Adversarial Playground

A Visualization Suite for Adversarial Sample Generation

Andrew Norton

University of Virginia

April 28, 2017

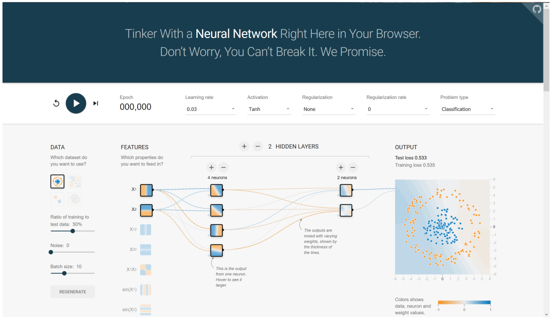
Outline

1. Review of Research
2. Fast Jacobian Saliency Map Apriori
   - Algorithm
   - Experiment
3. Adversarial Playground: The Webapp
   - System Organization
   - Attack Models
4. Discussion

Review of Research

TensorFlow Playground

Pedagogical tool to enhance understanding of neural networks (Yosinski et al. 2015):



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch 000,000
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10

FEATURES
Which properties do you want to feed in?

2 HIDDEN LAYERS
4 neurons
2 neurons

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

OUTPUT
Test loss 0.533
Training loss 0.535

Colors shows data, neuron and weight values.

- Entirely browser-based (JavaScript)
- Configurable Neural Network
- Trains in real-time and displays classifier

Adversarial Machine Learning: The “Big Picture”

- General topic: Machine Learning in presence of an adversary
- This project: Evasion (attacking side)

Evasion	Generate a sample that is similar to samples in class y, but is not classified in y
Targeted	The generated sample is classified in a specified class, y_{target}
Untargeted	The generated sample may be in any class that is not y

- “Similar to” is vague; various norms are used to formalize (in practice L^p norms)

Formalizing Targeted and Untargeted Attacking

Let X be the vectorspace of all inputs with a norm $\|\cdot\|$, C be the set of possible classes, and $F : X \to C$ a classifier. Further, let $\mathbf{x} \in X$ be a starting sample.

Targeted

Let $y_t \in C$ be the target class. Then, the evading sample, x', is:

$$x' = \arg\min \left\{ \|x - x'\| \ : \ F(x') = y_t \right\} \tag{1}$$

Untargeted

The evading sample, x', is:

$$x' = \arg\min \left\{ \|x - x'\| \ : \ F(x') \neq F(x) \right\} \tag{2}$$

Norm Selection

The norm may be varied to yield different evasion algorithms. Most commonly, we choose one of three L^p norms:

- L^∞
- L^2
- L^0

In the next slides, let $r = x' - x$, where x' and x are the adversarial and original input samples, respectively.

L^∞ Norm

L^∞ Norm

Measures maximum difference between x and x' along a single feature.

$$\|r\|_\infty = \max_i \{|r_i|\} \tag{3}$$

Two evasion algorithms utilize the L^∞ norm:

- Fast Gradient Sign, Goodfellow et al. (2014)
- Berekley Algorithm, Carlini & Wagner (2016)

L^2 Norm

L^2 Norm

Standard Euclidean distance between x and x'.

$$\|r\|_2 = \left(\sum_i r_i^2\right)^{1/2} \tag{4}$$

Two evasion algorithms utilize the L^∞ norm:

- L-BFGS (original paper), Szegedy et al. (2013)
- Berekley Algorithm, Carlini & Wagner (2016)

L^0 Norm

L^0 Norm

Counts the number of features with nonzero difference between x and x'.

$$\|r\|_0 = \sum_i 1_{r_i \neq 0} \tag{5}$$

Two evasion algorithms utilize the L^0 norm:

- Jacobian Saliency Map Approach, Papernot et al. (2015)
- Berekley Algorithm, Carlini & Wagner (2016)

Jacobian Saliency Map Approach

Saliency map

A visualization tool from image processing to highlight meaningful pixels.

In adversarial ML, this highlights pixels with large impact on resulting class.

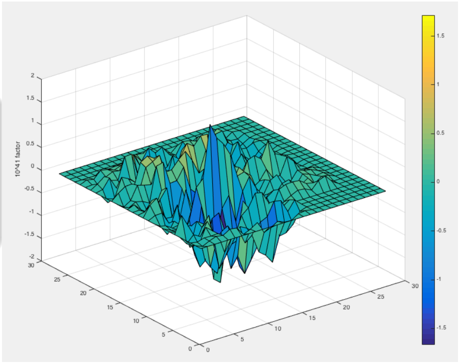


Figure: Saliency map of an input to LeNet. (Papernot et al. 2015.)

