Black-Box Adversarial Samples

Henil Vedant – MS in Cybersecurity
495670888

***Abstract*—** Adversarial examples are inputs that are created or moderated or altered in such a way that the purpose of confusing the neural network is obtained. These inputs are malicious and are not detectable by the naked eye but to the neural network, the consequences of such an attack can be catastrophic. There are several ways to implement these types of adversarial samples and each way has its own perks and cons. There are multiple approaches towards creation of adversarial samples and while all the approaches suggest minimizing the distance between the examples and in some way work towards reducing the cost function with respect to its exponential, it is very important to understand the type of noise injected and the type of defense mechanism that are already applied, this would help us train our CNN better in fewer number of epochs which ultimately means better adversarial sample. Less number of epochs can loosely be linked to image being poisoned more effectively. There are various existing ways of conducting black-box attacks but in the recent year's knowledge distillation and other counter measures have come up which try to increase the number of epochs it takes to create the adversarial sample**.**

***Keywords*—adversarial, FGSM, Perturbations, Black Box, White Box.**

I. Introduction

ML algorithms have an implicit nature which causes bias in the algorithms that leads to a vulnerability that compromises the security of system. Adversarial attacks are a machine learning approach that aims to trick machine learning models by providing deceptive input. The generation and detection of adversarial examples, which are inputs are often specially created to deceive classifiers. A white box attack is a scenario where the attacker has complete access to the target model, including the model's architecture and its parameters.[1][2] A black box attack is a scenario where an attacker has no access to the model and can only observe the outputs of the targeted model. An adversarial example or sample is an instance with a small intentional feature perturbations done in such a way that it leads to certain changes in the way the neural network analyzes the image. These changes can be in the form of adding noise or removing noise, the goal of that is to confuse the neural network in such a way that the image that belongs to 'X' where X represents the class vector is now categorized and labelled as 'Y' where Y is a different class label. Before we dig deep into adversarial examples, the question we answer is why we are interested in such samples and what is the goal that we are trying to achieve by injecting noise in our images in form of perturbations. Some of the recent studies have shown how a self-driving car crashes into another car because it ignores the stop sign. We have seen countless instances where AI and Neural Networks are now actively doing monitoring and image processing for sensitive and critical nature applications, adversarial attacks for such systems may harm our system and cause accuracy, performance and other issues which may degrade our quality of output.[2][4] Another example would be a spam mail fails to classify an email as a spam, the spam mail has been designed to resemble a normal email, but the goal here is to fool and cheat the system. A machine learning powered scanner which essentially scans for arms and weapons on Airport must be efficient and cannot be ringing false alarms every time a suitcase is processed or passed for screening. An increased number of systems using automation as the solution combined with inductance of image processing and computer vision in our day-to-day life, I was motivated to study and understand the adversarial sample generation and their affect on ML systems.

II. Problem Statement

Misclassification of images can mean that the ML System that was trained and tested to perform has not been successful in its goal of delivering the output. ML algorithms have an implicit nature which causes bias in the algorithms that leads to a vulnerability that compromises the security of system. Such a system is now compromised and the deployment or existence of such a pipeline would further cause more harm to our system.

III. MAIN CONTRIBUTION (NOVELTY)

The project conducted in the CIS-735 Class differs than previous method in the following way:

A. *Goal of project:*

- A more common and used approach is to use the cleverhans library to create adversarial samples and launch attacks, however using ready library does not let us fine-tune the setting in terms of increasing the accuracy of the attack, reducing the time and launching a more reliable and malicious attack.[1][8]
- F.G.S.M, Fast Gradient Sign Method is the algorithm used by the cleverhans, however, to build on it we intuitively increase the error function by increasing our total loss error and in some way this can be seen as a gradient ascent w.r.t the concave error function.[1][3]
- Depending on the application, I identify further settings under which we can vary our training per epoch to achieve a faster attack.
- This can be seen as an evaluation of various settings and their trade off's: 1) Accuracy vs Number of epochs trained, Utility vs Accuracy, utility here being the quality of noise being injected.
- I further note my observations and describe how different type of noise (ex. Gaussian, LaPlace) perform on our samples.

