



Carnegie Mellon University
Language Technologies Institute

Adversarial attacks on machine learning systems

Bhiksha Raj
Carnegie Mellon University

TSD 2019
13 Sep 2019

Acknowledgements..

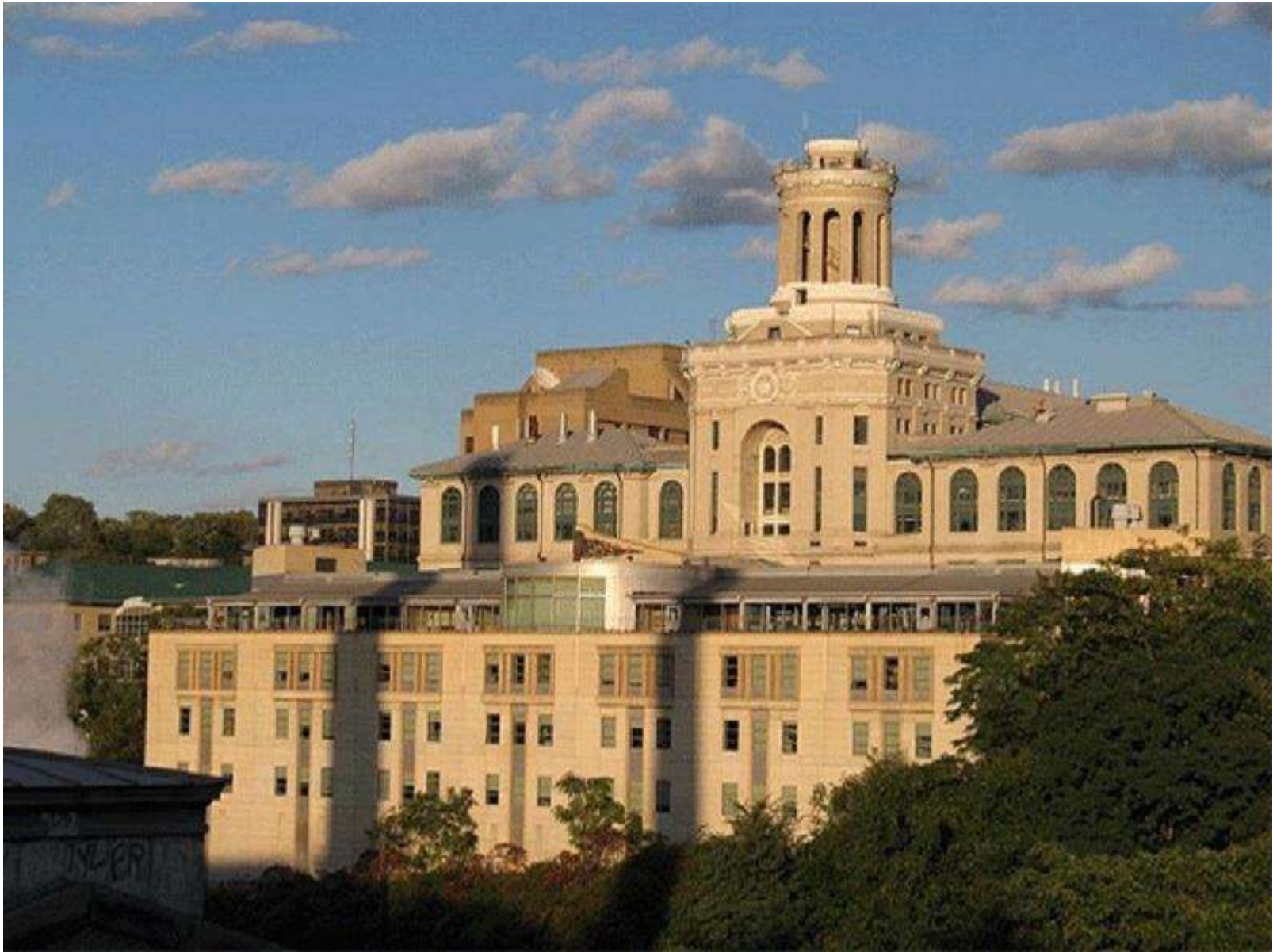
- Thanks (in alphabetic order) to:
 - Anders Oland
 - Ahmed Shah
 - Gerald Friedland
 - Nicolas Papernot
 - Joseph Keshet
 - Raphael Olivier
 - Rita Singh
- Whose material I am using (with permission)..
- Other collaborators:
 - Pulkit Agarwal
 - Nicholas Wolfe

Introducing me

- Bhiksha Raj
 - Professor
 - Carnegie Mellon University
 - Language Technologies
 - Electrical and Computer Engg.
 - Machine Learning
 - Music Technologies
- Research Areas
 - Automatic speech recognition
 - Audio intelligence
 - Machine learning and sparse optimization
 - **Deep learning**
 - **Data privacy**



CMU





Artificial Intelligence

Ranked in 2018, part of [Computer Science](#)

Artificial intelligence is an evolving field that requires broad training, so courses typically involve principles of computer science, cognitive psychology and engineering. These are the best artificial intelligence programs.

Specialty ✕ ▾

Rank	School name
#1	Carnegie Mellon University Pittsburgh, PA
#2	Massachusetts Institute of Technology Cambridge, MA
#3	Stanford University Stanford, CA
#4	University of California—Berkeley Berkeley, CA



Best Graduate Computer Science Programs

Ranked in 2018 | [Best Graduate Computer Science Programs Rankings](#)
[Methodology](#)

Earning a graduate degree in computer science can lead to positions in research institutions, government agencies, technology companies and colleges and universities. These are the top computer science schools. Each school's score reflects its average rating on a scale from 1 (marginal) to 5 (outstanding), based on a survey of academics at peer institutions.

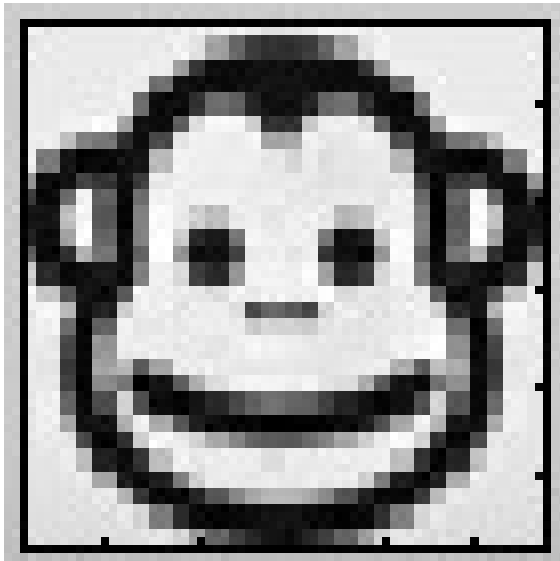
Specialty

Rank	School name	Score
#1 Tie	Carnegie Mellon University Pittsburgh, PA	5.0
#1 Tie	Massachusetts Institute of Technology Cambridge, MA	5.0
#1 Tie	Stanford University Stanford, CA	5.0
#1 Tie	University of California—Berkeley Berkeley, CA	5.0

The story of “O”



One day in summer 2013



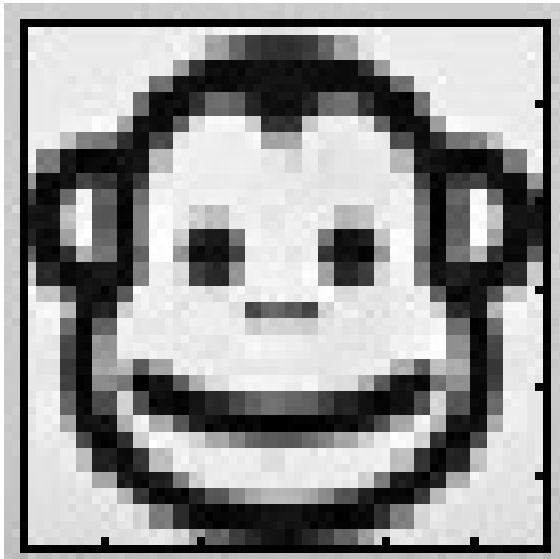
Boss, my
MNIST
recognizer
thinks this
monkey is the
number 2!



The story of O:

“O” (a PhD student) has just written his own ultra-efficient distributed matlab-based deep learning toolkit

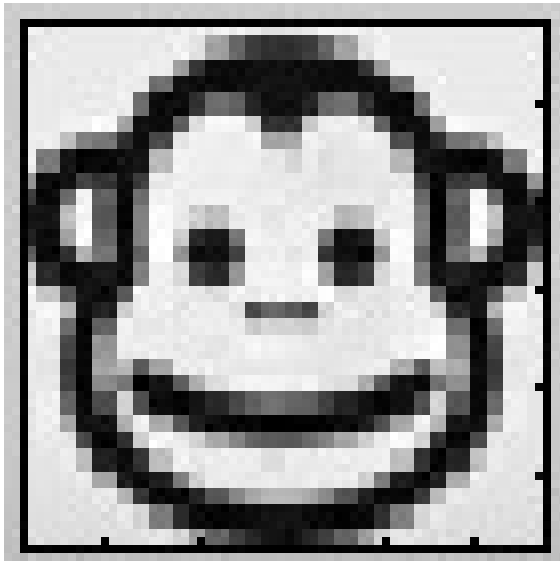
One day in summer 2013



2 2 2 2 2 2 2 2 2
2 2 2 2 0 1 2 2 2
2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2
2 0 1 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2

??

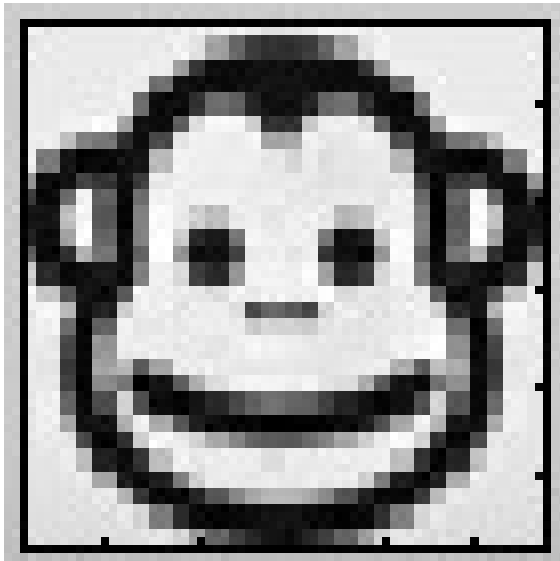
One day in summer 2013



No way! That
looks more like
an 8 or a 0



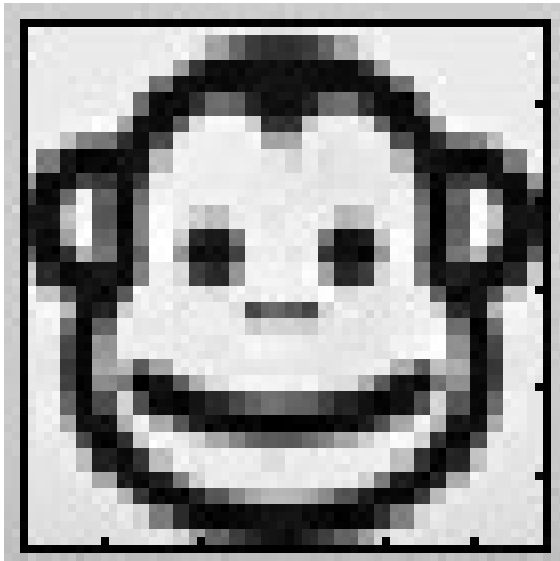
One day in summer 2013



Nope! It's the
number 2!



One day in summer 2013



Hm! I wonder why. Try erasing the smile.



One day in summer 2013



Now its an 8!



One day in summer 2013

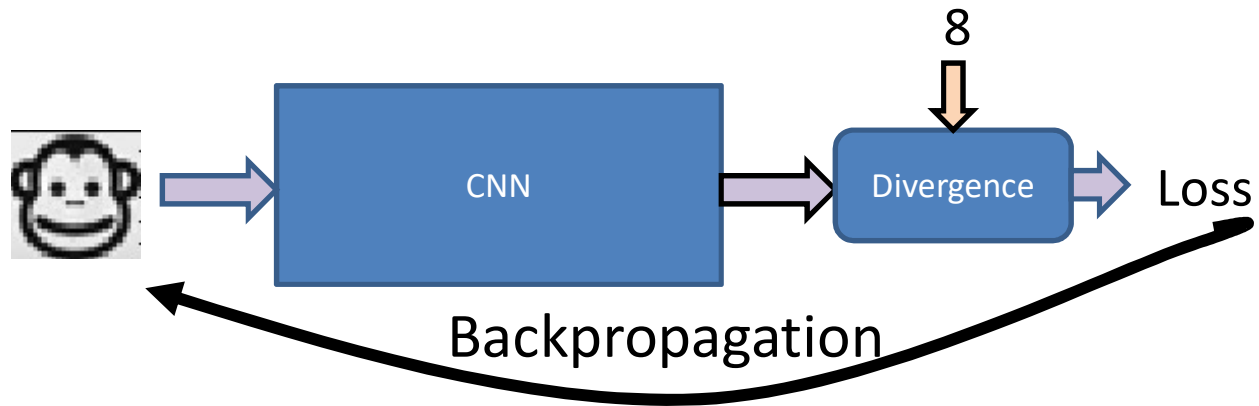
Can we
automatically figure
out how to edit it to
make it 8?



Sure! I
know how to
do it.



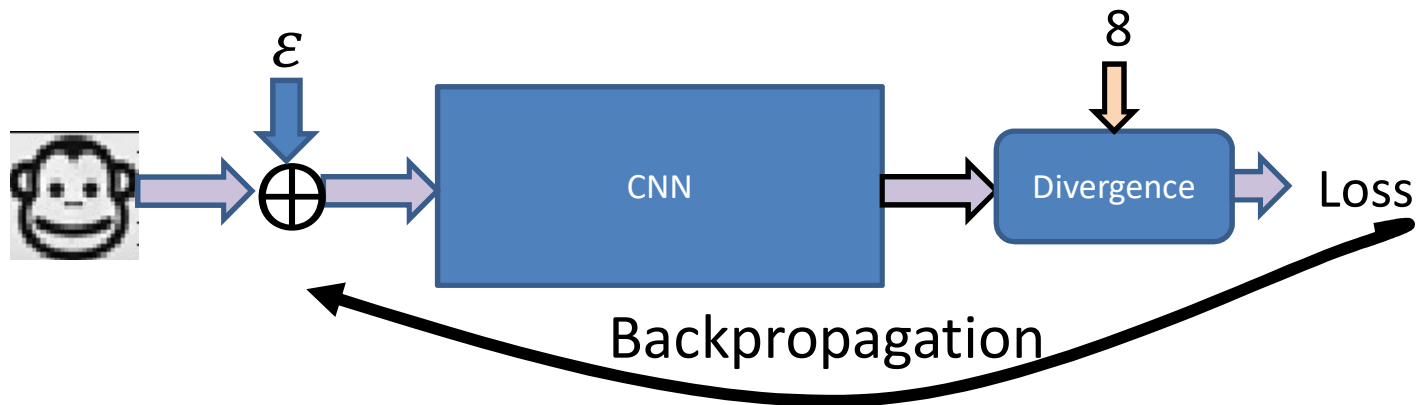
0 makes a monkey an 8



- Backpropagate the error all the way back to the input to modify the *input*

$$\operatorname{argmin}_x \operatorname{Div}(\operatorname{CNN}(x), 8)$$

0 makes a monkey an 8



- Backpropagate the error all the way back to the input to modify the *input*, but keep the corrections small

$$\operatorname{argmin}_{\epsilon} \operatorname{Div}(\operatorname{CNN}(x + \epsilon), 8) + \lambda \|\epsilon\|^2$$

0 makes a monkey 8

Boss, I made a
monkey 8!

