Neural Networks, Cost Functions, and What Happens When You Ignore Math

John McKay, PhD

Applied Research Laboratory

Pennsylvania State University

March 28, 2018

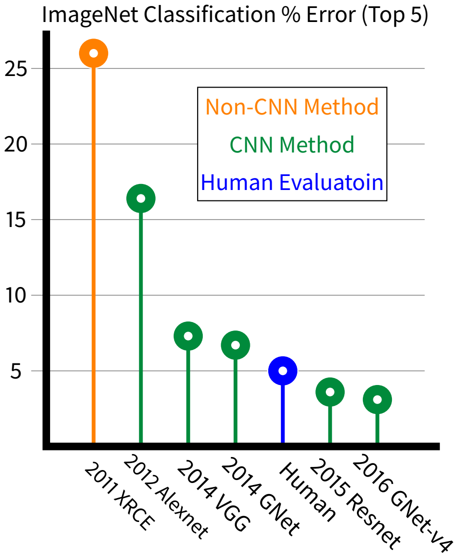
Pitt

Table of Contents

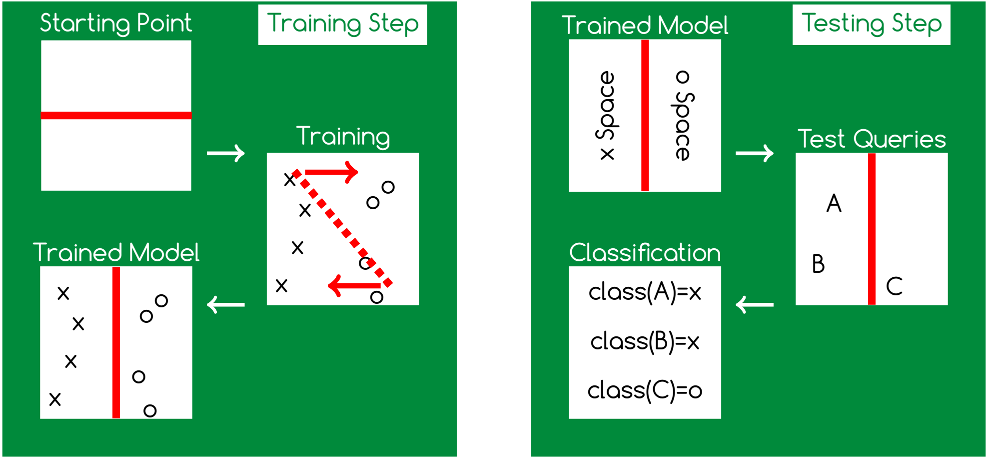
Outline

The Modern Convolutional Neural Network

- Convolutional neural networks (CNNs) are the state-of-the-art in image classification (and text classification and image super-resolution and object detection and...)
- In computer vision circles, seemingly every problem has seen a significant jump in CNN usage - to good effect.
- Much of that success has been had by researchers willing to bypass known mathematical warnings. This has largely played out well for CS/EE types.
- This talk is going to focus on a problem inherent to CNNs that relates to a lack of mathematical rigor.

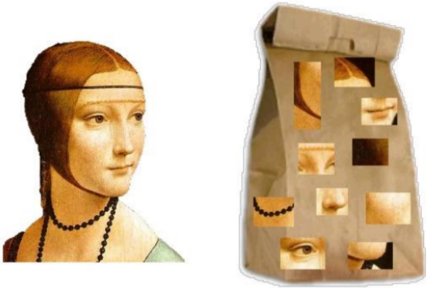


- ML is a field of algorithm development wherein data is used to tune parameters/weights towards some task (like classification). We are going to discuss **supervised** ML, meaning the data is labeled.



Key to Machine Learning: Extracted Features

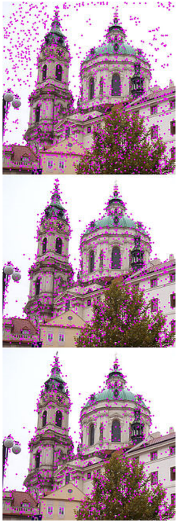
- Traditionally, **features** are extracted from data samples to focus the training/testing of the machine learning model.



MIT

Key to Machine Learning: Extracted Features

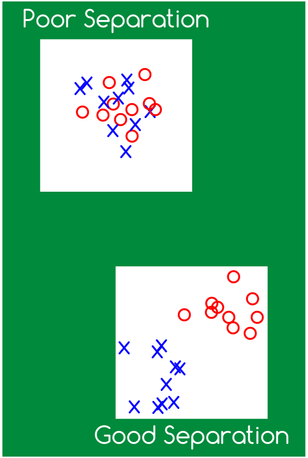
- Traditionally, **features** are extracted from data samples to focus the training/testing of the machine learning model.
- How these features are designed and the attributes they capture is of great interest; the more discriminatory they are, the easier a classifier will train and the better the algorithm will do.



Wikipedia

Key to Machine Learning: Extracted Features

- Traditionally, **features** are extracted from data samples to focus the training/testing of the machine learning model.
- How these features are designed and the attributes they capture is of great interest; the more discriminatory they are, the easier a classifier will train and the better the algorithm will do.

