Assignment #: 3

Venkata Sai Chelagamsetty

November 12th, 2022

- 1. (a) The security of any system is measured with respect to the adversarial goals and capabilities that it is designed to defend against—the system's threat model. Threat model is a model in which potential threats, such as structural vulnerabilities or the absence of appropriate safeguards, can be identified and enumerated, and countermeasures prioritized. Threat model answers questions like What kind of threats might a system face?, What kind of capabilities might we expect adversaries to have?, What are the limits on what the adversary might be able to do to us?.
 - (b) An attack occurs when someone attempts to exploit a vulnerability. A compromise occurs when an attack is successful. In other words, when a compromise occurs, a vulnerability is exploited.
 - (c) A security policy is a document that expresses clearly and concisely what the protection mechanisms are to achieve. Its a statement of the security we expect the system to enforce. A security model is a specification of a security policy:

it describes the entities governed by the policy,

it states the rules that constitute the policy.

Essentially It ensures that the CIA-Triad is maintained. It answers questions like how data is accessed? What level of security is required? and what should happen when these requirements aren't met.

(d) Confidentiality ensures that computer-related assets are accessed only by authorized parties Integrity requires that computer system assets and transmitted information be capable of modification only by authorized parties

Availability: The degree to which data or systems are accessible and in functioning condition.

(e) In ML systems, in terms of confidentiality, the attacks are classified as

Model Extraction: An adversary aims to discover the structure or parameters of the model by observing its predictions.

Whereas in terms of Integrity, the attacks are classified as

Poisoning attacks: An adversary tries to manipulate the training dataset in order to control the prediction behavior of a trained model such that the model will label malicious examples into a desired classes e.g., labeling spam e-mails as safe).

and, evasion attacks: The attacker manipulates input samples at test time to evade (cause a misclassification) a trained classifier at test time.

(f) i. Accuracy =
$$100 * \frac{Attacksclassifiedasattacks+Benignclassifiedasbenign}{Totalno.oflogins}$$

 $Accuracy = 100 * \frac{121806}{130200}$

Accuracy = 93

ii. $P(attack) = \frac{No.ofattacks}{Totallogins}$

 $P(Attack) = \frac{11250}{130200}$

P(attack) = 0.0864

iii.

$$P(flag|attack) = \frac{Attackclassifiedasattack}{Totalactualattacks}$$

$$P(flag|attack) = \frac{11086}{11250}$$
(2)

$$P(flag|attack) = \frac{11086}{11250} \tag{2}$$

$$P(flag|attack) = 0.9854 \tag{3}$$

iv.

$$P(flag) = \frac{traffic predicted a sattack}{Total logins}$$
(4)

$$P(flag) = \frac{20036}{130200} \tag{5}$$

$$P(flag) = 0.1538\tag{6}$$

v.

$$P(benign|flag) = \frac{P(flag|benign)P(benign)}{P(flag)}$$
(7)

$$P(benign) = 1 - P(attack) \tag{8}$$

$$P(benign) = 0.9136 \tag{9}$$

$$P(flag|benign) = \frac{Benignclassifiedasattack}{Totalactualbenign}$$
(10)

$$P(flag|benign) = \frac{8950}{110000 + 8950} \tag{11}$$

$$P(flag|benign) = 0.07524 \tag{12}$$

$$P(benign|flag) = \frac{0.07524 * 0.9136}{0.1538}$$
 (13)

$$P(benign|flag) = 0.4469 \tag{14}$$

- (a) The simplest and most accurate score function would be : $2a_1 + a_2 a_3$, If the score goes above 10, then the score is 10. If the score goes below 0, then the score is 0. (As attribute a_1 is correlated twice as strongly with SPAM as a_2 , and Attribute a3 is negatively correlated with SPAM exactly as strongly as a_3 is positively correlated with SPAM.
 - (b) The scores would be as follows:

M1	Spam	4
M2	Not Spam	1
M3	Spam	9
M4	Spam	2
M5	Not Spam	7
M6	Not Spam	2
M7	Not Spam	8
M8	Spam	3
M9	Spam	7
M10	Not Spam	2

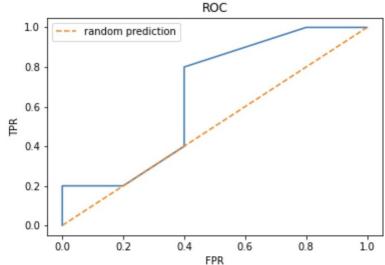
		Spam	Not Spam
(c)	Spam	4	2
	Not Spam	1	3

i. False positive rate: $\frac{2}{5} = 40\%$

ii. True positive rate: $\frac{4}{5} = 80\%$

iii. False negative rate: $\frac{1}{5} = 20\%$

iv. True negative rate: $\frac{3}{5} = 60\%$



(d) ROC is above.