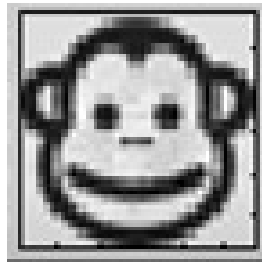


Neat! Perhaps
you can also
make it a 0?

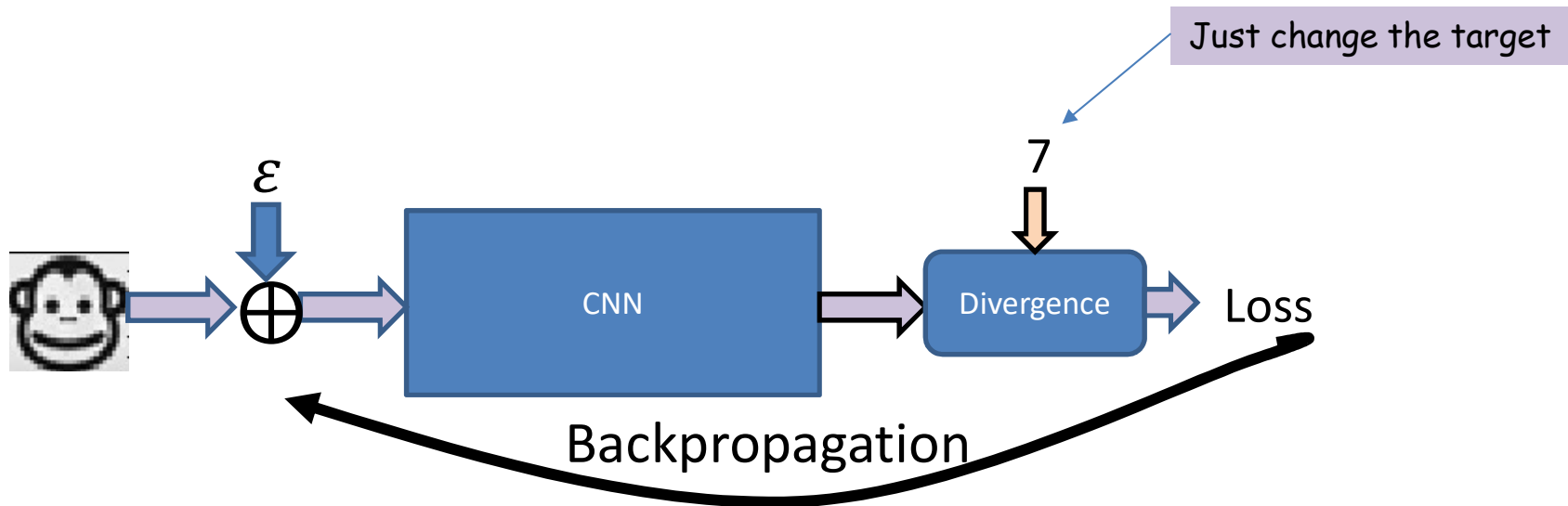


O can make a monkey anything

I can make it
anything!



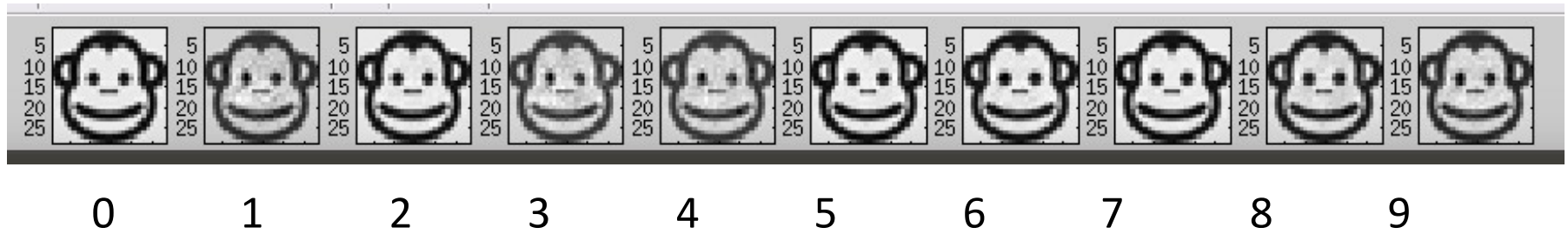
0 makes a monkey anything



- Backpropagate the error all the way back to the input to modify the *input*, but keep the corrections small

$$\operatorname{argmin}_{\epsilon} \operatorname{Div}(\operatorname{CNN}(x + \epsilon), 7) + \lambda \|\epsilon\|^2$$

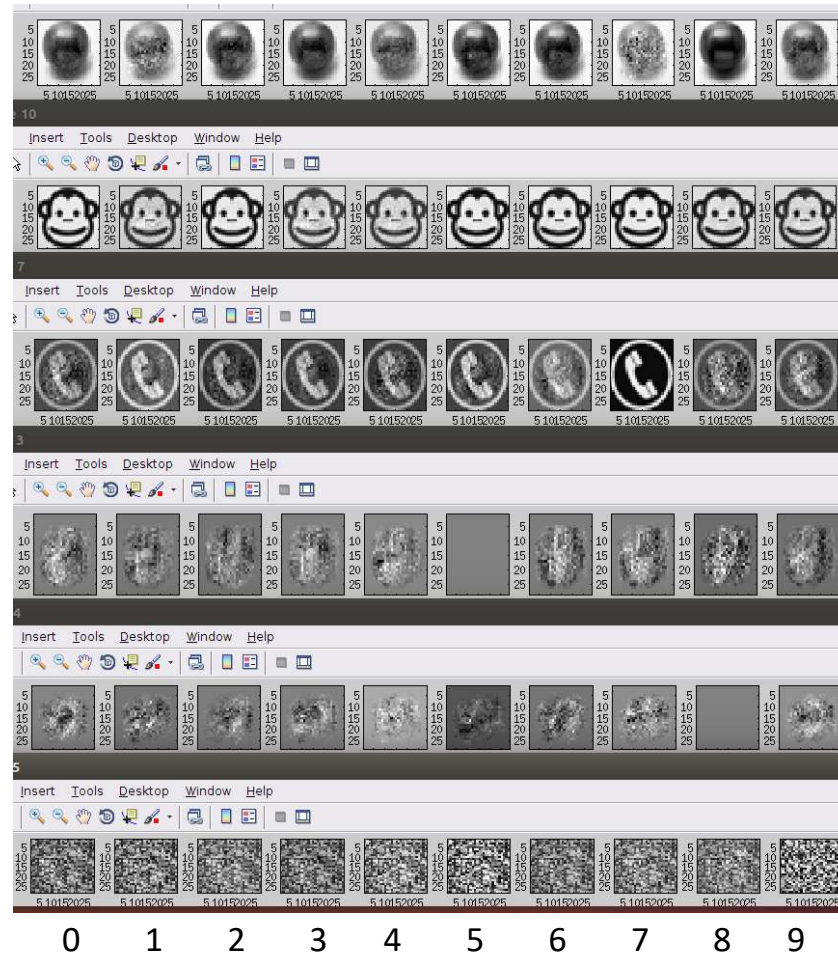
The monkey digits



- Monkey figures can be minimally edited to make O's MNIST CNN recognize them as any digit of our choice!

Other figures

- In fact, you can do this with *any* figure



Fooling a classifier

- Any input can be minimally perturbed to fool the classifier into classifying it as any class!
 - Perturbations can be so small as to be imperceptible to a human observer

The monkey distance



3

Unfortunately we were late to the party!



- Spammers had been fooling spam filters for *decades* already

The History of Email

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)

The History of Email Spam

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)
- The first email spam: 1 May 1978
 - By Digital Equipment Corporation
 - Although it wasn’t *called* “spam” until April 1993

DIGITAL WILL BE GIVING A PRODUCT PRESENTATION OF THE NEWEST MEMBERS OF THE DECSYSTEM-20 FAMILY; THE DECSYSTEM-2020, 2020T, 2060, AND 2060T. THE DECSYSTEM-20 FAMILY OF COMPUTERS HAS EVOLVED FROM THE TENEX OPERATING SYSTEM AND THE DECSYSTEM-10 <PDP-10> COMPUTER ARCHITECTURE. BOTH THE DECSYSTEM-2060T AND 2020T OFFER FULL ARPANET SUPPORT UNDER THE TOPS-20 OPERATING SYSTEM. THE DECSYSTEM-2060 IS AN UPWARD EXTENSION OF THE CURRENT DECSYSTEM 2040 AND 2050 FAMILY. THE DECSYSTEM-2020 IS A NEW LOW END MEMBER OF THE DECSYSTEM-20 FAMILY AND FULLY SOFTWARE COMPATIBLE WITH ALL OF THE OTHER DECSYSTEM-20 MODELS.

WE INVITE YOU TO COME SEE THE 2020 AND HEAR ABOUT THE DECSYSTEM-20 FAMILY AT THE TWO PRODUCT PRESENTATIONS WE WILL BE GIVING IN CALIFORNIA THIS MONTH. THE LOCATIONS WILL BE:

TUESDAY, MAY 9, 1978 – 2 PM
HYATT HOUSE (NEAR THE L.A. AIRPORT)
LOS ANGELES, CA

THURSDAY, MAY 11, 1978 – 2 PM
DUNFEY’S ROYAL COACH
SAN MATEO, CA
(4 MILES SOUTH OF S.F. AIRPORT AT BAYSHORE, RT 101 AND RT 92)

A 2020 WILL BE THERE FOR YOU TO VIEW. ALSO TERMINALS ON-LINE TO OTHER DECSYSTEM-20 SYSTEMS THROUGH THE ARPANET. IF YOU ARE UNABLE TO ATTEND, PLEASE FEEL FREE TO CONTACT THE NEAREST DEC OFFICE FOR MORE INFORMATION ABOUT THE EXCITING DECSYSTEM-20 FAMILY.

The History of Email Spam defences

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)
- The first email spam: 1 May 1978
 - By Digital Equipment Corporation
 - Although it wasn’t *called* “spam” until April 1993
- Earliest attempts at prevention of SPAM: 1996
 - “Mail Abuse Prevention System” (MAPS → SPAM in reverse), uses blacklists

The History of adversarial attacks on Email Spam defences

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)
- The first email spam: 1 May 1978
 - By Digital Equipment Corporation
 - Although it wasn’t *called* “spam” until April 1993
- Earliest attempts at prevention of SPAM: 1996
 - “Mail Abuse Prevention System” (MAPS → SPAM in reverse), uses blacklists
- Earliest address spoofing attack: 1997

And counter defences

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)
- The first email spam: 1 May 1978
 - By Digital Equipment Corporation
 - Although it wasn’t *called* “spam” until April 1993
- Earliest attempts at prevention of SPAM: 1996
 - “Mail Abuse Prevention System” (MAPS → SPAM in reverse), uses blacklists
- Earliest address spoofing attack: 1997
- Earliest ML based spam filter: 20 April 2001
 - Spam Assassin

And counter adversarial attacks

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)
- The first email spam: 1 May 1978
 - By Digital Equipment Corporation
 - Although it wasn’t *called* “spam” until April 1993
- Earliest attempts at prevention of SPAM: 1996
 - “Mail Abuse Prevention System” (MAPS → SPAM in reverse), uses blacklists
- Earliest address spoofing attack: 1997
- Earliest ML based spam filter: 20 April 2001
 - Spam Assassin
- Earliest adversarial attack on and ML Spam filter: 21 April 2001

Spam becomes a thing of the past

- The first “E-mail” :1965
 - MIT’s “Compatible Time-Sharing System” (CTSS)
- The first email spam: 1 May 1978
 - By Digital Equipment Corporation
 - Although it wasn’t *called* “spam” until April 1993
- Earliest attempts at prevention of SPAM: 1996
 - “Mail Abuse Prevention System” (MAPS → SPAM in reverse), uses blacklists
- Earliest address spoofing attack: 1997
- Earliest ML based spam filter: 20 April 2001
 - Spam Assassin
- Earliest adversarial attack on and ML Spam filter: 21 April 2001
- **Bill Gates declares Spam “soon to be a thing of the past” : January 2004**

Spam filtering in 2000

- Spam filters are mostly Naïve Bayes classifiers

$$\frac{\prod_i P(X_i|spam)}{\prod_i P(X_i|notspam)} > \theta? \quad \begin{array}{l} \text{Yes} \Rightarrow \text{Spam} \\ \text{No} \Rightarrow \text{not Spam} \end{array}$$

- X_i s are “features” derived from the message
 - Typically words or word patterns
 - E.g. CRM114

The “goodword” attack

Free Xanax, Low cost HGH

Sound is drop. Line whether soft oxygen. Cross burn make suggest, minute. Cover part reason. Why fresh wire. Notice, are fact find hold. Move such light city, feet. Near hot, pick other busy, book.