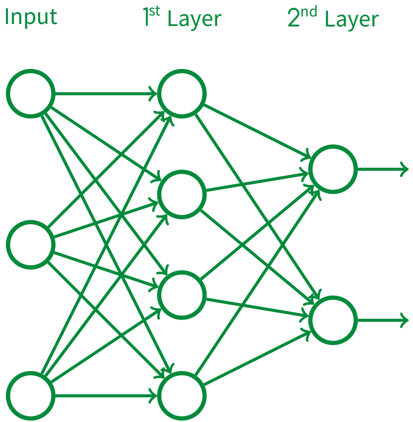


Outline

Neural Networks

- Neural Networks are nested affine transformations.
- Suppose we have a network with $L = 2$ layers and an (vectorized image) input $\boldsymbol{x}$:

$$\boldsymbol{a}^{(2)} = \overbrace{f_2(\quad \underbrace{f_1(\boldsymbol{x})}_{\text{First (Hidden) Layer}} \quad)}^{\text{Second (Output) Layer}}$$



Neural Networks

- Neural Networks are nested affine transformations.
- Suppose we have a network with $L = 2$ layers and an (vectorized image) input $\boldsymbol{x}$:

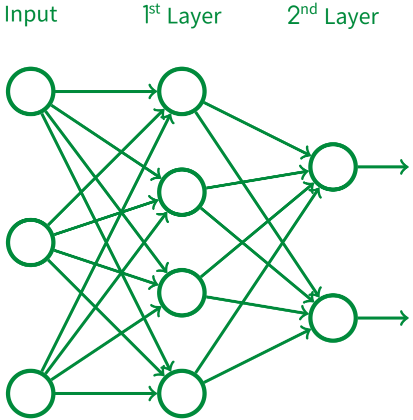
$$\boldsymbol{a}^{(2)} = W_2\sigma(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

- W_1 and W_2 are the *weights* and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are *biases*.
- σ is *activation function*, one of the components that makes the network nonlinear. In the old days, σ was typically a sigmoid to mimic neurons which is difficult to train. The community now prefers *ReLU* activations:

$$\sigma(x) = x \text{ if } x > 0 \text{ and } \sigma(x) = 0 \text{ otherwise}$$

- We use $\boldsymbol{z}$ as:

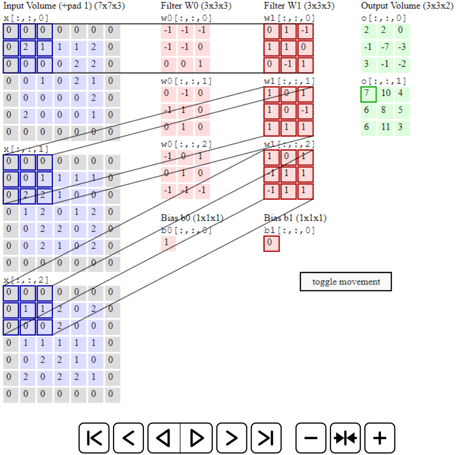
$$\boldsymbol{z}_1 = \sigma(\boldsymbol{a}^{(1)})$$



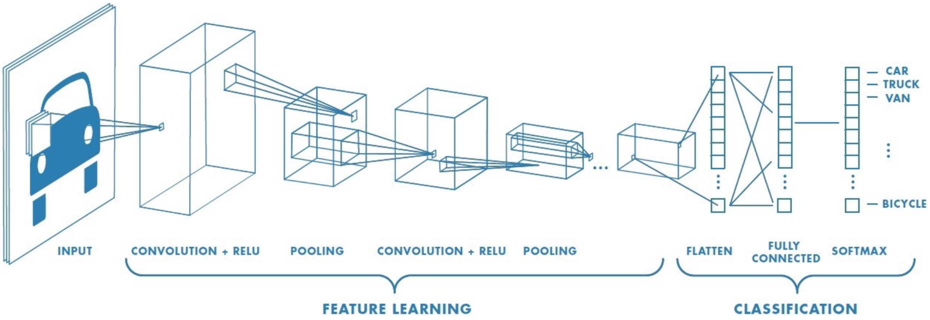
Convolutional Neural Network

- Traditional neural networks work well for text analysis and time series analysis, but for high dimensional problems (image processing) matrices make training cumbersome/impossible.
- Modern networks typical share weights to exploit locality. In other words, we convolve filters across activation maps:

$$a^{(n+1)} = w_{n+1} * z_n + b_{n+1}$$



Stanford cs231



Raghav Prabhu

- The output activation function is typically a *softmax*:

$$\mathbf{z}_j = \sigma(\mathbf{a})_j = \frac{\exp(a_j)}{\sum_{k=1}^{K} \exp(a_k)} \text{ for } j = 1, \ldots, K$$

for a classification problem with K classes. This approximates a one-hot vector.

