Homework 4: Part 2 (Written Part)

18739 Spring 2020

Due: Sunday May 3, 2020 10:20am PST / 1:20pm EST
Late days can be applied as normal to this homework.

Academic Integrity Policy

This is an individual homework. Discussions are encouraged but you should write down your own code and answers. No collaboration on code development is allowed. We refer to CMU's Policy on Cheating and Plagiarism (`https://www.cmu.edu/policies/`). Any code included in your submission will be examined under Similarity Check automatically.

Late Day Policy

Your solutions should be uploaded to Gradescope (`https://www.gradescope.com/`) by the deadline. You have 8 late days in total to spend during the whole course but no more than 3 days can be applied to any one homework.

Written Exercises

If a homework contains written exercises, you will need to submit a `.pdf` file. Please indicate your Andrew ID and name on the first page. We will not accept hard-copies and do not hand-write your answers. Latex (Overleaf: `https://www.overleaf.com`) and Markdown are two recommended languages to generate the clean layout but you are free to use other software. You will need to submit the written part to HomeworkX-PDF on Gradescope, separately from you code submission.

Coding Exercises

Submit the coding portions of homeworks to Gradescope according to the following procedure:

1. Compress your `.py` and other requested files into a zip file called `submission.zip` (No other name is allowed otherwise Gradescope will fail to grade your submission). Do NOT put all files into a folder first and compress the folder. **Create the zip file directly on the top of all required files**. Each homework will specify the `.py` and other files expected to be included in the zip.
2. Submit `submission.zip` to Gradescope. Your code implementation will be auto-graded by Gradescope and you have an infinite number of submissions before the deadline but only the last submission will count towards the grading. Allow the server to take some time (around 2 min) for execution, which means that do not submit until the last moment when everyone is competing for resources.

Contents

Introduction

Before you get started You will be attacking, defending, and analyzing models trained for MNIST, the character recognition dataset we covered in Homework 1. First copy over the mnist dataset from homework 1 (`mnist.pkz`). The release of this homework includes a pre-trained model in `model.MNIST.h5`. You can load this model and mnist data by using the methods and classes of `hw4_mnist`:

```
import hw4_mnist

data = hw4_mnist.load_data()

model = hw4_mnist.MNISTModel()
model.load()
```

Note that `model` here is of type `HW4Model` which encapsulates several aspects of a model that you will need in this homework, including references to its input, logits, probits, and prediction tensors. You will also find a tensorflow session in models of this type.

Visualizations The written questions in this homework include open ended hypothesis framing and experimentation. As part of these exercises, you will be asked to produce visualizations. The most convenient method to produce them are using `matplotlib` from within or without a jupyter notebook. You can consult the following links to aid you in this regard. The `examples.ipynb` notebook also includes several visualization methods that can be helpful.

- https://matplotlib.org/3.2.1/tutorials/index.html – Matplotlib tutorials.
- https://ercanozturk.org/2017/12/16/python-matplotlib-plots-in-latex/ – Saving `pdf` using matplotlib and including them in latex documents.

Bonuses and open ended questions The homework includes various sources of bonus points.

- Part II-3, Mądry's adversarial training, counts as a separate bonus homework worth about 1/4 of normal homework. It can be submitted anytime before May 5.
- We offer bonus points on this homework for finding bugs in our released solution:

  [bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.
- We also offer participation grade bonus credit for coordinating with students your work on the open-ended exercises of this homework. See the *[participation bonus]* notes in the exercises.. These involve both initial public posts regarding the settings chosen for the open ended questions as well as the post-deadline result. Please indicate in your posts whether you are using your own solutions to the coding exercises when performing the experiments or our provided solution.

The open-ended questions give you a lot of choices in how to approach them and involve some form of hypothesis forming, experimental design, and implementation to verify or disprove the hypotheses. The questions do not require your hypotheses to be verified but do require your experimental design to have a chance at addressing the hypotheses one way or another. Points will be deducted if your experimental designs are sufficiently flawed that their outcomes can be pre-determined by a student with rudimentary knowledge of the subject matter.