- Introducing a set of words and word patterns that are much more frequent in good email than spam will fool the naïve Bayes filter
- E.g. D. Lowd, C. Meek, *Good word attacks on statistical spam filters*, 2nd Conf. Email and Anti-Spam (CEAS), Mountain View, CA, USA, 2005
 - Formalized an already popular technique
 - Do not even need to know *which features* the classifier uses, though knowing helps

Naïve Bayes classifiers are linear classifiers

$$\frac{\prod_i P(X_i|spam)}{\prod_i P(X_i|notspam)} > \theta?$$

- Translates to

$$\sum_i (\log P(X_i|spam) - \log P(X_i|notspam)) - \log \theta > 0?$$

- Can be rewritten as

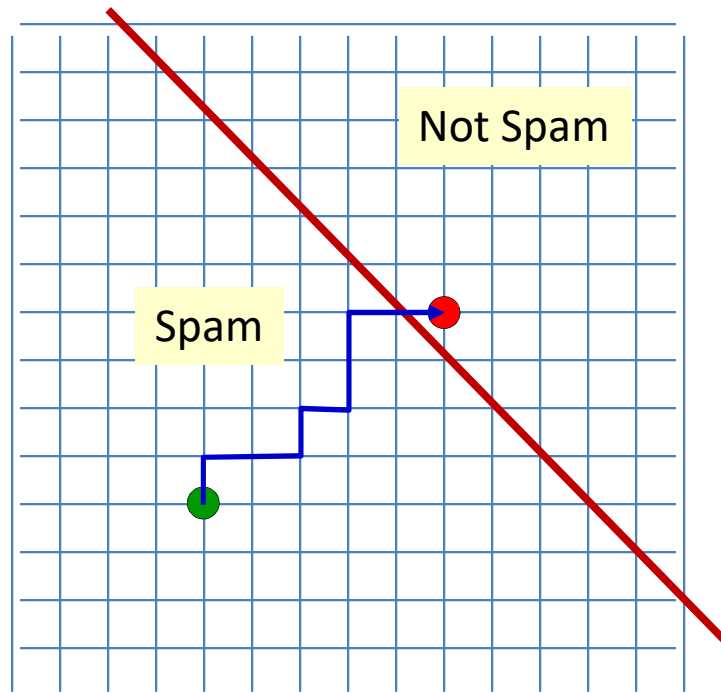
$$\sum_i f(X_i) - b > 0?$$

- Or more generally as

$$\sum_i \lambda_i f(X_i) - b > 0?$$

- Which actually translates to a Bayes classifier using *maximum entropy* distribution estimates
- Which is also a very popular Spam filtering mechanism

Beating linear classifier spam filters

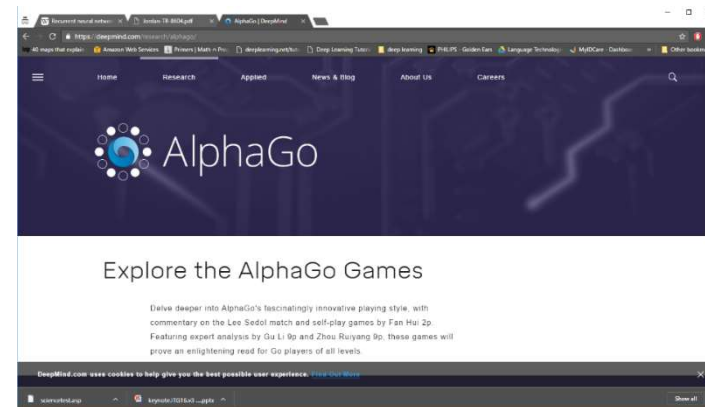
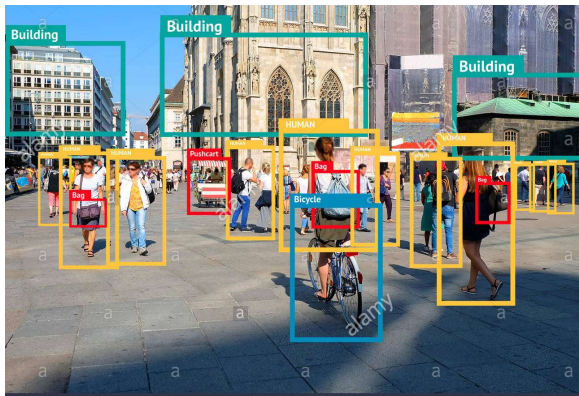
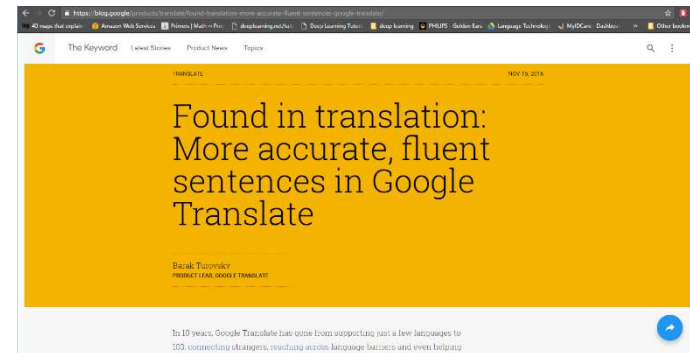


- N. Dalvi, P. Domingos, Mausam, S. Sanghai, D. Verma, *Adversarial classification*. International Conference on Knowledge Discovery and Data Mining, 2004
- Integer programming algorithm to determine minimal edits to “convert” a spam to not spam
 - Also works for other problems in other domains

2004-present: adversarial attacks on simple linear and non-linear classifiers

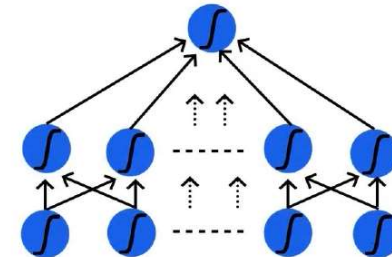
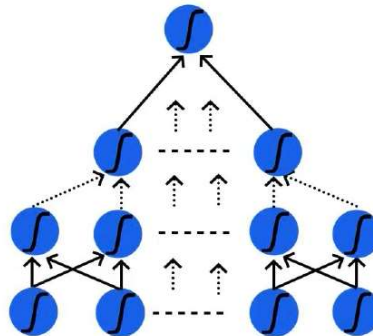
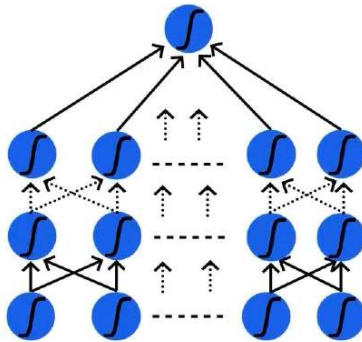
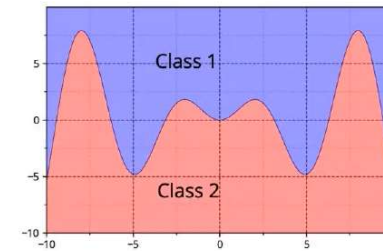
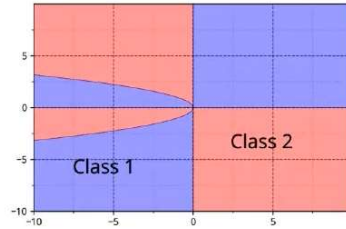
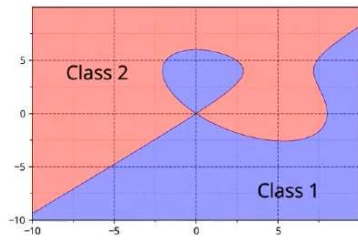
- On perceptrons
- On logistic regressions and softmax
- On SVMs
 - And SVMs with non-linear kernels
- Even on how to “poison” training data to make learned SVMs misbehave
 - And how to defend
- Biggio, Battista, and Fabio Roli. *Wild patterns: Ten years after the rise of adversarial machine learning*. Pattern Recognition 84 (2018)

2010s.. Neural networks rule..



- Neural network systems have established the state of the art in many many tasks...

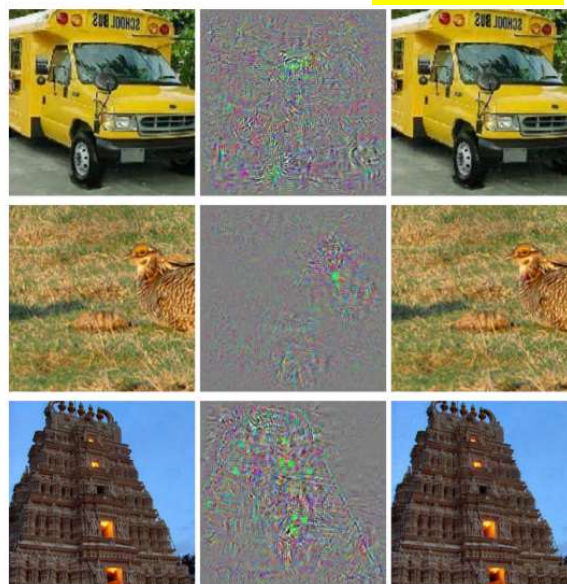
Nnets are universal approximators



- Can approximate anything!
- Surely they're more robust than simple naïve classifiers?

Szegedy et al. *Intriguing properties of neural networks.* ICLR 2014

Ostrich



- Adding often imperceptible noise to images can result in targeted misclassification
- Finding the noise that will cause images to be misclassified:

$$\hat{n} = \operatorname{argmin}_n \lambda |n| + L(f(x + n; \theta), y_{false})$$

- Subject to $(x + n) \in [0,1]^m$ (noisified images stay in valid range of pixel values)
- Basically “O”s method

Goodfellow 2014



“panda”
57.7% confidence

“gibbon”
99.3% confidence

$$\hat{x} = x + \delta \operatorname{sign}(\nabla_x L(f(x; \theta), y_{\text{true}}))$$

- Intentional modification
 - Modify only final bit of pixel values, to *maximize* the error between network output and true class
 - One-step process, without iteration
- Goodfellow, Shlens and Szegedy. *Explaining and harnessing adversarial examples*. *arXiv:1412.6572* (2014).

Many other attack methods

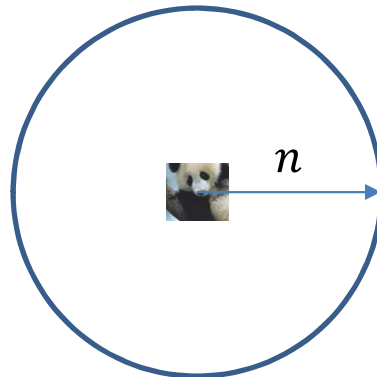
- Generally based on two approaches
- Norm minimization: Minimization of noise as a regularizer, for loss minimization

$$\hat{n} = \operatorname{argmin}_n \lambda |n|_p + L(f(x + n; \theta), y)$$

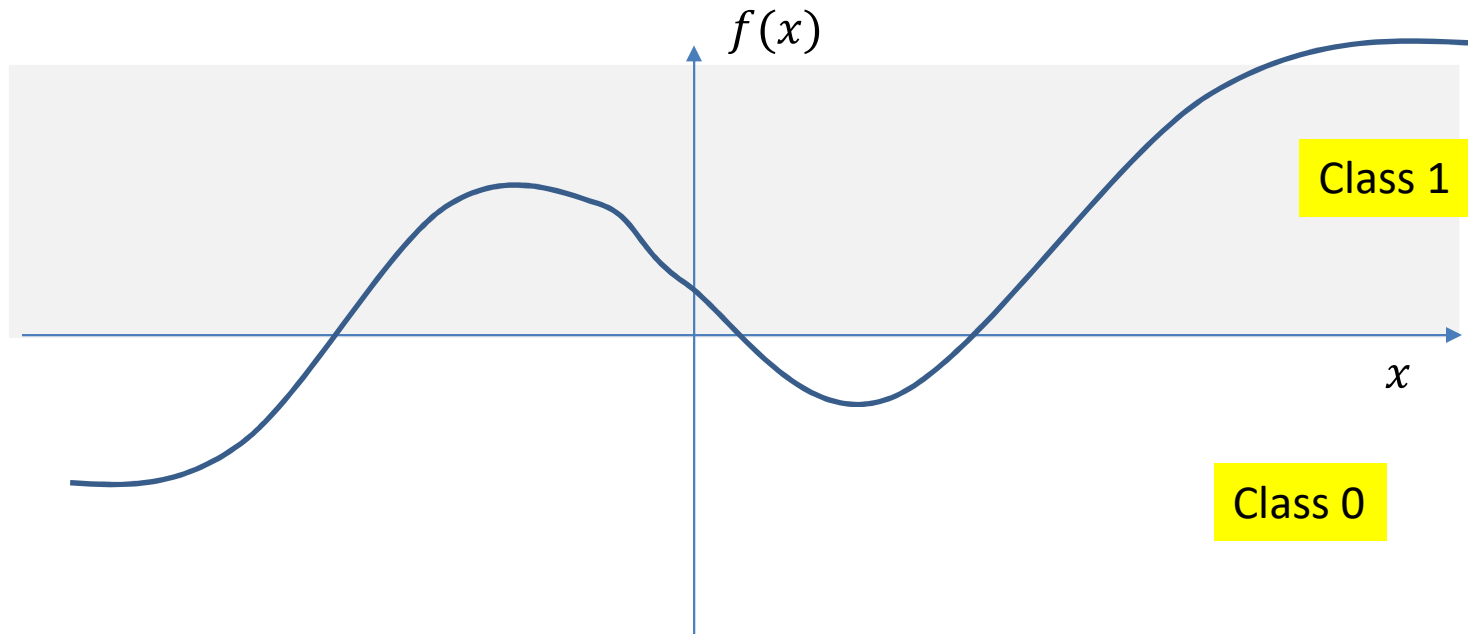
- Norm-constrained minimization: Impose hard constraints on noise while minimizing loss

$$\hat{n} = \operatorname{arg} \min_{n: |n|_p < \delta} L(f(x + n; \theta), y)$$

- In all cases, the modified data is encouraged or constrained to remain within a small “radius” of the original data, to maintain perceptual similarity