Training

- NNs require a cost function to "optimize" the weights and biases to map a sample $\boldsymbol{x}$ to a correct label $\boldsymbol{y}(\boldsymbol{x})$. Typically...
  - for image classification, the *cross-entropy* function is a common choice:

$$C_{CE} = -\frac{1}{K}\sum_{k=1}^{K} \boldsymbol{y}(\boldsymbol{x})_k \log(\boldsymbol{z}_k^L) + (1 - \boldsymbol{y}(\boldsymbol{x})_k)\log(1 - \boldsymbol{z}_k^L)$$

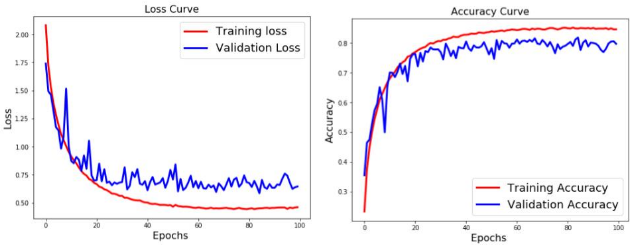
  - for image processing (super-resolution, denoising, etc), the mean squared error is popular:

$$C_{OLS} = \frac{1}{2n}\sum_{\boldsymbol{x}} ||\boldsymbol{y}(\boldsymbol{x}) - \boldsymbol{z}^L(\boldsymbol{x})||_2^2$$

- Even with filters (and pooling) we end up with millions of parameters. How do we train these images? *Backpropagation* and *Stochastic Gradient Descent*.

Training

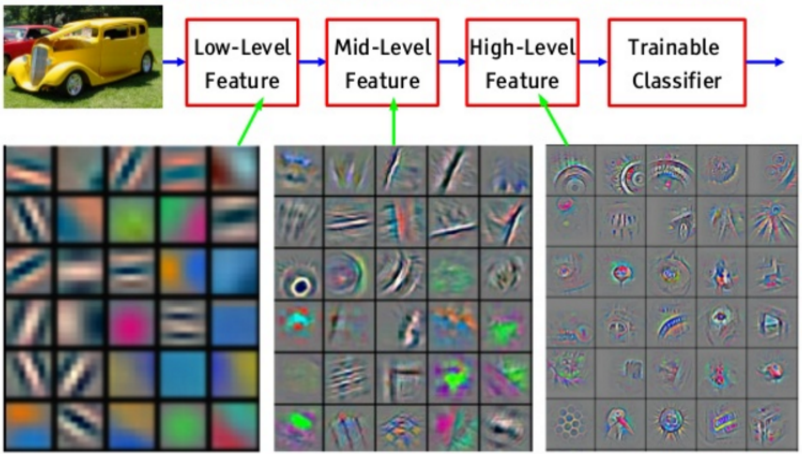
- The tl;dr on backpropagation: we use a forward pass through the network with a training sample and then backpropagate error through each layer using the chain rule many, many times to get the gradient.
- With these gradients, we could use a gradient descent scheme to train the network but if we have millions of training samples, this can be prohibitively slow. Thus, we use stochastic gradient descent on batches of samples.



Learn OpenCV

A Trained Neural Network

- When this works (which is more often than not) we end up with a cascade of finer and finer edge filters.



Yann LeCunn

- The strength of deep learning/neural networks is the auto-feature-generation. Circumventing human bias allows a direct path to a compelling solution.

Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun

Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

*Rectified activation units (rectifiers) are essential for state-of-the-art neural networks. In this work, we study rectifier neural networks for image classification from two aspects. First, we propose a Parametric Rectified Linear Unit (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. Second, we derive a robust initialization method that particularly considers the rectifier nonlinearities. This method enables us to train extremely deep rectified models directly from scratch and to investigate deeper or wider network architectures. Based on our PReLU networks (PReLU-nets), we achieve **4.94%** top-5 test error on the ImageNet 2012 classification dataset. This is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [29]). To our knowledge, our result is the first to surpass human-level performance (5.1%, [22]) on this visual recognition challenge.*

and the use of smaller strides [33, 24, 2, 25]), new nonlinear activations [21, 20, 34, 19, 27, 9], and sophisticated layer designs [29, 11]. On the other hand, better generalization is achieved by effective regularization techniques [12, 26, 9, 31], aggressive data augmentation [16, 13, 25, 29], and large-scale data [4, 22].

Among these advances, the rectifier neuron [21, 8, 20, 34], *e.g.*, Rectified Linear Unit (ReLU), is one of several keys to the recent success of deep networks [16]. It expedites convergence of the training procedure [16] and leads to better solutions [21, 8, 20, 34] than conventional sigmoid-like units. Despite the prevalence of rectifier networks, recent improvements of models [33, 24, 11, 25, 29] and theoretical guidelines for training them [7, 23] have rarely focused on the properties of the rectifiers.

In this paper, we investigate neural networks from two aspects particularly driven by the rectifiers. First, we propose a new generalization of ReLU, which we call

He *et al* CVPR 2015

What Can Go Wrong?

