IFT6164-Adversarial ML **Mehrnaz Mofakhami**
Prof: Gauthier Gidel **March 30, 2022**

Hacker Report on the Adversarial Example Games paper

As the hacker for the adversarial example games paper **[1]** I implemented the toy example mentioned in the paper and analyzed my results in the section 1 of this report. I also tried to understand the codebase available on github and play with it for which I explained my experience in section 2.

1 Reproduction of the simple setup: Binary classification with logistic regression

In this section, I will provide my results of the reproduction of the simple setup introduced in the paper in Propositions 2 and 3 and Figure 1. The code is implemented in PyTorch and can be found in this github repository.

1.1 Setup

As mentioned in the paper, I consider a deterministic generator that given a critic function, computes $g(x, y)$ by sampling a number of points uniformly from the ϵ-neighborhood of x with respect to a given norm and choosing the point that has the maximum loss. I will provide results using L_2 and L_∞ norms. I tested critic functions with linear and polynomial features with degrees 3 and 5 (called Linear, Poly3 and Poly5 respectively) similar to the paper. Their equations are given in the following. The experiments are on the two moon dataset of scikit-learn and the loss function is the same as the one defined in equation (5) of the paper. More details on training are given in section 1.2.2.

Since the input data has two dimensions, i.e. $x = (x_1, x_2)$, it is easy to exactly define the critic functions:

- For the **Linear** classifier, the function is as follows:

$$f(x) = w^\top x + b$$

  where $w^\top x + b$ is implemented using the Linear model of PyTorch.

- For the **Polynomial** classifier of degree n, the function is as follows:

$$f(x) = w^\top \text{polyFeatures}(x)$$

  where polyFeatures is a function that transforms an input data matrix into a new data matrix of the given degree using the PolynomialFeatures function of scikit-learn. For example for $n = 2$, (x_1, x_2) will be transformed to $(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$ and then this vector is fed into a PyTorch Linear model whose bias has been set to False because the first element of the transformed vector acts as a bias.

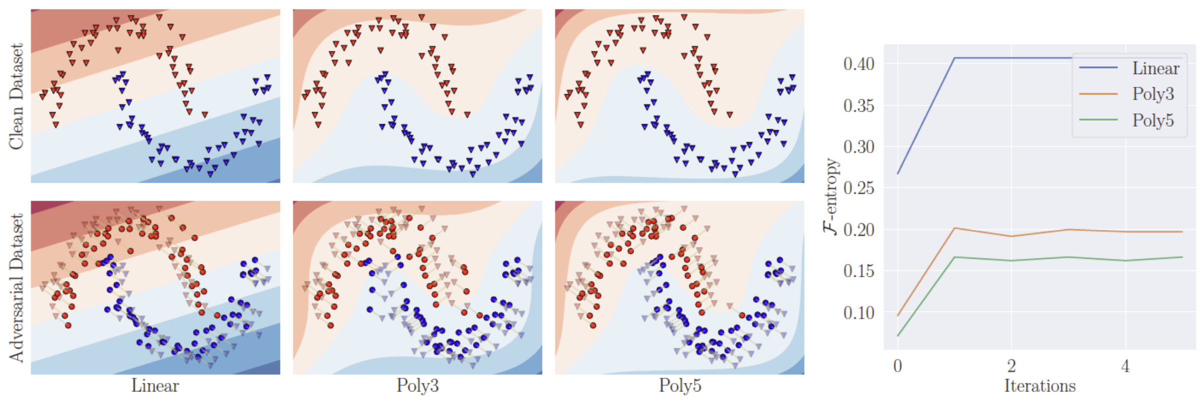

Figure 1: Illustration of Proposition 3 for three classes of classifiers in the context of logistic regression for the two moon dataset of scikit-learn [60] with linear and polynomial (of degree 3 and 5) features. **Left:** Scatter plot of the clean or adversarial dataset and the associated optimal decision boundary. For the adversarial dataset, each corresponding clean example is represented with a ▲/▲ and is connected to its respective adversarial example ●/●. **Right:** value of the $\mathcal{F}$-entropy for the different classes as a function of the number of iterations.

