50.039 Theory and Practice of Deep Learning

W5-S2 Introduction to Attacks and Defense on Neural Networks

Matthieu De Mari

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

About this week (Week 5)

1. What are **attacks** on Neural Networks (NNs)?
2. Why are attacks an **important concept** when studying NNs?
3. What are the different **types of attacks** and what is the intuition behind basic attacks?
4. How to **defend** against such attacks?

5. **State-of-the-art** of attacks and defense, **open questions** in research.

About this week (Week 5)

1. What are **attacks** on Neural Networks (NNs)?
2. Why are attacks an **important concept** when studying NNs?
3. What are the different **types of attacks** and what is the intuition behind basic attacks?
4. How to **defend** against such attacks?
5. **State-of-the-art** of attacks and defense, **open questions** in research.

In the last episode

Noising samples is sometimes good enough to produce adversarial samples and make a trained Neural Network malfunction.

- This exploits the intrinsic properties/limits of Neural Networks.
- The problem, however, is that noising is **too random**, and is often not guaranteed to work.

Can we implement **more advanced attacks**, with higher **success rates**?

- Can we “**target**” these attacks to produce adversarial samples with expected effects on Neural Networks?
- And later, can we **defend** against these attacks?

Some more taxonomy on attacks

Definition (untargeted attack):

The objective of an **untargeted attack** is to produce an attack sample, which will simply be misclassified.

Noising was an **untargeted attack**, as we attempted to modify a sample in such a way that it would classified as anything but its ground truth label.

Definition (targeted attack):

The objective of a **targeted attack** is to produce an attack sample, which will be misclassified as a specific class.

As such, **targeted attacks** are often **more complex** than **untargeted ones**.

E.g., modify a picture of a **dog (original label)** so it is misclassified as a **cat (target label)**.

Some more taxonomy on attacks

Definition (black-box attack):

A **black-box attack** does not exploit any properties of the model.

Black-box attacks assume that they can **only try inputs and access the outputs of the model under attack**.

Noising was therefore a **black-box** and **untargeted attack**.

Definition (white-box attack):

A **white-box attack** attempts to exploit properties of the model, e.g. its gradients, logits, weights, etc.

White-box attacks therefore assume that the model as a whole can be accessed, including its **weights** and **gradients**.

White-box attacks attempt to **learn** how the model works, to make it malfunction in a certain way.

Some more taxonomy on attacks

Definition (one-shot attack):

A **one-shot attack** attempts to produce a single attack sample, and if this attack fails, it simply retries on a different sample.

Noising was therefore a **one-shot attack**. It attempted to noise a sample to have it misclassified.

However, if this attempt failed, it simply tried on another sample.

Definition (iterated attack):

An **iterated attack** attempts to produce an attack sample, like the one-shot attacks.

However, it will try to **adjust the said sample** until it either

- **makes the model malfunction (in an expected way)**,
- or **reaches a maximal number of allowed iterations**.

The iterated attacks are often more robust and efficient.

About attacks

Adversarial Machine Learning can be very creative and is currently a very active research field. In this lecture, and in the interest of time, we will only cover some of the basic ones.

- What matters is to understand the intuition behind these basic attacks, more specifically how we might use information about the model to tailor our attacks and enhance their efficacy.
- In the next lecture, we will then discuss some more advanced attack techniques, for general knowledge.
- To summarize, keep in mind that the potential for attacks is quite unlimited and researchers have been very creative…!

About attacks

Basic attacks (to be discussed today):

1. **Untargeted**, **one-shot**, **white-box** gradient attack
2. **Untargeted**, **one-shot**, **white-box** fast gradient sign attack
3. **Untargeted**, **iterated**, **white-box** fast gradient sign attack
4. **Targeted**, **one-shot**, **white-box** fast gradient sign attack
5. **Targeted**, **iterated**, **white-box** fast gradient sign attack

About attacks

Basic attacks (to be discussed today):

1. **Untargeted**, **one-shot**, **white-box** **<u>gradient</u>** attack
2. **Untargeted**, **one-shot**, **white-box** fast **<u>gradient</u>** sign attack
3. **Untargeted**, **iterated**, **white-box** fast **<u>gradient</u>** sign attack
4. **Targeted**, **one-shot**, **white-box** fast **<u>gradient</u>** sign attack
5. **Targeted**, **iterated**, **white-box** fast **<u>gradient</u>** sign attack

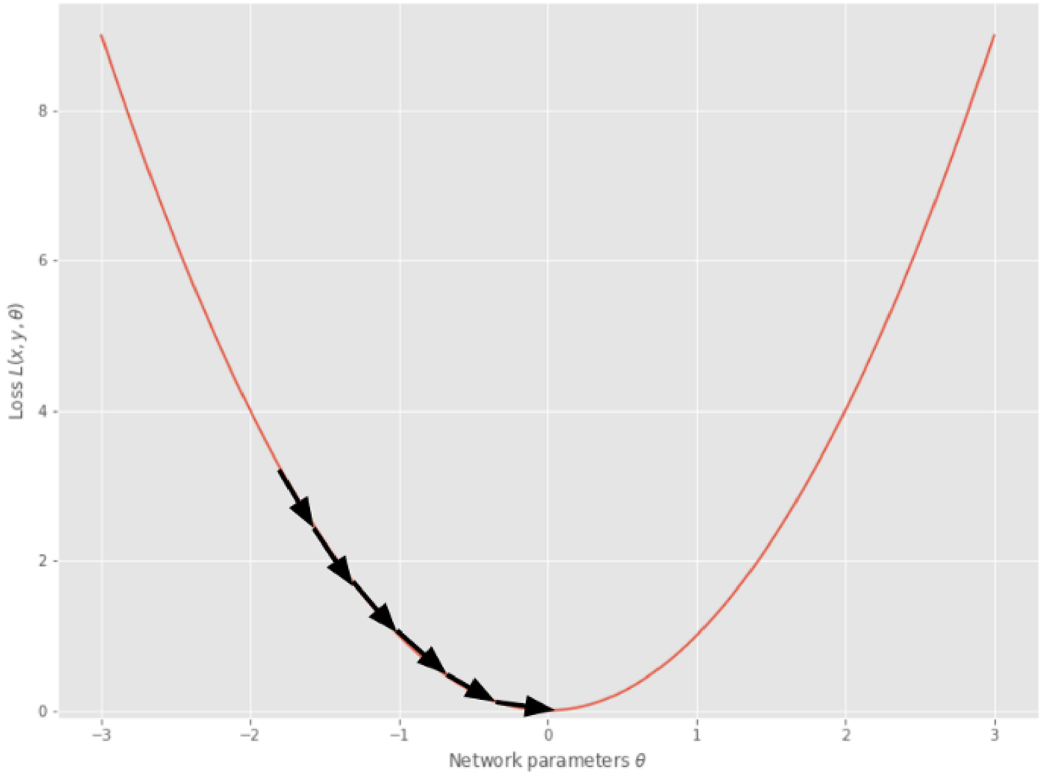
Note: "Gradient" seems to be the important keyword here, but why and how are gradients used for attacks?

A reminder on gradient descent

When **training** a neural network, we attempt to adjust the parameters of a model θ, to minimize a loss function $L(x, \theta, y)$.

- We typically use an optimizer, which implements some version of the **gradient descent** algorithm.

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(x, \theta, y)$$



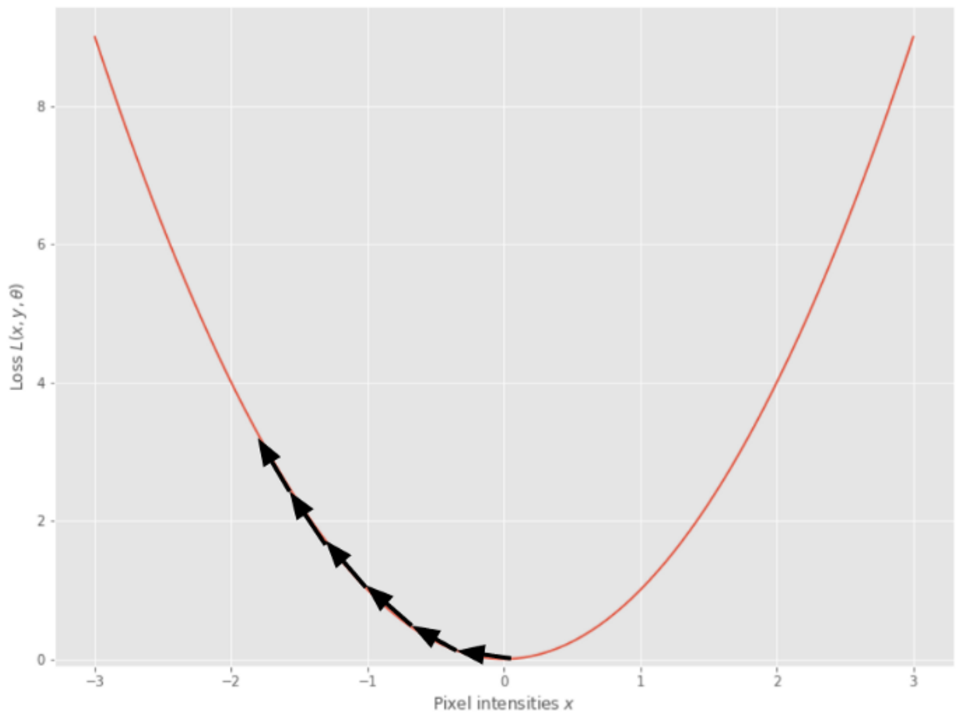
In a sense, gradients tell us how to improve the model performance by adjusting its parameters meaningfully.

Using Gradient Ascent to Attack

A possible approach to "smarter" attacks would then **turn this process on its head**.

- If we held the parameters of the model θ as constants and differentiated the loss with respect to some input sample x,
- We could then modify a sample x and create a new "somewhat similar" sample $\tilde{x}$, in a such a way that the expected loss of the model would increase.

- To do so, we would simply have to use some **gradient ascent** on this sample x.



Using Gradient Ascent to Attack

A possible approach to "smarter" attacks would then **turn this process on its head**.

- If we held the parameters of the model θ as constants and differentiated the loss with respect to some input sample x,
- We could then modify a sample x and create a new "somewhat similar" sample $\tilde{x}$, in a such a way that the expected loss of the model would increase.

- To do so, we would simply have to use some **gradient ascent** on this sample x.

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(x, \theta, y)$$

(Gradient **descent** on **parameters**, a.k.a. **"training"** a model)

Using Gradient Ascent to Attack

A possible approach to "smarter" attacks would then **turn this process on its head**.

- If we held the parameters of the model θ as constants and differentiated the loss with respect to some input sample x,
- We could then modify a sample x and create a new "somewhat similar" sample $\tilde{x}$, in a such a way that the expected loss of the model would increase.

- To do so, we would simply have to use some **gradient ascent** on this sample x.

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(x, \theta, y)$$

(Gradient **descent** on **parameters**, a.k.a. **"training"** a model)

$$\tilde{x} \leftarrow x + \alpha \nabla_x L(x, \theta, y)$$

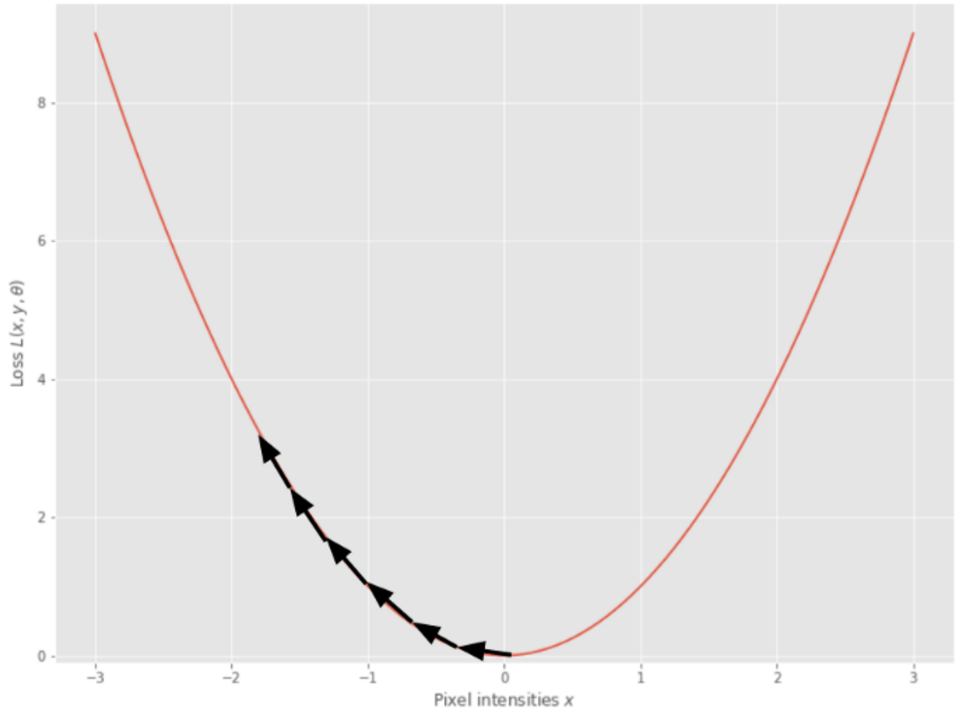
(Gradient **ascent** on a **sample**, a.k.a. "**attacking**" a model)

A note on losses, softmax and gradients

Most gradient-based attacks can operate on the gradients computed from the loss function $L(x, \theta, c)$, for instance, to move away from the original class c.

