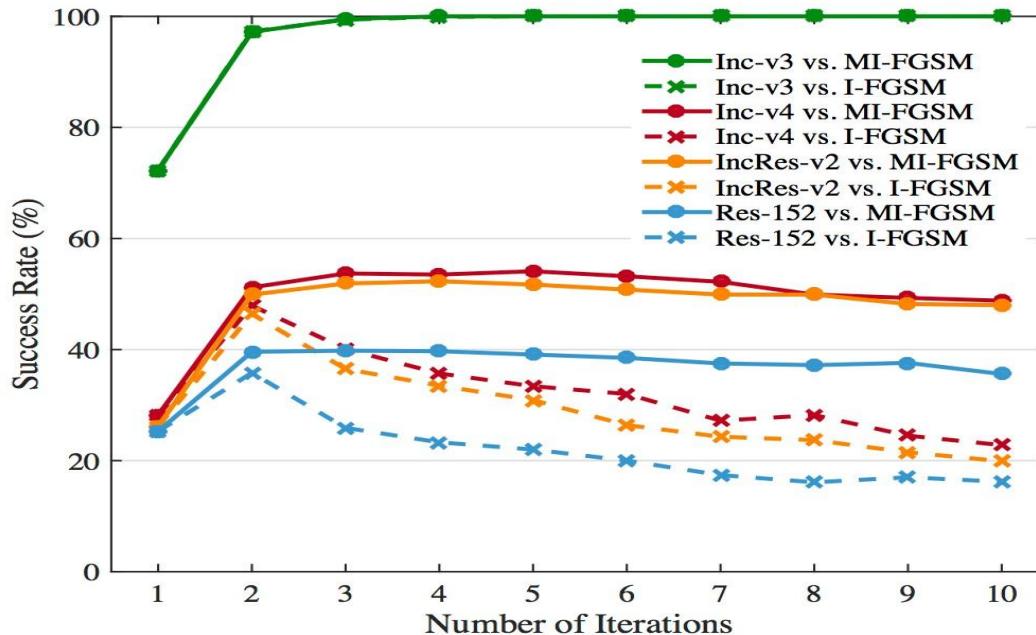
- 6: Update \mathbf{x}_{t+1}^* by applying the sign gradient as

$$\mathbf{x}_{t+1}^* = \mathbf{x}_t^* + \alpha \cdot \text{sign}(\mathbf{g}_{t+1}); \quad (7)$$

- 7: **end for**
- 8: **return** $\mathbf{x}^* = \mathbf{x}_T^*$.

Winning attack at NIPS 2017

Boosting Adversarial attacks with Momentum (MI-FGSM)



- **Deep Fool:**
 - Iteratively “linearizes” the loss function at an input point (taking the tangent to the loss function at that point), and applies the minimal perturbation necessary.

ADVERSARIAL EXAMPLES IN THE PHYSICAL WORLD

Alexey Kurakin

Google Brain

kurakin@google.com

Ian J. Goodfellow

OpenAI

ian@openai.com

Samy Bengio

Google Brain

bengio@google.com

ABSTRACT

Most existing machine learning classifiers are highly vulnerable to adversarial examples. An adversarial example is a sample of input data which has been modified very slightly in a way that is intended to cause a machine learning classifier to misclassify it. In many cases, these modifications can be so subtle that a human observer does not even notice the modification at all, yet the classifier still makes a mistake. Adversarial examples pose security concerns because they could be used to perform an attack on machine learning systems, even if the adversary has no access to the underlying model. Up to now, all previous work has assumed a threat model in which the adversary can feed data directly into the machine learning classifier. This is not always the case for systems operating in the physical world, for example those which are using signals from cameras and other sensors as input. This paper shows that even in such physical world scenarios, machine learning systems are vulnerable to adversarial examples. We demonstrate this by feeding adversarial images obtained from a cell-phone camera to an ImageNet Inception classifier and measuring the classification accuracy of the system. We find that a large fraction of adversarial examples are classified incorrectly even when perceived through the camera.