IV. LITERATURE REVIEW :

Based on the literature review it is possible to study the adversarial samples after dividing them into two categories such as follows:

A. *White Box Adversarial Samples:*

The underlying assumption for white box adversarial samples is that we have complete access to the ML architecture of our target model, this lets us create our samples specifically as per our model requirments. This can help us in creating very lethal mallicious attack examples that can be missclassified with high accuracy and fool the system.[1][2] A white box attack is a scenario where the attacker has complete access to the target model, including the model's architecture and its parameters, recent studies also show how if we have access to the training dataset we can infer various membership attacks on our system and reveal sensitive information about the data used to train and that can be used to reverse-engineer a more lethal attack, however, in real life scenario it is highly improbable or unlikely the attacker would have access to all the parameters and therefore this type of setting is only studied for understanding our system and model better.[2][4].

B. *Black Box Adversarial Samples:*

The underlying assumption for black box adversarial samples is more practical in real life scenarios and would be the basic assumption under which we will be conducting our experiments. In this scenario the attacker knows very little to nothing about the Machine Learning Model it plans to attack, and A black box attack is a scenario where an attacker has no access to the model and can only observe the outputs of the targeted mode.[2]

Poisoning Attacks:

The attacker influences the training data or its labels to cause the model to underperform during deployment. Hence, Poisoning is essentially adversarial contamination of training data. malicious users inject fake training data with the aim of corrupting the learned model[2][3]. To understand the Poisoning Attacks let us consider a scenario, the training data that was used to train the ML model is accessible to the attacker, he modifies the data in such a way that the ML model now classifies this previously correctly classified data as False class now and inserts it to the other side of the decision boundary, since the changes here are made at the training data, what it means is that the attacker is successful in training the system to misclassify all such examples of that label to a different label, this means that the result we were expecting is not delivered by the system but in fact we have a different expected output, however for the mL model it would still assume that it is categorizing and delivering the correct output, that is due to the implicit nature of neural networks and how they classify images. It is also called as adversarial contamination of training data. In **data poisoning attacks**, adversaries try to manipulate training data in an attempt [4].

- To decrease the overall performance (i.e., accuracy) of an ML model, to induce misclassification to a specific test sample or a subset of the test sample, or to increase training time.

- If an attacker has a specified target label to which a specific test sample is misclassified, the attack is called a targeted data poisoning attack; otherwise, it is called an untargeted data poisoning attack. Adversaries are assumed to either be able to contribute to the training data or have control over the training data.

Evasion Attacks:

Evasion attacks are the most prevalent and most researched types of attacks. The attacker manipulates the data during deployment to deceive previously trained classifiers. Since they are performed during the deployment phase, they are the most practical types of attacks and the most used attacks on intrusion and malware scenarios. Model stealing or model extraction involves an attacker probing a black box machine learning system in order to either reconstruct the model or extract the data it was trained on.[3][4].

This is especially significant when either the training data or the model itself is sensitive and confidential.

Model extraction attacks can be used, for instance, to steal a stock market prediction model, which the adversary could use for their own financial benefit.

Types of algorithms used:

L-BGFS:

Adversarial examples are often generated based on either first-order gradient information (such as the FGSM and DeepFool attack or gradient approximations. We first discuss some classical gradient-based attacks. In this case, the goal of an attacker is either to minimize the norm of the added perturbation that is needed to cause misclassification or to maximize the loss function model f with respect to an input data.[5][10].

It can be defined:

as subject to the constraints, $C(x') = t$ and $x' \in [0, 1]n$, where ‖ . ‖p is the Lp -norm. Finding a precise solution to this problem is very challenging since the constraint $C(x') = t$ is non-linear. Szegedy et al. replaced the

constraint $C(x') = t$ with the continuous loss function L, such as the cross-entropy for classification. That is, they solved the following optimization problem instead :

min $c \parallel x' - x \parallel p + L(x', t)$,

Fast Gradient Sign Method (F.G.S.M):

Unlike the L-BFGS attack, the fast gradient sign method (FGSM) proposed by Goodfellow et al. [6] focuses on finding an adversarial perturbation limited by $L\infty$-norm efficiently rather than producing an optimal adversarial example. In the training of an ML model, a given loss function is minimized to find an optimal parameter set Θ for a classifier C so that C classifies most training data correctly. In contrast, FGSM maximizes the loss function as it tries to make the classifier perform poorly on the adversarial example x'. Therefore, the FGSM perturbed image for an untargeted attack is constructed by solving the following maximization problem.[5][10].