Saliency Map Choice

Many saliency maps may be defined, but Papernot et al. suggest the following for algorithms that decrease features:

$$S(\mathbf{X}, y_t)[i] = \begin{cases} 0 & \text{if } \frac{\partial \mathbf{F}_{\mathbf{y_t}}(\mathbf{X})}{\partial \mathbf{X}_i} > 0 \text{ or } \sum_{j \neq i} \frac{\partial \mathbf{F}_{\mathbf{j}}(\mathbf{X})}{\partial \mathbf{X}_i} < 0 \\ \left| \frac{\partial \mathbf{F}_{\mathbf{y_t}}(\mathbf{X})}{\partial \mathbf{X}_i} \right| \cdot \sum_{j \neq i} \frac{\partial \mathbf{F}_{\mathbf{j}}(\mathbf{X})}{\partial \mathbf{X}_i} & \text{else} \end{cases} \tag{6}$$

Informally: Important pixels decrease likelihood of classification in y_t or increase overall likelihood for other classes.

Pseudocode (JSMA)

Algorithm 1 **Crafting adversarial samples for LeNet-5**
$\mathbf{X}$ is benign image, y_t is target class, $\mathbf{F}$ is the classifier function, Υ is maximum distortion, and θ is the change made to pixels.

```
Input: $\mathbf{X}$, $y_t$, $\mathbf{F}$, $\Upsilon$, $\theta$
 1: $\mathbf{X}' \leftarrow \mathbf{X}$
 2: $\Gamma = \{1 \ldots |\mathbf{X}|\}$                              ▷ search domain is all pixels
 3: $\texttt{max\_iter} = \left\lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \right\rfloor$      ▷ Modify up to $\Upsilon$ percent of image
 4: $s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$                                  ▷ source class
 5: while $s \neq y_t$ & $\texttt{iter} < \texttt{max\_iter}$ & $\Gamma \neq \emptyset$ do
 6:     $p_1, p_2 = \texttt{saliency\_map}(\nabla \mathbf{F}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$
 7:     Modify $p_1$ and $p_2$ in $\mathbf{X}^*$ by $\theta$              ▷ In practice, $\theta = 1$
 8:     Remove $p_j$ from $\Gamma$ if $p_j == 0$ or $p_j == 1$
 9:     $s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$
10:     $\texttt{iter}++$
11: end while
12: return $\mathbf{X}'$
```

Saliency Pseudocode (JSMA)

Algorithm 2 Papernot's Saliency Map Feature Selection

$\nabla\mathbf{F}(\mathbf{X})$ is the forward derivative, Γ the features still in the search space, and t the target class. Directly from Papernot et al. (2015).

```
Input: $\nabla\mathbf{F}(\mathbf{X})$, $\Gamma$, $t$
1: for each pair $(p, q) \in \Gamma^2,\ p \neq q$ do
2:     $\alpha = \sum_{i=p,q} \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}$
3:     $\beta = \sum_{i=p,q} \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}$
4:     if $\alpha < 0$ and $\beta > 0$ and $-\alpha \times \beta > \max$ then
5:         $p_1, p_2 \leftarrow p, q$
6:         $max \leftarrow -\alpha \times \beta$
7:     end if
8: end for
9: return $p_1, p_2$
```

Fast Jacobian Saliency Map Apriori

Fast Jacobian Saliency Map Apriori

Problem with JSMA

- Saliency map search is very slow
- All-pair search is $\Theta\left(\binom{|\Gamma|}{2}\right) = \Theta\left(|\Gamma|^2\right)$

Key Idea: Don't search *all pairs* for optimum; eliminate some coordinates *a priori*.

- Instead of all pairs (p, q) where $p, q \in \Gamma$, restrict p
- Select top k points from Γ when ordered by α value
- Force p to be from these top k points

Saliency Pseudocode (FJSMA)

Algorithm 3 **Fast Jacobian Saliency Map Apriori Feature Selection**
$\nabla \mathbf{F}(\mathbf{X})$, Γ, and t as in Algorithm 2, k is a small constant

```
Input: $\nabla \mathbf{F}(\mathbf{X})$, $\Gamma$, $t$, $k$
 1: $K = \arg\operatorname{top}_{p \in \Gamma} \left( -\frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i};\ k \right)$
 2: for each pair $(p, q) \in K \times \Gamma,\ p \neq q$ do
 3:     $\alpha = \sum_{i=p,q} \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}$
 4:     $\beta = \sum_{i=p,q} \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}$
 5:     if $\alpha < 0$ and $\beta > 0$ and $-\alpha \times \beta >$ max then
 6:         $p_1, p_2 \leftarrow p, q$
 7:         $max \leftarrow -\alpha \times \beta$
 8:     end if
 9: end for
10: return $p_1, p_2$
```

Benefit and Cost

We experience the usual tradeoff in heuristics: speed against accuracy:

Benefit *Much* faster runtime for each iteration. Instead of being $\Theta\left(|\Gamma|^2\right)$, the algorithm is $\Theta\left(k \cdot |\Gamma|\right)$, where $k \ll |\Gamma|$ is a small constant.

Cost Possible reduction in evasion rate – it could be that the optimum of $-\alpha\beta$ occurs for some small value of α but a very large value of β.