- To do so, by **increasing the loss** for said sample x and class c, using **gradient ascent** on the loss.

$$\tilde{x} \leftarrow x + \alpha \nabla_x L(x, \theta, c)$$

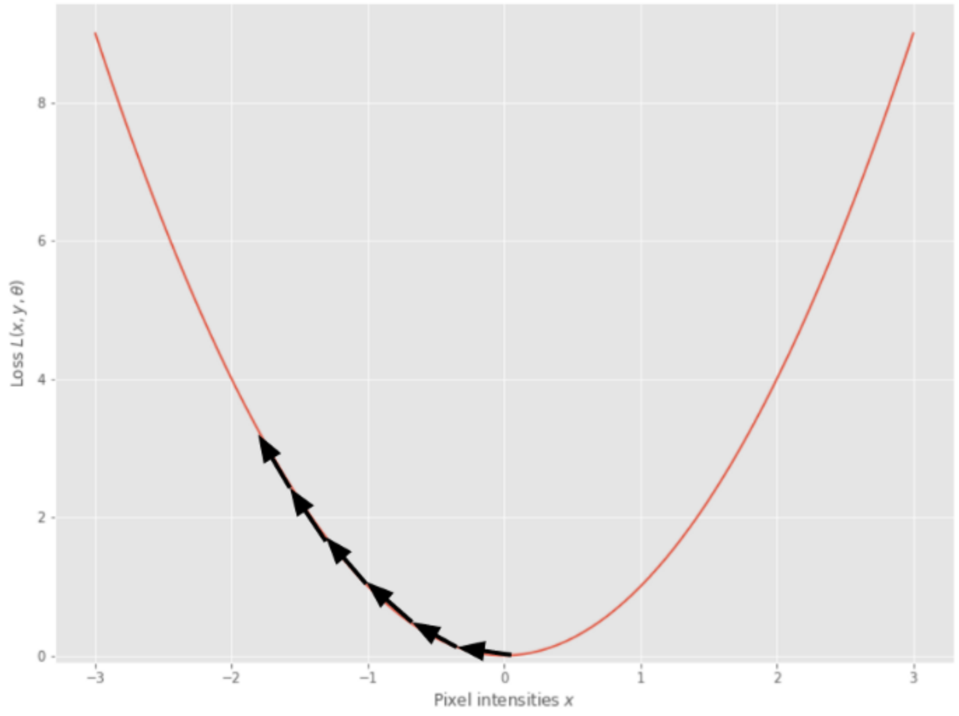


A note on losses, softmax and gradients

Some papers however mention that it would be preferable to use the logits $f_c(x)$ to **minimize the value of these logits (≈ final vector before softmax and argmax decision)**.

- In that case, we would use **gradient descent** to minimize the probability of class c to be chosen!

$$\tilde{x} \leftarrow x - \alpha \nabla_x f_c(x)$$



Note: this second approach (logits) often works better, because of the softmax might end up messing up the gradients sometimes (to be seen later).

Untargeted Gradient Attack

Definition (untargeted gradient attack):

The **untargeted gradient attack** takes a single sample x, of original class $c \in C$ and attempts to produce a sample $\tilde{x}$ of class $\tilde{c} \in C$, with $\tilde{c} \neq c$.

$$c = argmax_{i \in C}\left(f_i(x)\right)$$

And then two options…

Untargeted Gradient Attack (option #1)

Definition (untargeted gradient attack):

The **untargeted gradient attack** takes a single sample x, of original class $c \in C$ and attempts to produce a sample $\tilde{x}$ of class $\tilde{c} \in C$, with $\tilde{c} \neq c$.

$$c = argmax_{i \in C}\left(f_i(x)\right)$$

And then two options…

- **Option #1:** Look for the most probable class $c \in C$ and use gradient **ascent** to move the sample **away from its original class**, with step ϵ.

$$\tilde{x} \leftarrow x + \epsilon \nabla_x L(x, \theta, c)$$

The attack is successful if

$$\tilde{c} = argmax_{i \in C}\left(f_i(\tilde{x})\right) \neq c$$

Untargeted Gradient Attack (option #2, not implemented in notebooks)

Definition (untargeted gradient attack):

The **untargeted gradient attack** takes a single sample x, of original class $c \in C$ and attempts to produce a sample $\tilde{x}$ of class $\tilde{c} \in C$, with $\tilde{c} \neq c$.

$$c = argmax_{i \in C}\left(f_i(x)\right)$$

And then two options...

- **Option #2:** Look for the least probable class $c^* \in C$ and use gradient **descent** to move the sample **in the direction of the least probable class**, with step ϵ.

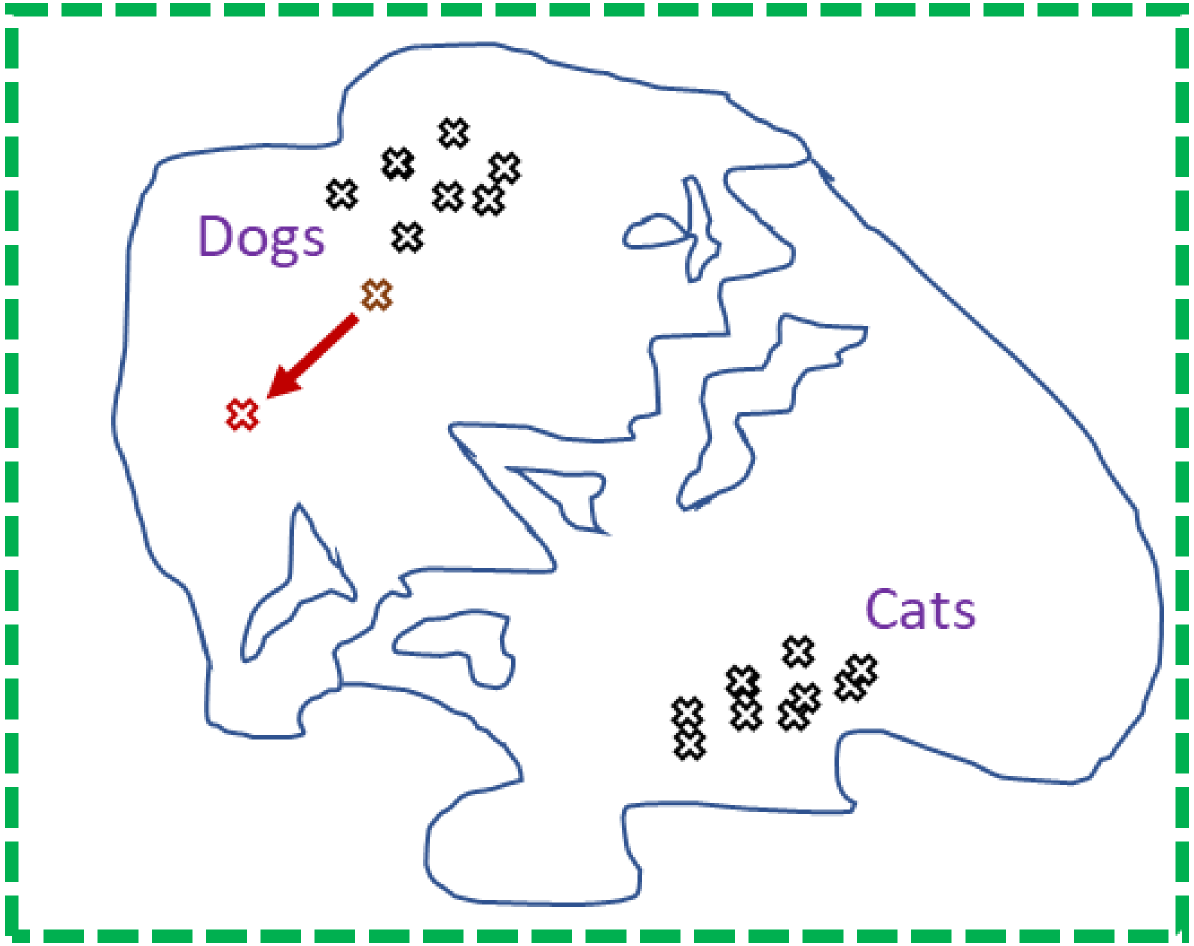
$$c^* = argmin_{i \in C}\left(f_i(x)\right)$$
$$\tilde{x} \leftarrow x - \epsilon \nabla_x L(x, \theta, c^*)$$

The attack is successful if

$$\tilde{c} = argmax_{i \in C}\left(f_i(\tilde{x})\right) \neq c$$

Why gradient attack works better than randomly noising

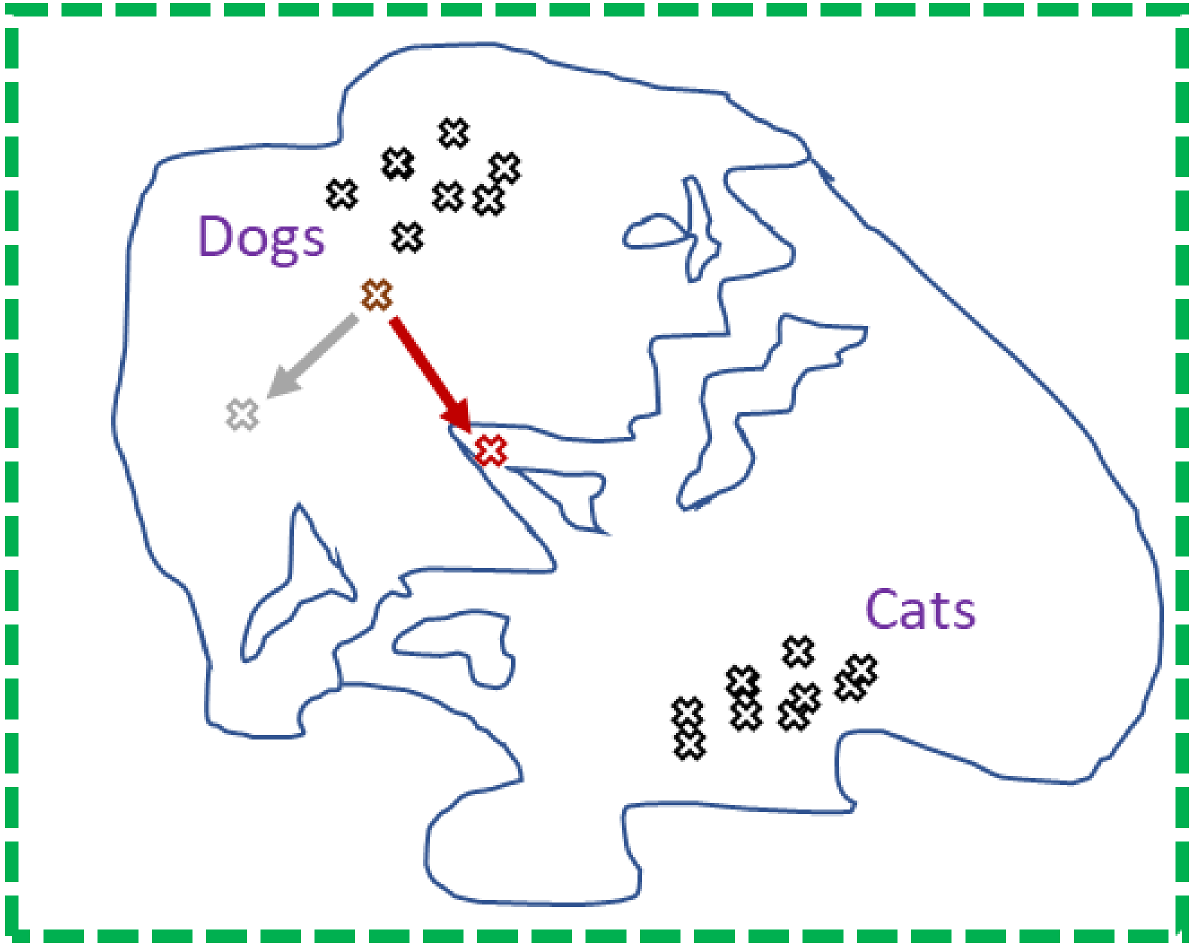
- When randomly noising a **sample** to make an **attack sample**, we move **randomly** in the feature map.



Why gradient attack works better than randomly noising

- When randomly noising a **sample** to make an **attack sample**, we move randomly in the feature map.

- When using gradient attack, we move in **a more meaningful direction**, which might help our **original sample** become **misclassified**.



Untargeted gradient attack code

The **untargeted gradient attack (option #1)** takes a single sample x, of original class $c \in C$ and attempts to produce a sample $\tilde{x}$ of class $\tilde{c} \in C$, with $\tilde{c} \neq c$.

- It uses gradient ascent to move the original sample **away from the most probable class** (i.e. its original one) to generate an attack sample.

```
1  def ugm_attack(image, epsilon, data_grad):
2  
3      # Create the attack image by adjusting
4      # each pixel of the input image
5      eps_image = image + epsilon*data_grad
6  
7      # Clipping eps_image to maintain pixel
8      # values into the [0, 1] range
9      eps_image = torch.clamp(eps_image, 0, 1)
10 
11     # Return
12     return eps_image
```

