Adversarial Playground

A Visualization Suite for Adversarial Sample Generation

Andrew Norton

University of Virginia

June 5, 2017

Outline

System Demonstration

Overview

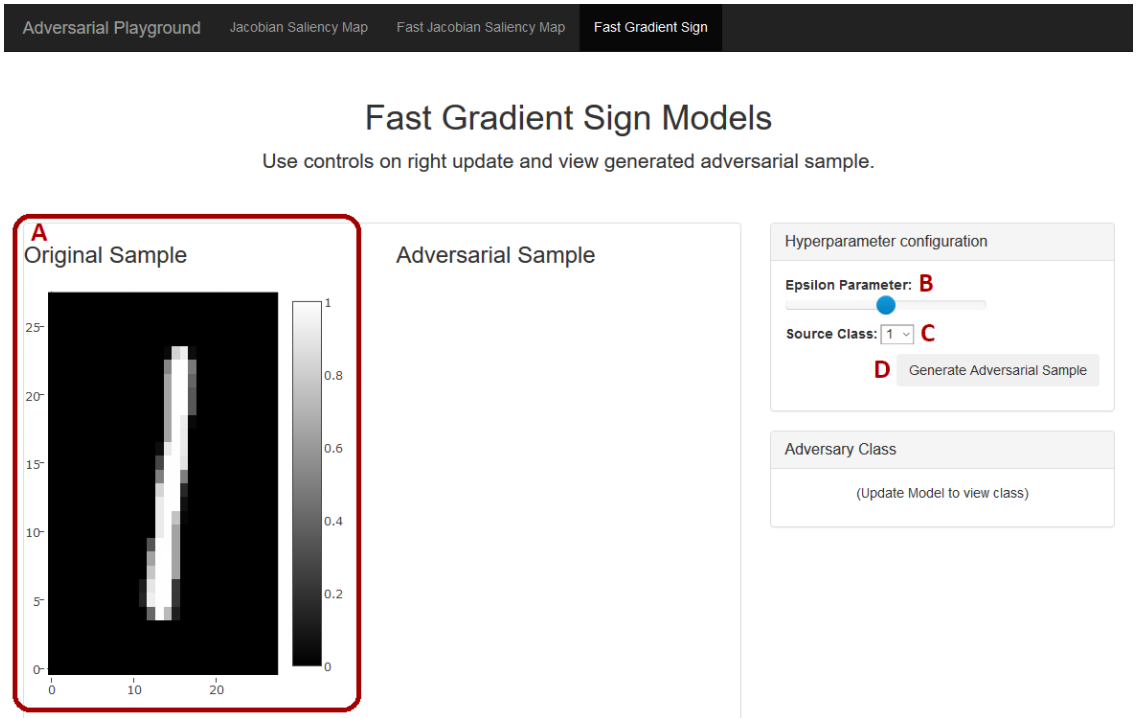
Adversarial Playground is a toolkit that visually represents the results of three evasion models, with both targeted and untargeted attacks.

- Fast Gradient Sign [Untargeted]
- Jacobian Saliency Map Approach (JSMA) [Targeted]
- Fast Jacobian Saliency Map Apriori (FJSMA) [Targeted]

Each model has a distinct link on the global navigation bar at the top of the web interface.