Figure 1: Results of the paper

1.2 Results

In this section, I will provide the results of my experiments on reproduction of Figure 1 of the paper and proposition 2.

1.2.1 Proposition 3 and Figure 1

Below you can compare the results of the paper (Figure 1) with the reproduction results (Figures 2 and 3). The first row of figure 2 illustrates the scatter plot of the clean dataset and the contour graph for each classifier after being trained on the clean dataset. The lines between red and blue regions show the decision boundary, and we can see from the plots and the $\mathcal{F}$-entropies that as the class of critic functions gets larger, the best classifier found within that class—i.e. the classifier found at the end of the minimization step (for more information see 1.2.2)—fits the data better in the sense that it has lower empirical risk. The values of $\mathcal{F}$-entropy for each of the three classes represents the empirical risk of the best classifier found within that class after training on either the clean dataset (iteration 0) or the adversarial dataset (other iterations) and we can see that this value decreases as the class of classifiers gets larger, confirming Proposition 3. In the second row, the clean dataset is shown by red and blue points as before and the adversarial dataset generated by the generator in the last iteration is shown by green and yellow points. Each clean sample is connected to its corresponding adversarial example by a dashed blue line. The contour graph shows the decision boundary and the level sets of the classifier trained on adversarial examples. We can see that the optimal generator constructs adversarial examples by pushing the clean dataset towards the decision boundary.

Two main conclusions can be drawn from the plot of $\mathcal{F}$-entropy. One, as said before, is that $\mathcal{F}$-entropy takes smaller values for larger classes of classifiers, which is what propositions 3 in the paper claims. Secondly, the fact that the plots have converged to a value shows that the min-max game has converged to an equilibrium for all three classes where neither generator nor the critic can improve themselves.