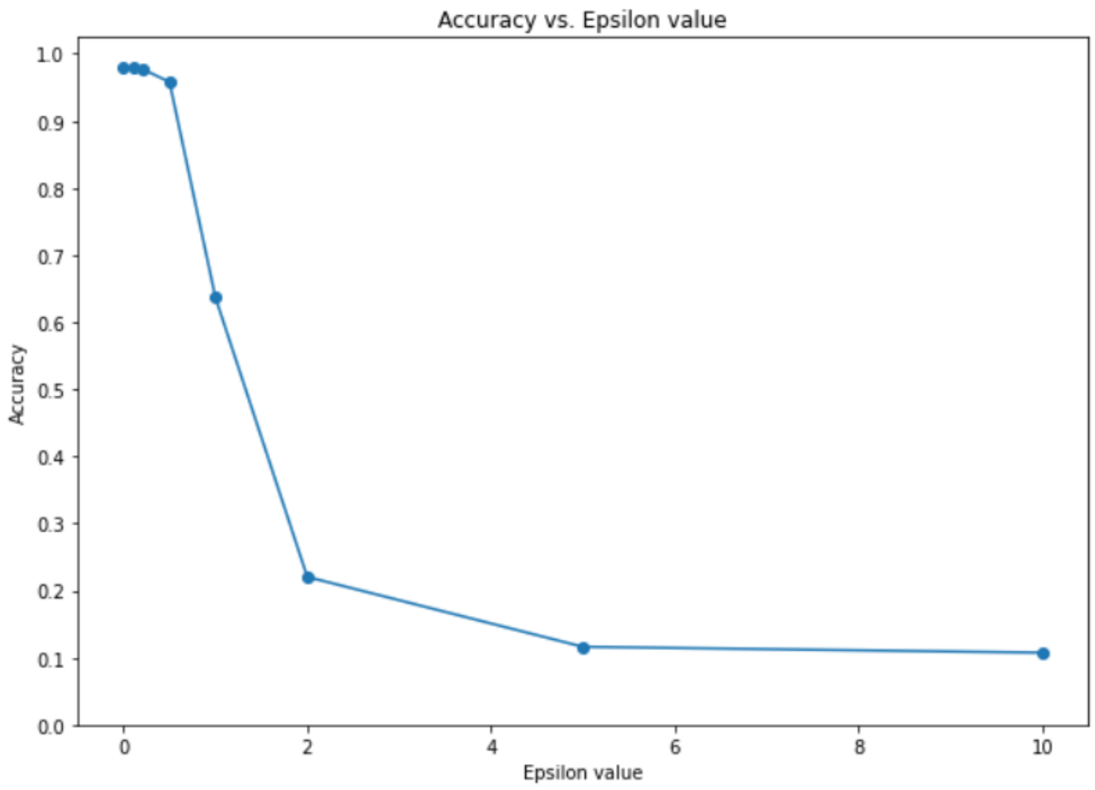
Line 5 easily implements

$$\tilde{x} \leftarrow x + \epsilon \nabla_x L(x, \theta, c)$$

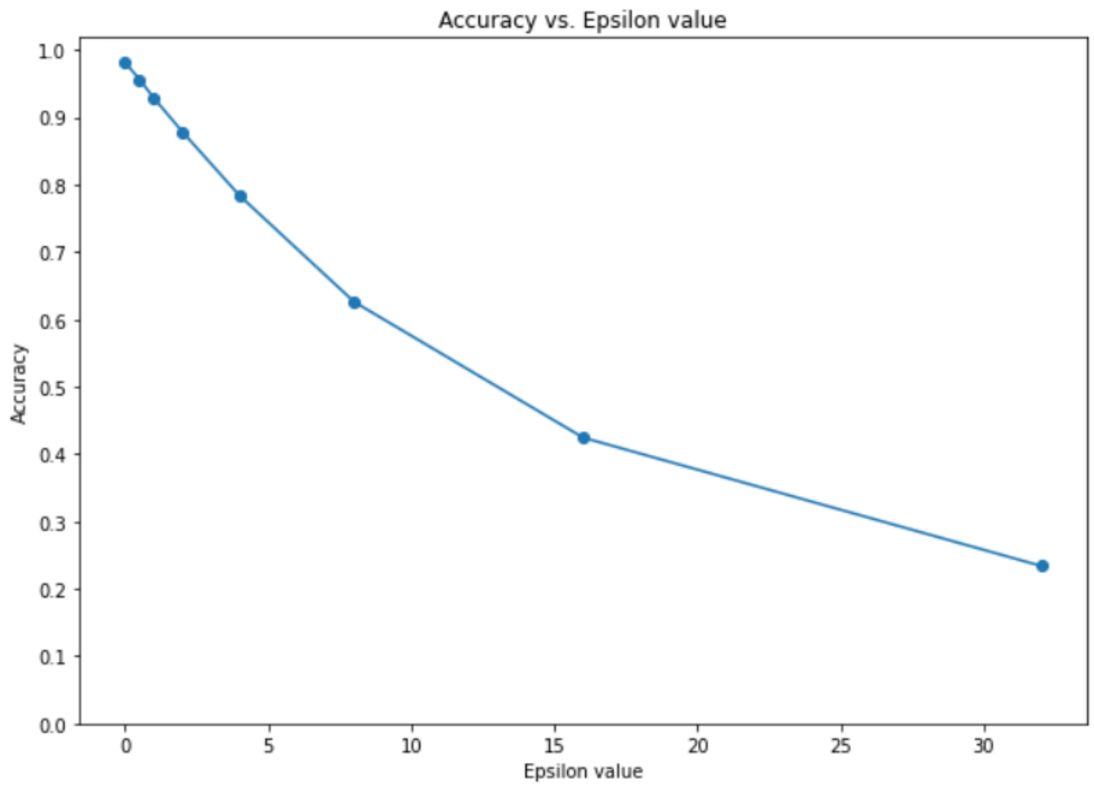
(Taken from notebook 2.)