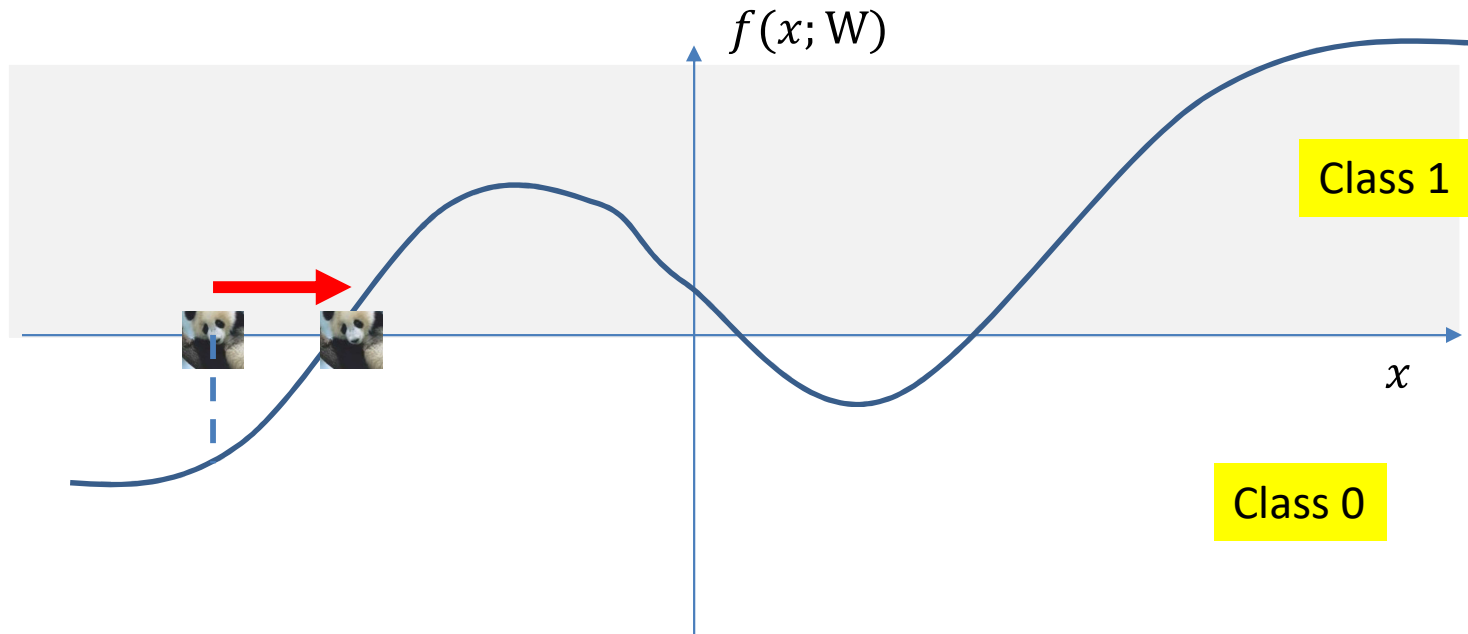


Deepfool



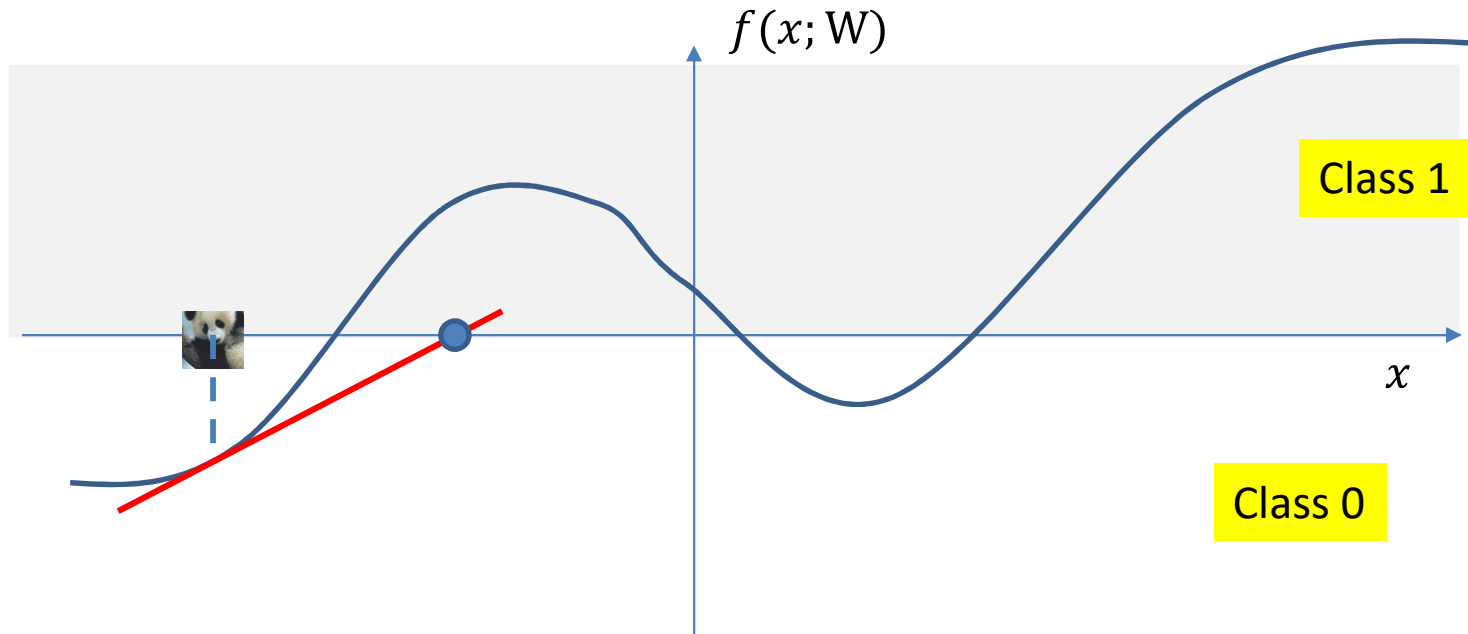
- The network is actually a discriminator $f(X)$
 - For binary classification, when $f(X) \geq 0$, the input X is classified as one class, when $f(X) < 0$ it is a different class
 - Multi-class classifiers can be viewed as a collection of such discriminators

Deepfool



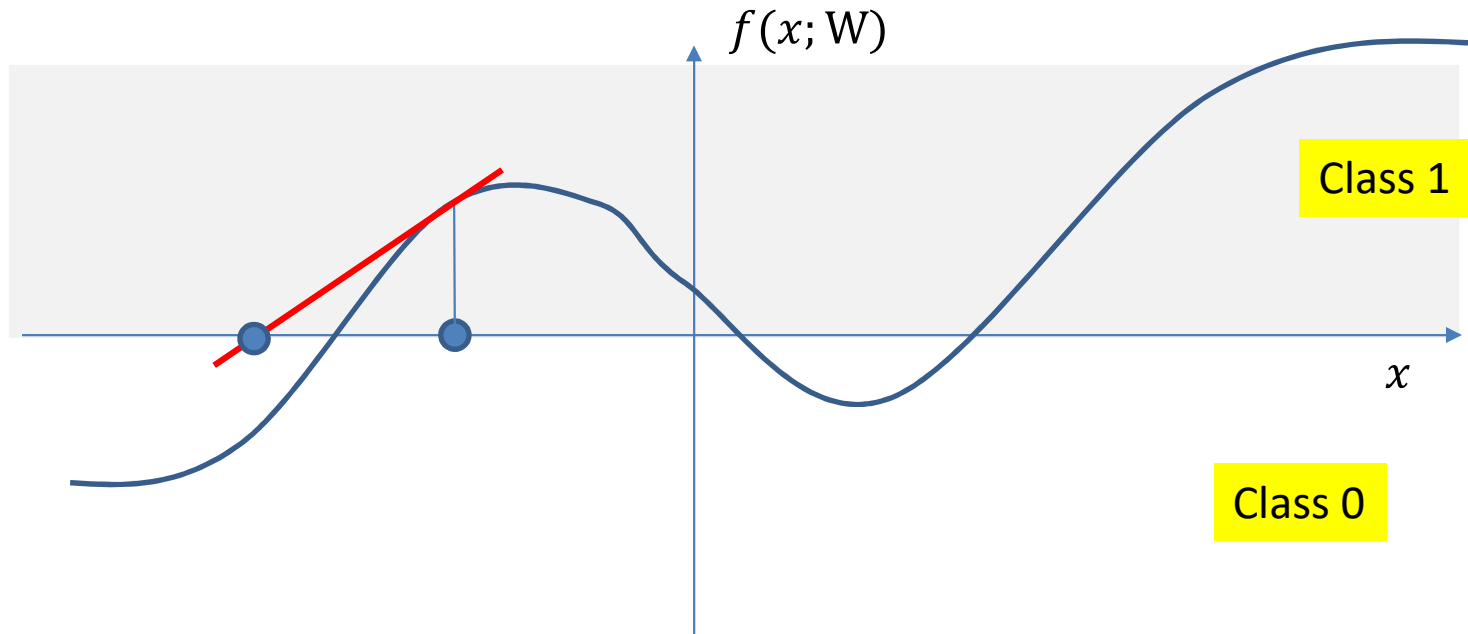
- The network is actually a discriminator $f(X)$
 - For binary classification, when $f(X) \geq 0$, the input X is classified as one class, when $f(X) < 0$ it is a different class
 - Multi-class classifiers can be viewed as a collection of such discriminators
- To change the classification output for an input, shift it by the minimum amount so that $f(X)$ changes sign

Deepfool



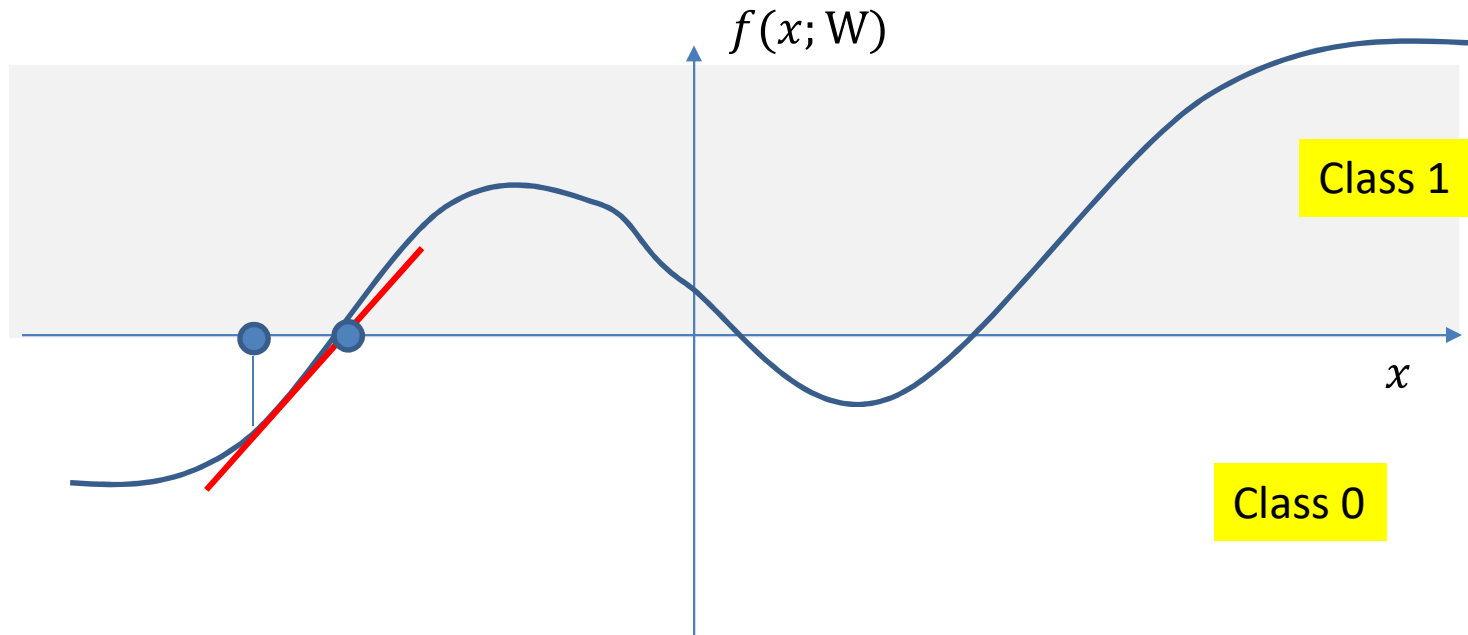
- Iteratively linearize the function and find location of 0
 - Until a location where the actual $f(X) = 0$ is found

Deepfool



- Iteratively linearize the function and find location of 0
 - Until a location where the actual $f(X) = 0$ is found

Deepfool




- Iteratively linearize the function and find location of 0
 - Until a location where the actual $f(X) = 0$ is found

Techniques are increasingly sophisticated

- And increasingly efficient!
- And versatile

You can simply have an instance misclassified...

Aim: Modify cat image so that its *not* classified as cat

$$\hat{n} = \arg \max_{n: |n|_p < \delta} L(f(x + n; \theta), y_{true})$$


- To just misclassify an input, find noise to *maximize* the error between the network output and the *true* label

Or even choose what it is misclassified as

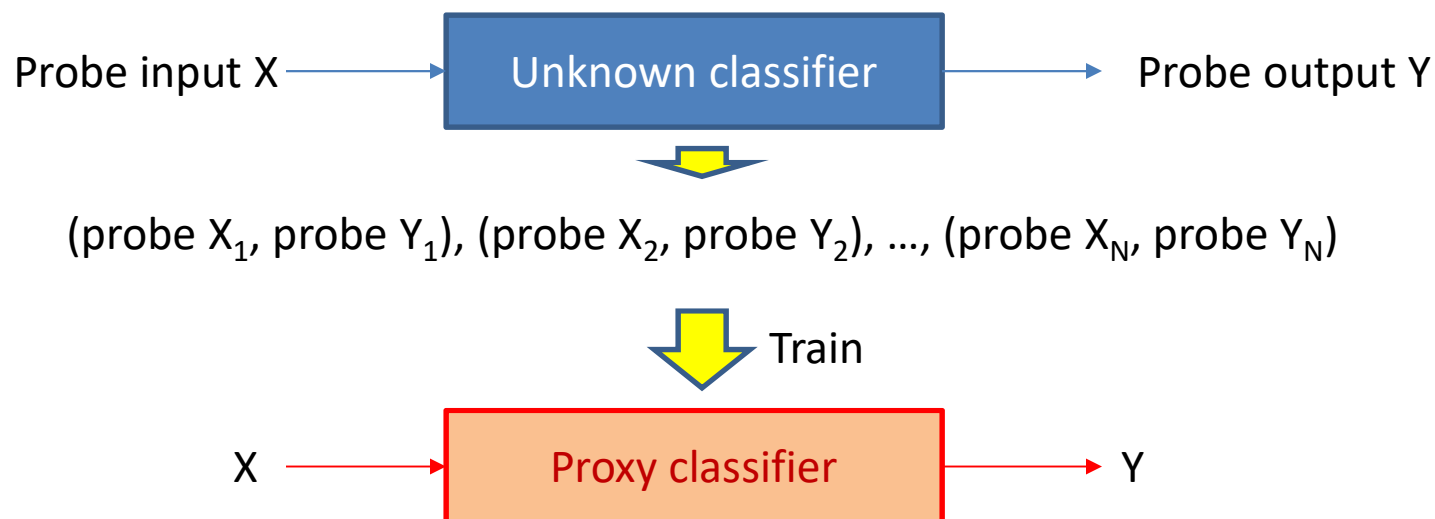
Aim: Specifically modify cat image so that it is classified as a bottle

$$\hat{n} = \arg \min_{n: |n|_p < \delta} L(f(x + n; \theta), \text{bottle})$$



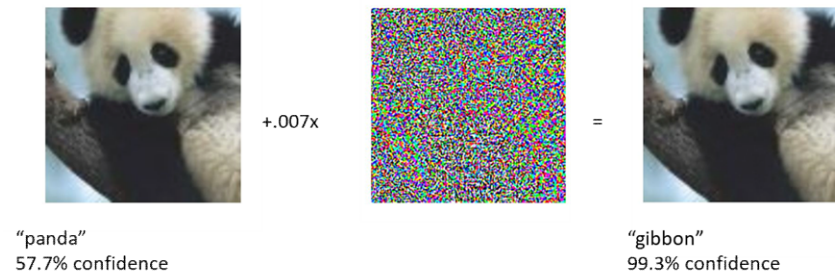
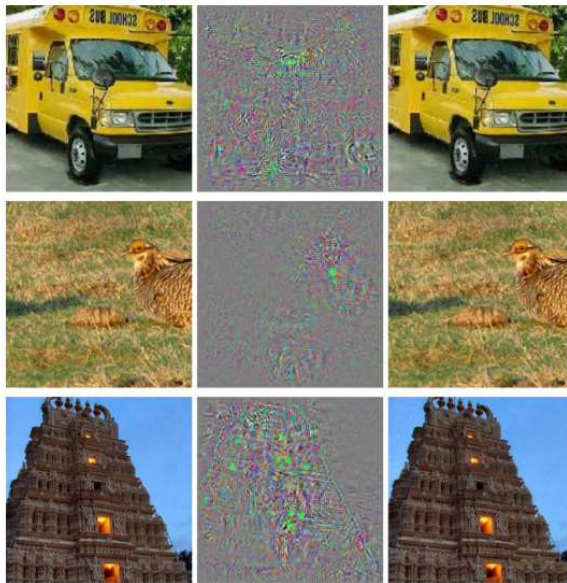
- Find noise to *minimize* the error between the network output and the *desired bogus label*

You don't even need to know the classifier



- Just probe the unknown classifier to obtain input-output pairs
- Train a proxy classifier with the probe data
- Use the proxy classifier to build your adversarial inputs
 - They will transfer to the original classifier!