- While NNs offer unprecedented performance, we have to make some questionable steps.
- Among the issues:
  - What are these learned features looking for?
  - Does the stochastic gradient descent ever converge to a minimum? (non-convex, NP hard problem[1])
  - What does the cost manifold look like?
  - How robust are they to noise?
  - Do they overfit?

What Can Go Wrong?

- While NNs offer unprecedented performance, we have to make some questionable steps.
- Among the issues:
  - What are these learned features looking for?
  - Does the stochastic gradient descent ever converge to a minimum? (non-convex, NP hard problem[1])
  - What does the cost manifold look like?
  - How robust are they to noise?
  - Do they overfit?

The Rest of this Talk...

- For the rest of this talk, we are going to discuss ways in which deep learning/neural networks can fooled.
- Key context: deep learning is **the state-of-the-art**. It is difficult to justify using SVMs/random forests/etc. for most problems when a neural network can *significantly* improve performance.
- Note as well that these other machine learning strategies can also be fooled - potentially in the exact same way.

Outline

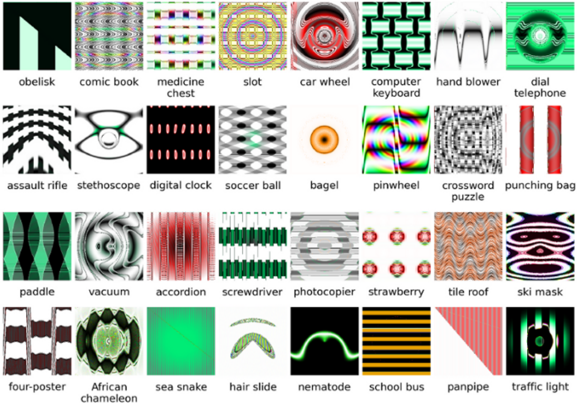
Setting

- Assume for the following that we have a well-trained neural network for a classification task.
- It is easiest to illustrate the following ideas with images, but they apply for any domain.

Outline

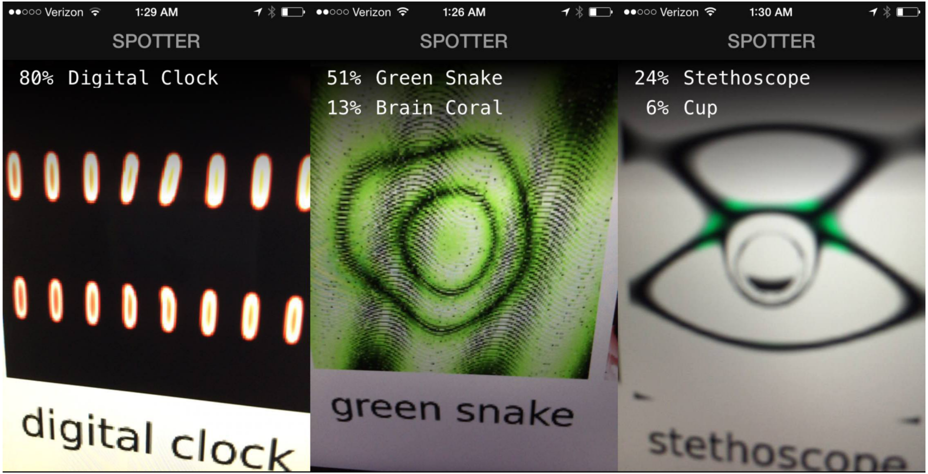
What Is the Network Looking At?

- We did not constrain the network to filters that we can understand.
- Research has shown that deformations of objects into gibberish can still earn high scores from a neural network. The images below all have $> 99\%$ confidence from a well-trained model.