Untargeted gradient vs. epsilon noising

- Epsilon Noising Method

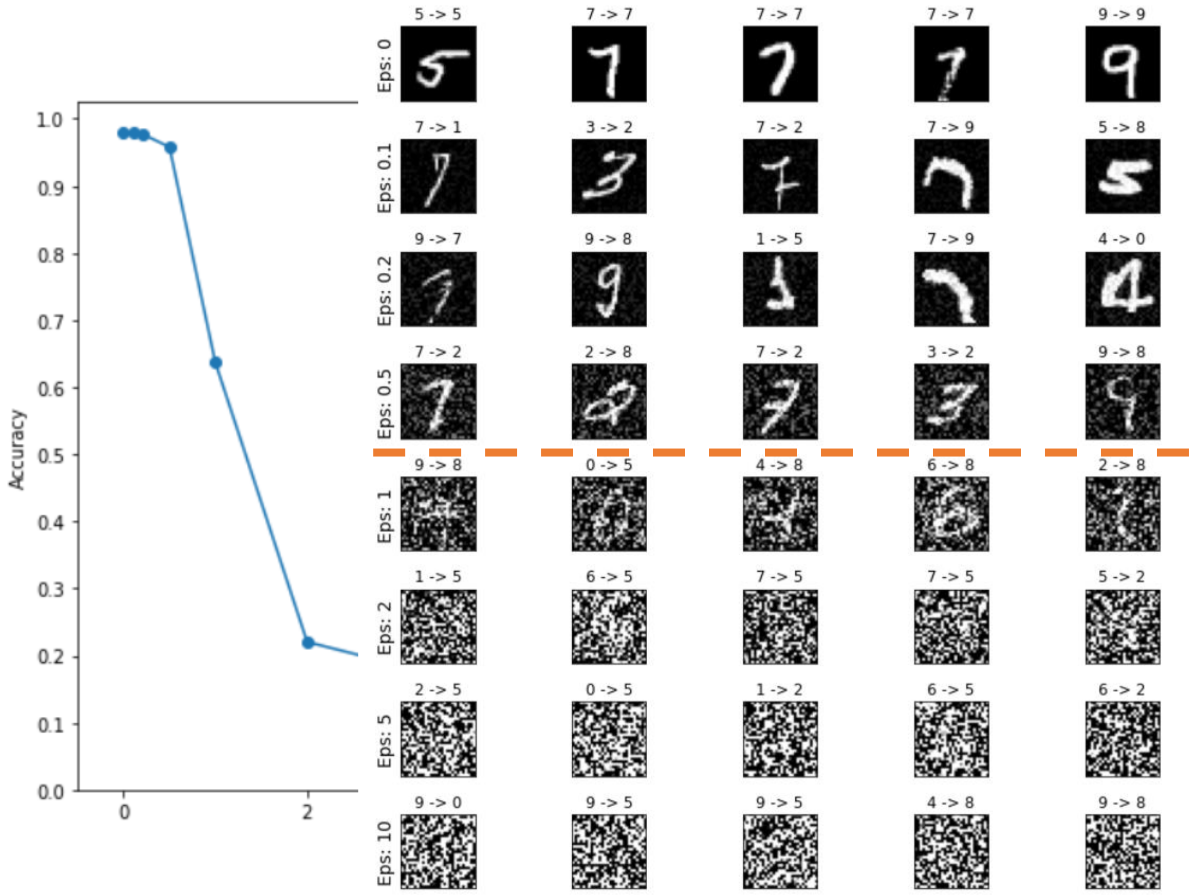


- Untargeted Gradient Method

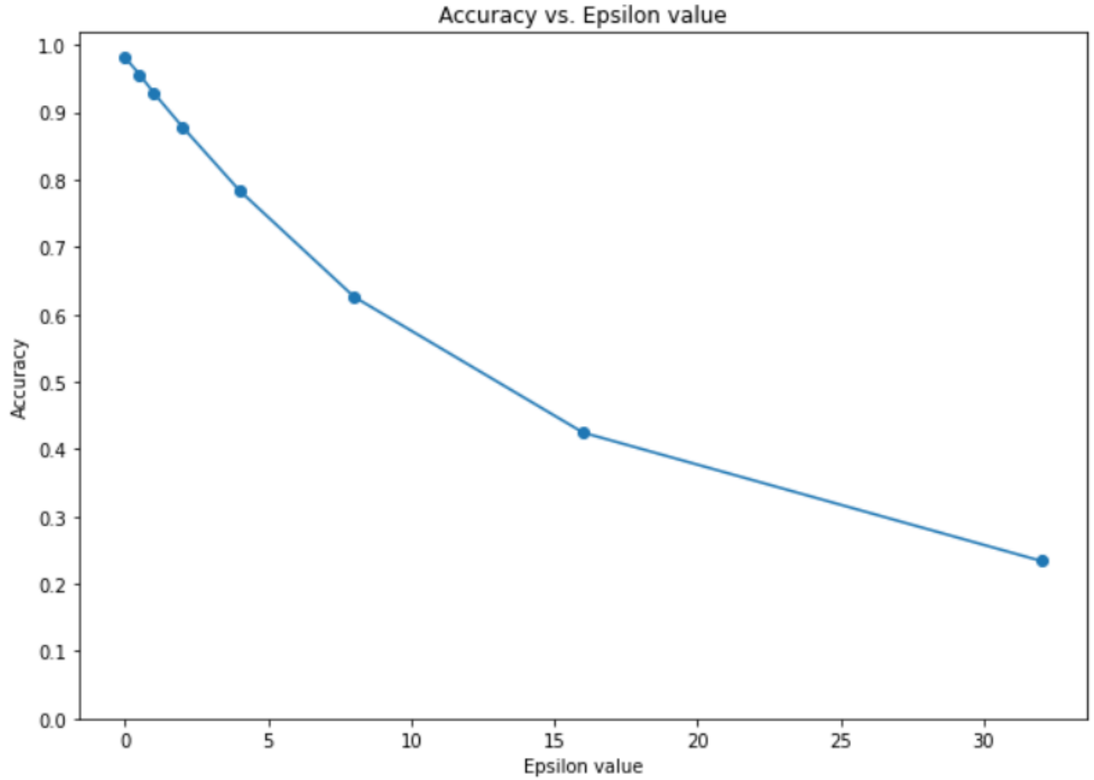


Untargeted gradient vs. epsilon noising

- Epsilon Noising Method

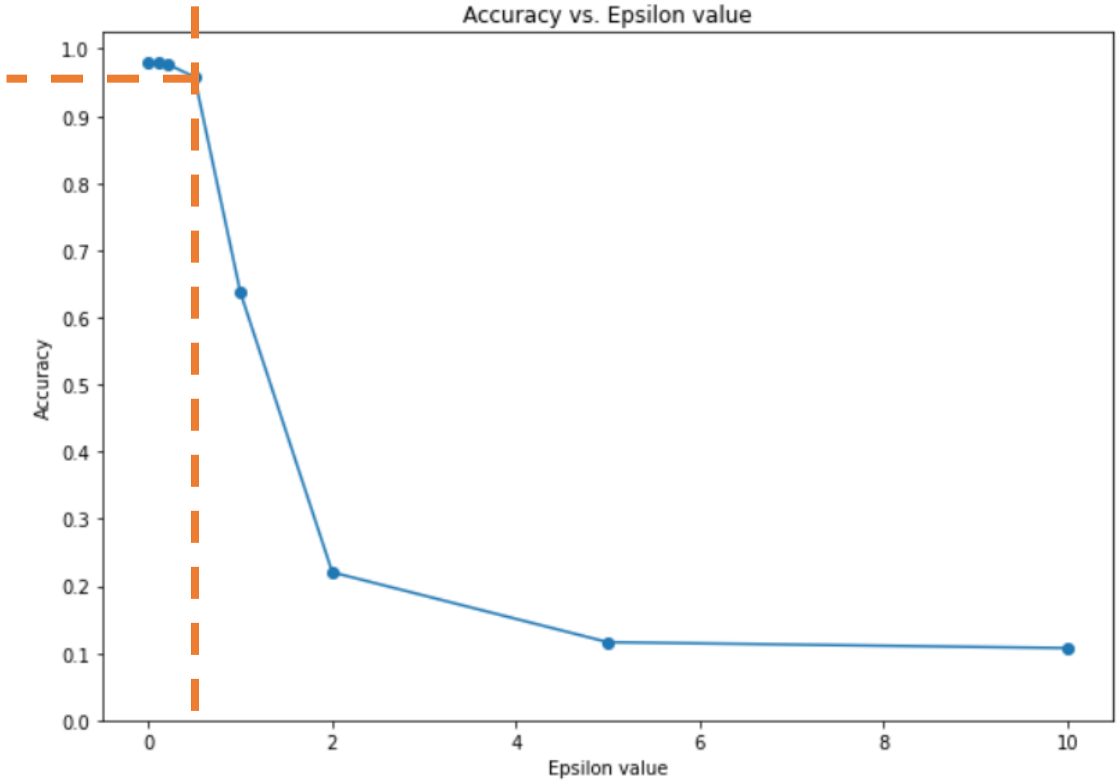


- Untargeted Gradient Method

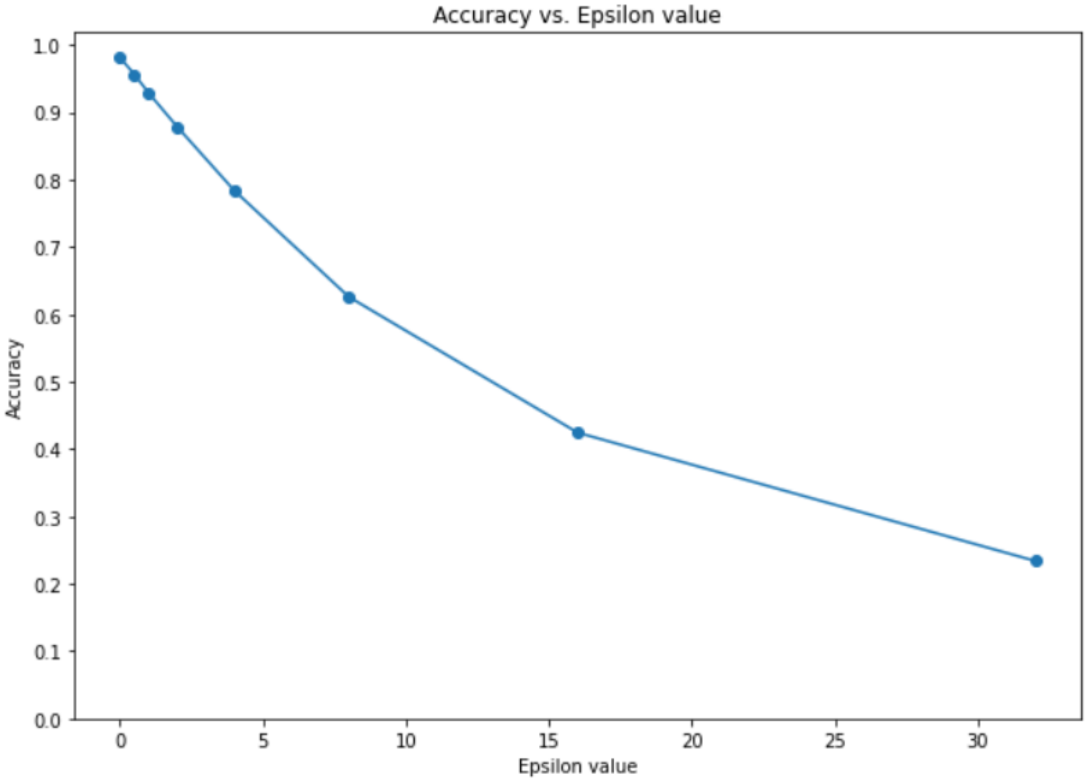


Untargeted gradient vs. epsilon noising

- Epsilon Noising Method

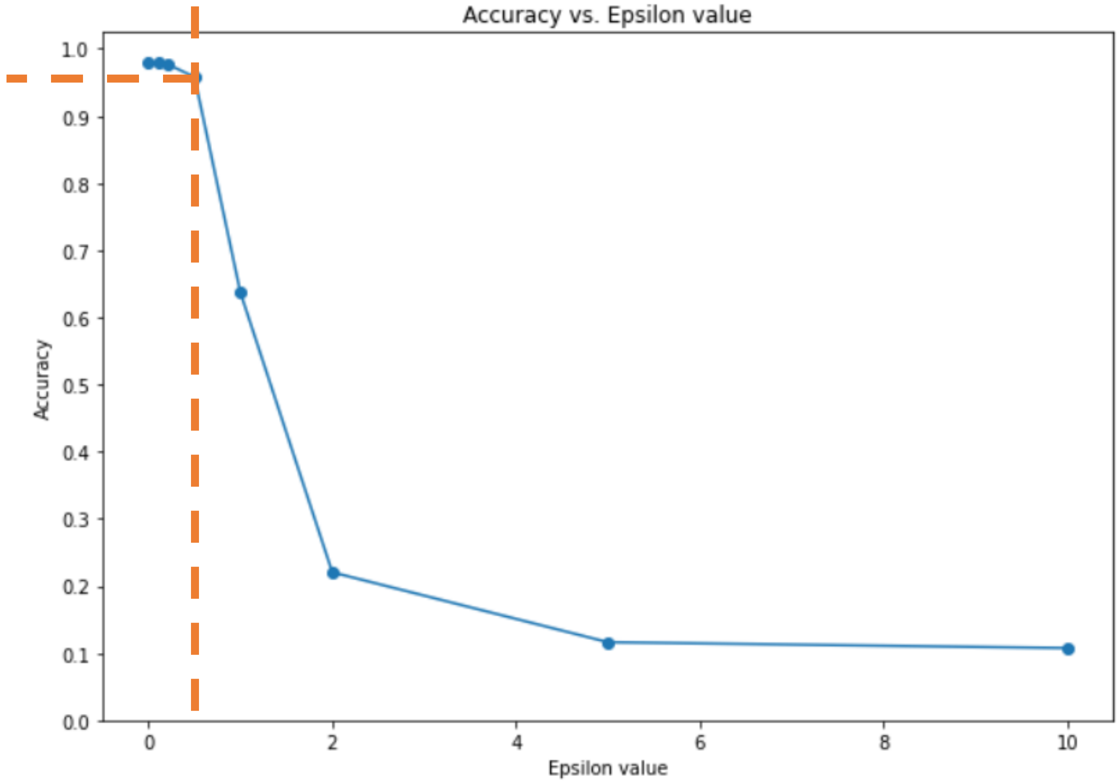


- Untargeted Gradient Method

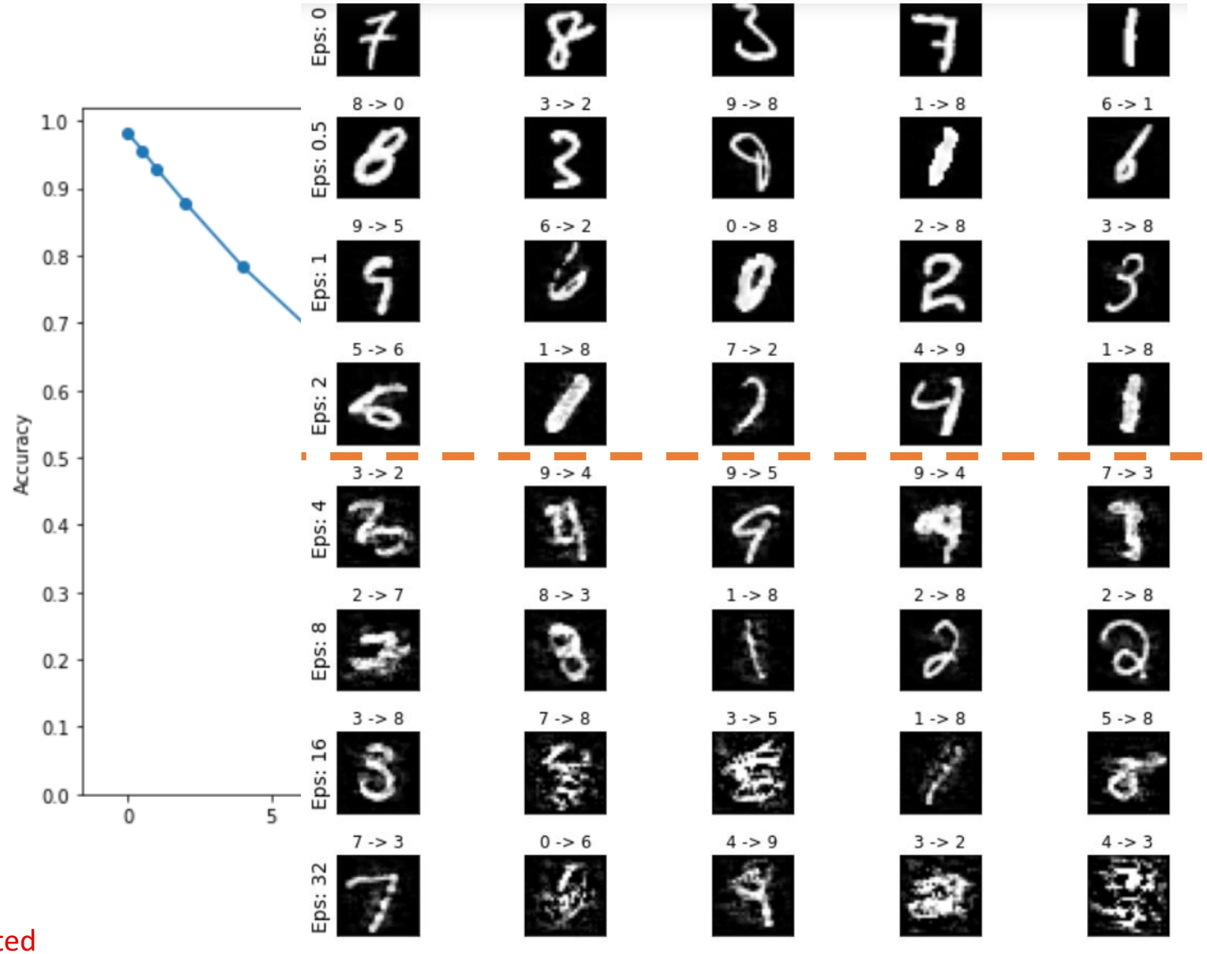


Untargeted gradient vs. epsilon noising

- Epsilon Noising Method

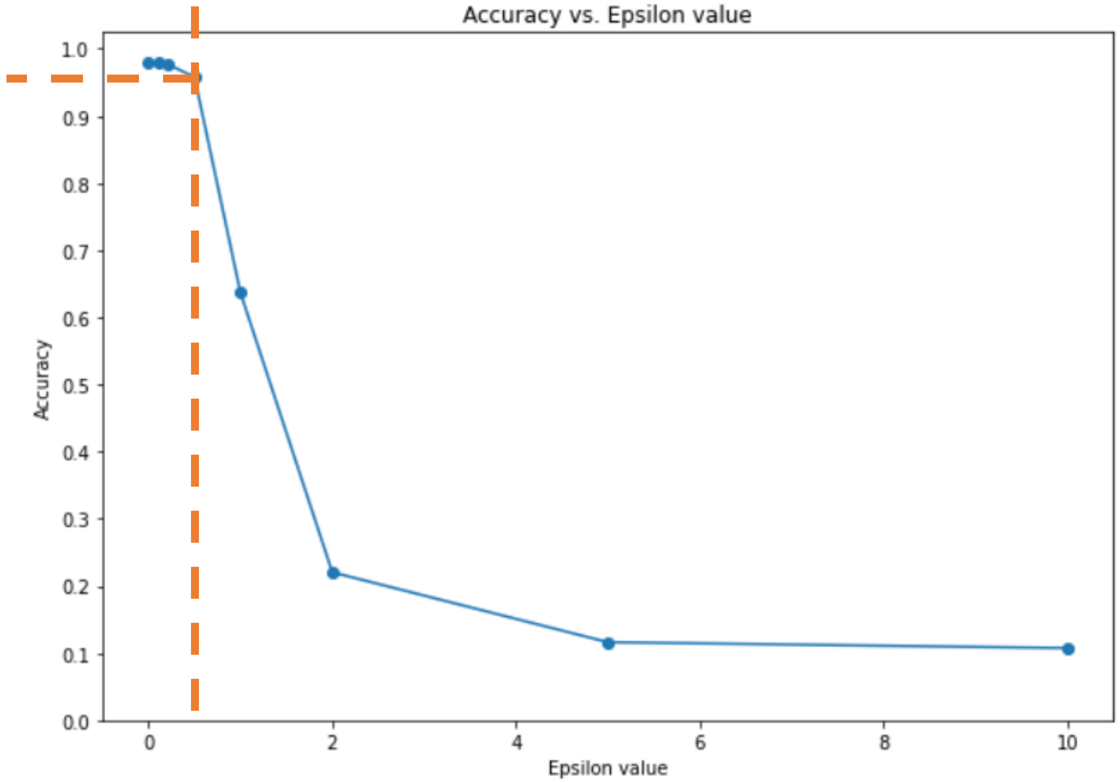


- Untargeted Gradient Method

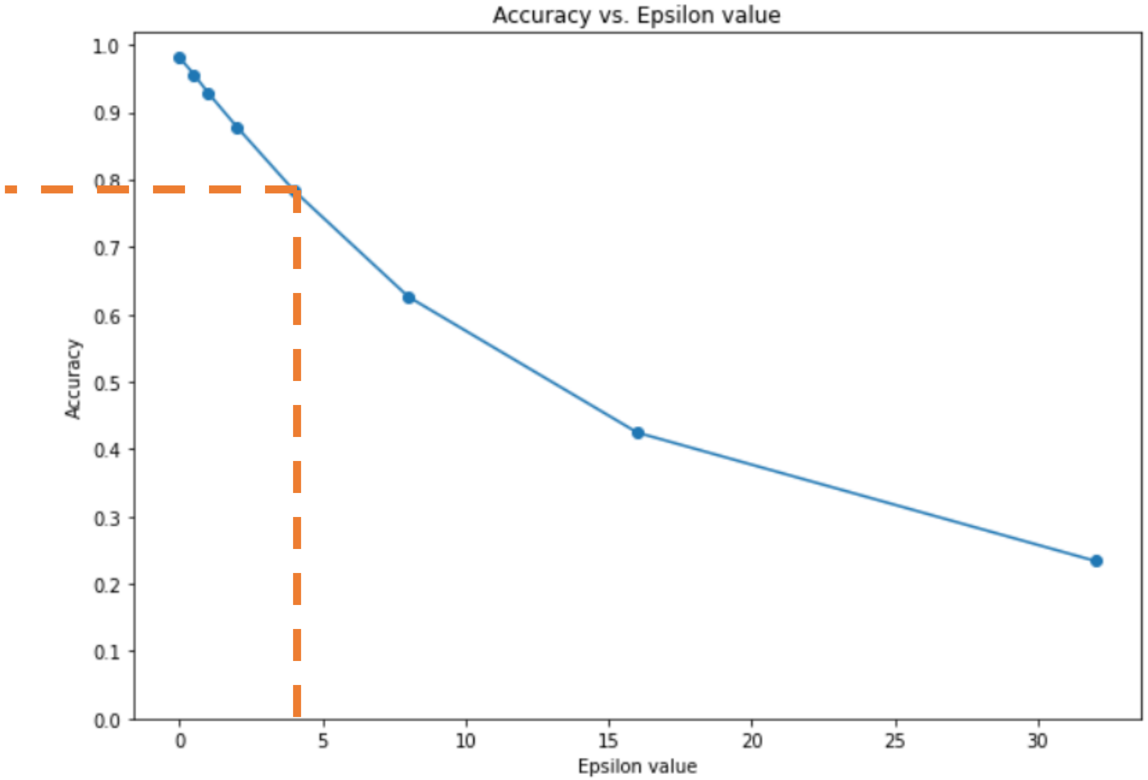


Untargeted gradient vs. epsilon noising

- Epsilon Noising Method

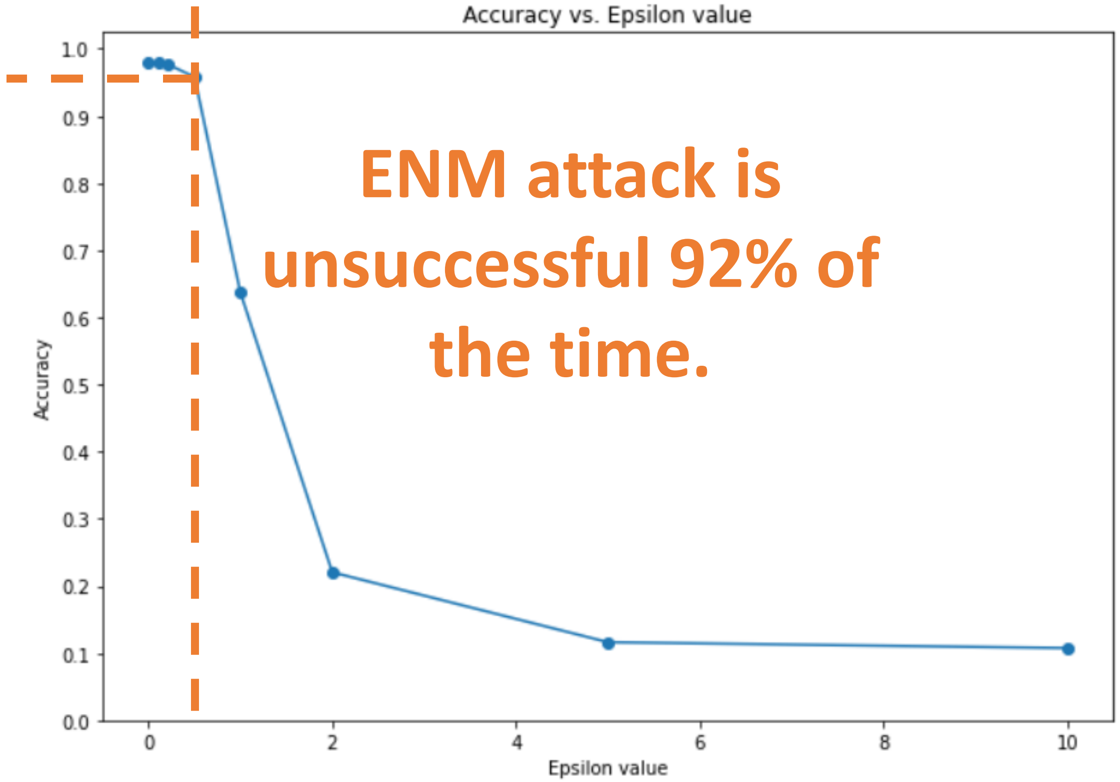


- Untargeted Gradient Method

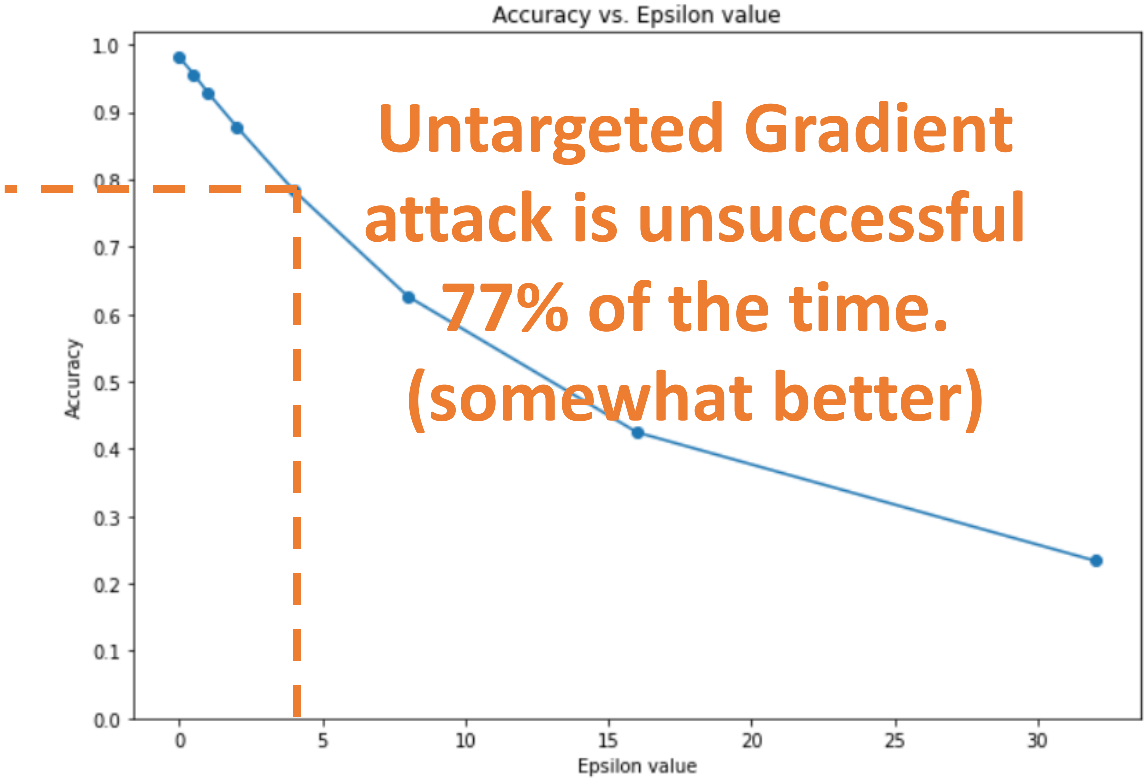


Untargeted gradient vs. epsilon noising

- Epsilon Noising Method

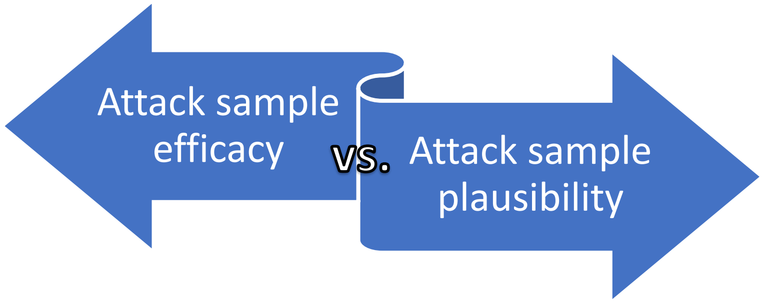


- Untargeted Gradient Method



Untargeted gradient attack recap

- Like the Epsilon Noising Method from lecture 1, the Untargeted Gradient Method is an attack which is subject to the same **tradeoff** we identified earlier.



- However, **it performs better than the Epsilon Noising Method, as it is able to produce plausible attack samples that seem to fool the models more often** ($\sim 92\%$ vs. $\sim 77\%$, failure rates).

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails $\sim 77\%$ of the time).

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails $\sim 77\%$ of the time).
- It is a **one-shot** attack, which does not necessarily make sense.
  - The gradient descent algorithms takes multiple steps (batches + epochs) during training to converge…
  - Why would a single step of gradient attack be enough?
  - We should repeat the gradient attack multiple times (i.e. make it **iterated**).

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails $\sim 77\%$ of the time).
- It is a **one-shot** attack, which does not necessarily make sense.
- It is **untargeted**.
  - It attempts to invalidate the sample by moving away from its original label,
  - or in the direction of the least probable class.
  - This seems to indicate we can orient the direction in which we move and therefore **target** classes…

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails ~77% of the time).
- It is a **one-shot** attack, which does not necessarily make sense.
- It is **untargeted**.
- Its **plausibility is not too great.**

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails $\sim 77\%$ of the time).
- It is a **one-shot** attack, which does not necessarily make sense.
- It is **untargeted**.
- Its **plausibility is not too great.**
- (It has a **heavy computational cost**, as it requires the gradients from the model to be applied on a sample.)

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails ~77% of the time).
- It is a **one-shot** attack, which does not necessarily make sense.
- It is **untargeted**.
- Its **plausibility is not too great.**
- (It has a **heavy computational cost**, as it requires the gradients from the model to be applied on a sample.)

Our next attack, the **Fast Gradient Sign Method** attempts to solve this heavy computational issue and help make more plausible samples.

Fast Gradient Sign Method (FGSM)

Definition (Fast Gradient Sign Method attack):

The **Fast Gradient Sign Method attack** only uses the **sign** of the gradient to create an attack sample.

$$\tilde{x} \leftarrow x + \epsilon \nabla_x L(x, \theta, c)$$