- 3. (a) The size of the convoluted matrix is 22×22
 - (b) We get the following 3×3 matrix:

4	6	5
6	6	8
9	8	8

4. Code

5. I used a subset of first 100 samples from the test dataset and used all the four methods given in the code i.e FGSM, Basic iterative method, Saliency map, and Universal Perturbation. The model summary is given in the code.

The accuracies seem to be really good without much tuning, though I have tested 2 different CNN models and implemented the better one in that case.

Finally, we can infer that the CNN and ANN models are trained on a dataset, but when we generate the adversarial samples, we basically generate data that has different distribution compared to the test and training set. Hence the CNN and ANN models perform so poorly. When we have the augmented dataset, almost half the dataset comes from the same distribution as the training dataset, therefore the accuracies are nearly 50 in such cases. And as the test data is from the same distribution, accuracy is very high.

3

In [1]: # Imports. import random import os print(os.getcwdb()) import network.network as Network import network.mnist loader as mnist loader import pickle import matplotlib.pyplot as plt import numpy as np # Set the random seed. DO NOT CHANGE THIS! seedVal = 41random.seed(seedVal) np.random.seed(seedVal) %matplotlib inline b'C:\\Users\\HP\\Desktop\\problem4' Use a pre-trained network. It has been saved as a pickle file. Load the model, and continue. The network has only one hidden layer of 30 units, 784 input units (MNIST images are \$ 28 \times 28 = 784 \$ pixels large), and 10 output units. All the activations are sigmoidal. # Load the pre-trained model. with open('network/trained network.pkl', 'rb') as f: u = pickle. Unpickler(f) u.encoding = 'latin1' net = u.load()# Helpful function to load the MNIST data. training data, validation data, test data = mnist loader.load data wrapper() The neural network is pretrained, so it should already be set up to predict characters. Run predict(n) to evaluate the \$ n^{th} \$ digit in the test set using the network. You should see that even this relatively simple network works really well (~97% accuracy). The output of the network is a one-hot vector indicating the network's predictions: def predict(n): # Get the data from the test set x = test data[n][0]#print(x) # Print the prediction of the network print('Network output: \n' + str(np.round(net.feedforward(x), 2)) + '\n') print('Network prediction: ' + str(np.argmax(net.feedforward(x))) + '\n') print('Actual image: ') # Draw the image plt.imshow(x.reshape((28,28)), cmap='Greys') # Replace the argument with any number between 0 and 9999 predict(8384) Network output: [[0.] [0.] [0.] [0.] [0.] [0.] [0.] [0.] [1.] [0.]] Network prediction: 8 Actual image: 0 5 10 15 20 25 10 15 20 To actually generate adversarial examples we solve a minimization problem. We do this by setting a "goal" label called \$ \vec y_{goal} \$ (for instance, if we wanted the network to think the adversarial image is an 8, then we would choose \$ \vec y_{goal} \$ to be a one-hot vector with the eighth entry being 1). Now we define a cost function: $S C = \frac{1}{2} | vec y_{goal} - \hat y(vec x)|^2_2$ where $\ | \ \ |^2_2 \$ is the squared Euclidean norm and $\ \ \$ is the network's output. It is a function of $\ \$ \vec x \$, the input image to the network, so we write $\ \$ \hat $\$ \vec x) \$. Our goal is to find an \$ \vec x \$ such that \$ C \$ is minimized. Hopefully this makes sense, because if we find an image \$ \vec x \$ that minimizes \$ C \$ then that means the output of the network when given \$ \vec x \$ is close to our desired output, \$ \vec y_{goal} \$. So in full mathy language, our optimization problem is: \$\$ \arg \min_{\vec x} C(\vec x) \$\$ that is, find the $\ \$ \vec x \$ that minimizes the cost \$ C \$. To actually do this we can do gradient descent on \$ C \$. Start with an initially random vector \$ \vec x \$ and take steps (changing \$ \vec x \$) gradually in the direction opposite of the gradient \$ \nabla_x C \$. To actually get these derivatives we can perform backpropagation on the network. In contrast to training a network, where we perform gradient descent on the weights and biases, when we create adversarial examples we hold the weights and biases constant (because we don't want to change the network!), and change the inputs to our network. Helper functions to evaluate the non-linearity and it's derivative: def sigmoid(z): In [4]: """The sigmoid function.""" **return** 1.0/(1.0+np.exp(-z)) def sigmoid prime(z): """Derivative of the sigmoid function.""" return