Fast Gradient Sign

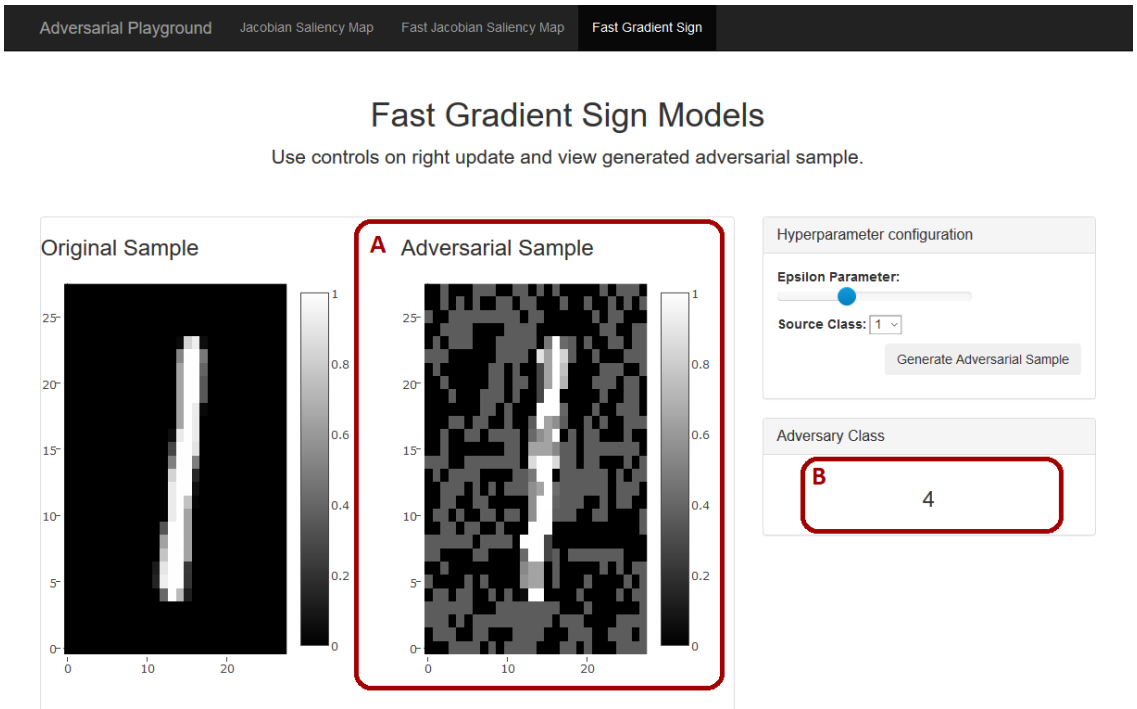
When the user loads the Fast Gradient Sign option, they are immediately presented with several options:



A. Source image
B. Attacking power (hyperparameter $\in [0, 1]$)
C. Source class selection
D. Button to generate evasive sample

Fast Gradient Sign

After generating the adversarial sample, two noticeable additions appear on the page:

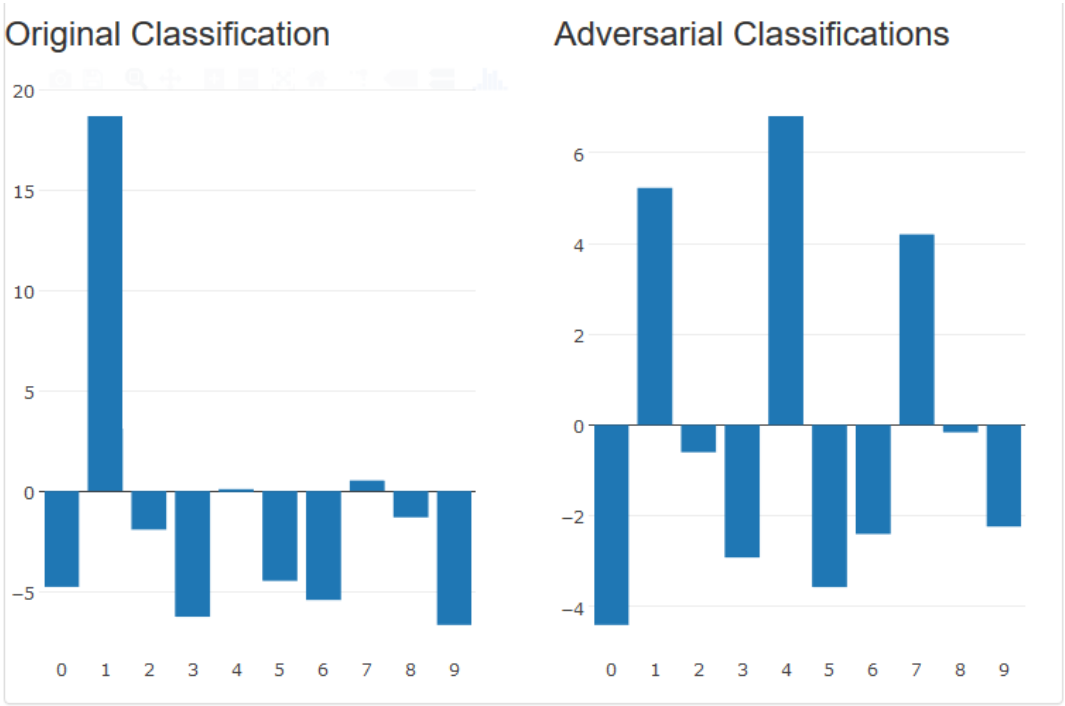


A. Generated Adversarial Image

B. Classification of generated image

Fast Gradient Sign

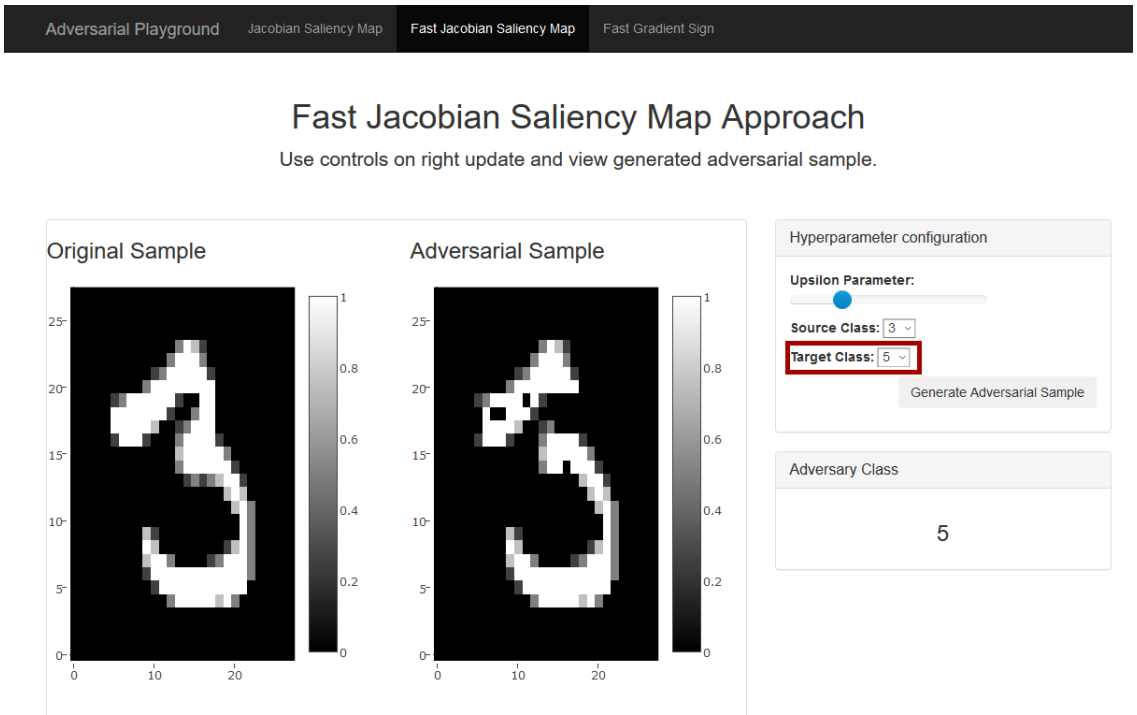
Scrolling down, the user is presented with the classifier's output for the original and generated samples



These are unnormalized values, but the value of each class' value is representative of the likelihood the image is of that class. The index with greatest value is the predicted class.

JSMA/FJSMA

The user interfaces for both the Jacobian Saliency Map and the FJSMA methods are nearly the same as that for Fast Gradient Sign.
We need only add a "target" class option (boxed below):