(Gradient attack)

$$\tilde{\boldsymbol{x}} \leftarrow \boldsymbol{x} + \boldsymbol{\epsilon}\, \mathbf{sign}(\boldsymbol{\nabla}_x L(x, \theta, c))$$

(FGSM attack)

```python
def fgsm_attack(image, epsilon, data_grad):
    # Get element-wise signs of each element of the data gradient
    data_grad_sign = data_grad.sign()

    # Create the attack image by adjusting each pixel of the input image
    eps_image = image + epsilon*data_grad_sign

    # Clipping eps_image to maintain pixel values into the [0, 1] range
    eps_image = torch.clamp(eps_image, 0, 1)

    # Return
    return eps_image
```

Fast Gradient Sign Method (FGSM)

Definition (Fast Gradient Sign Method attack):

The **Fast Gradient Sign Method attack** only uses the **sign** of the gradient to create an attack sample.

$$\tilde{x} \leftarrow x + \epsilon \nabla_x L(x, \theta, c)$$

(Gradient attack)

$$\tilde{\boldsymbol{x}} \leftarrow \boldsymbol{x} + \boldsymbol{\epsilon}\, \mathbf{sign}(\boldsymbol{\nabla}_x L(x, \theta, c))$$

(FGSM attack)

Important property: this also helps to make more plausible samples, as it will, by design, verify $||\tilde{\boldsymbol{x}} - \boldsymbol{x}||_\infty \leq \boldsymbol{\epsilon}$.

- (**Plausibility constraint** that we did not have it in the previous attacks!)

```python
def fgsm_attack(image, epsilon, data_grad):
    # Get element-wise signs of each element of the data gradient
    data_grad_sign = data_grad.sign()

    # Create the attack image by adjusting each pixel of the input image
    eps_image = image + epsilon*data_grad_sign

    # Clipping eps_image to maintain pixel values into the [0, 1] range
    eps_image = torch.clamp(eps_image, 0, 1)

    # Return
    return eps_image
```

Remember

- $\boldsymbol{L^0}$ **norm:** bounds the total number of pixels in $\tilde{x}$ that can be modified with respect to x (though they can be modified by any amount).
- $\boldsymbol{L^1}$ **norm:** bounds the average absolute distance between the values of pixels in $\tilde{x}$ and the corresponding pixels in x.
- $\boldsymbol{L^2}$ **norm:** bounds the total squared distance between the values of pixels in $\tilde{x}$ and the corresponding pixels in x. Often referred to as the Euclidean distance.
- $\boldsymbol{L^\infty}$ **norm:** bounds the maximum difference between any pixel in $\tilde{x}$ and the corresponding pixel in x. Often referred to as max norm.

$$\|\tilde{x} - x\|_\infty = \max_{i,j}\left(\left|\tilde{x}_{i,j} - x_{i,j}\right|\right)$$

Preferred one!

Testing the FGSM attack

The FGSM attack works just fine, and it might even make the **model completely malfunction!**

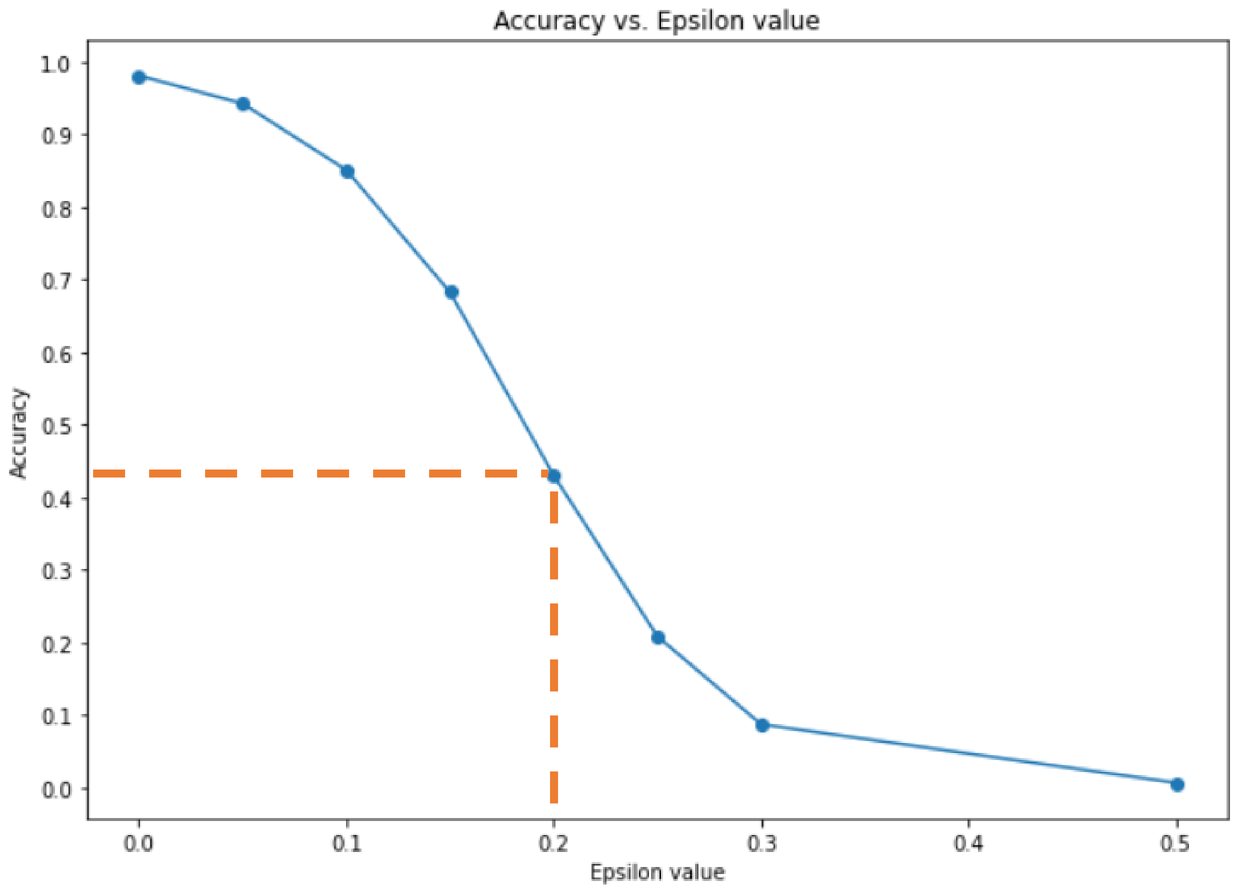
- In the noising approach, the model had to guess randomly and ended up getting a 10% accuracy for large values of epsilon.
- Here, the FGSM will strongly push the model to malfunction, eventually leading to a **0%** accuracy.

```
epsilons = [0, .05, .1, .15, .2, .25, .3, .5]
accuracies = []
examples = []

# Run test() function for each epsilon
for eps in epsilons:
    acc, ex = test(model, device, test_loader, eps)
    accuracies.append(acc)
    examples.append(ex)
```

```
Epsilon: 0 - Test Accuracy = 9810/10000 = 0.981
Epsilon: 0.05 - Test Accuracy = 9426/10000 = 0.9426
Epsilon: 0.1 - Test Accuracy = 8510/10000 = 0.851
Epsilon: 0.15 - Test Accuracy = 6826/10000 = 0.6826
Epsilon: 0.2 - Test Accuracy = 4301/10000 = 0.4301
Epsilon: 0.25 - Test Accuracy = 2082/10000 = 0.2082
Epsilon: 0.3 - Test Accuracy = 869/10000 = 0.0869
Epsilon: 0.5 - Test Accuracy = 63/10000 = 0.0063
```

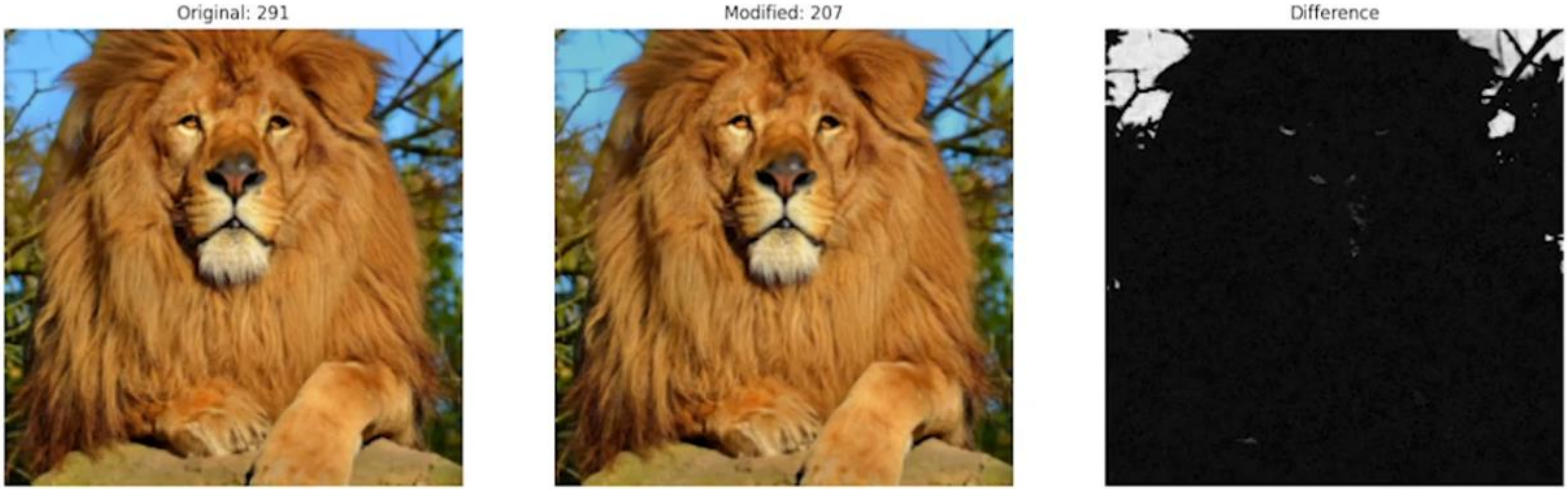
From Notebook 3.