1 INTRODUCTION



Adversarial examples can be printed out on normal paper and photographed with a standard resolution smartphone and still cause a classifier to, in this case, label a “washer” as a “safe”.

Source: <http://www.cleverhans.io/security/priveeasy.html>

Adversarial Patch

Tom B. Brown, Dandelion Mané*, Aurko Roy, Martín Abadi, Justin Gilmer
`{tombrown,dandelion,aurkor,abadi,gilmer}@google.com`

Abstract

We present a method to create universal, robust, targeted adversarial image patches in the real world. The patches are universal because they can be used to attack any scene, robust because they work under a wide variety of transformations, and targeted because they can cause a classifier to output any target class. These adversarial patches can be printed, added to any scene, photographed, and presented to image classifiers; even when the patches are small, they cause the classifiers to ignore the other items in the scene and report a chosen target class.

1 Introduction

Deep learning systems are broadly vulnerable to adversarial examples, carefully chosen inputs that cause the network to change output without a visible change to a human [15, 5]. These adversarial examples most commonly modify each pixel by only a small amount and can be found using a number of optimization strategies such as L-BFGS [15], Fast Gradient Sign Method (FGSM) [5], DeepFool [10], Projected Gradient Descent (PGD) [10] as well as the recently proposed Least squares

<https://arxiv.org/pdf/1712.09665.pdf>

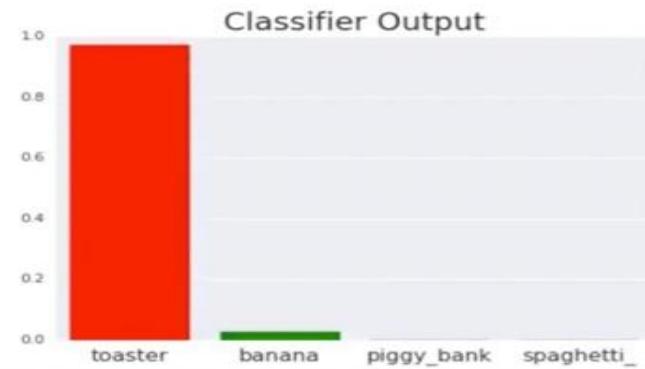
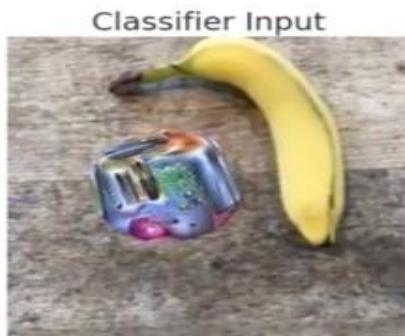
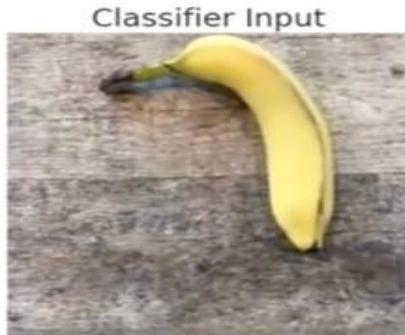
Classifier Input



Classifier Output



Demo



Discussion

How do you defend A.I. systems from these attacks?

- **Adversarial training**
 - Generate a lot of adversarial examples and explicitly train the model not to be fooled by each of them
 - Improves the generalization of a model when presented with adversarial examples at test time.

Adversarial Training [Madry 2018]

- Adversarial Risk:

$$L_{adv}(F(x; \theta), y) = \max_{\|\delta\| \leq \epsilon} Loss(F(x + \delta; \theta), y)$$

- Training model to minimize (empirical) adversarial risk:

$$\min_{\theta} \sum_{(x,y) \sim D} \max_{\|\delta\| \leq \epsilon} Loss(F(x + \delta; \theta), y)$$

Adversarial Training [Madry 2018]