For example: Part II calls for you to compare the effectiveness of two of the defenses in that section against an attacker of your choosing. However, if you design the attacker to have zero success rate to begin with, you will not be able to gauge the defense effectiveness. **In this case you'd likely lose all of the points for that exercise.** You should set the attack parameters to be effective to begin with. Piazza posts already include many effective parameter settings and you can experiment yourself as well.

Starter code The starter code now includes implementations of the coding exercises. When answering the questions in this homework that require use of the solutions to coding exercise, please indicate whether you are using your own or the provided solution.

I Adversarial Attacks

I-1 Projected Gradient Descent

We have provided a version of the PGD attack to use in Part II of this homework as well as a general guideline for the next attack in this section. The code can be found in `hw4_part1.py`.

[bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

I-2 Carlini & Wagner L_2 Attack

In this part of the assignment, you will be implementing an adversarial attack of Carlini and Wagner [1], which was covered in the lecture on Mar. 26th. Specifically, you will be implementing the L_2 attack on page 9 of the paper. You will be implementing the loss function for the attack as well as the attacking procedures to generate the adversarial image.

An adversarial attack is one of two general types: *evasion(untargeted)* attack, in which the goal is to fool the model into making any incorrect prediction; or a *targeted* attack, in which the goal is to fool the model into making a incorrect classification into a particular class.

The two attacks are typically similar — the primary difference being in the loss used to compute the attack:

Definition 1 (Carlini-Wagner L_2 Attack in original input space)**.** *Consider an input image x and hyper-parameter c, the Carlini-Wagner L_2 attack finds a perturbation δ^* for the following optimization problem:*

$$\begin{aligned} &\textit{minimize loss } \|\delta\|_2^2 + c \cdot f(x+\delta) \\ &\textit{such that } x+\delta \in [0,1]^n \\ \textit{where } &f(x') = \max\left(\max\left\{Z(x')_i : i \neq t\right\} - Z(x')_t, -\kappa\right) (\textit{targeted attack}) \\ &f(x') = \max\left(Z(x')_t - \max\left\{Z(x')_i : i \neq t\right\}, -\kappa\right) (\textit{untargeted attack}) \end{aligned}$$

$Z(x)$ is the pre-softmax output(logits) of the attacked model on x, t is the target label for targeted attack or the correct label for evasion(untargeted) attack. Parameter κ encourages the optimization solver to find an adversarial instance with high confidence. In this homework, you should set $\kappa = 0$.

We can see that the loss consists of two parts, the first part minimizes the perturbation, while the second part maximizes the adversarial loss for the perturbed input. The hyper-parameter c controls how much weight to put on either part of the loss function. In the meantime, we also have to make sure that the adversarial image is within the valid domain between 0 and 1 (for floating point pixels). One way to realize this is through clipping the image during the optimization process, such as in projected gradient descent attack of Szegedy et al. [3].

In practice, clipping the input domain tends to get stuck in a flat loss region. It is common to convert inputs into tanh space, a *change of variables*. Instead of optimizing over δ, we optimize over w where the relationship between the two is the following equation:

$$\delta = \frac{1}{2}(\tanh(w) + 1) - x$$

This effectively clips the perturbed (adversarial) image $x + \delta$ to be valid since $-1 \leq \tanh(w) \leq 1$ makes $x + \delta \in [0,1]$ automatically. With the change of variables, we get Carlini-Wagner L_2 loss in tanh input space:

Definition 2 (Carlini-Wagner L_2 attack loss in tanh input space)**.**

$$\left\|\frac{1}{2}(\tanh(w)+1) - x\right\|_2^2 + c \cdot f\left(\frac{1}{2}(\tanh(w)+1)\right)$$

where f is as in prior definition.

We start the gradient descent to minimize the loss function from the original image x, thus in the beginning $\delta = 0$. In the tanh space, that translates to $w = \operatorname{arctanh}(2x - 1)$, then any optimization/gradient descent onward will happen in the tanh space over w. In the end, to get back to the input space, we need to convert the optimized w^* back to the input space where the resulting perturbed image $x + \delta = \frac{1}{2}(\tanh(w^*) + 1)$.

Coding Exercise 1. ~~[10 points] *Implement the Carlini-Wagner L_2 attack in `hw4_part1.py`. Most of the glue code is already provided for you in the `Attacker` class. You need to only define the various tensors of `CWL2Attacker._define_ops`.*~~

Requirements

1. In the paper the author uses a binary search to find the optimal hyper-parameter c. The examples of `examples.ipynb` has reasonable instantiations of parameters which you can use to get started. Investigating hyper-parameters including this one will be part of the written homework exercises.

