Adversarial Examples for Neural Networks

# New Method

The method proposed to be put against the Fast Sign Gradient Method is a “Projected Gradient Descent Method”.

This is an iterative optimization-based category method that is utilized in creating adversarial examples. It opens the gap by randomly nudging the input image to a direction in which the model’s loss function is maximized (or in which the model's accuracy is decreased) so long as the modified image remains at a predetermined distance from the original input, the .

The PGD attack in this way cycles over the perturbed picture and advances it in the slope of the loss function's gradient. Eventually the PGD attack aims to find an adversarial example whose distortion is imperceptible to the human observer, but which maximally deceive the model.

## Pseudocode

PGD\_Attack(input\_image x, true\_label y, model f, epsilon, alpha, num\_iterations):

1. Initialize x\_adv = x

2. For each iteration:

a. Calculate the gradient: grad = Gradient(Loss(f(x\_adv), y))

b. Update x\_adv: x\_adv = x\_adv + alpha \* sign(grad)

c. Clip x\_adv: x\_adv = Clip(x\_adv, x - epsilon, x + epsilon)

3. Return x\_adv

## Experimental Results

The table below shows the accuracy of the adversarial image as a function of Epsilon. The tests directly calculated the accuracy of the model on adversarial images without making use of error bars and confidence levels for both the methods.

|  |  |  |
| --- | --- | --- |
| Epsilon | Fast Gradient Sign Method | Projected Gradient Descent |
| 0.05 | 0.335 | 0.296 |
| 0.1 | 0.014 | 0.007 |
| 0.15 | 0.002 | 0 |
| 0.2 | 0 | 0 |

## Critical Evaluation

From the test results, it is quite evident that the Projected Gradient Descent method is far superior when compared to Fast Sign Gradient Descent Method. PGD scored lower accuracy in all the tests and reached 0 accuracy before FSGM could.

However, PGD is a lot more computationally expensive when compared to FSGM and takes much longer to attack, thus a middle ground may be more suitable. This middle ground can be the “Basic Iterative Method”, that boasts lower accuracy than FSGM, alongside lower complexity than PGD.

# Appendix

    def gradient(self,x,y):

        '''

        This method finds the gradient of the cross-entropy loss

        of an image-label pair (x,y) w.r.t. to the image x.

        Input

            x: the input image vector in ndarray format

            y: the true label of x

        Output

            a vector in ndarray format representing

            the gradient of the cross-entropy loss of (x,y)

            w.r.t. the image x.

        '''

*# Forward pass*

        self.forward(x)

*# Compute the gradient of the loss w.r.t. z4*

        grad\_z4 = self.p.copy()  *# Make a copy to avoid modifying self.p*

        grad\_z4[y] -= 1

*# Backpropagation to compute the gradient w.r.t. x*

        grad\_h3 = np.dot(grad\_z4, self.W4.T)

        grad\_z3 = grad\_h3 \* (self.h3 > 0)

        grad\_h2 = np.dot(grad\_z3, self.W3.T)

        grad\_z2 = grad\_h2 \* (self.h2 > 0)

        grad\_h1 = np.dot(grad\_z2, self.W2.T)

        grad\_z1 = grad\_h1 \* (self.h1 > 0)

        grad\_x = np.dot(grad\_z1, self.W1.T)

        return grad\_x

    def fgsm\_attack(self, x, y):

        '''

        This method generates the adversarial example using the

        Fast Gradient Sign Method (FGSM) for an image-label pair (x,y).

        Input

            x: an image vector in ndarray format, representing

               the image to be corrupted.

            y: the true label of the image x.

        Output

            a vector in ndarray format, representing

            the adversarial example created from image x.

        '''

*# Calculate the gradient of the loss w.r.t the input image*

        grad\_x = self.gradient(x, y)

*# Generate the adversarial example using FGSM*

        adversarial\_x = x + self.eps \* np.sign(grad\_x)

        return adversarial\_x

    def pgd\_attack(self, x, y, num\_iterations=10):

        '''

        This method generates the adversarial example using the

        Projected Gradient Descent (PGD) for an image-label pair (x,y).

        Input

            x: an image vector in ndarray format, representing

               the image to be corrupted.

            y: the true label of the image x.

            num\_iterations: the number of iterations for PGD attack

        Output

            a vector in ndarray format, representing

            the adversarial example created from image x.