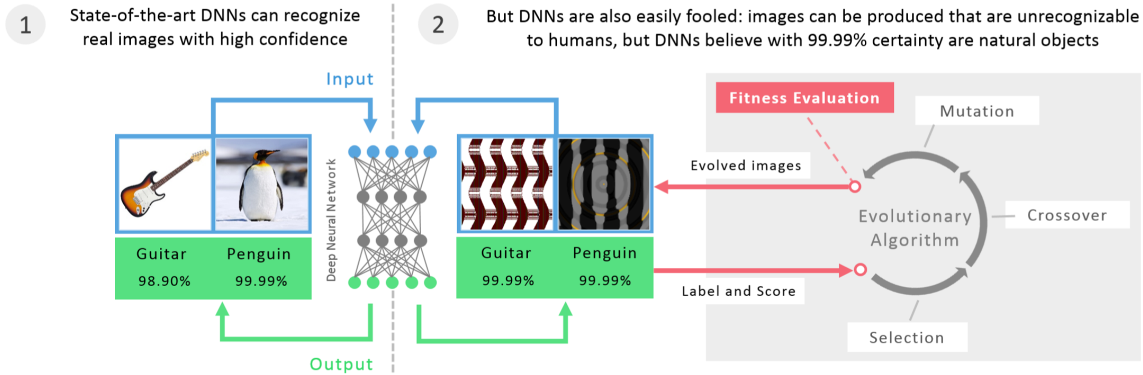


Nguyen *et al* CVPR 2015

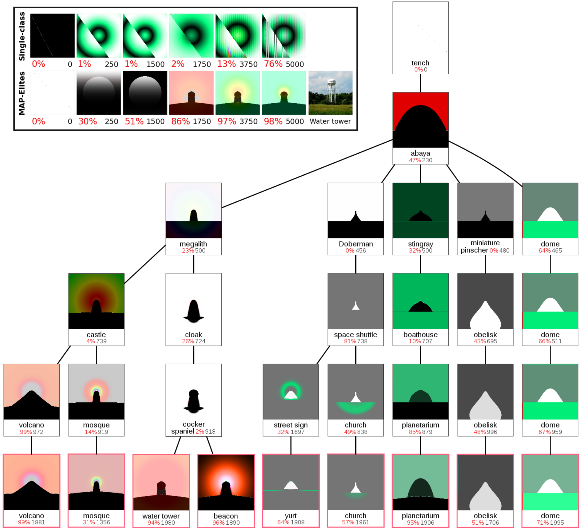
Pernicious Gibberish



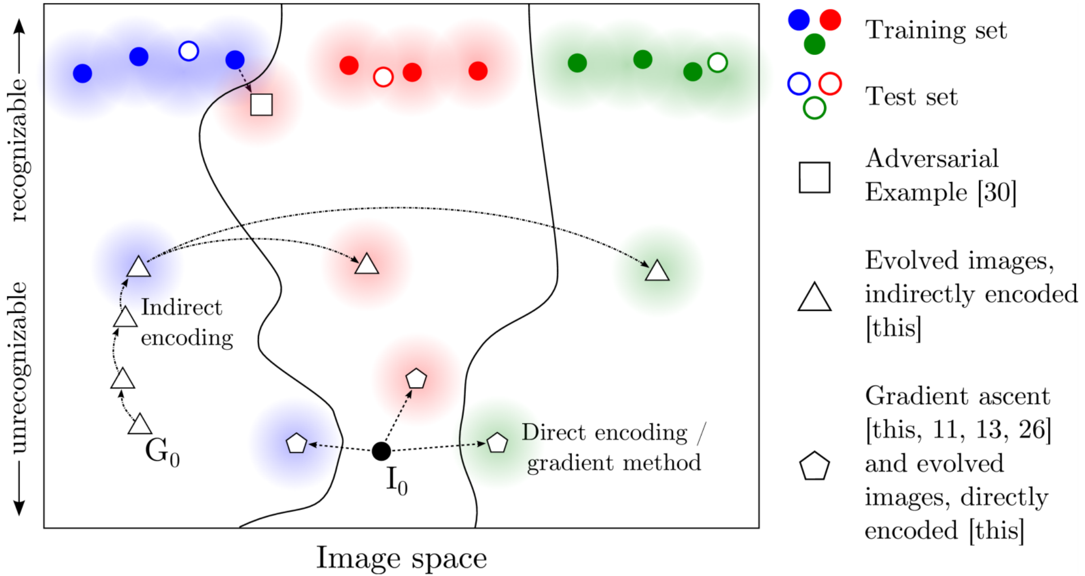
Nguyen *et al* CVPR 2015



Nguyen *et al* CVPR 2015



Nguyen *et al* CVPR 2015

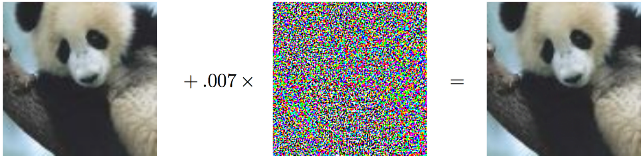


Nguyen *et al* CVPR 2015

Outline

How robust is the model to being fooled?

- What if someone wants to actively fool a network? What if we have an **adversarial attack**?

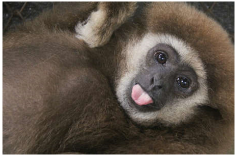


$\boldsymbol{x}$
"panda"
57.7% confidence

$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

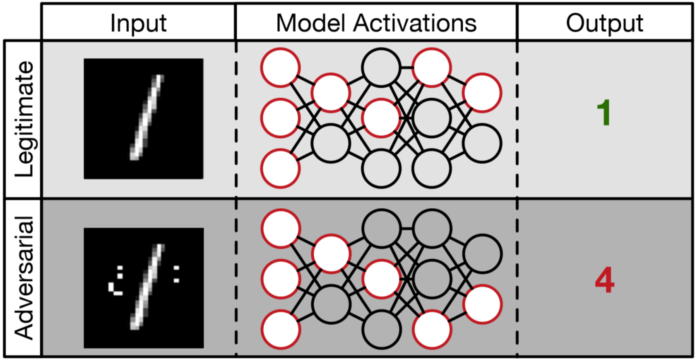
$\boldsymbol{x} + \epsilon\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