2. Since the optimization will be done in the tanh space, you will need to convert the starting input images to tanh space and convert the final attacks back to [0,1] space.

[bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

Written Exercise 2. [1 points]

What is the advantage of using f of Definition 2 compared to the common cross-entropy loss used in image classification?

Written Exercise 3. [2 points]

The PGD attack implementation in the starter code has two methods/modes for keeping the adversarial example close to the attacked image: clipping and projection (PGD is named after the latter mode). Describe the difference between these two modes in terms of both implementation and the effect they have on the resulting adversarial images. Inspecting adversarial examples produced using both modes is helpful here.

Written Exercise 4. [2 points]

The CWL2 attack has yet another means of keeping the adversarial example close to the attacked image. What is that means and in what ways does it differ from projection in PGD? Also, what is the effect of the difference on quantitative or your qualitative perception of distortion.

Written Exercise 5. [5 points]

Evaluate the performance of the three methods (PGD with clipping, PGD with projection, and CWL2) in terms of success, quantitative distortion measures, and subjective (human) distortion perception.

In order to offer a fair comparison, you will have to keep as many settings as possible fixed and consistent across methods while finding good settings for the parameters not shared by the methods. That is, the c parameter of PGD is not comparable to either c or k parameters of CWL2 while the number of steps and the model and instances being attacked should be identical.

Include enough information about your evaluation that it can be replicated, visualize the quantitative results, and describe the qualitative results.

[participation bonus] *In a public Piazza post, post the parameters you have found to have the best attack performance. After the due date, post the conclusions of this exercise.*

For the last exercises of this section, pick one of the three attack methods. Make a note of the method selected in your written submission. Your answers should then be relative to your chosen method.

Written Exercise 6. [5 points]

An intriguing property of adversarial examples is that they tend to transfer from model to model; an example that tricks one model may very well be successful at tricking another model (trained for the same task). Investigate this possibility in one of these ways:

- *(Option A) Train two MNIST models on separate disjoint subsets of the training data. Construct adversarial examples for one of the models and evaluate their success on both.*
- *(Option B) Same analysis as Option A but use two different MNIST model architectures (you can use MNISTModelRegular as the second) trained on the same data.*

- *(Option C) Use both different architectures and different training subsets for your two models.* [2 bonus points] *Bonus points available for using two wildly different MNIST models having comparable validation accuracy.*

In your written answer, state your option and answer: did the adversarial examples made for the first model successfully fool the second? Provide the numbers/visualiations supporting your conclusion.

[participation bonus] *Coordinate with your classmates. Post your choice of attack method and details of the two models being analyzed on a public Piazza post. After the due date, post a summary of your conclusions with regards to transferability of attacks made by the method between those two models.*

II Defense of Adversarial Attack

There are many defense techniques that are under development in the community. In this part, we will be working on defending a model against PGD attack described in [2] with two simple defending methods. We included an implementation of the PGD attack as part of the release materials, in `hw4_part1.py`. We will be building up towards Mądry's defense but that final defense is a BONUS exercise and requires significant additions to the classes we have provided with this homework.

II-1 Defense 1: Data Augmentation

The first approach is to use adversarial examples for training. This can be done to either finetune an existing model or augment training data with the adversarial examples for a model trained normally, then retrain with the combination of original data and adversarial examples from scratch.

Coding Exercise 7. ~~[5 points] *Implement the augmentation defense in the* `AugmentDefender` *class of* `hw4_part2.py`*. You will need to implement both the finetuning and retraining from scratch modes. As part of your implementation, please return from the* `defend` *call the set of adversarial examples generated during the defense and their correct classes.*~~

[bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

II-2 Defense 2: Towards Mądry's adversarial training

One problem with the first defense is that it might defend against the given adversarial examples, the attacks against it might be different than the attacks on the model it was based on. To remedy this, the second defense instead finds new adversarial examples while the training is in progress. In each batch, you will generate a batch of adversarial examples based on the parameters of the model being trained so far. You will use these examples as training instances instead of the normal instances.