Testing the FGSM attack



lion

golden retriever



Background pixels were changed and this led to an entirely different classification result?! This indicates that our model probably has a wrong classification logic somewhere…

From [Goodfellow2015].

Some more taxonomy on attacks

Definition (one-shot attack):

A **one-shot attack** attempts to produce a single attack sample, and if this attack fails, it simply retries on a different sample.

Noising was therefore a **one-shot attack**. It attempted to noise a sample to have it misclassified.

However, if this attempt failed, it simply tried on another sample.

Definition (iterated attack):

An **iterated attack** attempts to produce an attack sample, like the one-shot attacks.

However, it will try to **adjust the said sample** until it either

- **makes the model malfunction in an expected way**,
- or **reaches a maximal number of allowed iterations**.

The iterated attacks are often more robust and efficient.

Restricted

Iterative FGSM attack (from [Kurakin2016])

Definition (iterative Fast Gradient Sign Method attack):

The **iterative Fast Gradient Sign Method attack** will repeat the FGSM attack until it reaches a maximal number of iterations or makes the model malfunction.

$$\boldsymbol{x_0 = x}$$
$$\boldsymbol{x_{n+1} \leftarrow x_n + \epsilon\, \mathbf{sign}(\nabla_{x_n} L(x_n, \theta, c))}$$

Core idea behind iterating: gradient descent was used for several iterations to <u>train</u> our model, so why should our <u>attacks</u> be using only one iteration of gradient ascent?

Some more taxonomy on attacks

Definition (untargeted attack):

The objective of an **untargeted attack** is to produce an attack sample, which will simply be misclassified.

Noising was an **untargeted attack**, as we attempted to modify a sample in such a way that it would classified as anything but its ground truth label.

Definition (targeted attack):

The objective of a **targeted attack** is to produce an attack sample, which will be misclassified as a specific class.

As such, **targeted attacks** are often **more complex** than **untargeted ones**.

E.g., modify a picture of a **dog (original label)**, so it is misclassified as a **cat (target label)**.

Targeted FGSM attack

Definition (targeted Fast Gradient Sign Method attack):

The **targeted Fast Gradient Sign Method attack** will use the FGSM attack but will use the gradients of a targeted class $\tilde{\boldsymbol{c}}$.

This follows the same logic as moving towards the least probable class as in Gradient attack option #2, but you can use it with any class of your choice $\tilde{\boldsymbol{c}}$ instead of the least probable one.

This attack uses gradient descent to move the sample towards the targeted class $\tilde{\boldsymbol{c}}$.

$$\tilde{\boldsymbol{x}} \leftarrow \boldsymbol{x} - \boldsymbol{\epsilon}\, \mathbf{sign}(\boldsymbol{\nabla}_{\boldsymbol{x}} \boldsymbol{L}(\boldsymbol{x}, \boldsymbol{\theta}, \tilde{\boldsymbol{c}}))$$

Untargeted gradient attack recap

This being said, it suffers from several problems:

- Its **efficacy is still rather low** (fails $\sim 77\%$ of the time).
- It is a **one-shot** attack, which does not necessarily make sense.
- It is **untargeted**.
- Its **plausibility is not too great.**
- (It has a **heavy computational cost**, as it requires the gradients from the model to be applied on a sample.)

Iterative and Targeted FGSM attack

Definition (iterative targeted Fast Gradient Sign Method attack):

The **iterative targeted Fast Gradient Sign Method attack** will use the FGSM attack but will use the gradients of a targeted class $\tilde{\boldsymbol{c}}$.

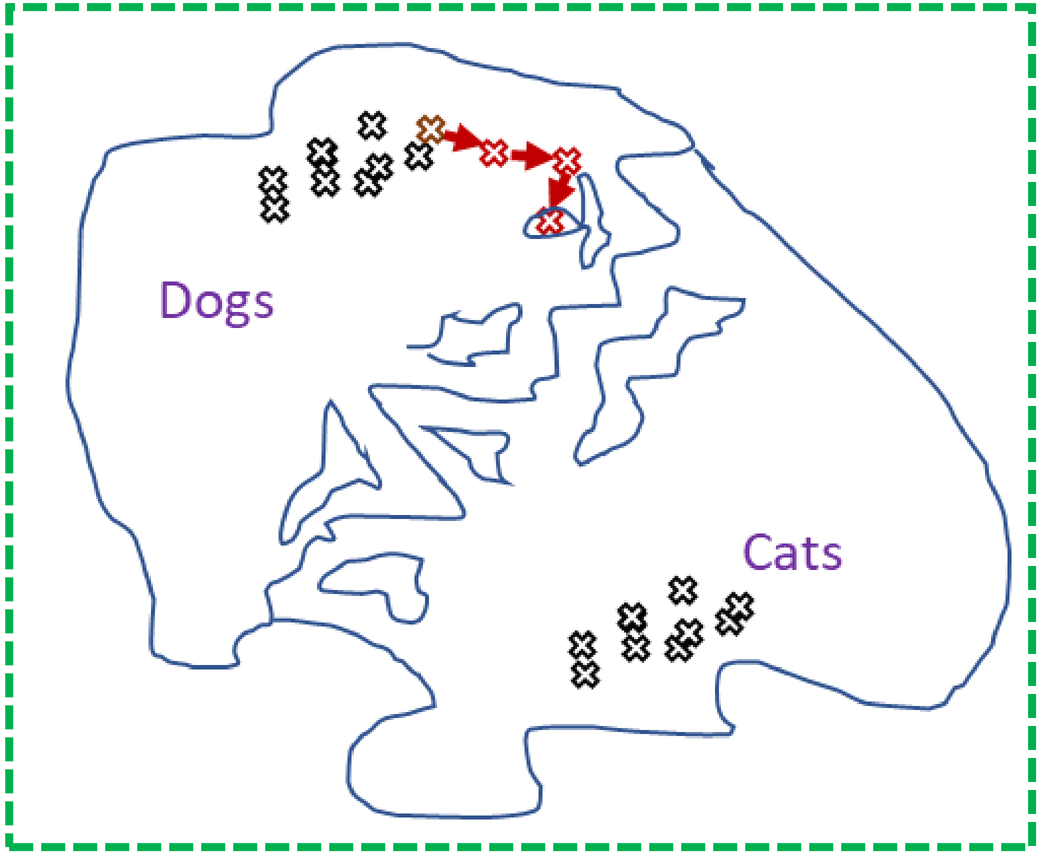
This follows the same logic as moving towards the least probable class as in Gradient attack option #2, but you can use it with any class of your choice $\tilde{\boldsymbol{c}}$ instead of the least probable one. This attack uses gradient descent to move the sample towards the targeted class $\tilde{\boldsymbol{c}}$.

This is repeated until it reaches a maximal number of iterations or makes the model malfunction with targeted class $\tilde{\boldsymbol{c}}$.

$$\boldsymbol{x_0} = \boldsymbol{x}$$
$$\boldsymbol{x_{n+1}} \leftarrow \boldsymbol{x_n} - \boldsymbol{\epsilon}\,\mathbf{sign}(\boldsymbol{\nabla_x L(x_n, \theta, \tilde{c})})$$

Why gradient attack works better than randomly noising

- When randomly noising a **sample** to make an **attack sample**, we move randomly in the feature map.
- When using a gradient-type attack, we move in a more meaningful direction, which might help our **original sample** become **misclassified**.
- **Iterating** allows for smaller steps and **better plausibility** in general (smaller changes in original image).



```python
def itfgsm_attack(image, epsilon, model, orig_class, target_class, iter_num = 10):

    # Convert target class to a LongTensor with one element
    # (Expected format for the F.nll_loss later on)
    target_class_var = Variable(torch.from_numpy(np.asarray([target_class])))
    target_class_torch = target_class_var.type(torch.LongTensor)
    worked = False

    for i in range(iter_num):
        # Zero out previous gradients
        image.grad = None
        # Forward pass
        out = model(image)
        # Calculate loss
        pred_loss = F.nll_loss(out, target_class_torch)

        # Do backward pass and retain graph
        #pred_loss.backward()
        pred_loss.backward(retain_graph = True)

        # Add noise to processed image
        eps_image = image - epsilon*torch.sign(image.grad.data)
        eps_image.retain_grad()

        # Clipping eps_image to maintain pixel values into the [0, 1] range
        eps_image = torch.clamp(eps_image, 0, 1)

        # Forward pass
        new_output = model(eps_image)
        # Get prediction
        _, new_label = new_output.data.max(1)

        # Check if the new_label matches target, if so stop
        if new_label == target_class_torch:
            worked = True
            break
        else:
            image = eps_image
            image.retain_grad()

    return eps_image, worked, i
```

From Notebook 4.

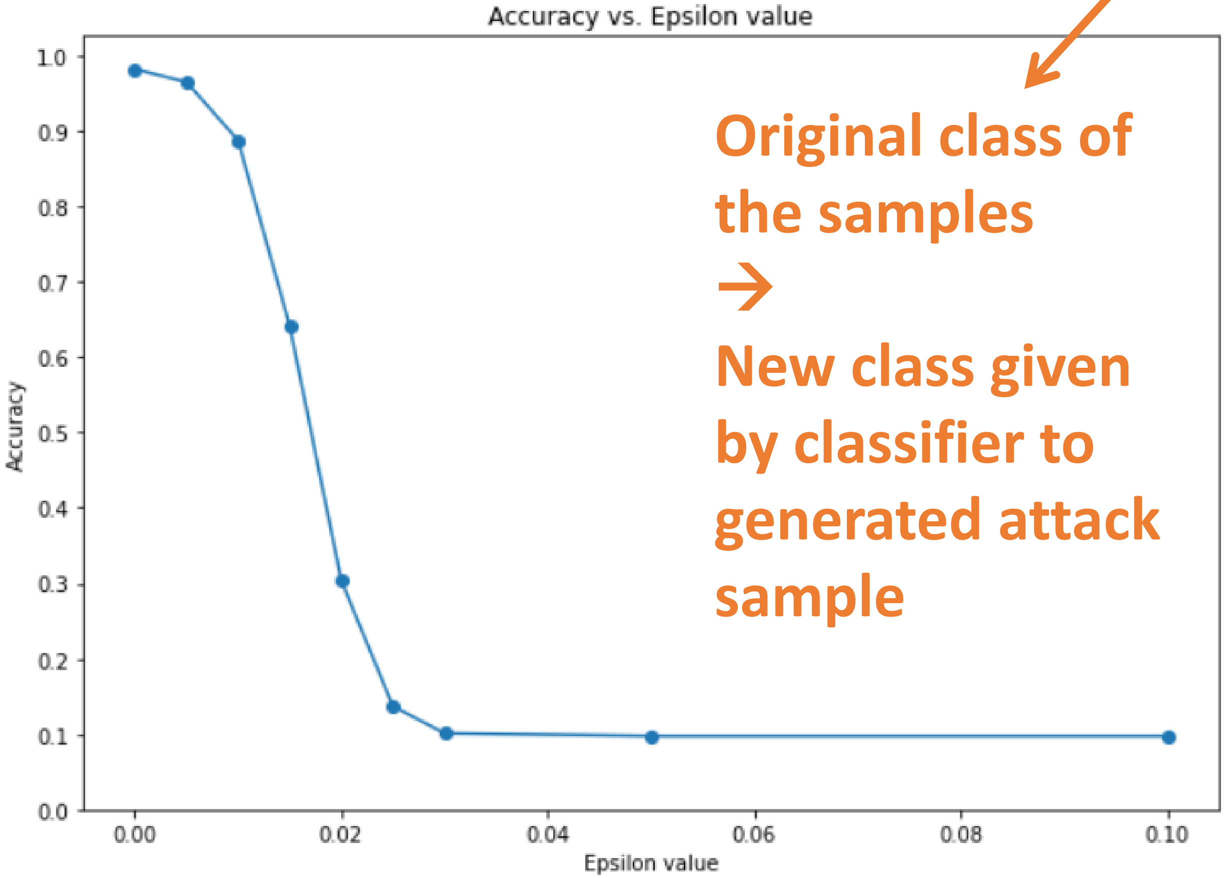
Testing the ITFGSM attack

```
def test(model, device, test_loader, epsilon):

    # Target class
    target_class = 8

    # Counter for correct values (used for accuracy)
    correct_counter = 0

    # List of successful adversarial samples
    adv_examples_list = []

    # Loop over all examples in test set
    for image, label in test_loader:

        # If the initial label is already matching the target class,
        # do not bother attacking, skip current image
        if target_class == label.item():
            correct_counter += 1
            continue

        # Send the data and label to the device
        image, label = image.to(device), label.to(device)

        # Set requires_grad attribute of tensor to force torch to
        # keep track of the gradients of the image
        # (Needed for the ugm_attack() function!)
        image.requires_grad = True

        # Pass the image through the model
        output = model(image)
        # Get the index of the max log-probability
        init_pred = output.max(1, keepdim = True)[1]

        # If the initial prediction is wrong, do not bother attacking, skip current image
        if init_pred.item() != label.item():
            continue

        # Call TFGSM Attack
        eps_image, worked, iterations = itfgsm_attack(image, epsilon, model, label, target_class)
```