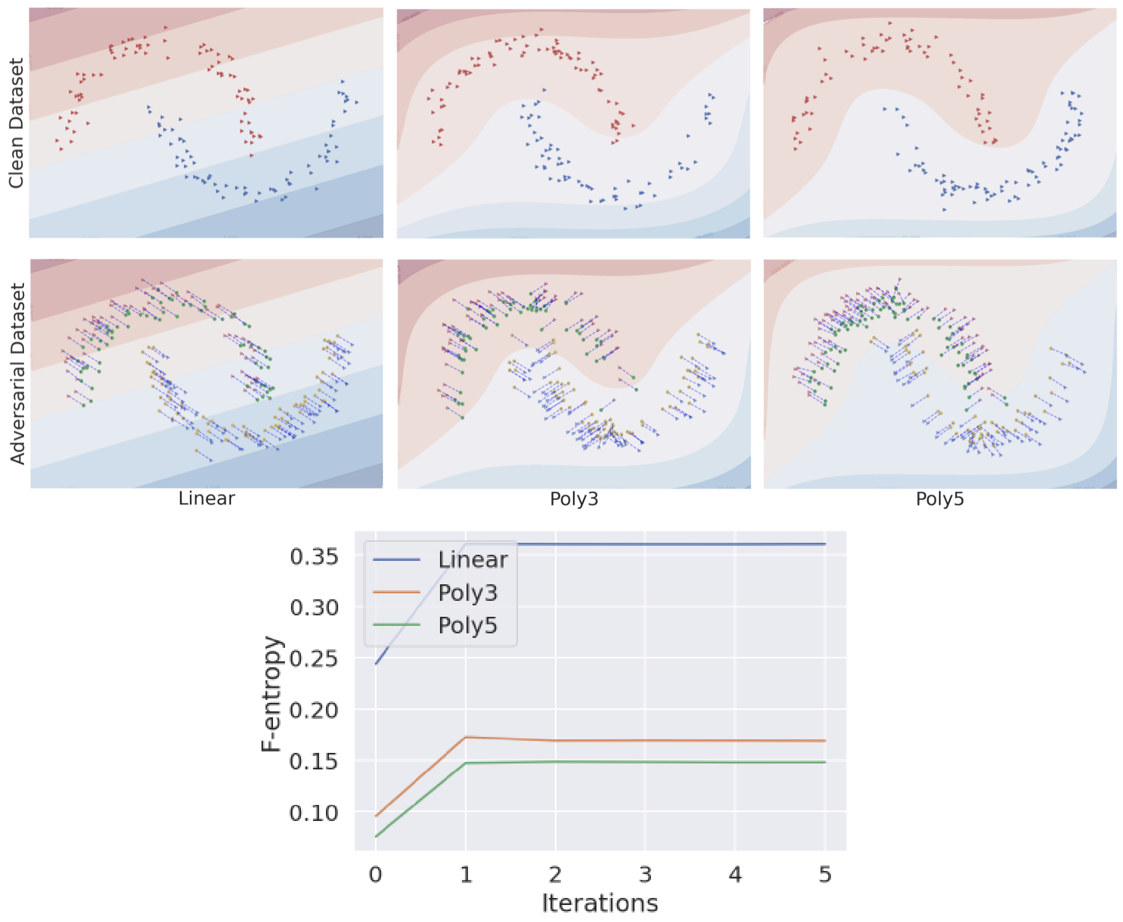

Figure 2: Reproduction results of Figure 1 for a logistic regression task on the two-moon dataset of scikit-learn for classifiers with Linear and polynomial features of degrees 3 and 5 and L_∞ norm for the ϵ-neighborhood. **Top:** The first row is an illustration of the clean dataset and the associated decision boundary. The second row shows how the clean dataset is transformed into the adversarial dataset after convergence of the min-max game. Each line connects a benign point to its corresponding adversarial example. **Bottom:** Values of $\mathcal{F}$-entropy on validation dataset for the three classes of classifiers as a function of the number of maximization steps.