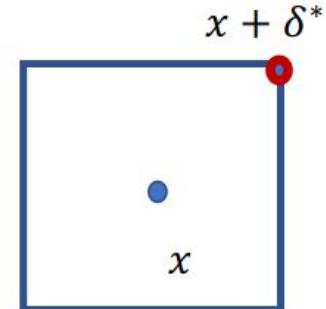
- Goal: Training model to minimize (empirical) adversarial risk:

$$\min_{\theta} \sum_{(x,y) \sim D} \max_{\|\delta\| \leq \epsilon} \text{Loss}(F(x + \delta; \theta), y)$$

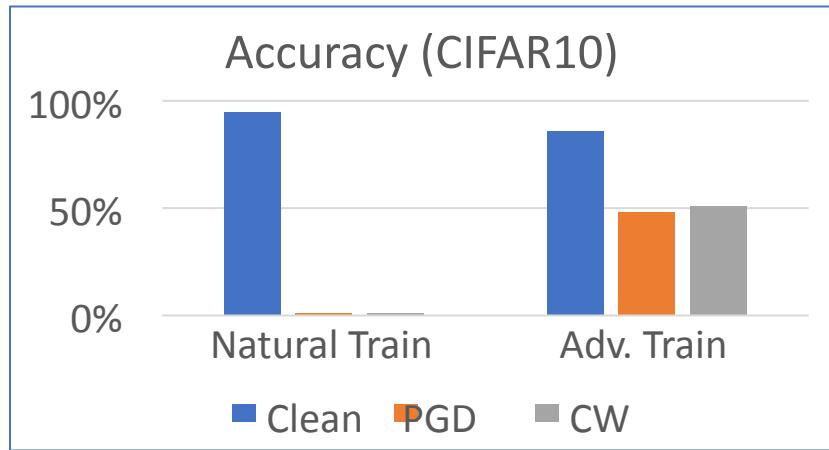
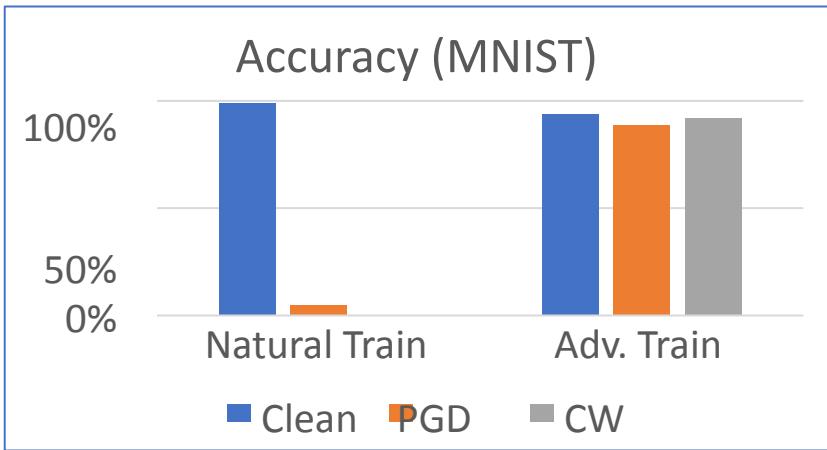
- Solution: Adversarial Training

- For each batch:

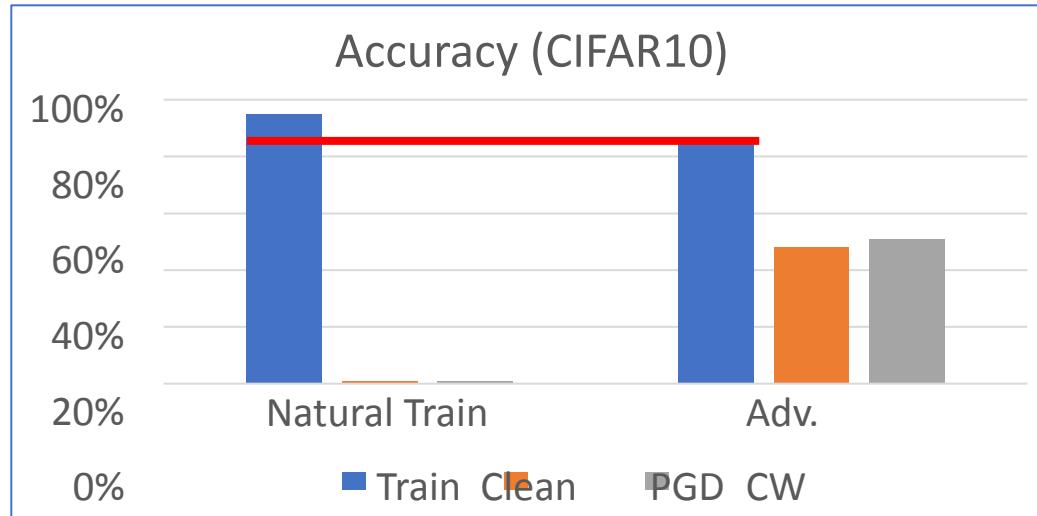
1. Solve the inner maximization problem to find optimal δ^*
2. Update θ the to minimizing loss value on $x + \delta^*$