        '''

*# Initialize the perturbed image as the original image*

        x\_adv = x.copy()

*# Perform PGD attack for a specified number of iterations*

        for \_ in range(num\_iterations):

*# Compute the gradient of the loss w.r.t the perturbed image*

            grad\_x = self.gradient(x\_adv, y)

*# Update the perturbed image using PGD*

            x\_adv += self.eps / num\_iterations \* np.sign(grad\_x)

*# Clip the perturbed image to ensure it stays within the epsilon bound*

            x\_adv = np.clip(x\_adv, x - self.eps, x + self.eps)

        return x\_adv

def evaluate\_on\_adversarial\_fgsm(clf, X\_test, Y\_test, nTest, eps):

    clf.set\_attack\_budget(eps)

    correct = 0

    encountered\_labels = set()  *# Keep track of encountered label pairs*

    for i in range(nTest):

        x, y = X\_test[i], Y\_test[i]

        x\_adv = clf.fgsm\_attack(x, y)  *# Perform FGSM attack*

        y\_pred = clf.predict(x\_adv)

        if y\_pred == y:

            correct += 1

*# Check if the label pair has been encountered before*

        label\_pair = (y, y\_pred)

        if label\_pair not in encountered\_labels:

*# Print original and adversarial images*

            print(f"{i}- Original image (Label: {y}), Adversarial image (Label: {y\_pred})")

            plt.figure(figsize=(8, 4))

            plt.subplot(1, 2, 1)

            plt.imshow(x.reshape(28, 28), cmap='gray')

            plt.title('Original Image')

            plt.axis('off')

            plt.subplot(1, 2, 2)

            plt.imshow(x\_adv.reshape(28, 28), cmap='gray')

            plt.title('Adversarial Image')

            plt.axis('off')

            plt.show()

*# Mark the label pair as encountered*

            encountered\_labels.add(label\_pair)

    acc = correct / nTest

    return acc

*# Set the number of test examples and epsilon*

nTest = len(Y\_test)  *# Or a smaller number if you want to test on a subset*

epsilon = 0.1

*# Evaluate the accuracy on adversarial examples using FGSM*

acc\_adv\_fgsm = evaluate\_on\_adversarial\_fgsm(clf, X\_test, Y\_test, nTest, epsilon)

print(f"Accuracy on adversarial examples using FGSM: {acc\_adv\_fgsm}")

def evaluate\_on\_adversarial\_pgd(clf, X\_test, Y\_test, nTest, eps, num\_iterations=10):

    clf.set\_attack\_budget(eps)

    correct = 0

    encountered\_labels = set()  *# Keep track of encountered label pairs*

    for i in range(nTest):

        x, y = X\_test[i], Y\_test[i]

        x\_adv = clf.pgd\_attack(x, y, num\_iterations)  *# Perform PGD attack*

        y\_pred = clf.predict(x\_adv)

        if y\_pred == y:

            correct += 1

*# Check if the label pair has been encountered before*

        label\_pair = (y, y\_pred)

        if label\_pair not in encountered\_labels:

*# Print original and adversarial images*

            print(f"{i}- Original image (Label: {y}), Adversarial image (Label: {y\_pred})")

            plt.figure(figsize=(8, 4))

            plt.subplot(1, 2, 1)

            plt.imshow(x.reshape(28, 28), cmap='gray')

            plt.title('Original Image')

            plt.axis('off')

            plt.subplot(1, 2, 2)

            plt.imshow(x\_adv.reshape(28, 28), cmap='gray')

            plt.title('Adversarial Image')

            plt.axis('off')

            plt.show()

*# Mark the label pair as encountered*

            encountered\_labels.add(label\_pair)

    acc = correct / nTest

    return acc

*# Set the number of test examples and epsilon*

nTest = len(Y\_test)  *# Or a smaller number if you want to test on a subset*

epsilon = 0.1

num\_iterations = 10

*# Evaluate the accuracy on adversarial examples using PGD*

acc\_adv\_pgd = evaluate\_on\_adversarial\_pgd(clf, X\_test, Y\_test, nTest, epsilon, num\_iterations)

print(f"Accuracy on adversarial examples using PGD: {acc\_adv\_pgd}")