ALGORITHM USED for PROJECT:

GRADIENT ASCENT- It can be seen as F.G.S.M under some relaxed conditions, we move towards the cost function in gradient ascent as opposed to moving away from cost function in F.G.S.M , this helps us increase our error function and at the same time create our adversarial sample with lower number of perturbations.[10]

- In multiclass logistic regression the probability that the model outputs yn yn of NN inputs xnxn matches their targets tntn is given by:

  $p(t|y(x,w))=\prod n=1N\prod k=1Kytn,kn,k$p(t|y(x,w))=∏n=1N∏k=1Kyn,ktn,k

  We assumed that inputs and class memberships are independent and identically distributed. The target tn/tn of one input xn/xn is a vector with K elements following one-hot-encoding (the true label class is 1, all others are 0).

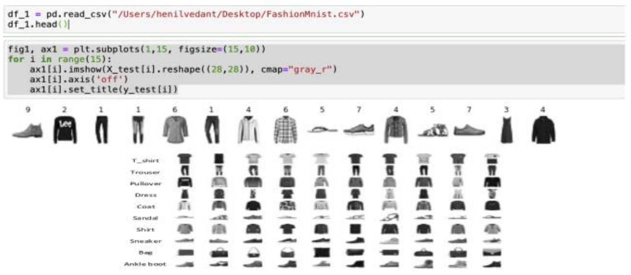
Maximizing the probability of matches above is also called the maximum likelihood approach.

Each class in a multiclass logistic regression has its own weight vector and inputs are passed with weights through the soft-max function to obtain the model output:

$yn,k=\exp(wTkxn)$ $\sum Kc=1\exp(wTcxn)$yn,k=ex(wkTxn)∑c=1 Kexp(wcTxn)

V. Experimental Setup and Results :

1. We will be using F-MNIST DATASET, it is a dataset of various clothing styles and shoes and can be seen as follows:

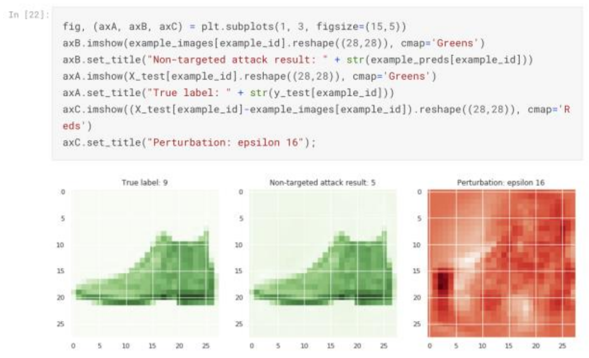


```
df_1 = pd.read_csv("/Users/henilvedant/Desktop/FashionMnist.csv")
df_1.head()
```

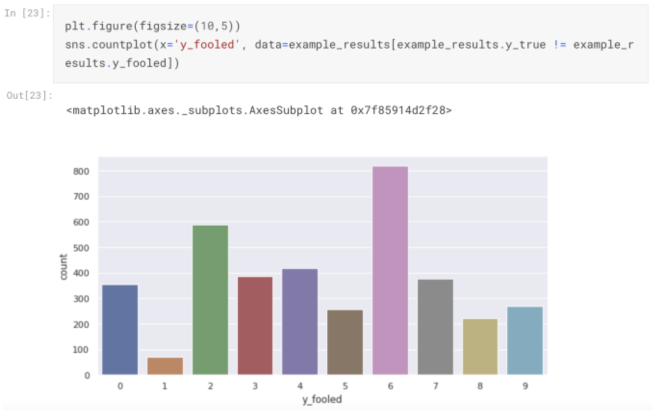
```
fig1, ax1 = plt.subplots(1,15, figsize=(15,10))
for i in range(15):
    ax1[i].imshow(X_test[i].reshape((28,28)), cmap="gray_r")
    ax1[i].axis('off')
    ax1[i].set_title(y_test[i])
```

RESULTS:

Here we can see how we fool the CNN to misclassify the target image between its true label and false label.



```
In [22]:
fig, (axA, axB, axC) = plt.subplots(1, 3, figsize=(15,5))
axB.imshow(example_images[example_id].reshape((28,28)), cmap='Greens')
axB.set_title("Non-targeted attack result: " + str(example_preds[example_id]))
axA.imshow(X_test[example_id].reshape((28,28)), cmap='Greens')
axA.set_title("True label: " + str(y_test[example_id]))
axC.imshow((X_test[example_id]-example_images[example_id]).reshape((28,28)), cmap='R
eds')
axC.set_title("Perturbation: epsilon 16");
```



```
In [23]:
plt.figure(figsize=(10,5))
sns.countplot(x='y_fooled', data=example_results[example_results.y_true != example_r
esults.y_fooled])

Out[23]:
<matplotlib.axes._subplots.AxesSubplot at 0x7f85914d2f28>
```

Based on these I note certain observations:

1) Images have natural and non-natural fooling targets; I further establish a more formal definition

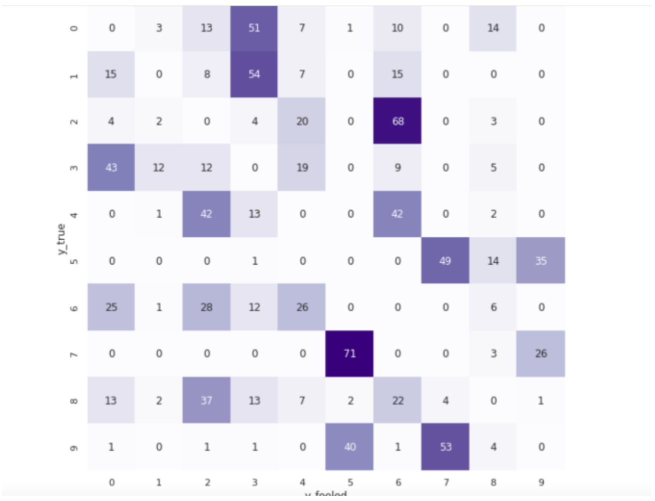
for the notion of natural and non-natural fooling targets.

2) Injecting random noise is of no use, the noise injected must be mathematical noise calculated to misclassify the boundaries.
3) Number of epochs and the accuracy to fool the target can be established.

Natural and non-natural fooling targets:

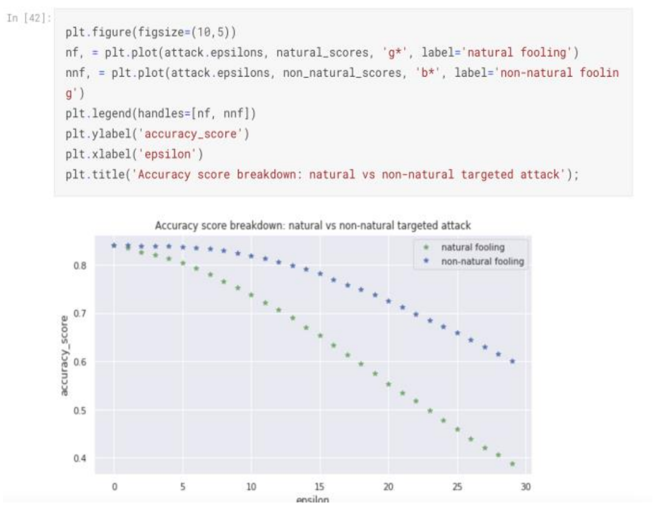
The idea of natural and non-natural fooling targets comes from the observation that under certain images at certain epochs while analyzing the image at certain points we see the neural network almost always fails impeccably.

We implement a covariance matrix to understand the target class and its natural and non-natural fooling targets and show the observation.