But these are only artificial, right?



- Synthetic examples, where you add noise to pre-recorded images
 - Using significant computation in each case
- Doesn't carry over to real-life where you will generally not have the ability to carefully manipulate an image with iterative algorithms

Right!



Sharif, Mahmood, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter.
"Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition."
ACM SIGSAC Conference on Computer and Communications Security, 2016.

Right!



Eykholt, Kevin, et al. "Robust physical-world attacks on deep learning visual classification." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

These only work on images though..



- Are attacks limited to images?
 - Attacking simple binary or multi-class classifiers
- Will it work on harder tasks like speech recognition
 - With effectively infinite classes...

Fooling a speech recognizer



- Example from Nicolas Carlini

Fooling a speech recognizer



[Reveal Transcription] “okay google browse to evil dot com”



[Reveal Transcription] “without the dataset the article is useless”



[Reveal Transcription] [original, no speech is recognized]



[Reveal Transcription] “okay google browse to evil dot com”

- Example from Nicolas Carlini

But this requires the loss to be differentiable

$$\hat{n} = \operatorname{argmin}_n \lambda |n|_p + L(f(x + n; \theta), y)$$

- Wont work if we're trying to introduce non-differentiable errors, right?
 - Like ASR or MT errors
 - Or image segmentation errors..

The *Proxy* loss

$$\bar{l}(\hat{x}, y) = P_{\gamma \sim N(0,1)} [f(\hat{x}, y; \theta) - f(\hat{x}, \hat{y}; \theta) < \gamma] L(y, \hat{y})$$

- $f(x, y; \theta)$ is the score assigned to class y by the network
 - \hat{y} is the highest scoring class
- Finds the expected error between actual output and target output
 - This turns out to be a differentiable function of n
- Houdini: Fooling Deep Structured Prediction Models, Cisse, Adi, Neverova, Keshet, 2017

Image segmentation example



True segmentation



(Cisse, Adi, Neverova, & Keshet, 2017)

Image segmentation example



Borrow segmentation from this image

(Cisse, Adi, Neverova, & Keshet, 2017)

Image segmentation example



Adversarially modified image

(Cisse, Adi, Neverova, & Keshet, 2017)

Image segmentation example



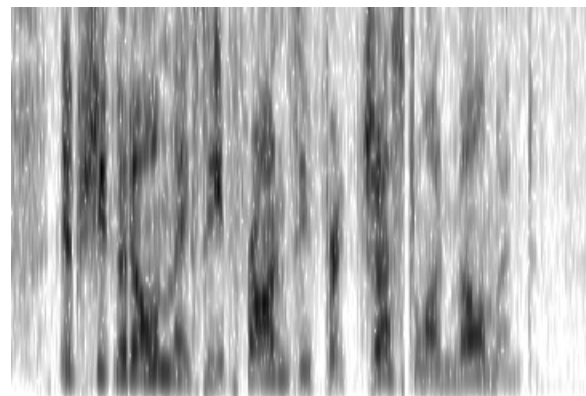
...and its rubbish segmentation

(Cisse, Adi, Neverova, & Keshet, 2017)

Speech recognition (Google Voice)



Original:
if she could only see Phronsie for just one
moment



Adversarial:
if she ou down take shee throws purhdress luon ellwon

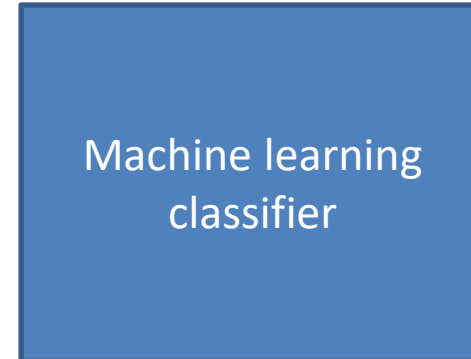
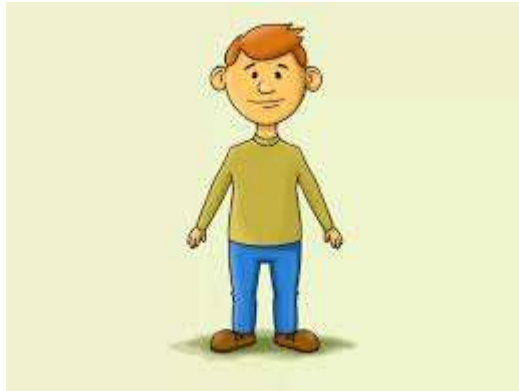


(Cisse, Adi, Neverova, & Keshet, 2017)

So why are classifiers so fragile

- Perceptual reasoning
- Statistical reasoning

Perceptual reasoning



- We're actually working with *two* classifiers
 - Human perception (typically)
 - The ML classifier
- We want to modify the data such that the two classify the data differently
 - Malicious modification

Human perception is very forgiving

On átkíns or the soùth beach diet, try our diet páтч. A new cutting edge, advanced áppétite sùpprèssant, mètabólism bôôster, and ènérgy ènháncer...all in one. The perfect supplement to ássist you in lôsiñg those extrá pôúñds just in time for sùmmèr

Lèarñ the trúth about losing wèight.

All ordérs backéd by our nó risk, monéy back Guarántèe!

Shìpped Discrèetly.

[Why wait, the solution is now](#)

No further €máils plèáse
<http://thesedealzwontlast.com>

- *We want* to find patterns

Human perception is very forgiving



(a)



(b)



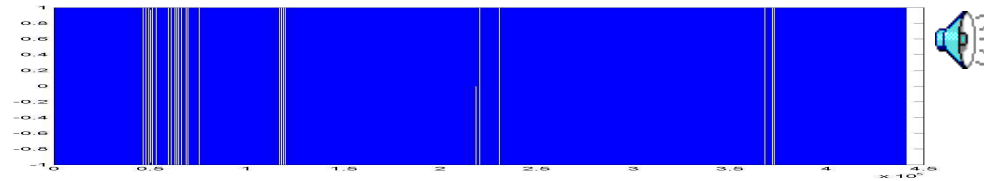
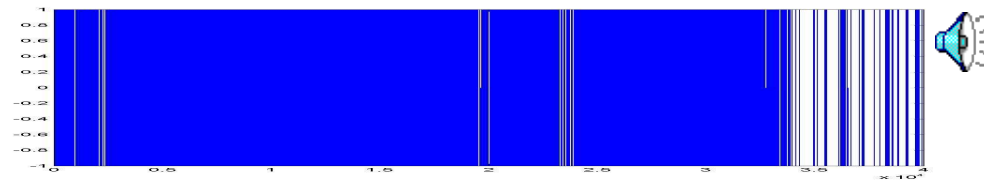
(c)



(d)

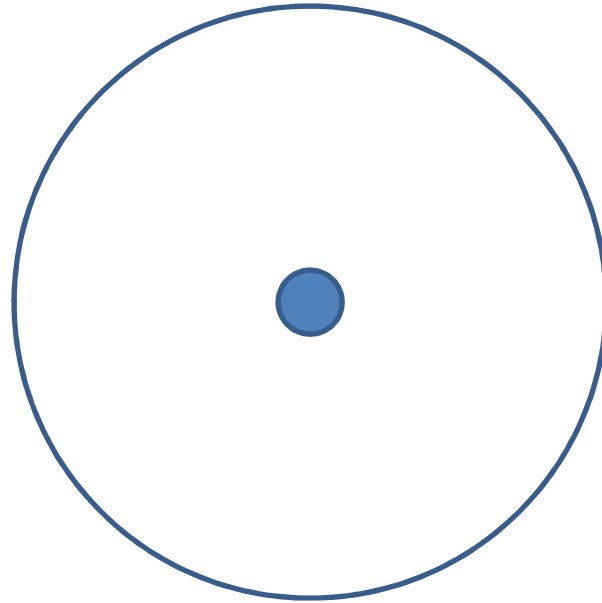
- *We want* to find patterns

Human perception is very forgiving



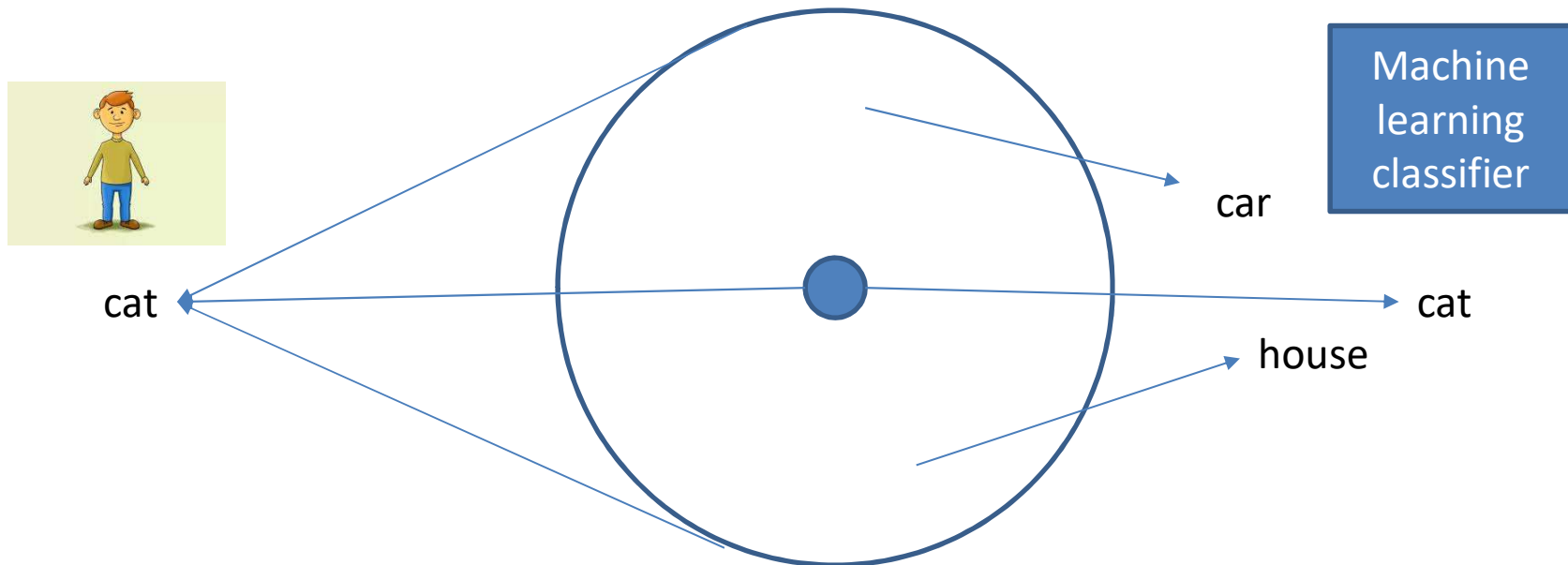
- *We want* to find patterns
 - Tom Sullivan and Schubert, for the curious

The perceptual radius



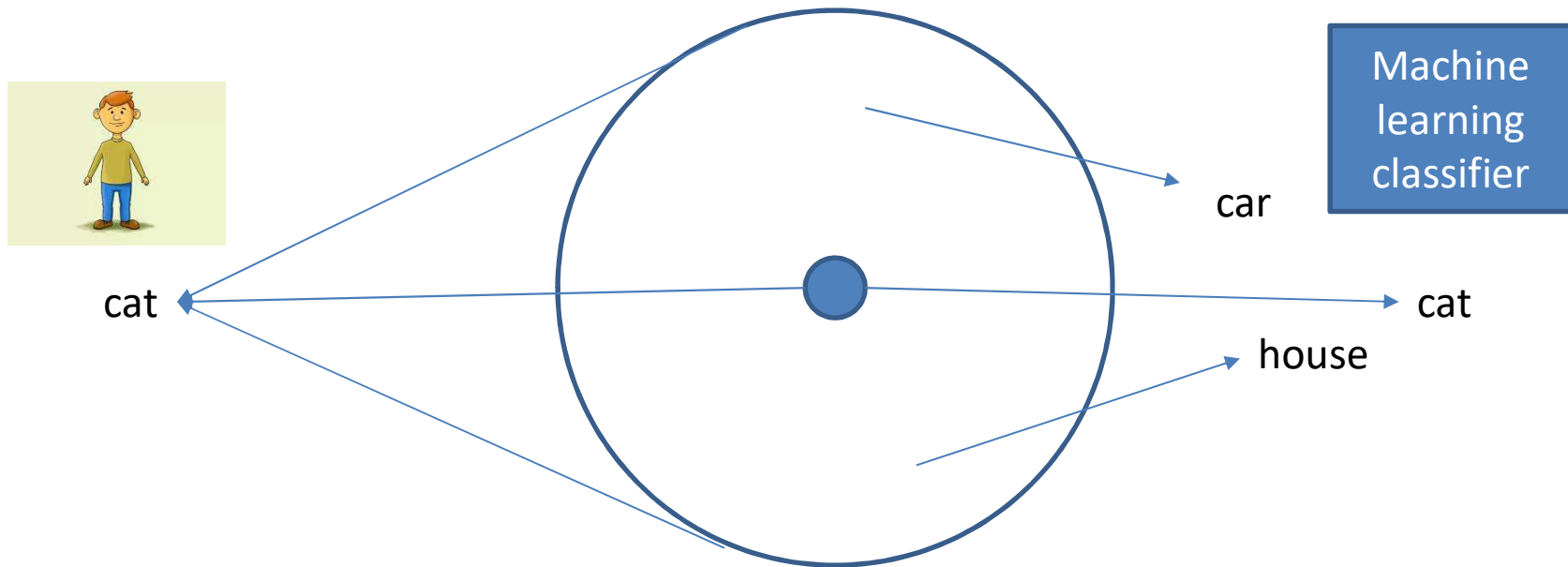
- There is a “ball” of modifications around any valid pattern that we are tolerant to
 - ML algorithms, on the other hand, are sensitive these variations

The perceptual rationale



- Adversarial attacks search for points within this ball for which the ML algorithm responds differently than we do
 - Since we don't really know the perceptual ball, they model it instead as a physical ball of small radius
 - E.g. $x + n$, $\|n\|_p < \epsilon$
 - If the model physical ball lies within the perceptual ball, the found solutions will be valid adversarial instances