Coding Exercise 8. ~~[5 points] *Implement the augmentation defense in the* `PreMadryDefender` *class of* `hw4_part2.py`*. You will need to implement both the finetuning and retraining from scratch modes. As part of your implementation, please return from the* `defend` *call the set of adversarial examples generated during the defense and their correct classes.*~~

[bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

Written Exercise 9. [2 points]

What are the strengths and weaknesses of the two defense methods compared to each other?

Written Exercise 10. [5 points]

Compare the effect of two defense methods on their imparted resistance to adversarial attacks and reduction in model accuracy (on natural examples) under equivalent conditions. That is, they should each be allotted the same number of training steps against the same attacker. Resistance should be measured in terms of the reduction in adversarial success rate and increase in distortion.

You will need to evaluate the chosen attacker on a dataset before defenses; run the defenses on two copies of the attacked model; then evaluate the attacker on the defended models and same attack set again. Provide visualizations for the results in this exercise and include enough details that your results can be replicated.

As the unit of training is a batch, make sure that the total number of batches for both defenses are identical.

[participation bonus] *Coordinate with your classmates with regards to the attacker chosen, its parameters, the defense parameters used, and other choices made. After the deadline, also post your results.*

Written Exercise 11. [2 points]

In the previous exercise you made sure that the amount of training effort in both defenses is the same for a fair comparison. Are there other reasons why the comparison might not have been fair? What else (other than diminished attacker performance) should you consider if you need to determine which defense to deploy?

For the last exercises in this section pick one of the two defenses and one of the three attacks. You may use same attacker as you used to answer the last exercises of the prior section.

Written Exercise 12. [5 points]

Choose an attacker and their effective parameters. Choose a defense. Compare the resistance impaired by the defense and reduced accuracy on the dataset used in the deference vs. a separate dataset not used in the defense. That is, you will need to inspect attacker effectiveness and model performance on two different subsets, one of them used for defense, and the other not, both before and after defense.

Include enough details to be replicated and discuss/visualize your results. What do the results imply about the effectiveness of the defense in the real world?

[participation bonus] *Coordinate with your classmates with regards to the attacker chosen, its parameters, the defense and parameters used, and the subsets selection made. After the deadline, also post your results.*

II-3 (BONUS) Defense 3: Mądry's adversarial training

This is a bonus exercise. It is due anytime before May 5. If you attempt this exercise, send your solutions to the instructors before the deadline. We will also conduct an interview in which you will need to explain your solution to earn full bonus credits. Full bonus credits will also involve written questions in this section. Full credit will account for about 1/4 of a homework's worth of bonus credit.

In this part you will implement Mądry's training described in the Lecture on April 7. Like in the previous defense, you need to consider adversarial examples during the training process itself. As described by Mądry et al. [2], the adversarial training can be written as a minimization problem:

$$\min_{\theta} \rho(\theta, x, y) \tag{1}$$

where

$$\rho(\theta, x, y) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \max_{\delta_p \in \mathcal{S}} L(\theta, x + \delta_p, y)$$

We view $\max_{\delta_p \in \mathcal{S}} L(\theta, x + \delta_p, y)$ as an adversarial attacker and instead of training for a loss with respect to an instance x, the training is done with respect to an adversarially perturb instance $x + \delta_p$. The difference between this and defense 2, however, is that here the adversarial instances are not concrete images. In terms of tensorflow, we need to train on symbolic tensors that represent the adversarial training process which produces adversarial examples.

Coding Exercise 13. [* bonus points]

Implement a symbolic versions of the MNIST model in the `HW4MNISTModelSymbolic` and `PGDAttackerSymbolic` classes in the in the last parts of `hw4_part2.py`.

To implement the loss above, we need to solve min-max optimization, which can be divided into two steps: 1) solve inner-maximization 2) optimize the outter-minimizaiton. However, training with adversarial loss can be much more difficult than natural loss function, therefore, in this homework, we use the following variation that allows us to more easily train the model.

$$\rho(\theta, x, y) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\alpha L(\theta, x, y) + (1-\alpha) L(\theta, x + \delta_p^*, y)]$$

where

$$\delta_p^* = \arg\max_{\delta_p \in \mathcal{S}} L(\theta, x + \delta_p, y)$$

and α is a hyper-parameter that controls the trade-off between the natural loss and the adversarial.