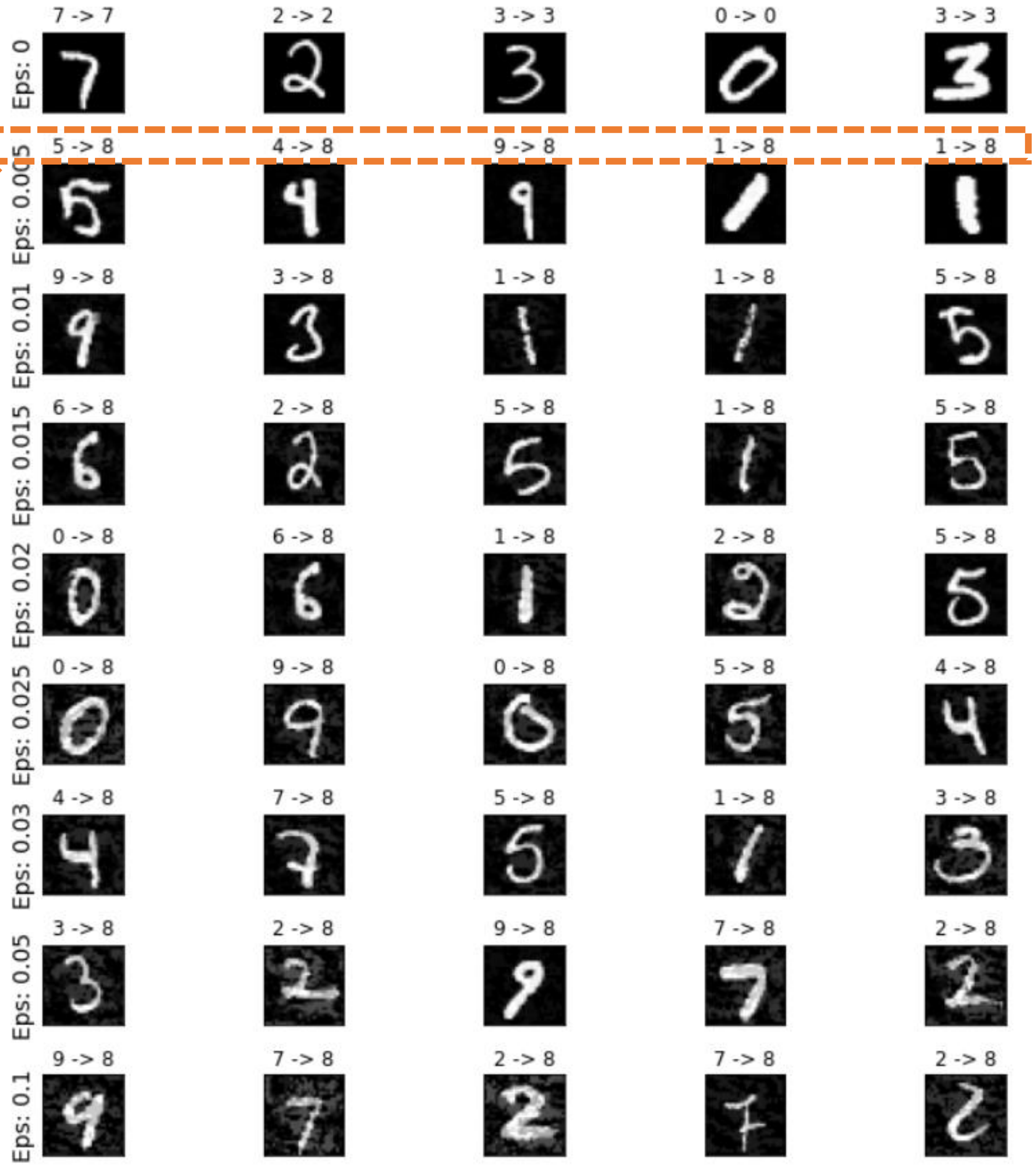
If sample is already of the target class, attack cannot happen…

We cannot modify a picture of an 8 so that it becomes misclassified as an 8! Skip these.

Testing the ITFGSM attack

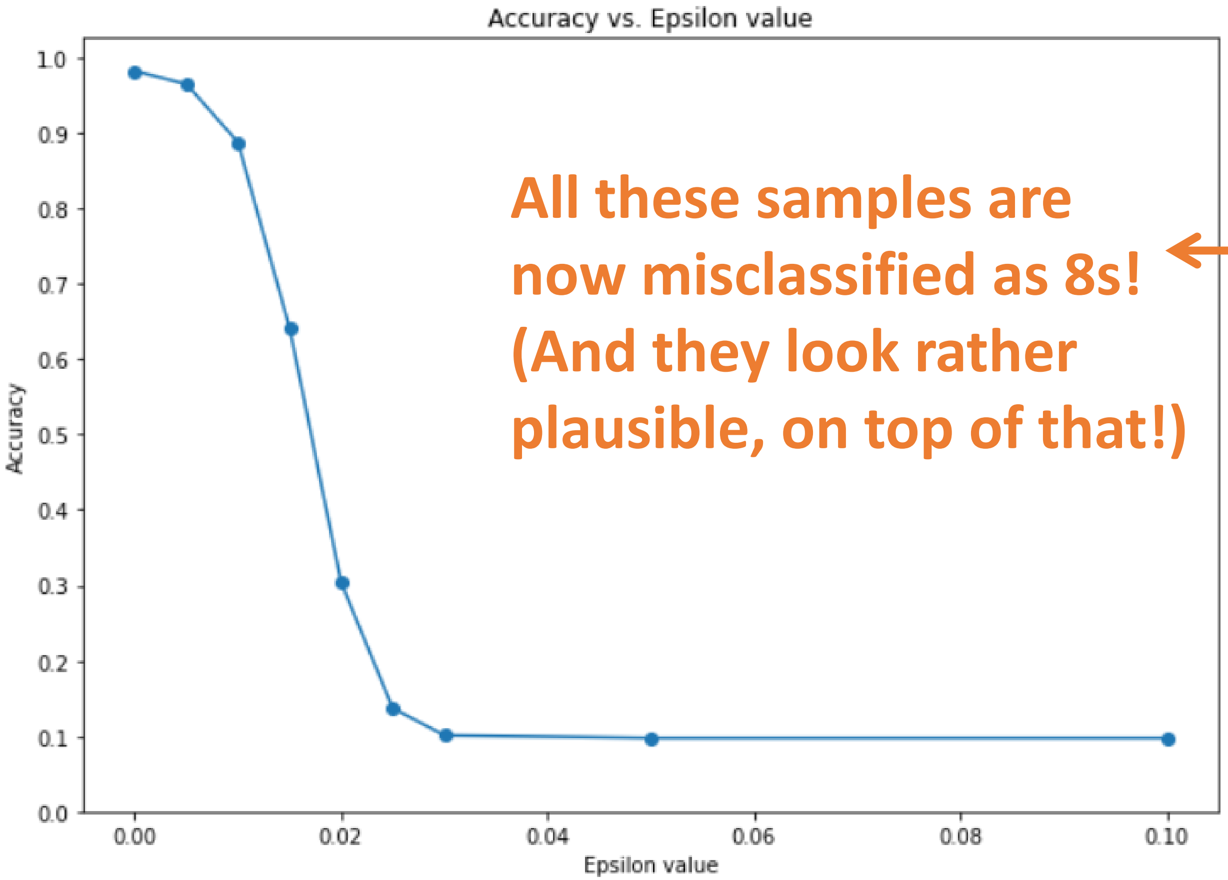


Original class of the samples → New class given by classifier to generated attack sample