The perceptual rationale



- Machine learning algorithms that are provided training samples, only learn the function at the sample, but not the entire perceptual circle around it
 - Which cannot even be characterized in most cases

Statistical reasoning

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Target function:
 $Y = X_1 \text{ XOR } X_2$

- Consider an ML algorithm that has been provided this training data
 - Trivial to learn
 - Simple XOR

Spurious inputs

X_1	X_2	X_3	Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	0
1	1	1	0

Target function:
 $Y = X_1 \text{ XOR } X_2$

- Now the algorithm has been provided this new table instead
 - The target function is still $X_1 \text{ XOR } X_2$
 - X_3 is a *spurious input*

What will the algorithm learn

X_1	X_2	X_3	Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	0
1	1	1	0

X_1	X_2	X_3	Y
0	1	1	0
1	0	1	0

X_1	X_2	X_3	Y			X_1	X_2	X_3	Y			X_1	X_2	X_3	Y
0	1	1	0			0	1	1	1			0	1	1	1
1	0	1	1			1	0	1	0			1	0	1	1

- The algorithm can learn any of these patterns for the unseen input combinations
 - Only one is right for our target function
 - If it learns any of the others, the output for any combination of X_1 and X_2 can be made erroneous by choosing the right X_3

What will the algorithm learn

Each additional spurious bit of input adds an exponential number of ways for the algorithm to learn the wrong thing

This makes it foolable

X_1	X_2	X_3	Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	0
1	1	1	0

X_1	X_2	X_3	Y
0	1	1	0
1	0	1	0

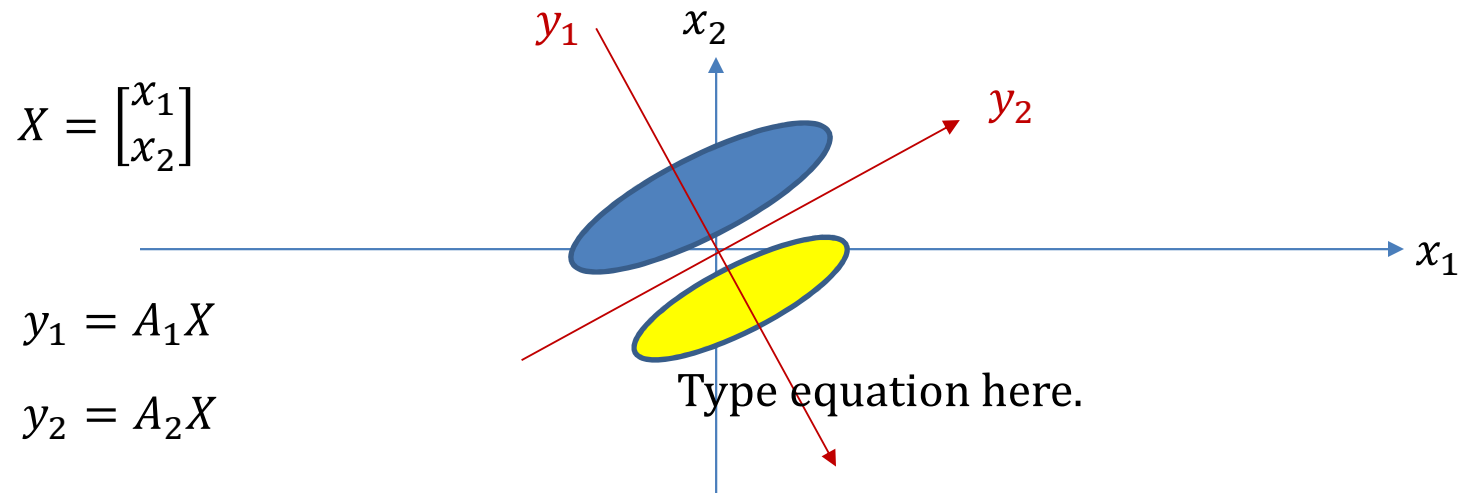
X_1	X_2	X_3	Y			X_1	X_2	X_3	Y			X_1	X_2	X_3	Y
0	1	1	0			0	1	1	1			0	1	1	1
1	0	1	1			1	0	1	0			1	0	1	1

- The algorithm can learn any of these patterns for the unseen input combinations
 - Only one is right for our target function
 - If it learns any of the others, the output for any combination of X_1 and X_2 can be made erroneous by choosing the right X_3

Sufficient statistic

- A sufficient statistic is the *minimal* function of the input that is sufficient to compute the output
- For the previous example (x_1, x_2) is a sufficient statistic
- (x_1, x_2, x_3) is *not* a sufficient statistic
 - It is overspecified

Sufficient statistic: Linear example

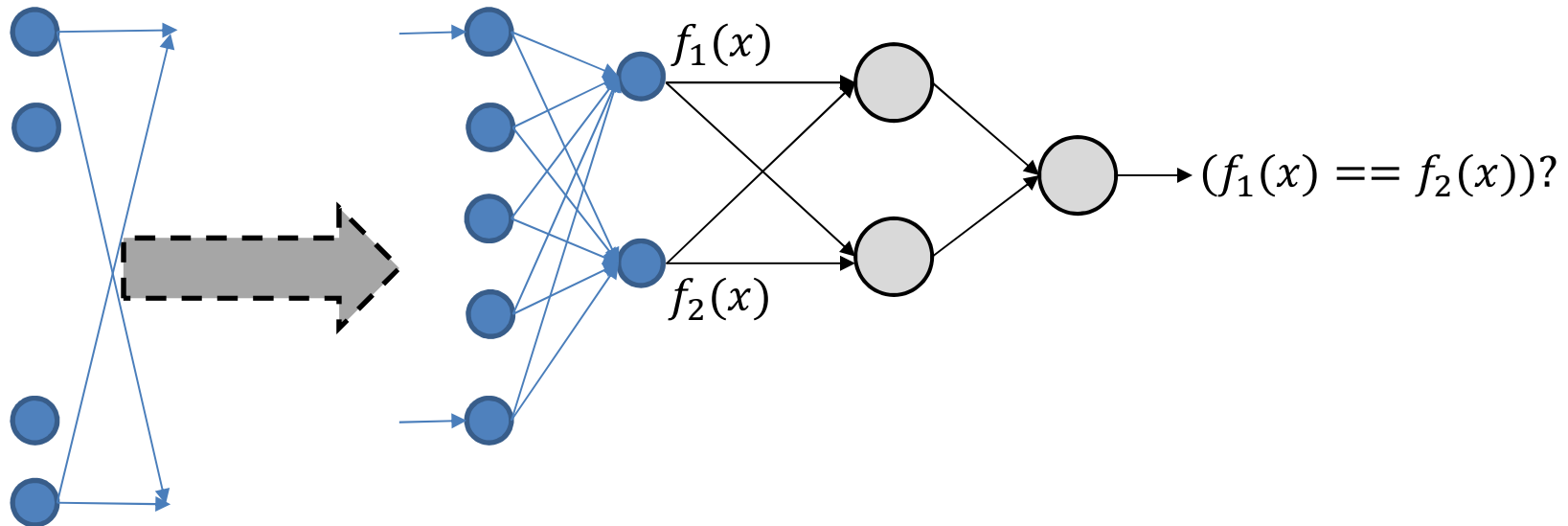


- Binary classification problem
 - Blue class vs. yellow class
- $y_1 = A_1 X$ is a sufficient statistic
 - (y_1, y_2) is not a sufficient statistic
 - (x_1, x_2) is not a sufficient statistic

Sufficient statistic

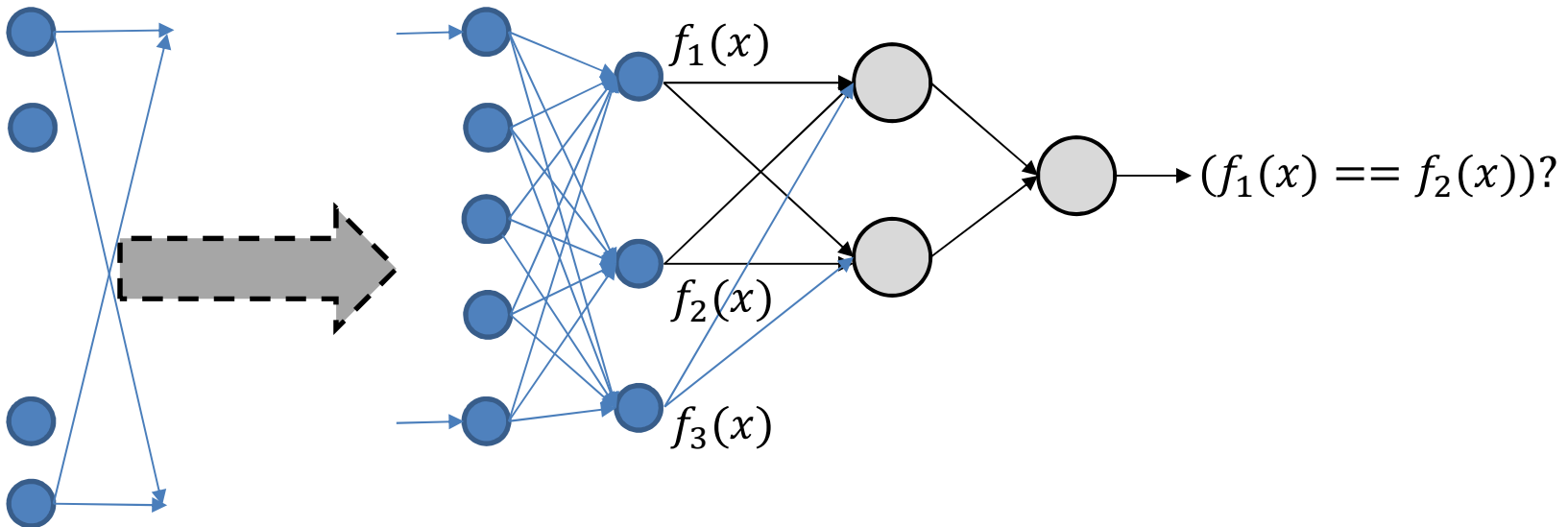
- Any classifier that operates on a non-sufficient statistic of the input is exponentially hard to learn and can be fooled by adversarial examples
- The input to any linear classifier that can be fooled by adversarial examples is *not* a sufficient statistic
 - B. Li, G. Friedland, J. Wang, R. Jia, C. Spanos, D. Song: "One Bit Matters: Understanding Adversarial Examples as the Abuse of Data Redundancies", submitted to NIPS 2018
- Summary: If you provide redundant input to the classifier, it can be fooled by an adversarial example

The susceptibility of networks



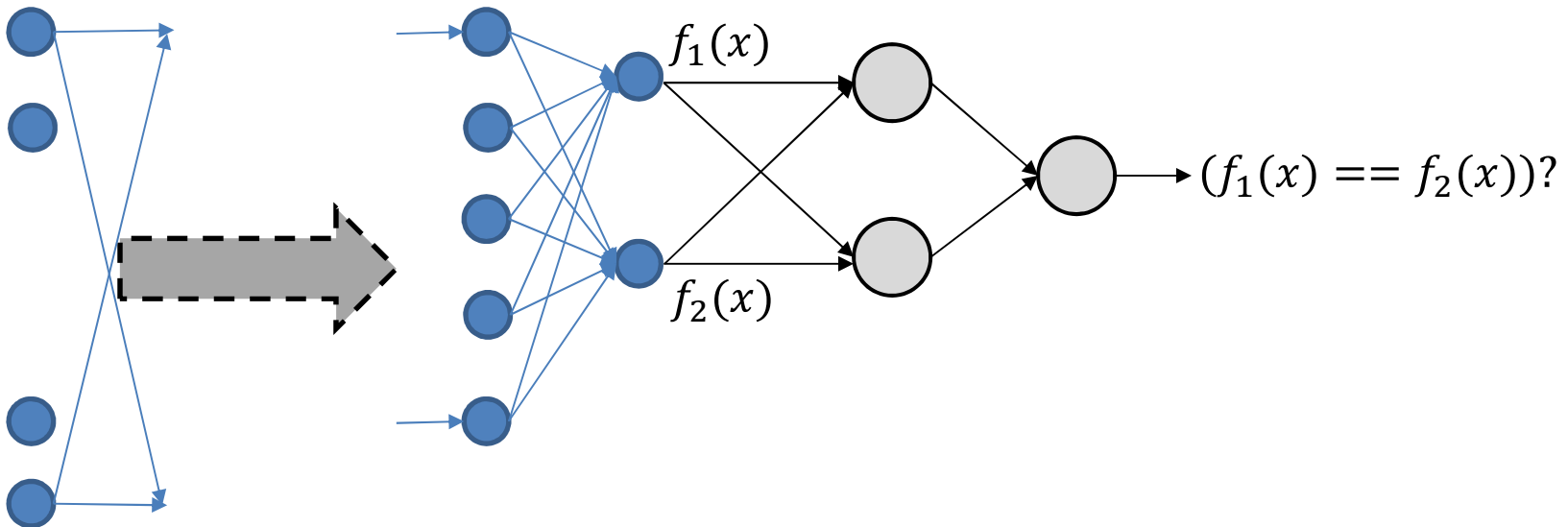
- Consider the example of $y = (f_1(x) == f_2(x))?$
 - $f_1(x)$ and $f_1(x)$ are two outputs at kth layer

The susceptibility of networks



- Consider the example of $y = (f_1(x) == f_2(x))$?
 - $f_1(x)$ and $f_1(x)$ are two outputs at k th layer
- If the network produces *three* features at the k th layer, this opens up the possibility of adversarial attack

Susceptibility of networks



- Adversarial attacks can only be prevented by having a “perfect” network
 - At least one layer that produces exactly sufficient statistic
- It is impossible to know what the minimal network architecture is for any given problem
- Any practical solutions will *always* be exploitable
 - By a more motivated attacker

Defences

- So how does one defend?