Experiment Design

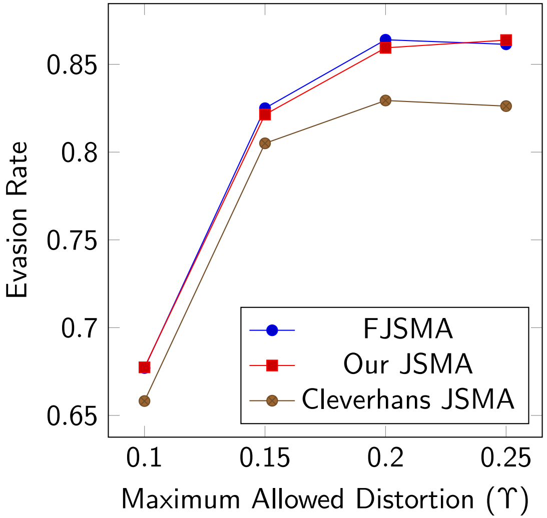
Model and Data:

- MNIST Dataset
- CNN from TensorFlow website for classification
- Used 5000 samples from MNIST

Measurement:

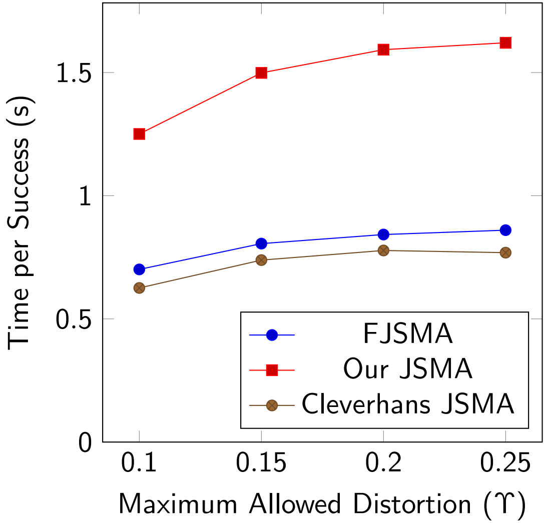
- Varied Υ (maximum L^0 difference between x and x')
- Evasion Rate $=$ (# Successful) / (# Attempted)
- Compared evasion rate and time of three implementations
  - `cleverhans` JSMA
  - JSMA (reimplementation, single-threaded)
  - FJSMA

Results: Evasion Rate



- Both implementations outperform `cleverhans`
- FJSMA and JSMA perform approximately equivalently

Results: Time for Successful Sample



- Clearly, FJSMA is faster than JSMA
- `cleverhans` is multithreaded and runs on all cores
- Our JSMA and FJSMA are *not* multithreaded
- FJSMA is slower than `cleverhans`, but I have not yet written a multithreaded version.

Adversarial Playground: The Webapp

Adversarial Playground

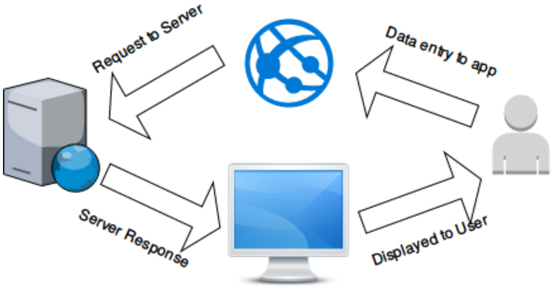
Goal: A comparable webapp to *TensorFlow Playground* for adversarial machine learning.

Design Specifications:

- Easy install
- Web based interface
- Multiple evasion models
- Targeted and Untargeted Methods
- Configure attack parameters

System Organization

Due to computation complexity, the server/GPU is involved.



1. User selects model type
2. User configures model and chooses sample
3. Data is sent to server
4. TensorFlow backend generates adversarial sample
5. Images describing sample information are returned
6. Images are displayed to the user

Attack Models

Currently, the webapp supports three attack models

- Fast Gradient Sign, adapted from `cleverhans`.
- Jacobian Saliency Map Approach (our own implementation)
- Fast Jacobian Saliency Map Apriori

Live Demonstration

Live Demo.

Discussion

Results

Fast Jacobian Saliency Map Apriori

- Heuristically reduces search space
- Evasion rate similar to original JSMA.
- Speed is similar to multithreaded JSMA approach

Adversarial Playground: Webapp

- User-friendly tool for exploring behavior of different evasion algorithms
- Supports targeted and untargeted approaches
- Displays original and evasive sample, together with classification likelihoods

Future Work

FJSMA

- Comparisons against algorithms from Carlini & Wagner (2016).
- Comparisons on multiple models
- Create multithreaded version for comparison against `cleverhans`

Adversarial Playground: Webapp

- Move image/plot generation entirely to the client to speed up
- Improve instructions
- Drop-in model additions

Bibliography I

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pp. 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. URL `http://dl.acm.org/citation.cfm?id=645920.672836`.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016. URL `http://arxiv.org/abs/1608.04644`.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Ian J. Goodfellow, Nicolas Papernot, and Patrick D. McDaniel. cleverhans v0.1: an adversarial machine learning library. *CoRR*, abs/1610.00768, 2016. URL `http://arxiv.org/abs/1610.00768`.

Bibliography II

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015. URL `http://arxiv.org/abs/1511.07528`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL `http://arxiv.org/abs/1312.6199`.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015. URL `http://arxiv.org/abs/1506.06579`.