Coding Exercise 14. [* bonus points]

Implement the variation of Mądry's training in the `MadryDefender` class of `hw4_part2.py`.

The following are bonus written exercises about Mądry's adversarial training. These are not part of the normal homework submission. See instructions at the beginning of this section about the Bonus.

Written Exercise 15. [* bonus points]

Plot the natural test accuracy and adversarial test accuracy against different training epochs 2, 4, 6, 8. You can take use of the function `compare_and_show()` for plotting. Use $\alpha = 0.5$, $p = \infty$, $\epsilon = 0.3$.

Written Exercise 16. [* bonus points]

Plot the natural test accuracy and adversarial test accuracy against different $\alpha = \{0, 0.2, 0.5, 0.7\}$. Fix the training epochs to 4. You can take use of the function `compare_and_show()` for plotting. Use $p = \infty$, $\epsilon = 0.3$.

Written Exercise 17. [* bonus points]

Plot the natural test accuracy and adversarial test accuracy against different $\epsilon = \{2/255, 4/255, 8/255, 16/255\}$. Fix the training epochs to 4. You can take use of the function `compare_and_show()` for plotting. Use $p = \infty$, $\alpha = 0.5$.

Written Exercise 18. [* bonus points]

True or False and why: The original Madry's training is a certified robustness defense that the following statement holds for all training points $x \in \mathcal{D}_{train}$

$$f(x) = f(x + \epsilon) \quad \text{if } ||\epsilon||_p < \delta_p \tag{2}$$

where δ_p is the maximum allowed perturbation in ℓ_p space used by the PGD attack during the training. Why or why not?

Written Exercise 19. [* bonus points] ***(Super Bonus)***

Derive the certified robustness guarantee δ_p in ℓ_p space for a linear classifier $y = w^\top x + b$, $y = 0$ if $w^\top x + b < t$ and 1 otherwise. A certified robustness is a Ball $B(x, \delta_p)$ centered at x with radius δ_p that $y = y'$ for if $||x' - x||_p < \delta_p$. Find δ_p to determine the Ball. Why this may be not easy for a neural network?

III Representer Point Selection

In this part, you need to understand Yeh et al. [4][1] and finish the following questions related to the paper. The paper introduced the interpretability of neural networks which can help to identify if a training point is excitatory or inhibitory for predicting a given test point.

From the given input training points $\mathbf{X}$ to corresponding labels $\mathbf{Y}$, we consider a neural network as our prediction model which takes the form of:

$$\hat{\mathbf{y}}_i = \sigma(\Phi(\mathbf{x}_i, \mathbf{\Theta})) \tag{3}$$

where $\mathbf{\Theta}$ are all the parameters of the neural network model and σ is the activation function after last layer in the model. The goal is to understand to what extent does one particular training point $\mathbf{x}_i$ affect the prediction $\hat{\mathbf{y}}_t$ of a test point $\mathbf{x}_t$ as well as the learned weight parameters $\mathbf{\Theta}$.

The parameters are separated to $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ which represent parameters used in the last intermediate layer and all the parameters used to generate the output before last layer, respectively. In our case the last layer is a softmax activation. We assume that the intermediate output before last layer is $\mathbf{f}_i$. As a result, the whole model can be represented into two parts:

- from input $\mathbf{x}_i$ to intermediate output $\mathbf{f}_i$: $\mathbf{f}_i = \Phi_2(\mathbf{x}_i, \mathbf{\Theta}_2)$.
- from $\mathbf{f}_i$ to output $\Phi(\mathbf{x}_i, \mathbf{\Theta})$: $\Phi(\mathbf{x}_i, \mathbf{\Theta}) = \mathbf{\Theta}_1 \mathbf{f}_i$.

III-1 Loss function

Notice that the representer point framework assumes models are trained to a stationary point with an L_2 regularizer on the post-features parameters Θ_1:

$$\mathbf{\Theta}^* = \arg\min_{\Theta} \frac{1}{n} \sum_{i}^{n} L(\mathbf{x}_i, \mathbf{y}_i, \mathbf{\Theta}) + \lambda \|\mathbf{\Theta}_1\|_2^2 \tag{4}$$

However, the starter MNIST model does not use regularization at all!