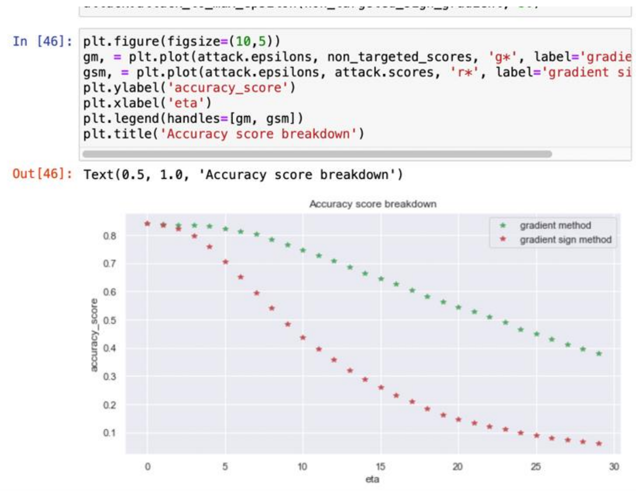
y_true \ y_fooled	0	1	2	3	4	5	6	7	8	9
0	0	3	13	51	7	1	10	0	14	0
1	15	0	8	54	7	0	15	0	0	0
2	4	2	0	4	20	0	68	0	3	0
3	43	12	12	0	19	0	9	0	5	0
4	0	1	42	13	0	0	42	0	2	0
5	0	0	0	1	0	0	0	49	14	35
6	25	1	28	12	26	0	0	0	6	0
7	0	0	0	0	0	71	0	0	3	26
8	13	2	37	13	7	2	22	4	0	1
9	1	0	1	1	0	40	1	53	4	0

In the next figure we plot a breakdown of accuracy w.r.t epsilon:



```
In [42]:
plt.figure(figsize=(10,5))
nf, = plt.plot(attack.epsilons, natural_scores, 'g*', label='natural fooling')
nnf, = plt.plot(attack.epsilons, non_natural_scores, 'b*', label='non-natural foolin
g')
plt.legend(handles=[nf, nnf])
plt.ylabel('accuracy_score')
plt.xlabel('epsilon')
plt.title('Accuracy score breakdown: natural vs non-natural targeted attack');
```

We see as the epsilon increases; we start suffering a significant loss in our accuracy thus noting their direct relation.



```
In [46]: plt.figure(figsize=(10,5))
         gm, = plt.plot(attack.epsilons, non_targeted_scores, 'g*', label='gradie
         gsm, = plt.plot(attack.epsilons, attack.scores, 'r*', label='gradient si
         plt.ylabel('accuracy_score')
         plt.xlabel('eta')
         plt.legend(handles=[gm, gsm])
         plt.title('Accuracy score breakdown')

Out[46]: Text(0.5, 1.0, 'Accuracy score breakdown')
```

Here we can see that the proposed method is faster than G.S.M for our trained samples.

VI. CHALLENGES

1. The goal of this project can be divided into 2 parts:
   a) We generate adversarial samples to trick the classifier into falsely classifying with confidence,
   b) We specifically train the classifier in such a way that it gets fooled with a class instance that we want.

2. Intuitively the targeted attack is for specifically targeting an instance class and

non-targeted is a bit more relaxed version of that constraint.

3. The above discrimination can help us for future works in building a robust system that is safe against adversarial attacks.
4. Jacobian distillation is one of the approaches that can intuitively provide defense against LaPlace and gaussian noise for adversarial samples.

5. Fooling takes place in regions where the model fails to draw good decision boundaries which of course depends on the model architecture/flexibility but on the input data quality and preprocessing as well.
6. Since some inputs are closer to each other in meanings of decision boundaries, there exist natural and non-natural fooling targets References.

VII. REFERENCES:

[1] Adversarial Attacks and Defences: A Survey - Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, Debdeep Mukhopadhyay, https://arxiv.org/abs/1810.00069.

[2] Practical Black-Box Attacks against Machine Learning- Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, Ananthram Swami, https://arxiv.org/abs/1602.02697.

[3] Towards Black-box Attacks on Deep Learning Apps, Hongchen Cao, Shuai Li, Yuming Zhou, Ming Fan, Xuejiao Zhao, Yutian Tang, https://arxiv.org/abs/2107.12732.

[4] 4Adverserial Machine Learning - https://en.wikipedia.org/wiki/Adversarial_machine_learning.

[5] 5 How Adverserial Examples work - https://wiki.davidl.me/view/Adversarial_Examples.

[6] 6- NIPS Completion Track- Google adversarial attack open challenge Kaggle dataset/F-MNIST https://nips.cc/Conferences/2017/CompetitionTrack.

[7] 7- Generative adversarial network, https://en.wikipedia.org/wiki/Generative_adversarial_network.

[8] 8- Black-box Adversarial Sample Generation Based on Differential Evolution, https://arxiv.org/abs/2007.15310.

[9] Black Box Attacks Cleverhans - http://www.cleverhans.io.

[10] Adversarial attacks an empirical survey - https://arxiv.org/pdf/2112.02797.pdf.