Adding adversarial samples to training

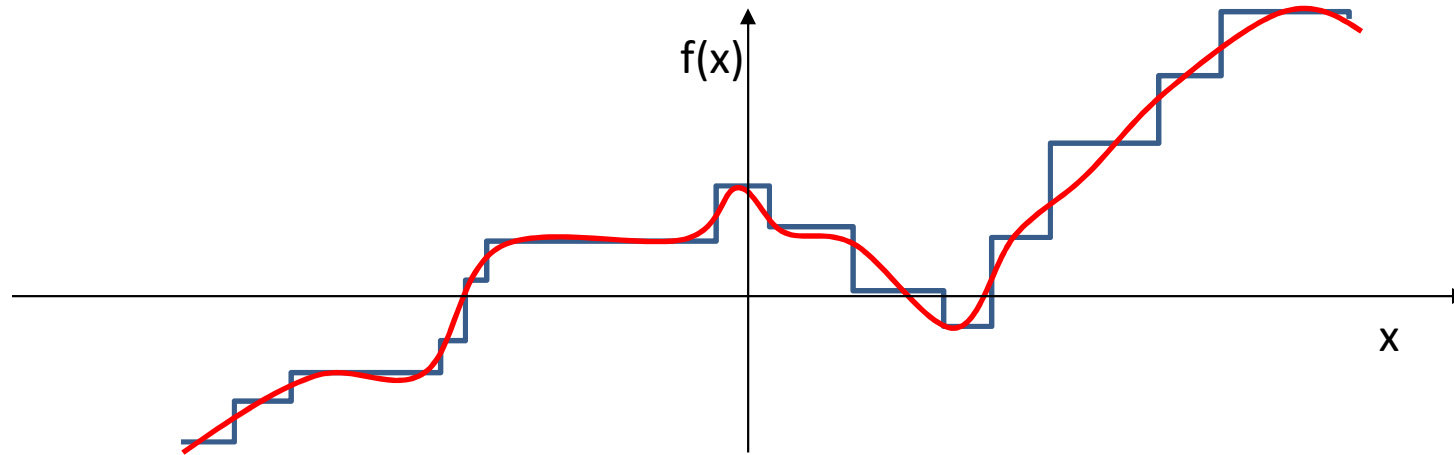
- Explicitly train against adversarial instances
- While training the network
- Iteratively:
 - Generate several adversarial instances
 - Instances that are misclassified by the classifier
 - E.g. Cats that are classified as tables
 - Add them (with correct labels) to training set and retrain
 - Szegedy et al., 2014, Goodfellow et al., 2014
 - Improve
- **Network remains exploitable..**

Making the function non-differentiable

$$\hat{n} = \operatorname{argmin}_n \lambda |n|_p + L(f(x + n; \theta), y)$$

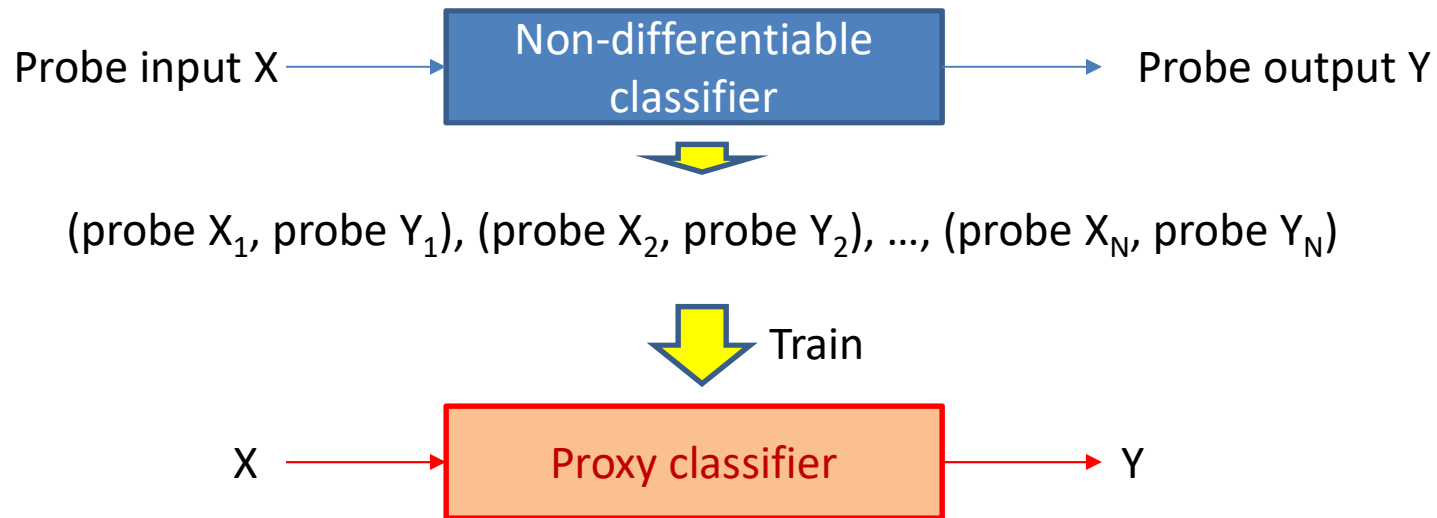
- Usual techniques for producing adversarial samples use gradients of $f(x + n; \theta)$
 - This is the network
- The network must be differentiable w.r.t x

Making the function non-differentiable



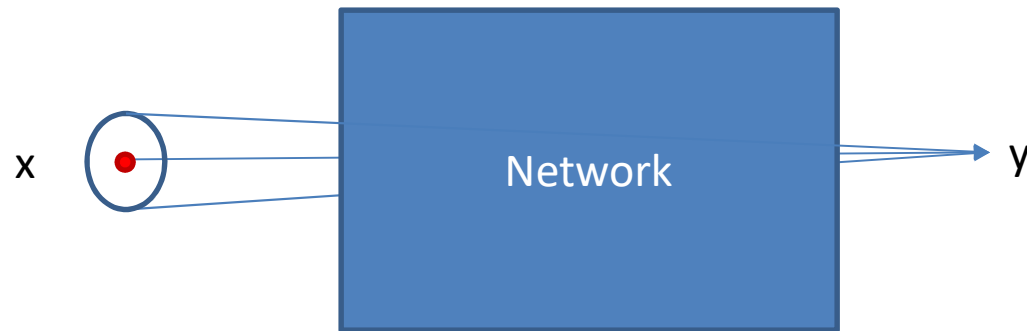
- Make $f(x + n; \theta)$ non-differentiable
 - A variety of ways
 - Quantize input
 - Randomize computations
 - Quantize activations
 - Etc.

Non-differentiable classifiers remain exploitable



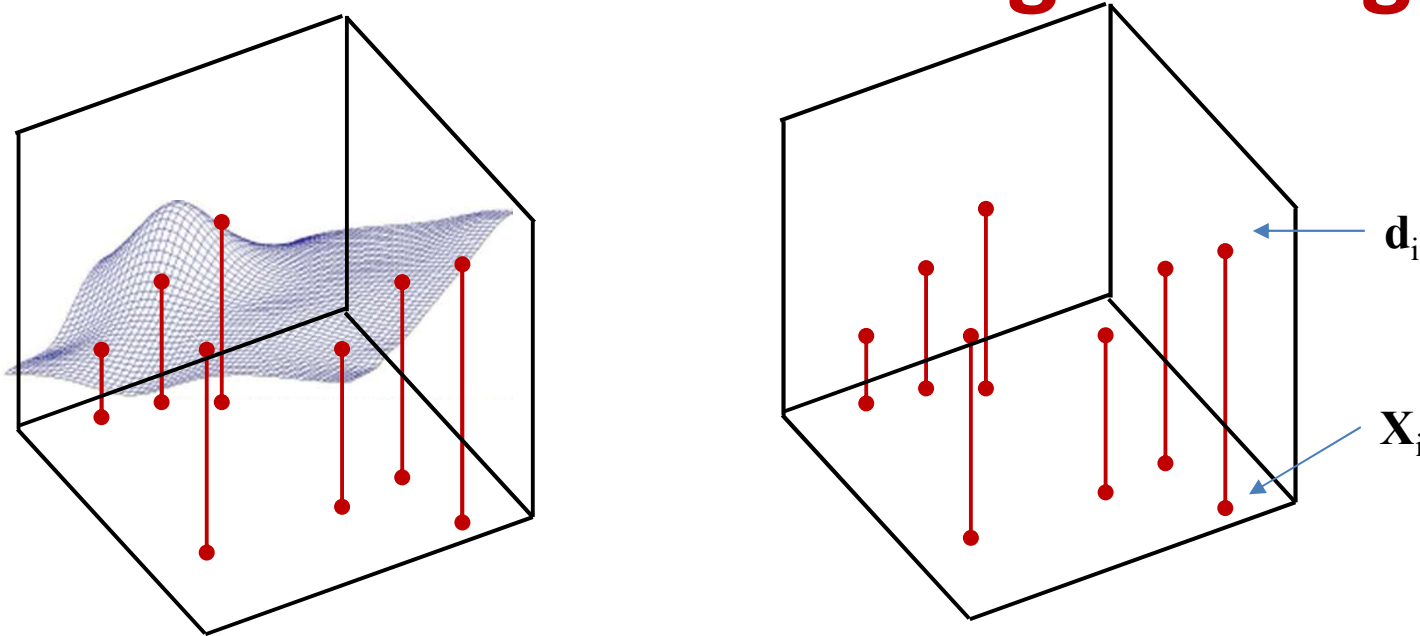
- Build differentiable proxy classifiers and fool them
 - The adversarial samples are transferrable

Making it robust to (perceptually) acceptable variations



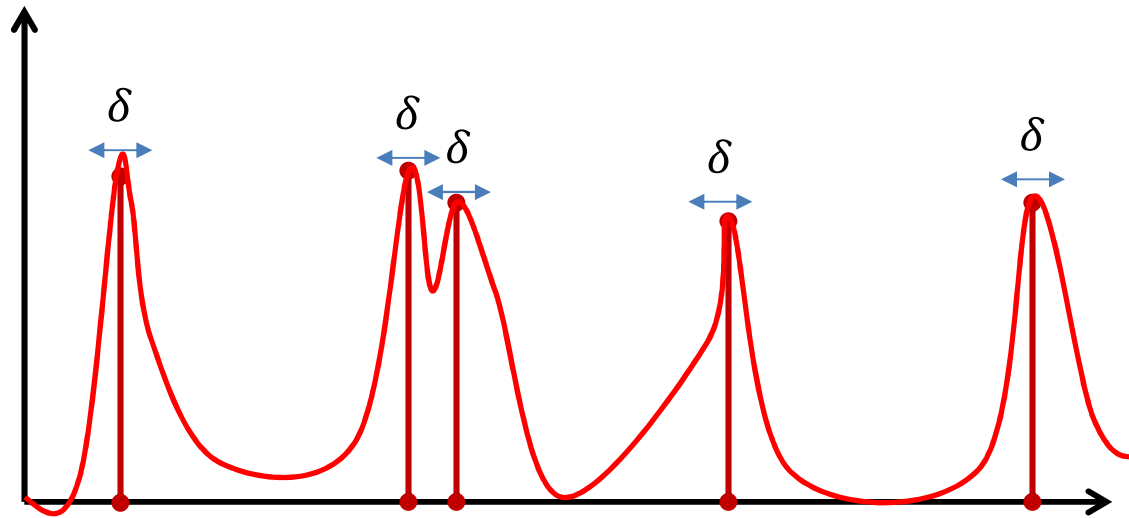
- Most successful approach to date: Train model to not change output within perceptual radius of each training input
 - Actual perceptual radius unknown; use metric balls instead
 - Wang and Kolter, 2018

Standard Machine Learning Paradigm



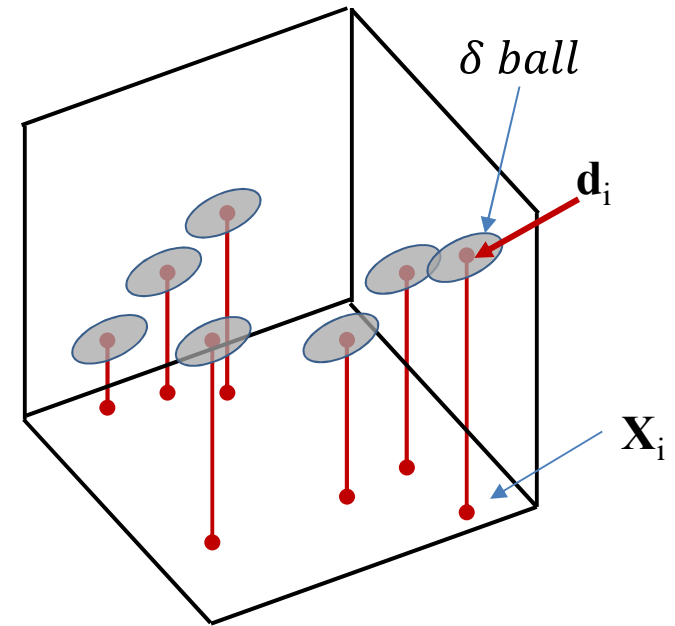
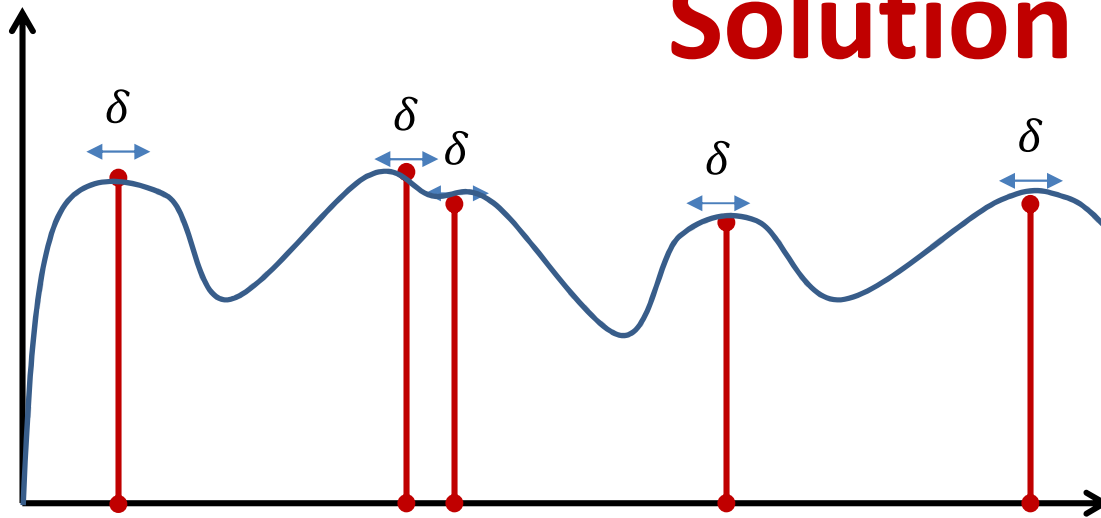
- Objective: Train a mapping from input to output
 - But given only input output pairs
- Solution: Learn the function such that the mapping is correctly learned for the specific input-output pairs provided

Standard Machine Learning Paradigm



- What really happens: No guarantee what the function learns even δ away from the training samples
- More generally, the output of the network can change very sharply within a δ region of any valid instance
- Adversarial instances exploit these δ regions
 - They are obtained by modifying valid instances by small amounts

Solution

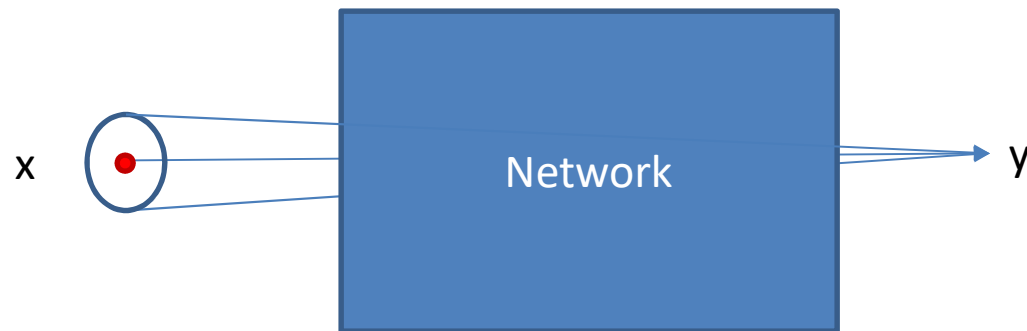


- Learn network such that
 - It outputs correct y value at each training x
 - It outputs a value close to y , in a δ ball around x

$$\hat{\theta} = \operatorname{argmin}_{\theta} \mathbb{E}_x \left[\max_{\|n\|_p < \delta} L(f(x + n; \theta), y) \right]$$

- Minimize the worst loss within the delta ball
 - Towards deep learning models resistant to adversarial attacks, Madry et al., 2017
 - Provable defenses against adversarial examples via the convex outer adversarial polytope, Wang and Kolter 2018

Making it robust to (perceptually) acceptable variations → Testing for adversariality



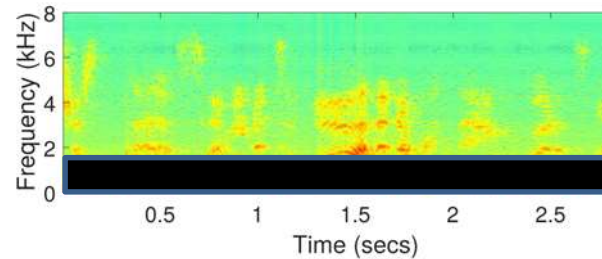
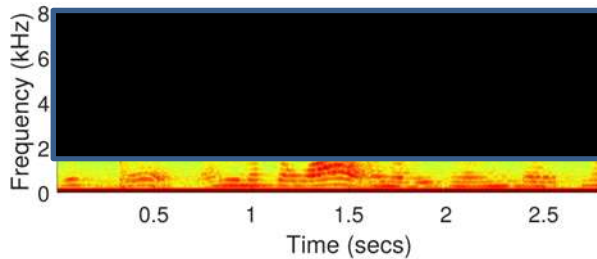
- The procedure can also be used to *verify* test instances
 - Does the rest of the ball produce the same output as the instance itself?
 - Wang and Kolter, 2018

What are we missing?