sigmoid(z)*(1-sigmoid(z)) Also, a function to find the gradient derivatives of the cost function, \$ \nabla_x C \$ with respect to the input $\$ \vec x \$, with a goal label of $\$ \vec y_{goal} \$. (Don't worry too much about the implementation, just know it calculates derivatives). def input derivative(net, x, y): """ Calculate derivatives wrt the inputs""" nabla b = [np.zeros(b.shape) for b in net.biases] nabla w = [np.zeros(w.shape) for w in net.weights] # feedforward activation = xactivations = [x] # list to store all the activations, layer by layer zs = [] # list to store all the z vectors, layer by layer for b, w in zip(net.biases, net.weights): z = np.dot(w, activation) + bzs.append(z) activation = sigmoid(z) activations.append(activation) # backward pass delta = net.cost derivative(activations[-1], y) * \ sigmoid prime (zs[-1])nabla b[-1] = deltanabla w[-1] = np.dot(delta, activations[-2].transpose()) for 1 in range(2, net.num_layers): z = zs[-1]sp = sigmoid prime(z)delta = np.dot(net.weights[-l+1].transpose(), delta) * sp nabla b[-1] = deltanabla w[-1] = np.dot(delta, activations[-1-1].transpose()) # Return derivatives WRT to input return net.weights[0].T.dot(delta) The actual function that generates adversarial examples and a wrapper function: (a) Non Targeted Attack def nonTargetedAdversarial(net, n, steps, eta): net : network object neural network instance to use n : integer our goal label (just an int, the function transforms it into a one-hot vector) steps : integer number of steps for gradient descent eta : float step size for gradient descent ###### Enter your code below ###### # Set the goal output goal = np.zeros((10,1))goal[n] = 1# Create a random image to initialize gradient descent with x = np.random.rand(784, 1)# Gradient descent on the input for i in range(steps): # Calculate the derivative $x = x - eta*(input_derivative(net, x, goal))$ # The GD update on x return x # Wrapper function def generate(n): n : integer goal label (not a one hot vector) ###### Enter your code below ###### # Find the vector x with the above function that you just wrote. x = nonTargetedAdversarial(net, n, 1000, 0.2) # Pass the generated image (vector) to the neural network. Perform a forward pass $\#print('Network\ Output: \n' + str(x) + '\n')$ #print('Network Prediction: ' + str(np.argmax(x)) + '\n') #print('Adversarial Example: ') $print('Network output: \n' + str(np.round(net.feedforward(x), 2)) + '\n')$ $print('Network\ prediction: ' + str(np.argmax(net.feedforward(x))) + '\n')$ print('Adversarial Example: ') plt.imshow(x.reshape(28,28), cmap='Greys') Now let's generate some adversarial examples! Use the function provided to mess around with the neural network. (For some inputs gradient descent doesn't always converge; 0 and 5 seem to work pretty well though. I suspect convergence is very highly dependent on our choice of random initial \$ \vec x \$. We'll see later in the notebook if we force the adversarial example to "look like" a handwritten digit, convergence is much more likely. In a sense we will be adding regularization to our generation process). generate(2) Network output: [[0.] [0. [0.99][0.] [0. [0.] [0. [0. [0. 1 [0.]] Network prediction: 2 Adversarial Example: 15 20 10 15 20 25 (b) Targeted Attack(s) Sweet! We've just managed to create an image that looks utterly meaningless to a human, but the neural network thinks is a '5' with very high certainty. We can actually take this a bit further. Let's generate an image that looks like one number, but the neural network is certain is another. To do this we will modify our cost function a bit. Instead of just optimizing the input image, \$ \vec x \$, to get a desired output label, we'll also optimize the input to look like a certain image, \$ \vec x_{target} \$, at the same time. Our new cost function will be $S C = \langle y_{goal} - y_{hat}(\vec{x}) |^2_2 + \lambda \langle \vec{x} - \vec{x}_{target} |^2_2$ The added term tells us the distance from our $\ \$ and some $\ \$ (which is the image we want our adversarial example to look like). Because we want to minimize \$ C \$, we also want to minimize the distance between our adversarial example and this image. The \$ \lambda \$ is hyperparameter that we can tune; it determines which is more important: optimizing for the desired output or optimizing for an image that looks like \$ \vec x_{target} \$. If you are familiar with ridge regularization, the above cost function might look suspiciously like the ridge regression cost function. In fact, we can view this generation method as giving our model a prior, centered on our target image. Here is a function that implements