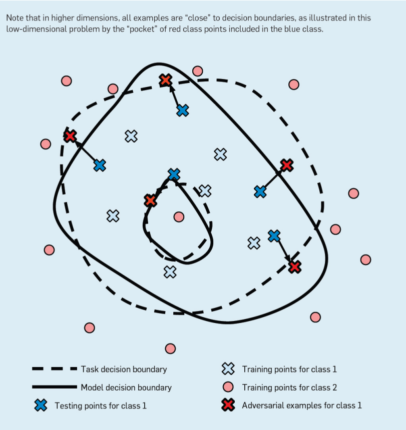


Goodfellow *et al* ICLR 2014

How Attacks Work: General Idea



Papernot *et al* 2016 arXiv



GoodFellow *et al* ACM Magazine 2018

How Attacks Work: White Box

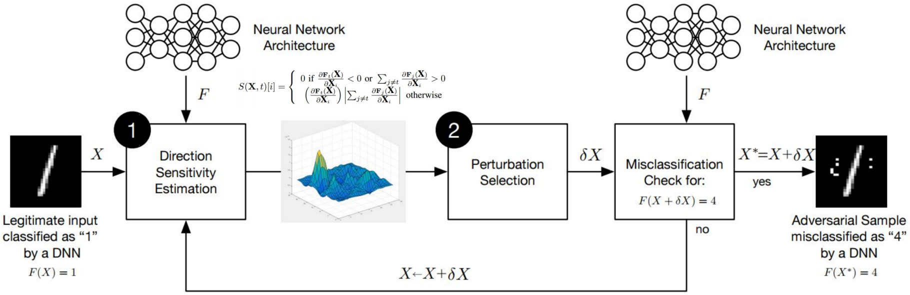
- Suppose we have a learned model $F(X) \mid \boldsymbol{\theta})$ for a set of parameters $\boldsymbol{\theta}$. For a *white box attack*, we know $\boldsymbol{\theta}$ and the architecture of F and want to design an adversary $X = X + \delta X$ that fools the network with a high degree of confidence. Since we also want the perturbation creating the adversary to be small, we look to solve

$$\arg\min_{\delta X} ||\delta X|| \text{ such that } F(X + \delta X) = \underbrace{Y_k}_{\text{one-hot vector for class } k} \neq \underbrace{Y_\ell}_{\text{class } \ell \neq k}$$

- Because of the complexity of a neural network, solving this problem is non-trivial (highly non-linear and non-convex [25]).

How Attacks Work: White Box

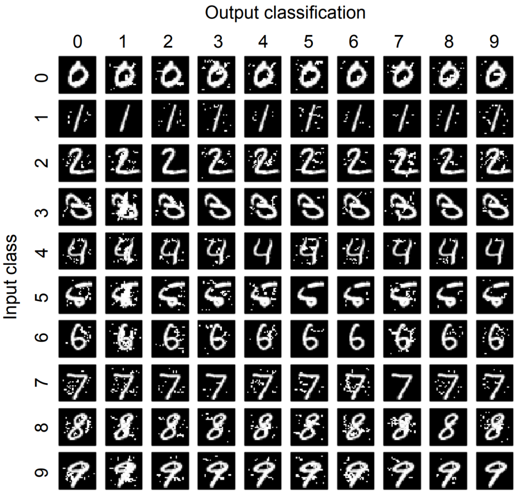
- Jacobian-based Saliency Map Approach:



Papernot *et al* ESSP 2016

How Attacks Work: White Box

- Jacobian-based Saliency Map Approach:



Papernot *et al* ESSP 2016

Black Box Attacks

- Black box attacks: **the adversary doesn't know model parameters**.
- These attacks are *harder* to deal with than white box attacks.

$$
\begin{tikzcd}
\text{Model A} \arrow[dr] & & \text{Adversarial Attack Function } Q_A(\cdot) \\
\text{Model B} \arrow[r] & \text{Utilize Model Parameters to Get Perturbations} \arrow[ur] \arrow[r] \arrow[dr] & \text{Adversarial Attack Function } Q_B(\cdot) \\
\text{Model C} \arrow[ur] & & \text{Adversarial Attack Function } Q_C(\cdot)
\end{tikzcd}
$$

White Box Attack

$$\text{Test Query } X \longrightarrow \text{Generate Adversary } Q_A(X) \longrightarrow \text{class}_A(Q_A(X)) \neq \text{class}_A(X)$$

Black Box Attack

$$
\begin{tikzcd}
 & \text{Generate Adversary } Q_A(X) \arrow[dr] \arrow[ddr] & \\
\text{Query } X \arrow[ur] & & \text{class}_B(Q_A(X)) \neq \text{class}_B(X) \\
 & & \text{class}_C(Q_A(X)) \neq \text{class}_C(X)
\end{tikzcd}
$$

Black Box Attacks

- Suppose we can generate $Y = F(X)$ for any X but do not have $\boldsymbol{\theta}$ or the original training data.
- **Strategy 1**: Iterate over synthetic data to train a new network $\hat{F}$ to mimic the output of F. With $\hat{F}$, we can then craft adversaries using $\hat{F}$ but apply them towards F.
- **Strategy 2**: Take an existing network trained for a similar task and use that as $\hat{F}$ in the strategy above.