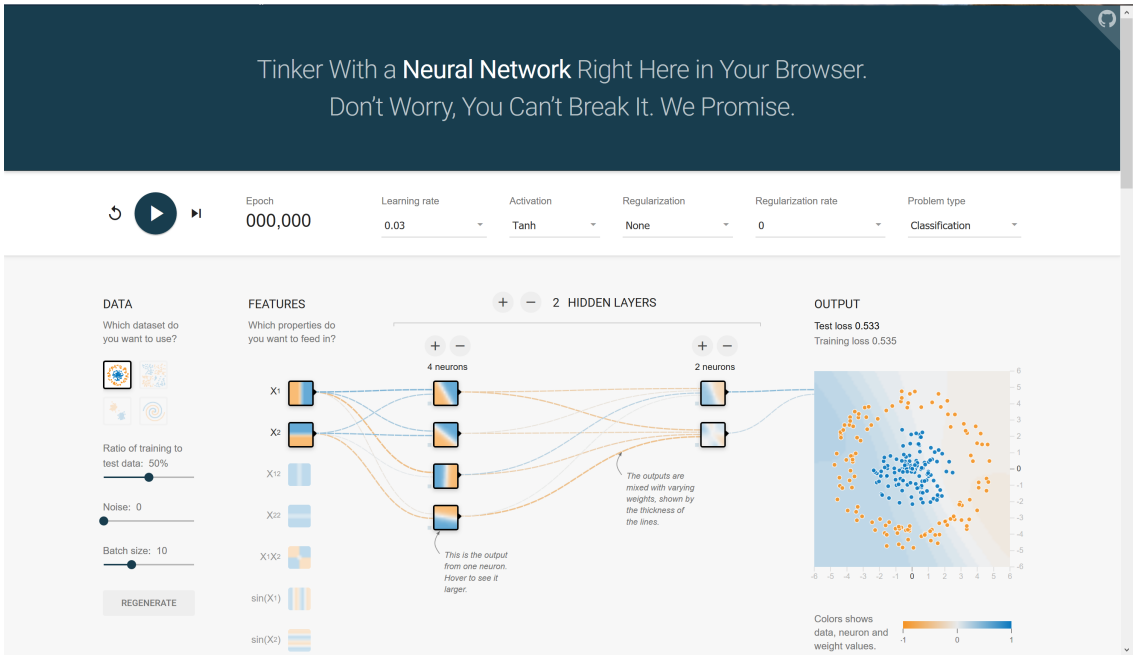


Notice the JSMA/FJSMA options modify fewer pixels, but to a greater extent.

Review of Research

TensorFlow Playground

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



- 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 and 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 and 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 and 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.

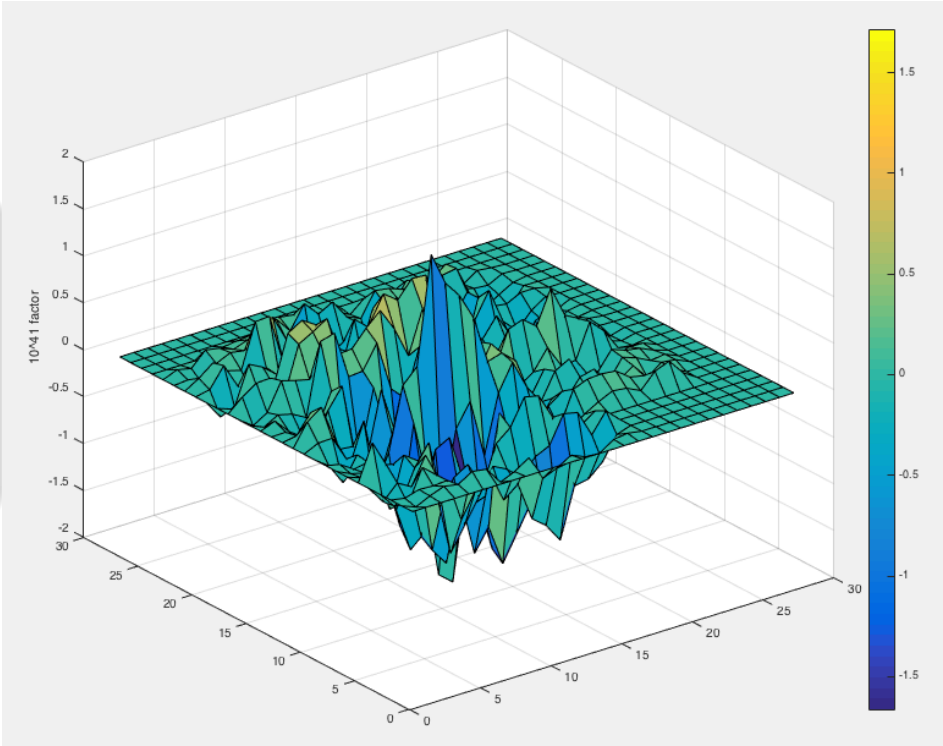


Figure: Saliency map of an input to LeNet. (Papernot et al. 2015.)