optimizing the modified cost function, called sneaky_adversarial (because it is very sneaky). Note that the only difference between this function and adversarial is an additional term on the gradient descent update for the regularization term: # Probably include a question on why not generate an image of 5, instead of having to gradient based optimization methods. (as in the above case) In [24]: def targetedAdversarial(net, n, x target, steps, eta, lam=.05): net : network object neural network instance to use n : integer our goal label (just an int, the function transforms it into a one-hot vector x target : numpy vector our goal image for the adversarial example steps : integer number of steps for gradient descent eta : float step size for gradient descent lam : float lambda, our regularization parameter. Default is .05 # Set the goal output goal = np.zeros((10,1))goal[n] = 1# Create a random image to initialize gradient descent with x = np.random.rand(784, 1)# Gradient descent on the input for i in range(steps): # Calculate the derivative $x = x - eta*(input_derivative(net, x, goal)+lam*(x-x_target))$ # The GD update on x, with an added penalty to the cost function return x # Wrapper function def generate_advSample(n, m): n: int 0-9, the target number to match m: index of example image to use (from the test set) # Find random instance of m in test set idx = np.random.randint(0,8000)while test data[idx][1] != m: idx += 1 # Hardcode the parameters for the wrapper function a = targetedAdversarial(net, n, test data[idx][0], 100, 1) x = np.round(net.feedforward(a), 2) print('\nWhat we want our adversarial example to look like: ') plt.imshow(test_data[idx][0].reshape((28,28)), cmap='Greys') plt.show() print('\n') print('Adversarial Example: ') plt.imshow(a.reshape(28,28), cmap='Greys') plt.show() print('Network Prediction: ' + str(np.argmax(x)) + '\n') print('Network Output: \n' + str(x) + '\n') return a Play around with this function to make "sneaky" adversarial examples! (Again, some numbers converge better than others... try 0, 2, 3, 5, 6, or 8 as a target label. 1, 4, 7, and 9 still don't work as well... no idea why... We get more numbers that converge because we've added regularization term to our cost function. Perhaps changing \$ \lambda \$ will get more to converge?) In [25]: # generate advSample(target label, target digit) adv ex = generate advSample(8, 2) What we want our adversarial example to look like: 0 · 5 15 20 25 25 0 10 15 20 Adversarial Example: 5 10 15 20 25 15 20 Network Prediction: 8 Network Output: [[0.] [0. [0.01] [0. .01 [0. [0. [0. 1 [1. 1 [0.]] (c) Protection against adversarial attacks Awesome! We've just created images that trick neural networks. The next question we could ask is whether or not we could protect against these kinds of attacks. If you look closely at the original images and the adversarial examples you'll see that the adversarial examples have some sort of grey tinged background. So how could we protect against these adversarial attacks? One very simple way would be to use binary thresholding. Set a pixel as completely black or completely white depending on a threshold. This should remove the "noise" that's always present in the adversarial images. Let's see if it works: def simple_defense(n, m): n: int 0-9, the target number to match m: index of example image to use (from the test set) 11 11 11 # Generate an adversarial sample. x = generate_advSample(n, m) # Perform binary thresholding on the generated sample. You can choose the thresho. for k in range(len(x)): **if** x[k] > 0.5: x[k] = 1else: x[k] = 0print("With binary thresholding: ") # Plot a grayscale image of the binarized generated sample. plt.imshow(x.reshape(28,28), cmap='Greys') plt.show() # Print the network's predictions. print("Prediction with binary thresholding: " + str(np.argmax(net.feedforward(x))) # The output of the network. print("Network output: print(str(np.round(net.feedforward(x), 2))) # binary thresholding(target digit, actual digit) simple defense (0, 3)What we want our adversarial example to look like: 5 15 20 25 10 15 25 20 Adversarial Example: 5 10 15 25 25 10 15 20 Network Prediction: 0 Network Output: [[0.96] [0.] [0. [0.02] [0. [0. [0. [0. [0. [0.]] With binary thresholding: 0 5 10 15 20 25 10 15 20 25 Prediction with binary thresholding: 3 Network output: [[0.] [0.] [0.] [1.] [0.] [0.] [0.] [0.] [0.] [0.]] Looks like it works pretty well! However, note that most adversarial attacks, especially on convolutional neural networks trained on massive full color image sets such as imagenet, can't be defended against by a simple binary threshold.