Black Box Attacks

- Suppose we can generate $Y = F(X)$ for any X but do not have $\boldsymbol{\theta}$ or the original training data.
- **Strategy 1**: Iterate over synthetic data to train a new network $\hat{F}$ to mimic the output of F. With $\hat{F}$, we can then craft adversaries using $\hat{F}$ but apply them towards F.

Algorithm 1 Papernot *et al* 2017 ASIA CCS

Input $F, \hat{F}, S_0, \lambda$
for $n = 0$ to N **do**
  // Label the substitute training set.
  $D = \{(X, F(X)) : X \in S_n\}$
  // Train $\hat{F}$ on D to evaluate parameters $\hat{\theta}$
  $\hat{\theta} = \text{train}(\hat{F}, D)$
  // Perform Jacobian-based data set augmentation
  $S_{n+1} = \{X + \lambda \text{sign}(J_{\hat{F}}[F(X)]) : X \in S_n\} \cup S_n$
end for
return $\theta_{\hat{F}}$

- **Strategy 2**: Take an existing network trained for a similar task and use that as $\hat{F}$ in the strategy above.

Black Box Attacks

- Suppose we can generate $Y = F(X)$ for any X but do not have $\boldsymbol{\theta}$ or the original training data.
- **Strategy 1**: Iterate over synthetic data to train a new network $\hat{F}$ to mimic the output of F. With $\hat{F}$, we can then craft adversaries using $\hat{F}$ but apply them towards F.
- **Strategy 2: Take an existing network trained for a similar task and use that as $\hat{F}$ in the strategy above.**

Table 1: **Error rates (in %) of adversarial examples transferred between models.** We use Step-LL with $\epsilon = {}^{16}/_{256}$ for 10,000 random test inputs. Diagonal elements represent a white-box attack. The best attack for each target appears in bold. Similar results for MNIST models appear in Table 7.

Top 1

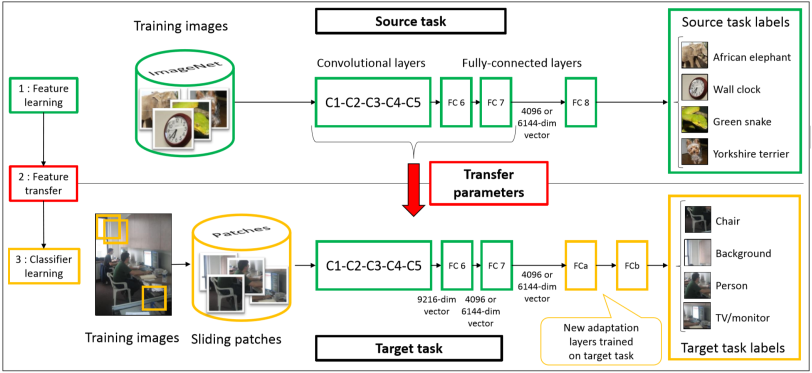
	Source				
Target	v4	v3	v3_{adv}	IRv2	IRv2_{adv}
v4	**60.2**	39.2	31.1	36.6	30.9
v3	43.8	**69.6**	36.4	42.1	35.1

Top 5

	Source				
Target	v4	v3	v3_{adv}	IRv2	IRv2_{adv}
v4	**31.0**	14.9	10.2	13.6	9.9
v3	18.7	**42.7**	13.0	17.8	12.8

Papernot *et al* ICLR 2018

- A key concept of modern NN theory is **transfer learning**, the ability to share weights among similar tasks.
- Hypothesis: learned human interpret-able image decision manifolds are largely the same.



Oquab *et al* CVPR 2014

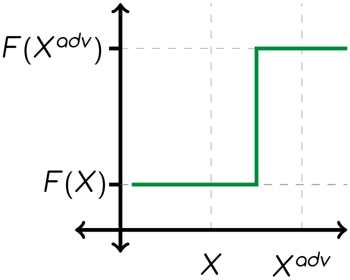
Why Do Black Box Attacks Work?

- A key concept of modern NN theory is **transfer learning**, the ability to share weights among similar tasks.
- Hypothesis: learned human interpret-able image decision manifolds are largely the same.

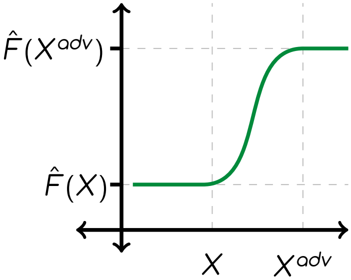
Method	mean Accuracy
HSV [27]	43.0
SIFT internal [27]	55.1
SIFT boundary [27]	32.0
HOG [27]	49.6
HSV+SIFTi+SIFTb+HOG(MKL) [27]	72.8
BOW(4000) [14]	65.5
SPM(4000) [14]	67.4
FLH(100) [14]	72.7
BiCos seg [7]	79.4
Dense HOG+Coding+Pooling[2] w/o seg	76.7
Seg+Dense HOG+Coding+Pooling[2]	80.7
CNN-SVM w/o seg	74.7
CNNaug-SVM w/o seg	**86.8**

Razavian *et al* CVPR 2014

- Black box attacks work even when models are designed to resist white box attacks.