Success of Adversarial Training



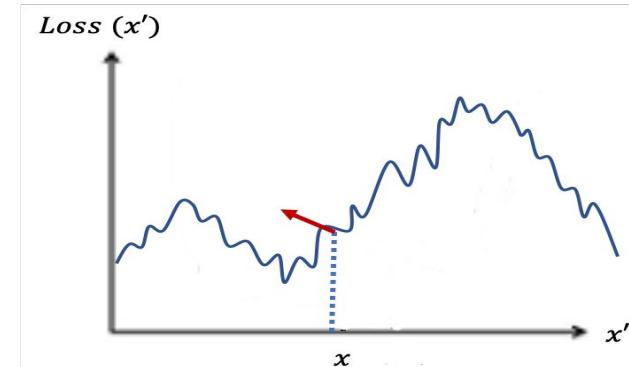
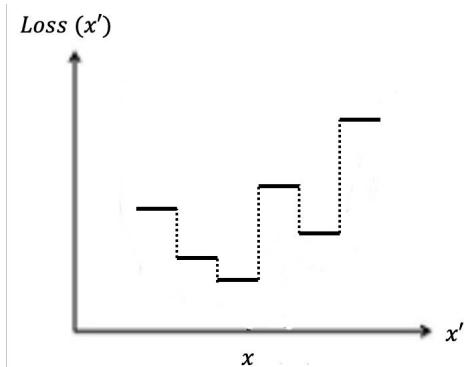
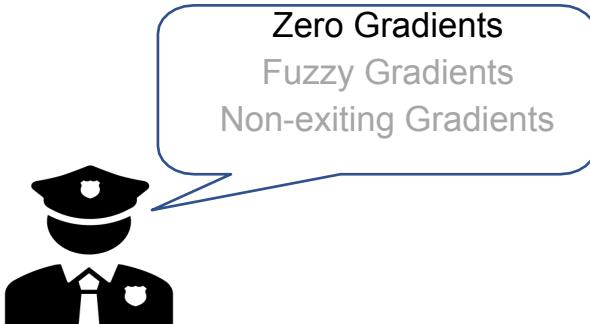
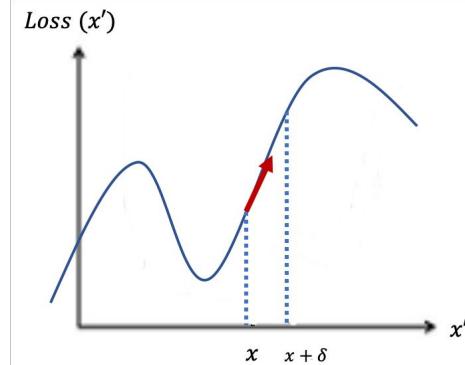
Adversarial Robustness may hurt Clean Accuracy