(f Re rc Re ve Re er Re (f Re ka Re di Re	equirement adversal quirement dist-packal quirement arsarial-roll quirement arguirement arguirement arguirement arguirement arguirement arguirement adversal quirement ages (from adversal	already sa sarial-robust already sa ages (from already sa obustness-to already sa sarial-robustneady sa sarial-robustneady sa sarial-robustneady sa already sa sarial-robustneady sa (from nu already sa om scikit-l	rial robustnes atisfied: nump ustness-toolbox atisfied: setu tness-toolbox) atisfied: scik adversarial-r atisfied: tqdm toolbox) (4.64 atisfied: six oolbox) (1.15. atisfied: scip ustness-toolbox atisfied: numb ustness-toolbox atisfied: impo atisfied: impo atisfied: llvm umba>=0.53.1-> atisfied: three learn<1.1.0,>=	1.4 M y>=1.18.0 x) (1.21.6 ptools in (57.4.0) it-learn<1 obustness- in /usr/lo 0) y>=1.4.1 in x) (1.7.3) a>=0.53.1 x) (0.56.4 rtlib-metalial-robust lite<0.40, adversarialadpoolctl>= 0.22.2->ad	B 7.4 MB/ in /usr/l) /usr/loca .1.0,>=0. toolbox) ocal/lib/p n /usr/lo data in / ness-tool >=0.39.0d l-robustn =2.0.0 in versarial	s ocal/lib/p l/lib/pyth 22.2 in /u (1.0.2) python3.7/c ython3.7/c cal/lib/py ocal/lib/p usr/local/ box) (4.13 ev0 in /us ess-toolbo /usr/loca -robustnes	oython3.7/dishon	dist-pa t-packa lib/pyt ages (f ges (fr ist-pac dist-pa n3.7/di ib/pyth 1) hon3.7/) (3.1.
React	equirement kages (fre equirement rom sciki- equirement -packages 1.1) equirement experiment experim	already sa om scikit-l already sa t-learn<1.1 already sa (from impo already sa b-metadata- collected py installed as dom py as np plotlib.pyp sorflow as arn.model_s .datasets i .models imp .layers imp	atisfied: thre learn<1.1.0,>= atisfied: jobl 1.0,>=0.22.2-> atisfied: typi ortlib-metadat atisfied: zipp ->numba>=0.53. packages: adve d adversarial-	adpoolctl> 0.22.2->ad ib>=0.11 i adversaria ng-extensi a->numba>= 0>=0.5 in / 1->adversa rsarial-ro robustness rt GridSea l opout, Flat	=2.0.0 in versarial n /usr/lo l-robustn ons>=3.6. 0.53.1->a usr/local rial-robu bustnesstoolbox-	/usr/loca -robustnes cal/lib/py ess-toolbo 4 in /usr/ dversarial /lib/pytho stness-too toolbox	al/lib/pyt ss-toolbox ython3.7/d bx) (1.2.0 'local/lib -robustne	hon3.7/) (3.1. ist-pac) /python ss-tool
ff	rom keras rom keras rom keras rom keras rom sklean accuracy rom art.at rom art.es Set the eedVal = a andom.seed p.random.s Defining UM_CLASSES ATCH_SIZE	.layers imp import bac .layers.act rn.metrics score(y_tes ttacks.evas stimators.c random seed 41 d(seedVal) seed(seedVal) seed(seedVal) s = 10 = 64	chend as K civation import import accura st, y_pred) sion import Fa classification ds. DO NOT CHA	axPooling2 t LeakyReLU cy_score stGradientN import Ke: NGE THIS!	J Method, B		iveMethod	, Unive
## See only	BM's ART like Load the x_train, y Normalize train = 1 test = ke Plot an e lt.title(' lt.imshow lt.show() rint(x_tra convert o train = 1	mnist data y_train), (ation keras.utils example ima 'Label is { (x_train[10].sh class vecto keras.utils	assignment, and assignment, and aset (x_test, y_tes s.normalize(x_t) normalize(x_t) age (after nor {}'.format(y_t) 00])	not any other t) = mnist train, axis est, axis=: malization rain[100]); class matr. al(y_train)	.load_data s=1) 1)) from the ices , NUM_CLA	e train se SSES)	t	
0 5 10 15	.set_image _train = ? _test = x_	e_data_form x_train.res	to_categorica mat('channels_ shape(x_train. ape(x_test.sha	first') shape[0], 3	_ 1, 28, 28			
33]: p (6 34]: p p (6 (1 27]: p	0 5 88, 28) rint(y_tra 60000, 10) rint(x_tra rint(x_tes 60000, 784 0000, 784 rint(x_tra 600, 784)	ain.shape) st.shape))) ain[0:100].	.shape)					