Trained with Smoothing Penalty



Other Model

Papernot *et al* IEEE ESSP 2018

Outline

- We can prevent white box attacks by training with adversarial examples on the model itself.

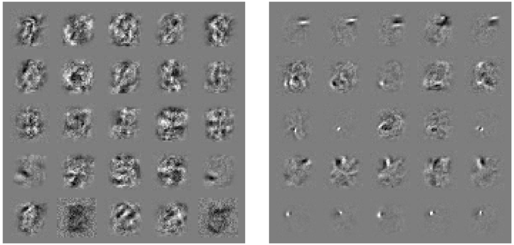
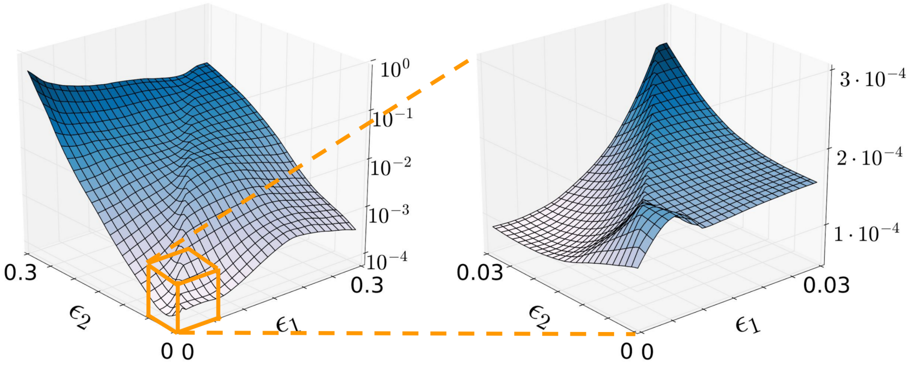


Figure 3: Weight visualizations of maxout networks trained on MNIST. Each row shows the filters for a single maxout unit. Left) Naively trained model. Right) Model with adversarial training.

Goodfellow *et al* ICLR 2015

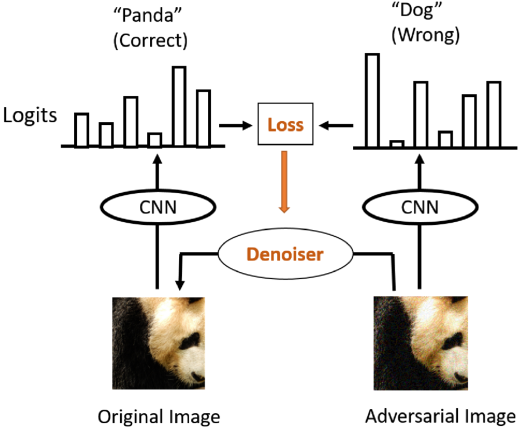
- Similar to white box, we can use train several models and train using a mixed batch of adversarial images.
- Ensemble adversarial training makes a model more robust to attacks and less useful in transferring attacks.
- Such models do have a lower ceiling in terms of general performance.



Tramèr *et al* ICLR 2018

Protection from Black Box Attacks

- Similar to white box, we can use train several models and train using a mixed batch of adversarial images.
- Ensemble adversarial training makes a model more robust to attacks and less useful in transferring attacks.
- Such models do have a lower ceiling in terms of general performance.

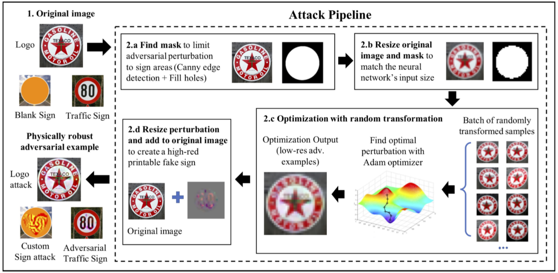


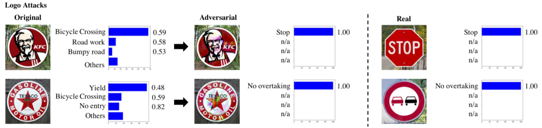
Liao *et al* CVPR 2018

Outline

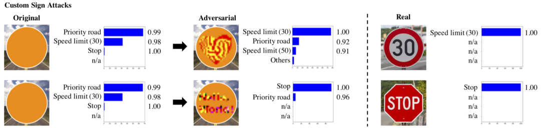
Summary

- Deep learning is the state-of-the-art. It's unavoidably the best choice for most classification tasks.
- It's mysterious design. We want the machine to craft its own features even though we won't be able to decipher their meaning.
- Adversarial images show the double edged sword of CNN feature generation. The incredible performance comes with vulnerabilities.
- We are not sure how generalizable these NNs are.





(a) Classification of Logo attack examples. The adversarial examples are classified with high confidence as a real traffic sign, in spite of being out of the dataset.



(b) Classification of Custom Sign attack examples. The adversarial examples are classified with high confidence as a real traffic sign, in spite of being custom made signs.

Sitawarin *et al* ACM CCS 2018