Gradient Obfuscation / Masking

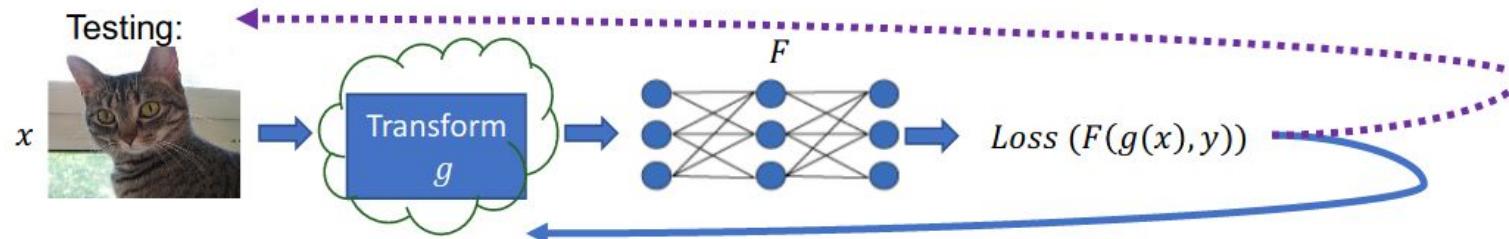
One Example : (FGSM)

$$\delta = \epsilon \cdot \text{sign} (\nabla_x \text{Loss} (F(x; \theta), y))$$



Adaptive attack on Gradient Obfuscation / Masking

- Input Transformation Defense [Guo 2018]



- Bit-depth Reduction g preserves most information of x : $g(x) \approx x$
- Key Idea: Use $h(x) = x$ to replace g when calculating gradients:

$$\frac{\partial \text{Loss}(F(\textcolor{red}{g}(x), y))}{\partial x} \approx \frac{\partial \text{Loss}(F(g(x), y))}{\partial g(x)} \cdot \frac{\partial h(x)}{\partial x}$$

- Use approximated gradients for PGD / FGSM, succeed attack!

Adversarial Robustness may hurt Clean Accuracy [TRADES, Zhang 2019]

- Adversarial Risk:

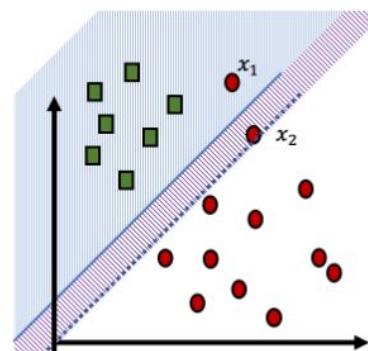
$$L_{adv}(F(x; \theta), y) = \max_{\|\delta\| \leq \epsilon} Loss(F(x + \delta; \theta), y)$$

- Robust Error = Standard Error + Boundary Error:

$$R_{adv}(F(x; \theta), y) = Prob. \{F(x) \neq y\} + Prob. \{\exists \delta \in B(X, \epsilon), F(x + \delta) \neq F(x)\}$$

- New Objective for Adversarial Training: (TRADES, Zhang 2019)

$$L_{Trades}(F(x; \theta), y) = Loss(F(x; \theta), y) + \max_{\|\delta\| \leq \epsilon} Loss(F(x + \delta; \theta), F(x; \theta))$$



Intriguing properties of neural networks

Christian Szegedy

Google Inc.

Wojciech Zaremba

New York University

Ilya Sutskever

Google Inc.

Joan Bruna

New York University

Dumitru Erhan

Google Inc.

Ian Goodfellow

University of Montreal

Rob Fergus

New York University

Facebook Inc.

Abstract

Deep neural networks are highly expressive models that have recently achieved state of the art performance on speech and visual recognition tasks. While their expressiveness is the reason they succeed, it also causes them to learn uninterpretable solutions that could have counter-intuitive properties. In this paper we report two such properties.

First, we find that there is no distinction between individual high level units and random linear combinations of high level units, according to various methods of unit analysis. It suggests that it is the space, rather than the individual units, that contains the semantic information in the high layers of neural networks.

Second, we find that deep neural networks learn input-output mappings that are fairly discontinuous to a significant extent. We can cause the network to misclassify an image by applying a certain hardly perceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input.