## ## CC C	nn_based_r nn_based_r nn_based_r nn_based_r nn_based_r nn_based_r nn_based_r nn_based_r	he CNN netwoodel = Second to the intermed model.add(Comodel.add(Model.add(Emo	Conv2D(32,(3,3 Conv2D(64,(3,3 MaxPooling2D(p Flatten()) Dense(128, act	ure.),activatio),activatio ool_size=(2	on='relu' 2,2),padd elu'))	,input_sha ing='same'	pe = (1, 2)	
p Tr Ep 60 y:	nn_based_onn_based_onn_based_or Compute inn_based_s rint('Test ain on 600 och 1/5 000/60000 0.9986 och 2/5	classifier classifier. predictions = cnn_bas the accuracy: t accuracy: 000 samples [========	<pre>ile (optimizer= = KerasClassi .fit (x_train, s on the test sed_model.pred cy on the test nn_based_model :', cnn_based_ s</pre>	fier(model= y_train, nl set. ict(x_test) setevaluate(x score1)	=cnn_base o_epochs= x_test,y_	d_model, coo, batch_s test)[1]	lip_value ize=BATCH	s=(0,1) SIZE)
60 y: Ep 60 y: /u `M us	0.9994 och 4/5 0000/60000 0.9989 och 5/5 0000/60000 0.9990 sr/local/dodel.state ed in Tens updates=se sed in Tens updates = set set accurace nn_based_r	[=====================================	3.7/dist-packa will be remov 0, as `updates e_updates	ges/keras/ed in a fuges/keras/ed in a fuges/ke	- 5s 91us - 5s 86us engine/tr ture vers ied autom engine/tr ture vers	/sample - /sample - aining_v1. ion. This atically. aining_v1. ion. This	loss: 0.0 loss: 0.0 py:2067: property py:2045:	032 - a 030 - a UserWar should UserWar
	lense (Denslense_1 (Denslense_1) lense_tal params	e) ======== nv2D) Conv2D) g2d (MaxPool	(None, (N	12544)	8)	Param # 320 18496 0 1605760 1290	==	
YC 37]: # a # N a # # # a	Define the nn_based_reann_base	to complete ANN netwood and the intermediate model.add (Immodel.add (I	diate layers. Dense(input_di (Dense(output_ (Dense(units = Dense(NUM_CLAS	m = x_train dim = 100, 10, kerne.	n.shape[1 init = ' l_initial], units = uniform', izer='unif	activatio	n = 're
a x x x a a # p # a a p Tr Ep 60 y:	nn_based_r nn_based_r nn_based_c _train = x _test = x _nn_based_c Get the p redictions Compute n n_score = rint('Test rain on 600 0000 0.9973	classifier x_train.res test.resha classifier. predictions s = ann_bas the accurace ann_based t accuracy: 000 samples	= KerasClassishape(60000, 7ape(10000, 784.fit(x_train, son the test sed_classifier cy on the test d_model.evaluation.	fier(model=84)) y_train, nh setpredict(x_ set. te(x_test,y)	=ann_base o_epochs= _test) y_test,ba	d_model, c	lip_value size=BATC ATCH_SIZE	s=(0,1); H_SIZE)
60 y: 60 y: 60 y: 60 y: 60 y: 60 y: 60 y:	0.9975 och 3/10 000/60000 0.9987 och 4/10 000/60000 0.9991 och 5/10 000/60000 0.9992 och 6/10 000/60000 0.9992 och 7/10 000/60000 0.9997 och 8/10				- 2s 32us - 2s 32us - 2s 32us - 2s 32us - 2s 32us	/sample - /sample - /sample - /sample -	loss: 0.0 loss: 0.0 loss: 0.0 loss: 0.0	072 - a 058 - a 044 - a 045 - a
y: Ep 60 y: Te 5]: a Mc	0.9997 poch 10/10 poch	[=====================================	Output (None, (None,	Shape 256)	- 2s 32us	/sample -	loss: 0.0	
Yc (6): X y p p p	raphica ou need = ['CNN-k = [cnn_ba lt.bar(X, lt.title() lt.show() lt.close()	to complement of the complemen	ete the follo	owing.	acy on	the tes	t data	
Yc uestion,	ert 2: G	to compl the following	ANN-ba e adversar ete the follo	owing.	•			
d	# This attack x_adv = return ef generat # The l attack x_adv = return ef generat # attack x_adv = return er generat # attack x_adv = return	attack is = FastGrad = attack.ge x_adv te_adv_exam Basic Itera = BasicIte = attack.ge x_adv te_adv_exam ck from the r link: htt = Saliency = attack.ge x_adv	mples_BasicIte ative Method i erativeMethod(enerate(x) mples_Saliency e Jacobian-bas tps://arxiv.or yMapMethod(cla enerate(x)	"Fast Grad assifier) rativeMethors the iteraclassifier; MapMethod(ed Saliency g/abs/1511 ssifier)	ed(classi ative ver classifie y Map Att.	fier, x): sion of FG r, x): ack (Paper	rnot et al	