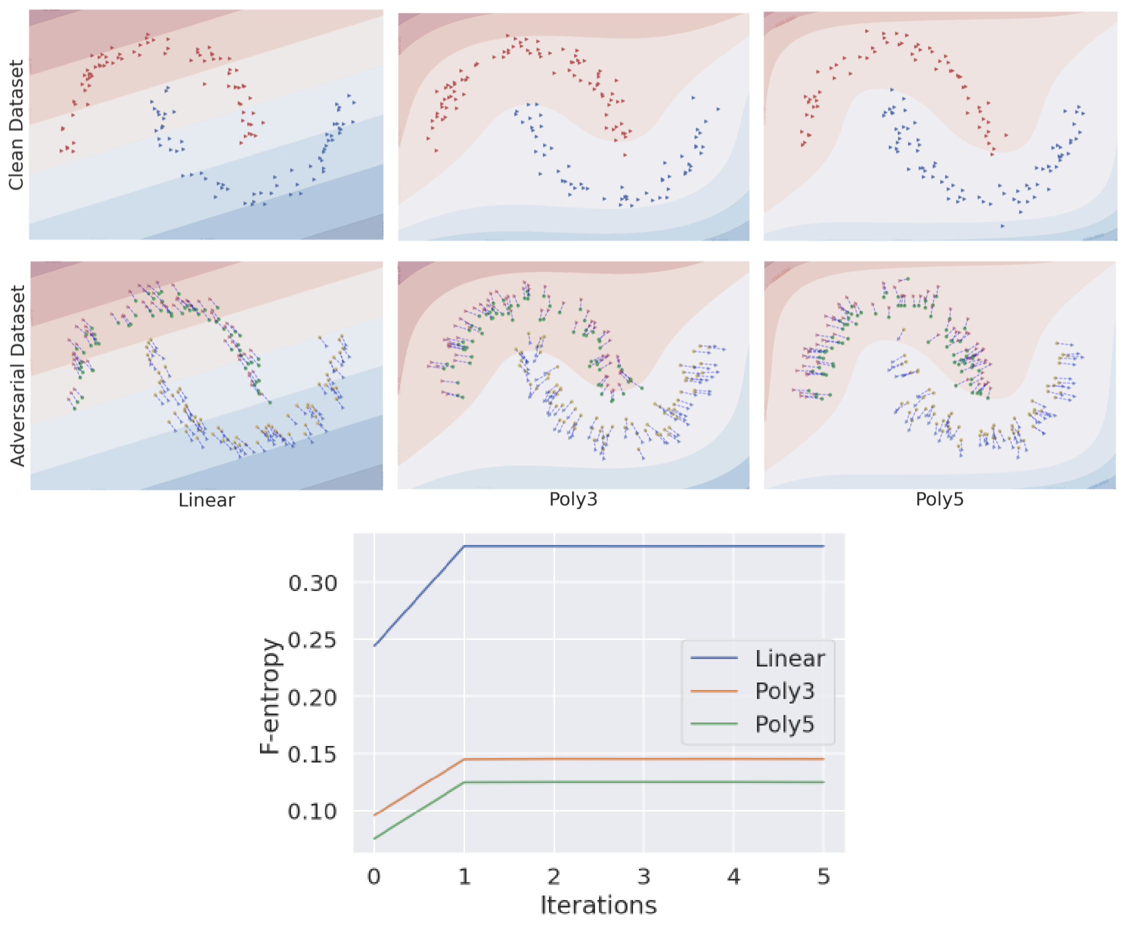

Figure 3: Similar to Figure 2, except for the norm used in calculating the ϵ-neighborhood which is L_2 in this figure.

Comparing figures 2 and 3, we can see the effect of the norm chosen to compute the ϵ-neighborhood around each point while constructing adversarial examples. In Figure 2, the ϵ-neighborhood around each point used in the maximization step is calculated with respect to the L_∞ norm—where points are sampled uniformly from a square of side size 2ϵ—whereas in Figure 3, the ϵ-neighborhood is calculated with respect to the L_2 norm, i.e. points are sampled uniformly from a circle of radius ϵ. Although not mentioned in the paper which norm is used, from the experiments I believe it's L_∞, so it is best to compare figures 1 and 2. In figure 3, for the linear case the blue dashed lines from the benign to adversarial examples mostly point perpendicular to the decision boundary, while for L_∞ this is not necessarily the case since the maximum happens at the corners of the ϵ-neighborhood. The other notable difference is at regions of points for which the part of the decision boundary they should move towards is curled; the change in the direction in these regions is more smooth for L_2 as highlighted in figure 4. The behavior of $\mathcal{F}$-entropy in figure 3 is similar to figure 2 though the entropies for L_2 are a bit smaller.

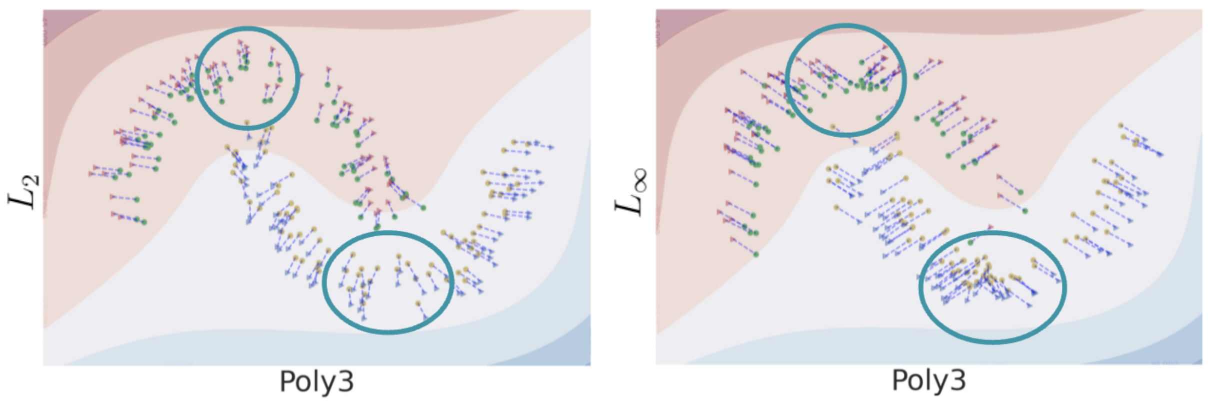

Figure 4: Comparison of L_2 and L_∞ as the norms used in the ϵ-neighborhood around each point in the maximization step

1.2.2 Training Details

Although this toy example seems to be an easy task, reproducing it was very challenging since the details mentioned in the paper on the model architectures and training setup are insufficient and it took me a very long time to find the right setup and hyperparameters that works and produces results that are close to the ones in the paper. In the following, I will provide the setup I used and the important tricks I found, without which the implementation does not produce the desirable results.