Coding Exercise 20. ~~[5 points] *Create a version of the starter MNIST model that keeps track of the features layer f and features an L_2 regularizer on the parameters that produce the final output from the features. You need to do this in the `MNISTModelRegular` class of `hw4_part3.py`.*~~

[bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

[1] https://arxiv.org/pdf/1811.09720.pdf

III-2 Representer Points

One we have a suitably setup and trained model, we can determine the impact of each training point on a target instance, as described by Theorem 3.1 of [4]. We will experiment with the sum in that theorem composed of a set of terms, the representer values $k(x_t, x_i, \alpha_i)$ and importance coefficients α_i. Here x_t is the target instance being explained and x_i are training instances. The representer values are a product of the importance coefficients with target/train instance feature similarity $f_i^T f_t$.

Coding Exercise 21. ~~[5 points] *Implement the feature similarity value, training instance importance coefficient value, and representer point value computations in the `Representer` class of `hw4_part3.py`. These should be done in the three methods `similarity`, `coefficients`, `values`.*~~

[bonus points]* Find bugs in our solution. Submit them via a public Piazza post. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

As mentioned in the source paper and implemented by you, the dot product between feature vectors f_i, f_t of two instances is a measure of similarity between them as "seen" by relatively deep point in a network. We could also consider the same measure on the input images themselves. Let $\hat{x} \overset{\text{def}}{=} \frac{x}{\|x\|_2}$ be the normalized version of x. Given flat inputs x_i and x_t, both of shape [784], we can view $\hat{x}_i^T \hat{x}_t$, or the dot product between their normalized versions, as a measure of their similarity in the input space.

Written Exercise 22. [4 points]

Determine whether MNIST instances of the same class are more similar to each other in their input space or more similar to each other in the feature space of an `MNISTModelRegular` model with non-zero λ parameter.

There are many options on how to approach this but be careful to not compare the similarity measurements in the two spaces to each other as they are not expected to have the same units or magnitude.

An example approach is outlined below:

- *For any instance i of class c of the training set, find the top 10 most similar training instance as measured by input space similarity or feature similarity.*
- *If n of the 10 are also class c, say this instance has $n/10$ class consistency in the input space or feature space.*
- *Take the average of the class consistency over a training subset.*
- *Compare the result for the two similarity measures.*

Whatever approach you take, make sure to include enough details for it to be replicated. Discuss the results and what they suggest about the latent representations inside the layers of neural networks.

[2 bonus points] *Use a clustering algorithm to answer the question posed in this exercise.*

Written Exercise 23. [3 points]

Perform the same exercise as above on adversarial examples. Specifically, are adversarial examples more similar to natural examples of the same class in the input space or in the feature space?

In the previous section we've seen how to augment training data with adversarial examples as a means of building resilience to adversarial attacks. Can we use a combination of a defense and representer value computation to identify adversarial examples? The last exercise is an open ended investigation of this question.

Written Exercise 24. [6 points]

Design a method to determine whether an input is adversarial to a model that has been trained using an adversarial example augmented dataset (that is, the `AugmentDefender` *from scratch defense). That is, the method will only have access to an instance that has not been used during training or defense, and has to determine whether it is adversarial or not. The method does not have the instance's true label so you cannot simply test whether the defended model produces the correct label.*

Note that there are many ways of doing this but we would like you to specifically use the Representer Value framework and the coding exercise solutions for the task.

Describe and implement the method, formulate a hypothesis with regards to its effectiveness, design and carry out an experiment to judge its actual effectiveness, and report and visualize the results. Make sure you include enough details to be replicated.

[* bonus points] *Bonus available for well thought-out methods. Success at detecting adversarial example is not necessary for full credit or bonus credit as long as the approach is reasonable.*

Submission

While this is a written exercises homework, most of the exercises call for replication details and involve some coding. As part of the submission, you should include all of the code and notebooks you have used to carry out the various experiments. Make sure you annotate in your code the exercise being addressed and make a note in each write-up where to find the code addressing it.

References

[1] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[2] Aleksander Mądry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL http://arxiv.org/abs/1312.6199.

[4] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. In *Advances in Neural Information Processing Systems*, pages 9291–9301, 2018.