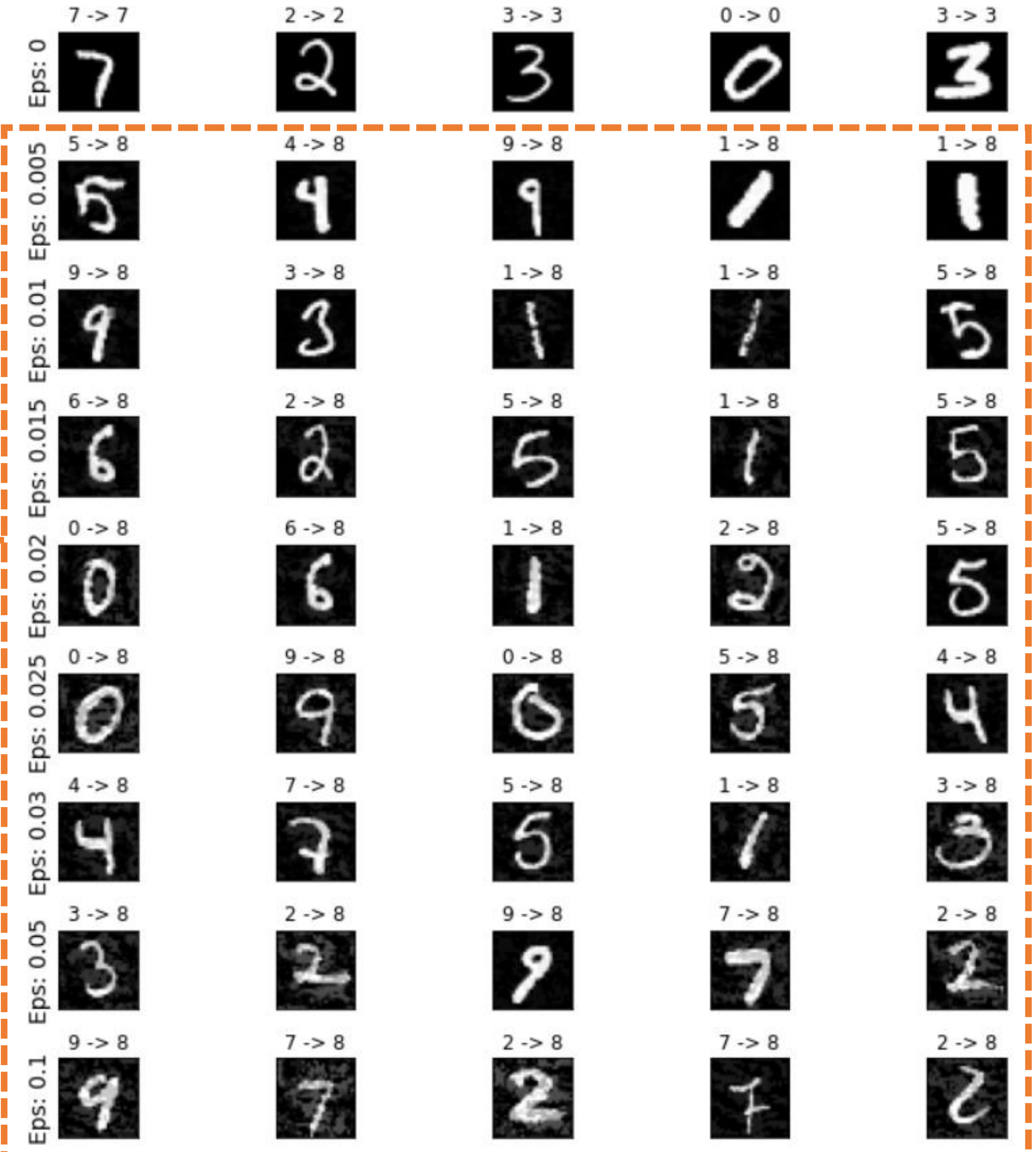


Restr

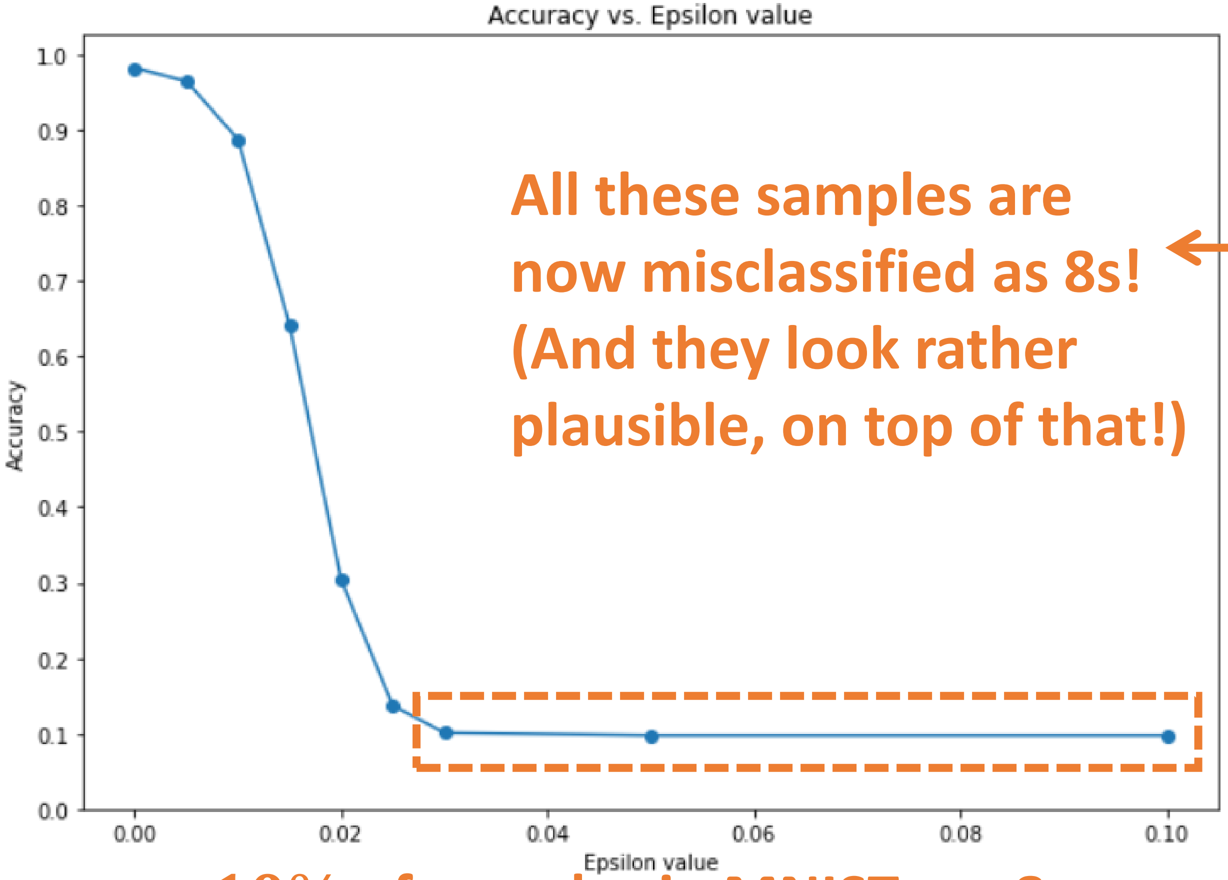
Testing the ITFGSM attack



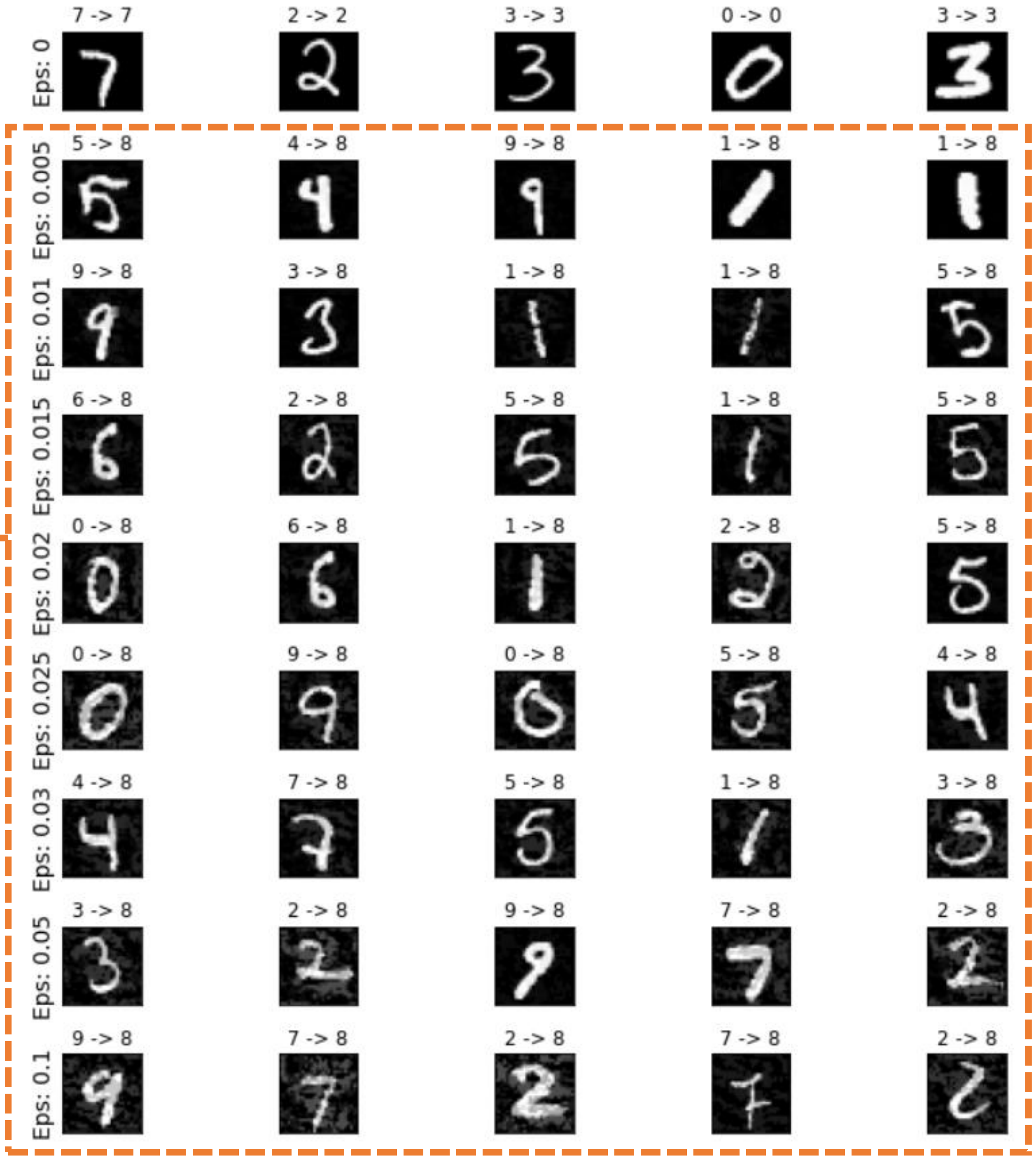
All these samples are now misclassified as 8s! (And they look rather plausible, on top of that!)



Testing the ITFGSM attack



All these samples are now misclassified as 8s! (And they look rather plausible, on top of that!)

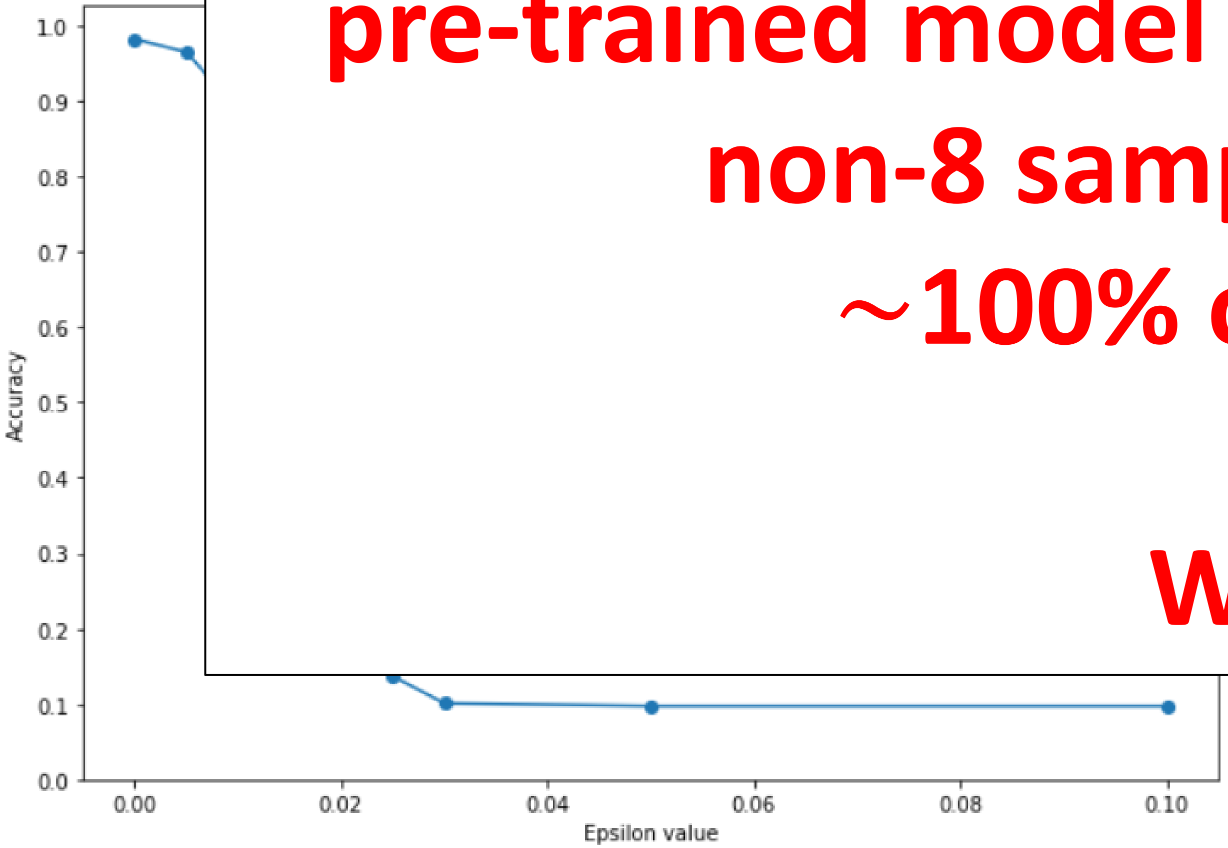


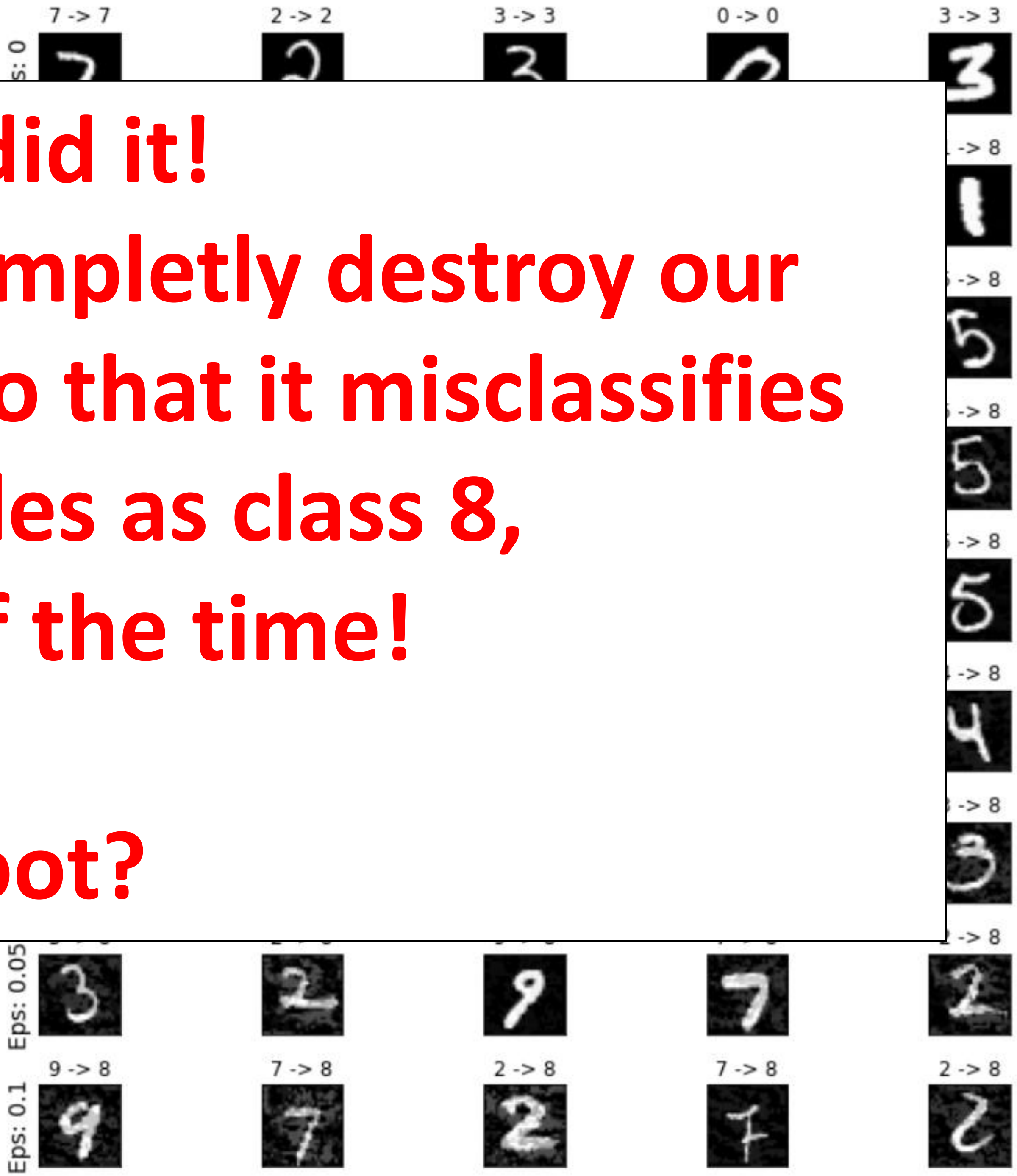
~10% of samples in MNIST are 8s and cannot be attacked

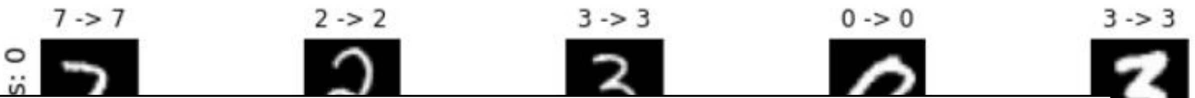
We did it!

We managed to completly destroy our pre-trained model so that it misclassifies non-8 samples as class 8, ~100% of the time!

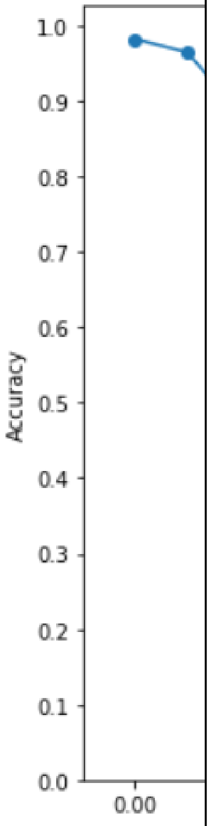
Woot?







(Wait, actually, that is scary, I never EVER want to trust a Neural Network again…!)



Remember, there is more… Reason #2: Defense

Definition (Defense on Neural Networks):

In adversarial machine learning, **defense** refers to machine learning techniques that attempt to **protect models from being attacked** by malicious attempts.

Important: defense mechanisms often rely on an understanding of how attacks work.

SOMETHING FOR LATER…

Conclusion (W5S2)

- Using gradient-based attacks can help produce attacks with higher success rates.
- Using Fast Gradient Sign Method gives an extra plausibility constraint, in the form of a max norm constraint between the original image and attack image.
- This can lead to a devastating attack!
- Iterations were used during training, so might as well use them in attacks as well.
- Iterating greatly helps with plausibility.
- Iterated FGSM can technically lead to a full failure of our pre-trained model…!
- Defense is very much needed!

Learn more about these topics

Out of class, for those of you who are curious

- [Goodfellow2015] **Goodfellow** et al., “**Explaining and Harnessing Adversarial Examples**“, 2015. https://arxiv.org/abs/1412.6572
- [Kurakin2016] **Kurakin** et al. “**Adversarial examples in the physical world**”, 2016. https://arxiv.org/abs/1607.02533
- Implementing more advanced gradient ascent, e.g. FGSM with gradient ascent and momentum as in [Dong2017] Y. Dong et al. “**Boosting Adversarial Attacks with Momentum**”, 2017. https://arxiv.org/abs/1710.06081

Learn more about these topics

Tracking important names (Track their works and follow them on Scholar, Twitter, or whatever works for you!)

- **Alexei Kurakin: Researcher** at **Google Brain**.
  http://kurakin.me/
  https://scholar.google.com/citations?user=nCh4qyMAAAAJ&hl=en
- **Ian Goodfellow: (Former?) director** at **Apple** and **PhD** from **Stanford**, wrote a book that is considered the Bible of Deep Learning, and inventor of Generative Adversarial Networks (for later).
  https://www.iangoodfellow.com/
  https://www.deeplearningbook.org/
  https://scholar.google.ca/citations?user=iYN86KEAAAAJ&hl=en

Learn more about these topics

Tracking important names (Track their works and follow them on Scholar, Twitter, or whatever works for you!)

- **Samy Bengio: Senior Director** at **Apple, inventor** of **PyTorch** (!), (and brother of Yoshua Bengio).
  https://bengio.abracadoudou.com
  https://scholar.google.com/citations?user=Vs-MdPcAAAAJ&hl=fr