1 Introduction

<https://arxiv.org/pdf/1312.6199.pdf>

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy

Google Inc., Mountain View, CA

{goodfellow, shlens, szegedy}@google.com

ABSTRACT

Several machine learning models, including neural networks, consistently misclassify *adversarial examples*—inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. Early attempts at explaining this phenomenon focused on nonlinearity and overfitting. We argue instead that the primary cause of neural networks’ vulnerability to adversarial perturbation is their linear nature. This explanation is supported by new quantitative results while giving the first explanation of the most intriguing fact about them: their generalization across architectures and training sets. Moreover, this view yields a simple and fast method of generating adversarial examples. Using this approach to provide examples for adversarial training, we reduce the test set error of a maxout network on the MNIST dataset.

1 INTRODUCTION

Szegedy et al. (2014b) made an intriguing discovery: several machine learning models, including state-of-the-art neural networks, are vulnerable to *adversarial examples*. That is, these machine learning models misclassify examples that are only slightly different from correctly classified examples drawn from the data distribution. In many cases, a wide variety of models with different architectures trained on different subsets of the training data misclassify the same adversarial example. This suggests that adversarial examples expose fundamental blind spots in our training algorithms.

How do you defend A.I. systems from these attacks?

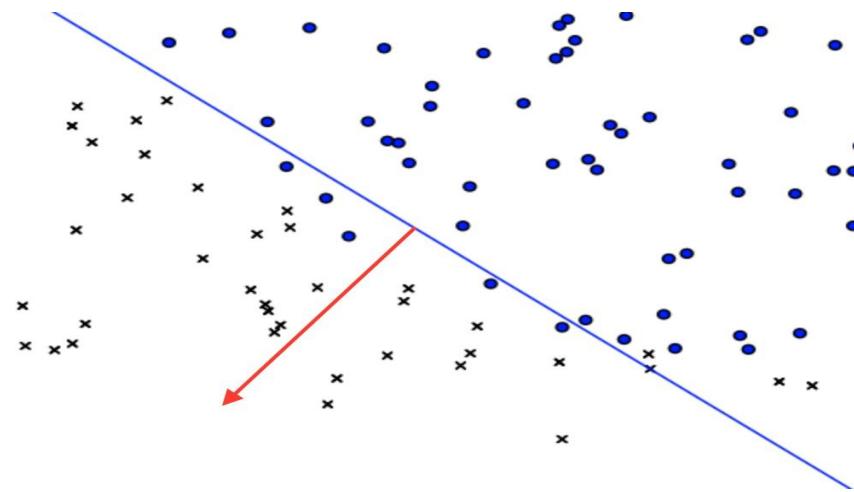
- **Defensive distillation** smooths the model's decision surface in adversarial directions exploited by the adversary.
 - Train the model to output probabilities of different classes, rather than hard decisions about which class to output.
 - Creates a model whose surface is smoothed in the directions an adversary will typically try to exploit.

Motivation

Are adversarial examples simply a fun toy problem for researchers?

Or an example of a deeper and more chronic frailty in our models?

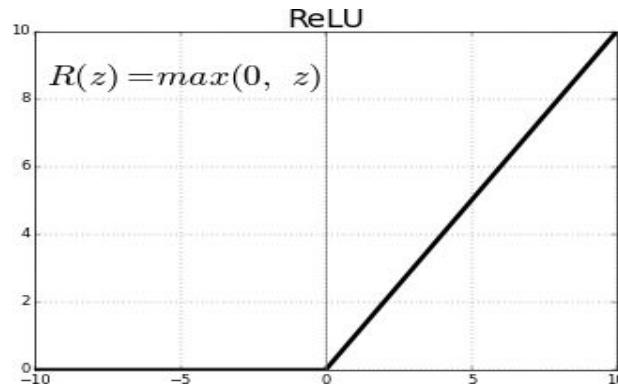
Model linearity



In the example above, if we move in a direction perpendicular to the decision boundary, we can, with a relatively small-magnitude vector, push ourselves to a place where the model is very confident in the wrong direction

- Linear models' behavior outside of the region where training data is concentrated is quite pathological.
- In a high-dimensional space, each individual pixel might only increase by a very small amount, but have those small differences contribute to a dramatic difference in the weights * inputs dot product.