Saliency Map Choice

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

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

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**

X is benign image, y_t is target class, **F** is the classifier function, Υ is maximum distortion, and θ is the change made to pixels.

Input: $\mathbf{X}$, y_t, $\mathbf{F}$, Υ, θ

$\mathbf{X}' \leftarrow \mathbf{X}$

$\Gamma = \{1 \ldots |\mathbf{X}|\}$ ▷ search domain is all pixels

$\texttt{max\_iter} = \lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \rfloor$ ▷ Modify up to Υ percent of image

$s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$ ▷ source class

while $s \neq y_t$ & $\texttt{iter} < \texttt{max\_iter}$ & $\Gamma \neq \emptyset$ **do**

  $p_1, p_2 = \texttt{saliency\_map}(\nabla \mathbf{F}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$

  Modify p_1 and p_2 in $\mathbf{X}^*$ by θ ▷ In practice, $\theta = 1$

  Remove p_j from Γ if $p_j == 0$ or $p_j == 1$

  $s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$

  $\texttt{iter} + +$

end while

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: $\nabla \mathbf{F}(\mathbf{X})$, Γ, t

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

$K = \text{arg top}_{p \in \Gamma} \left(-\frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_p};\ k \right)$
for each pair $(p, q) \in K \times \Gamma,\ p \neq q$ **do**
    $\alpha = \sum_{i=p,q} \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}$
    $\beta = \sum_{i=p,q} \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}$
    if $\alpha < 0$ and $\beta > 0$ and $-\alpha \times \beta >$ max **then**
        $p_1, p_2 \leftarrow p, q$
        $max \leftarrow -\alpha \times \beta$
    end if
end for
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

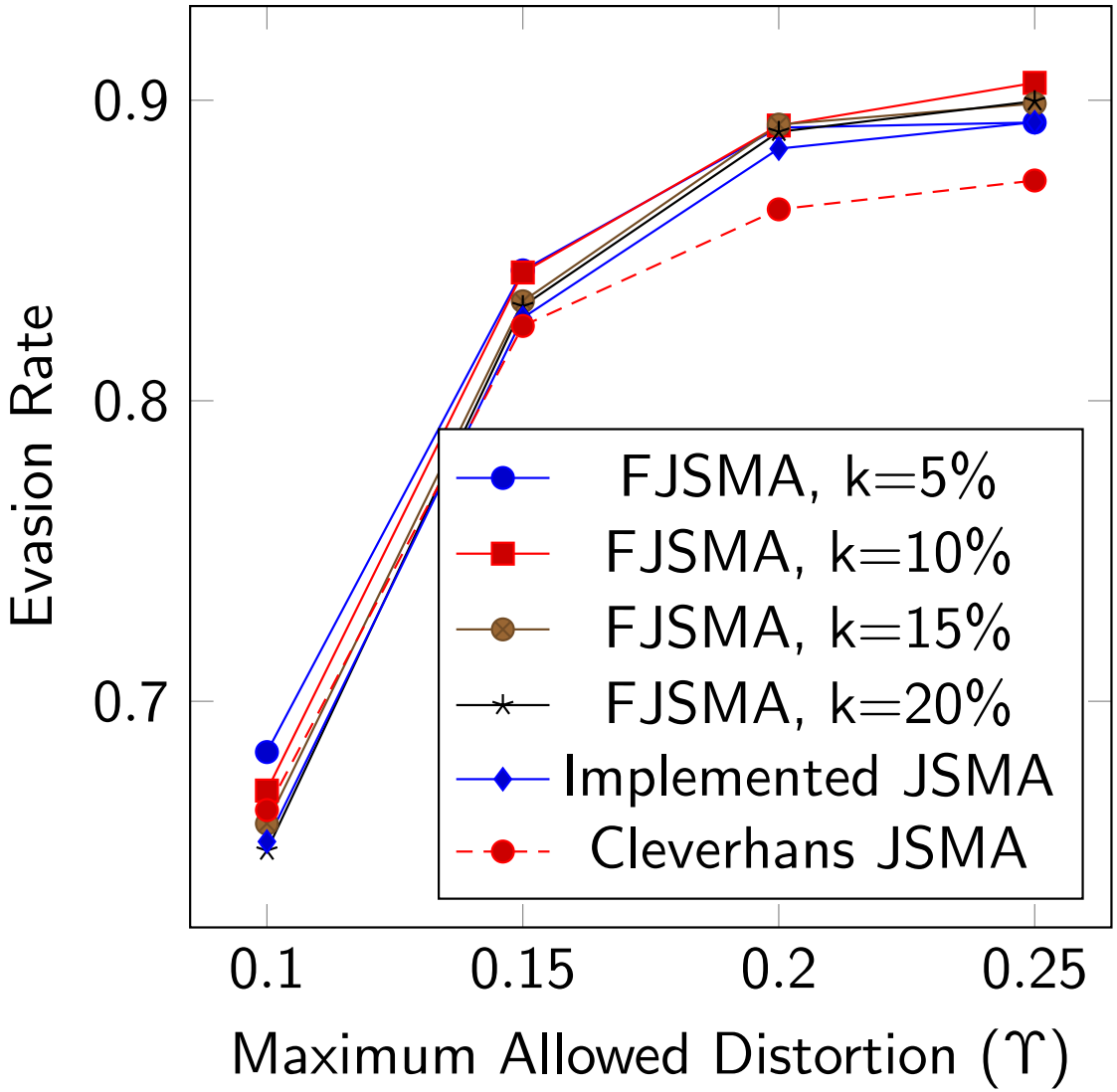
Model and Data:

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

Measurement:

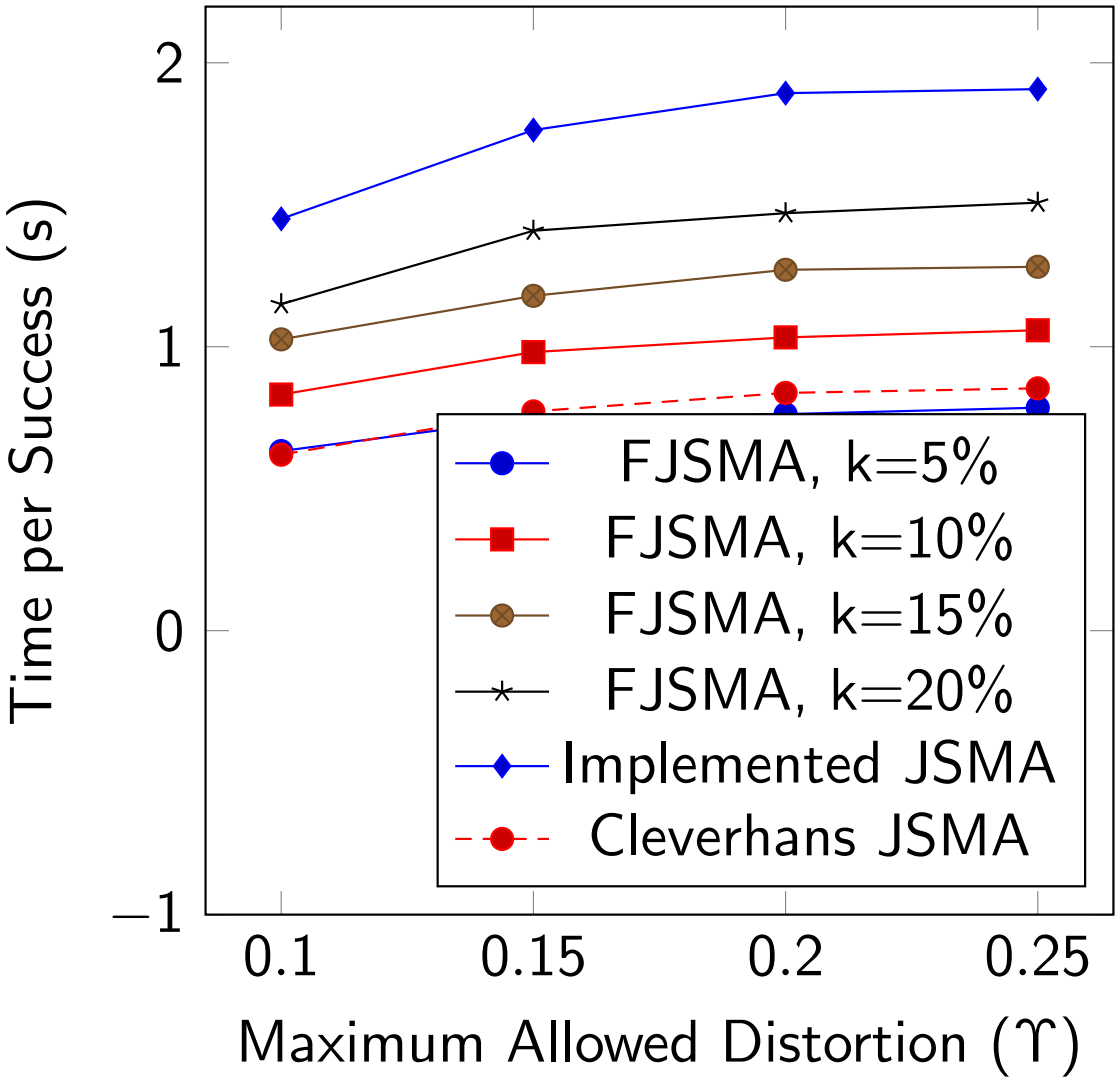
- Varied Υ (maximum L^0 difference between x and x')
- Evasion Rate = (# Successful) / (# Attempted)
- Compared evasion rate and time of three implementations
  - `cleverhans` JSMA
  - JSMA (reimplementation in similar fashion to FJSMA)
  - FJSMA