]:	attack x_adv = return Generate = x_train lassifier nn_adv_exa nn_adv_exa nn_adv_exa nn_adv_exa ig, ax = p ig.suptiti	= Universa = attack.ge x_adv examples f n[0:100] = cnn_base amples_FGSM amples_Basi amples_Sali amples_Univ	mples_Universa alPerturbation enerate(x) for CNN-based ed_classifier M = generate_a icIterativeMet iencyMapMethod versalPerturba ts(8, 4, share arial examples cnn adv exampl	model dv_examples hod = genes = generate tion = gene x='col', sh generated	s_FGSM(clarate_adv_eadv_eadv_eadv_eadv_eadv_eadv_eadv	assifier, examples_B mples_Sali _examples_ w', figsiz CNN-based	x) asicItera encyMapMe Universal e=(15,15) model')	thod(cla Perturba
WA ob	or i in ra for j i ax pre ax ax RNING:art abilities	ange(8): in range(4) [i, j].imsh edictions = [i, j].set_ .attacks.ev . DeepFool .attacks.ev		amples[j][: assifier.p: k Prediction l:It seems s as model l:It seems	i].squeezeredict(npon: {}'.fe	e()) expand_di prmat(np.a	ms(cnn_ad rgmax(pre model is its full model is	v_examp diction: predict attack
WA ok th	ARNING: art pabilities	. DeepFool .attacks.ex . DeepFool .attacks.ex . DeepFool .attacks.ex . DeepFool	vasion.deepfood expects logit vasion.deepfood expects logit vasion.deepfood expects logit vasion.deepfood expects logit vasion.deepfood expects logit	d:It seems as model l:It seems as model l:It seems as model l:It seems as model	that the output t that the output t that the output t that the output t	attacked o achieve attacked o achieve attacked o achieve attacked o achieve attacked	model is its full model is its full model is its full model is its full	attack predict attack predict attack predict attack
WAOR THE	RNING: art abilities RNING: art abilities RNING: art abilities	. DeepFool .attacks.ev . DeepFool .attacks.ev . DeepFool	vasion.deepfoodexpects logit vasion.deepfoodexpects logit vasion.deepfoodexpects logit Adversarial example Network Predicti	l:It seems s as model l:It seems s as model s:It seems s as model	that the output t that the output t	attacked o achieve attacked o achieve	model is its full model is its full	attack predict attack predict
0 10 20 Ne 0 10 20 Ne 0	twork Prediction	on: 7 on: 8	Network Predicti Network Predicti Network Predicti	on: 7 on: 7	Network F	Prediction: 1 Prediction: 4 Prediction: 3	Ne Ne	etwork Pred
0 10 20 Ne 0 10	twork Prediction	on: 3	Network Predicti	on: 3	Network F	Prediction: 3 Prediction: 4 Prediction: 0	Ne	etwork Pred
(177]: ## xx xx cc aa aa aa af ff	Generate = x_train = x.reshalassifier nn_adv_exann_exanneexannee	examples for [0:100] ape(100, 78 apples_FGSM amples_Basi amples_Sali amples_Universalic.subplotle('Adversalic.subplotle)	es_BasicIterat for ANN-based 84) ed_classifier M = generate_a icIterativeMet iencyMapMethod versalPerturba cs(8, 4, share arial examples ann_adv_exampl	dv_example: hod = gene: = generate tion = gene x='col', sh generated	s_FGSM(clarate_adv_eadv_examerate_adv_examerate_adv_examerate_adv_examerate_trownerey='roomet	examples_B mples_Sali _examples_ v', figsiz ANN-based	asicItera encyMapMe Universal e=(15,15) model')	thod(cla Perturba
WA ok	ax pre ax ARNING: art pabilities ARNING: art	<pre>in range(4) [i, j].imsh edictions = [i, j].set_ .attacks.ev . DeepFool .attacks.ev</pre>	vasion.deepfocexpects logit	assifier.p: k Prediction l:It seems s as model l:It seems	<pre>that the output t</pre>	expand_di ormat(np.a attacked o achieve	model is its full	predict attack
WA ok th	RNING: art abilities	. DeepFool .attacks.ex . DeepFool .attacks.ex . DeepFool .attacks.ex . DeepFool	vasion.deepfocexpects logite vasion.deepfocexpects logite vasion.deepfocexpects logite vasion.deepfocexpects logite vasion.deepfocexpects logite vasion.deepfocexpects logite	l:It seems model	that the output t that the output t that the output t that the output t	attacked o achieve attacked o achieve attacked o achieve attacked o achieve attacked	model is its full model is its full model is its full model is its full	attack predict attack predict attack predict attack