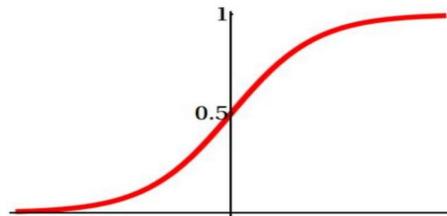
Model linearity



Within the space of possible nonlinear activation functions, modern deep nets have actually settled on one that is very close to linear: the Rectified Linear Units. (ReLU)

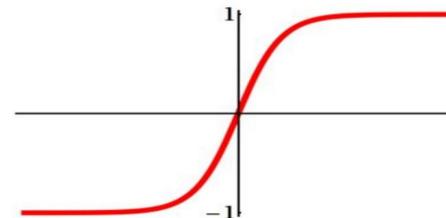
Model linearity

$$\sigma(\Sigma) = \frac{1}{1 + e^{-\Sigma}}$$



logistic (sigmoid, unipolar)

$$\tanh(\Sigma) = \frac{e^{\Sigma} - e^{-\Sigma}}{e^{\Sigma} + e^{-\Sigma}}$$

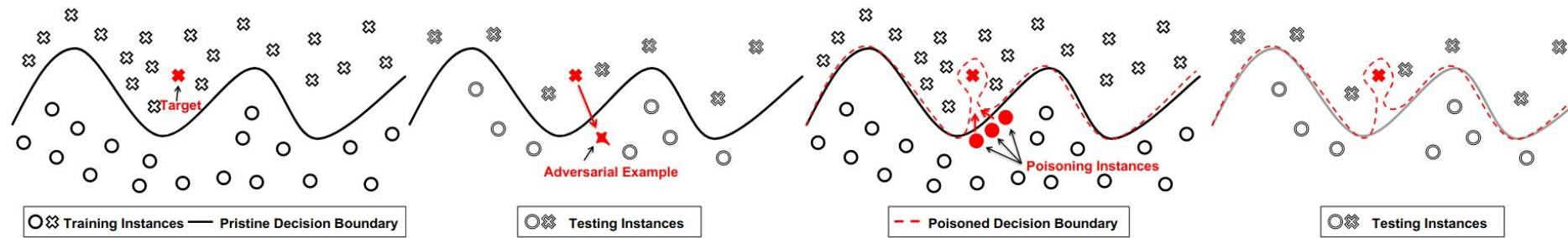


tanh (bipolar)

From Ian Goodfellow's key paper on the topic:

"Using a network that has been designed to be sufficiently linear—whether it is a ReLU or maxout network, an LSTM, or a sigmoid network that has been carefully configured not to saturate too much— we are able to fit most problems we care about, at least on the training set. The existence of adversarial examples suggests that being able to explain the training data or even being able to correctly label the test data does not imply that our models truly understand the tasks we have asked them to perform. Instead, their linear responses are overly confident at points that do not occur in the data distribution, and these confident predictions are often highly incorrect. ...One may also conclude that the model families we use are intrinsically flawed. Ease of optimization has come at the cost of models that are easily misled."

Data Poisoning



Source: <https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-suciu.pdf>

Data Poisining

- This technique involves an attacker inserting corrupt data in the training dataset to compromise a target machine learning model during training.
- Some data poisoning techniques aim to trigger a specific behavior in the model when it faces a specific pattern at inference time

Data Poisining

- Other data poisoning techniques aim to reduce the accuracy of a machine learning model on one or more output classes. In this case, the attacker would insert carefully crafted adversarial examples into the dataset used to train the model. These manipulated examples are virtually impossible to detect because their modifications are not visible to the human eye.

Data Poisoning

- How Robust are Randomized Smoothing based Defenses to Data Poisoning?
<https://arxiv.org/pdf/2012.01274.pdf>
 - Poisoning Against Certified Defenses generates poisoned data that have been optimized for specific adversarial robustness techniques.