- The techniques still approximate the perceptual ball with the metric ball
- Fight fire with fire: Use actual perceptual metrics
 - Use known perceptual properties for the data to build defences
 - Particularly effective for speech

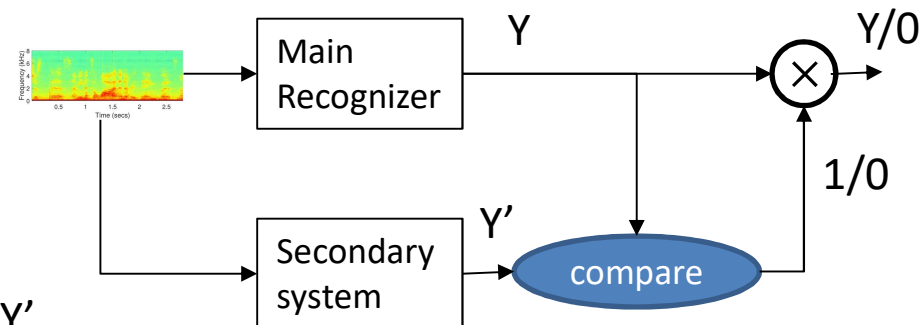
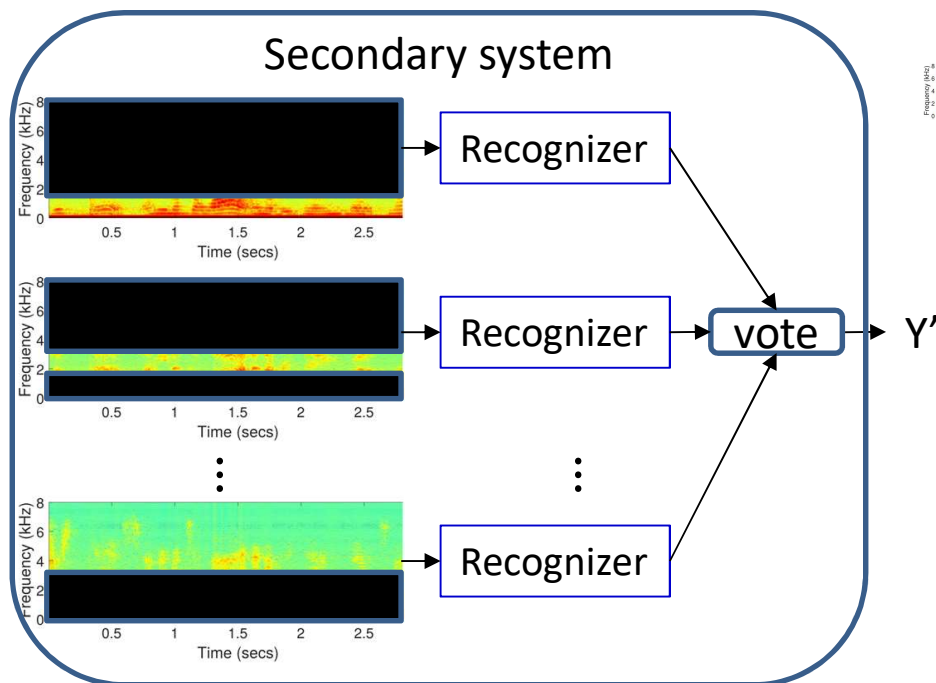
Detecting Adversariality: Spectral band redundancy

Both equally intelligible
(but not as intelligible as full-band speech)



- Fletcher's experiment: Speech that has been high-pass filtered 1800Hz is as intelligible as speech that is low-pass filtered at 1800Hz
 - And both are intelligible
 - Speech is highly redundant
- Exploit spectral redundancy to combat adversarial noise
 - Adversarial noise will affect some spectral bands more than others

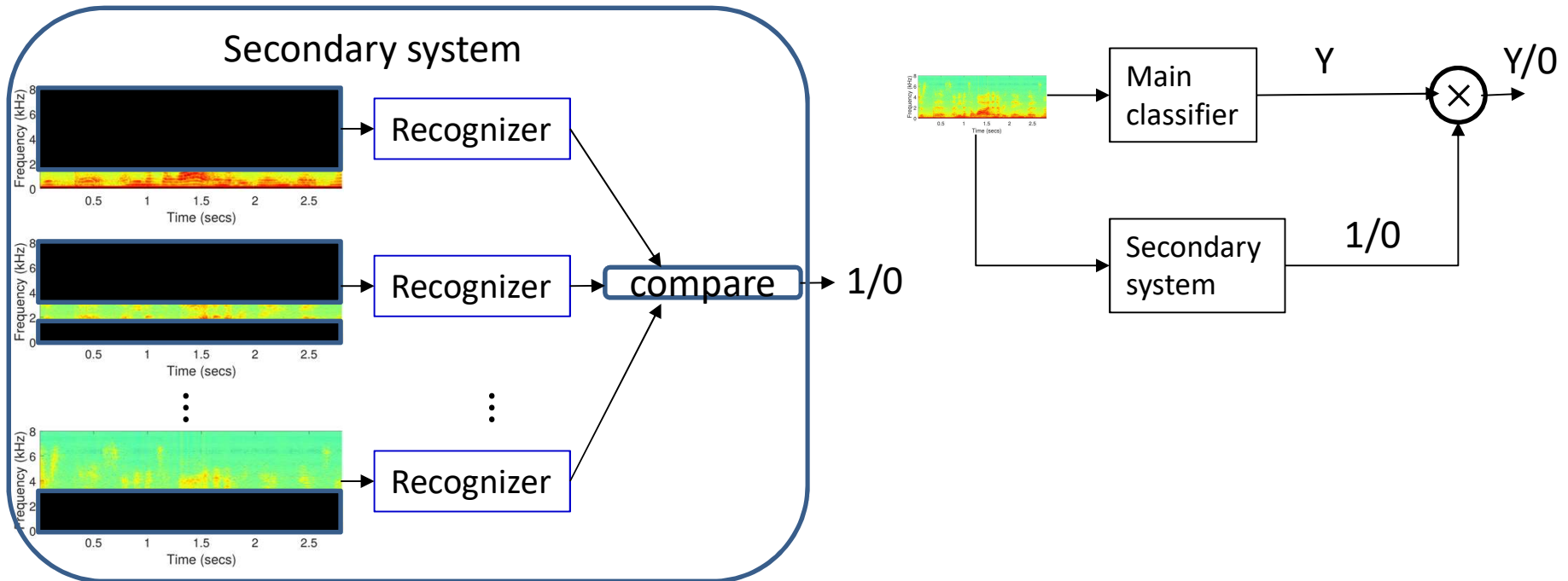
Detecting Adversariality: *Spectral band redundancy*



Y and Y' are recognized text

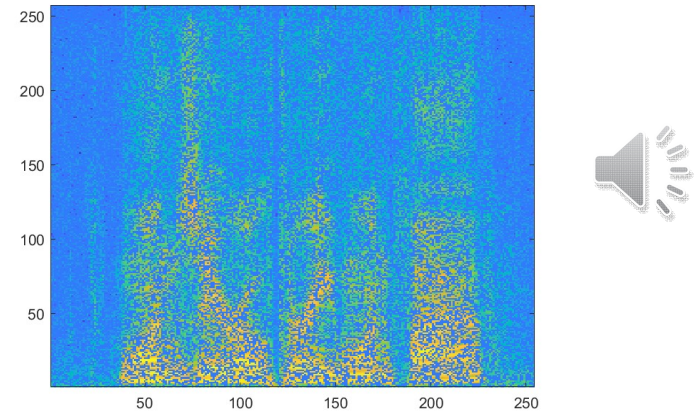
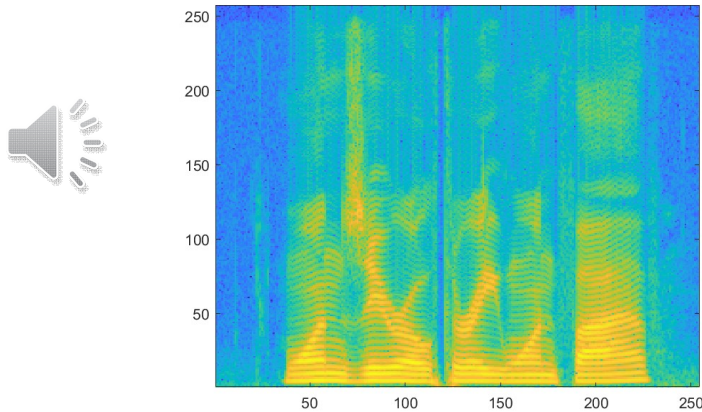
- Secondary verifier strategy
 - Filter signals into multiple bands
 - Recognize bands individually
 - Vote
- If secondary recognizer output does not match primary recognizer output, input is potentially adversarially modified

Detecting Adversariality: *Spectral band redundancy*



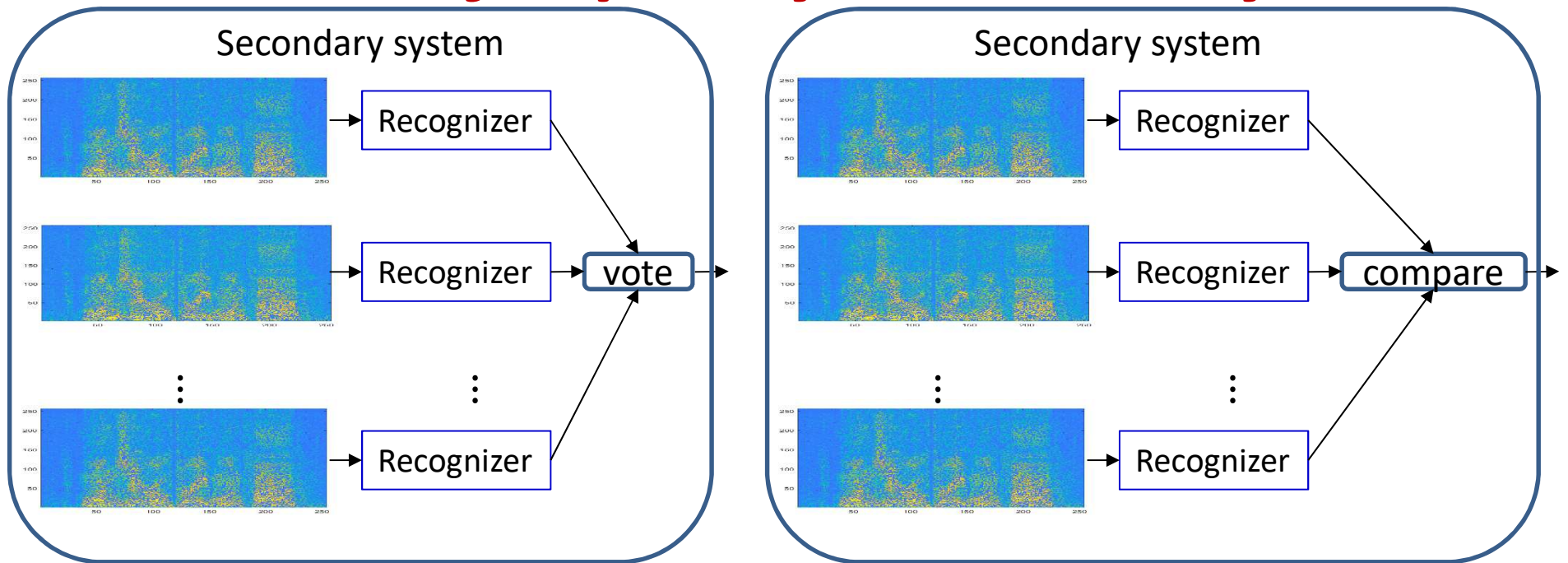
- If primary task is not speech recognition
 - E.g. speaker verification, health condition test, etc.
- Secondary system are still *recognizers*
 - Outputs are compared to one another to determine if input is adversarial
 - Adversarial inputs will increase variety and diversity of sub-band recognition outputs

Detecting Adversariality: *time-frequency redundancy*



- Speech remains highly intelligible after random time-frequency components of the audio are masked out
 - “Erase” up to 80% of randomly selected TF elements
- Erasure will likely eliminate adversarial noise

Detecting Adversariality: *time-frequency redundancy*



- Generate multiple random-masked versions of input
- Recognize all of them
- Secondary system can be used in the same manner as for the spectral-band redundancy method

The bad news

- None of the defences discussed so far are panaceas
 - Often don't work
- Attacks, meanwhile, get increasingly sophisticated
 - And increasingly pose real dangers

Looking ahead

- Will remain an area of research for the immediate future
 - Adversaries and defences constantly catching up to each other
- What I have not covered:
 - Poisoning the training set
 - Backdoor attacks
 - Exploiting adversarial samples
 - For watermarking, etc.

The Abrupt Stop