ok th ok th wa ok th	RNING: art abilities RNING: art abilities RNING: art abilities	. DeepFool .attacks.ev . DeepFool .attacks.ev	vasion.deepfoo expects logit vasion.deepfoo expects logit vasion.deepfoo expects logit Adversarial example	l:It seems s as model l:It seems s as model l:It seems s as model	that the output t that the output t	attacked o achieve attacked o achieve	model is its full model is its full	attack predict attack predict
10 20 Ne 0 10 20 Ne 0 10 20	twork Prediction	on: 2 on: 0	Network Predicti Network Predicti Network Predicti	on: 2 on: 8	Network F	Prediction: 5 Prediction: 0 Prediction: 4	Ne Ne	etwork Pred
0 10 20 Ne 0 10	twork Prediction	on: 8	Network Predicti	on: 8	Network F	rediction: 1 rediction: 0	Ne	etwork Pred
Pa ac po Yc	art 3: Colversarer ou need to we will lassifier nn_adv_exace 2 = x_test	reate a rial imag ance on to compl be using th t[0:10000] = cnn_base amples_FGSM t[0:10000]	new test some series ges generate the following the following series and the following series are generated.	set, base ated preset. Swing.	ed ent evious	y. Ťest	your cl	assifi
7]: #p a p AN 3]: #p c c p CN 2]: p	lassifier2 nn_adv_exa Evaluate redictions nn_based_s rint('ANN- N-based Mo Evaluate redictions nn_based_s rint('CNN- N-based Mo	ANN-based- s = ann_basecore = ann- based Model codel Score: CNN-based- s = cnn_basecore2 = cn- based Model codel Score: adv_example	sed_classifier M = generate_a -classifier on sed_classifier n_based_model. el Score: {}'. : 0.0162000004 -classifier on sed_classifier n_based_model el Score: {}'. : 0.1542000025 es_FGSM.shape)	the newly .predict(an evaluate(an format(ann 20212746 the newly .predict(cn .evaluate(con format(cnn	generate nn_adv_ex nn_adv_ex based_sc generate nn_adv_ex cnn_adv_ex	d adversar amples_FGS amples_FGS are)) d adversar amples_FGS	rial test M) M, y_test rial test M))[1] set
(1 (6 3]: x x CI ac YC	reate a dversar and an augment or i in rain augment of france if rance if rance if rance if some in the second contact and the second co	new au to completed_example ange (10000) ct adversar dom.randint	ete the follo	test set			timage	es +
# p a p AN 1]: # c	ann # Select else: ann ann_augmen print(ann_ redictions nn_based_s rint('ANN- IN-based Mo CNN-based nn_augment or i in ra # Select if rand	n_augmented ct actual s n_augmented nted_exampl augmented s = ann_bas score2 = an -based Mode odel Scores d-classifie ted_example ange (10000) ct adversar dom.randint n_augmented	d_examples.app samples. d_examples.app les = np.conca examples.shap sed_classifier nn_based_model el Score: {}'. : 0.5036000013 er es = [] o: rial samples. t(0, 1) == 0: d_examples.app	end(x_test tenate(ann_e) .predict(an_evaluate(ann_a)) 35144	_augmente	d_examples ted_exampl redictions pre2))	es)	
ch Ch Of ha	# Selections cnn_augment redictions nn_based_s rint('CNN- (N-based Mo lake a s f all the ave ou need to	n_augmented n_augmented nted_exampl s = cnn_bas score3 = cn -based Mode odel Scores single p e model to compl	d_examples.app d_examples.app les = np.conca sed_classifier nn_based_model el Score: {}'. : 0.5741000175 lot, where ls, on the follo	end(x_test tenate(cnnpredict(cn_evaluate(x) format(cnn_476074 ein you three ty owing.	[i]) _augmente nn_augmen x_test, p based_sc compa	d_examples ted_example redictions ore3)) re the test se	test act	you
3]: # X p p	nn_models Plot for = ['Orig: lt.bar(X, lt.title(' lt.show() lt.close() Tes	score = [a the ANN ba inal Test S ann_models Test Accur	cnn_based_scorenn_score, ann_score, ann_score, ann_sed model. Set','Augmentes_score, colored for the A	_based_sco: d Test Set =['r', 'g', NN-based mo	re2, ann_i ', 'Adver, ', 'b'])	oased_scor	e]	
0.4	4]							