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` has some lower-level `numpy` calls
- Our JSMA and FJSMA do *not* use these calls
- FJSMA is slower than `cleverhans`, but I will soon utilize similar `numpy` calls to improve speed

Adversarial Playground: The Webapp

Adversarial Playground

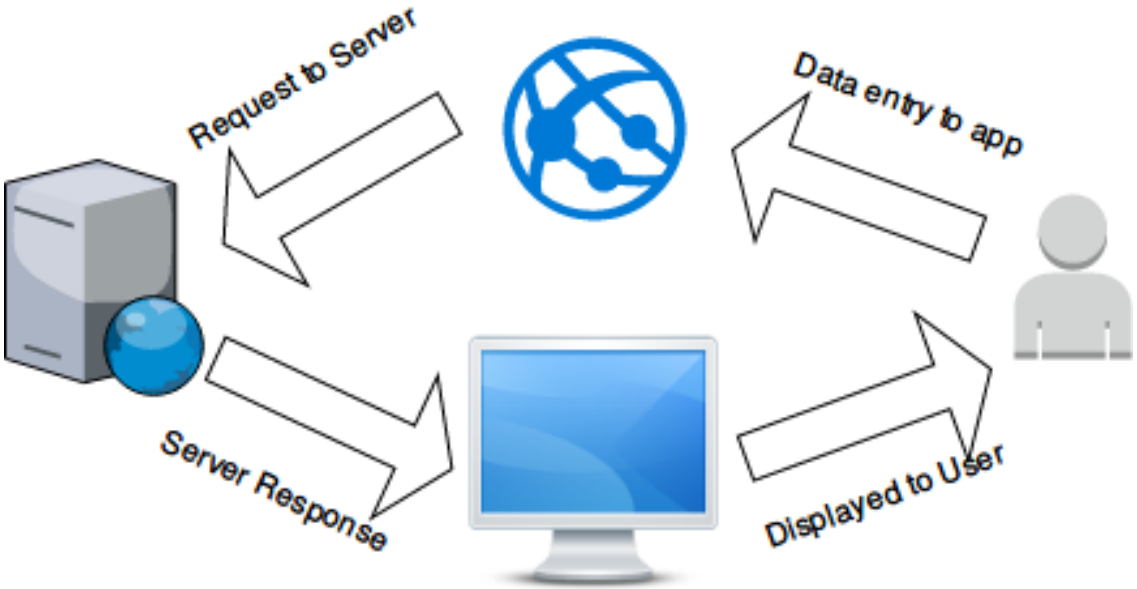
Goal: A comparable webapp to *TensorFlow Playground* for adversarial machine learning.

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

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 and 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

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

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015. URL `http://arxiv.org/abs/1511.07528`.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL `http://arxiv.org/abs/1312.6199`.

Bibliography II

J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015. URL `http://arxiv.org/abs/1506.06579`.