The two players in this min-max game are the generator and the critic whose structures are described in section 1.1. Considering the polynomial critic as described in 1.1 is a crucial step that was not mentioned in the paper; see section 1.2.3 for more information on this. I did not use any non-linearity to have plots that are similar to figure 1. Another important trick I found to be effective was to standardize data to have zero-mean and unit variance. In section 1.2.3 I will provide results showing that the training fails without this preprocessing and my opinions of why this transformation makes sense.

As mentioned in the paper, it is very important to fully solve the minimization and maximization steps at each iteration of training. To this end, I trained the critic using full-batch gradient descent until the absolute value of the difference in loss between two consecutive epochs is less than a given stopping threshold. I found the training and values of $\mathcal{F}$-entropy to be very dependent on the stopping threshold. I tuned this hyperparameter such that the values for $\mathcal{F}$-entropy are close to the ones in figure 1. For all three classes, I used SGD with a learning rate of 0.01, ϵ=0.2 and stopping threshold of 1e-5, and the number of samples to be drawn from the ϵ-neighborhood around each point in the maximization step was set to 500. Training all of the experiments was very quick and took 8 minutes on Google Colab for figure 2.

1.2.3 Failed experiments

Following are the settings I tested which failed to work, but are worth mentioning:

- As said in the Training Details section, I found it crucial to implement the polynomial critic in the way described in section 1.1. At first, I implemented it with equation 2.1 of **[2]** but it failed to work, meaning that no matter how much I trained the critic, it did not learn the decision boundaries as in figures 1 to 3 and at best it learned a linear

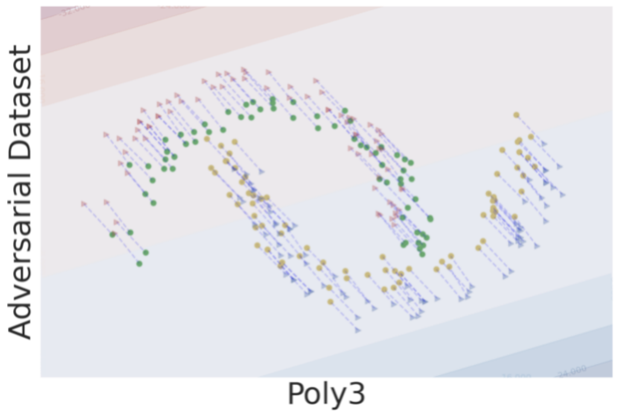

Figure 5: Failure case for an alternative polynomial critic function described in section 1.2.3

function. Figure 5 shows the result of this experiment with standardized inputs, ϵ=0.2, SGD with learning rate=1e-2, stopping threshold=1e-15 and L_∞ norm. Plots were similar with SGD+momentum or Adam as the optimizer and different stopping thresholds.

- As mentioned in section 1.2.2, preprocessing data to have zero mean and unit variance was an important step of my experiments. Figure 6 shows the results of training on the clean dataset with the exact setup described in 1.2.2 except that data is not standardized. We can see that the trained Poly5 classifier completely fails to classify the clean dataset and Poly3 also does not do a very good job if we compare the decision boundary with that of figures 1 to 3. The linear, however, learns to classify well even with non-standardized data. If we look at the range of data in figure 6, it makes sense why the results get worse as the degree grows. The first dimension of the red class ranges from -1.17 to 1.21, though the first dimension of the blue class ranges from -0.78 to 2.15. This difference gets significant if numbers are raised to the power of 5 and makes learning difficult as we see in the results.

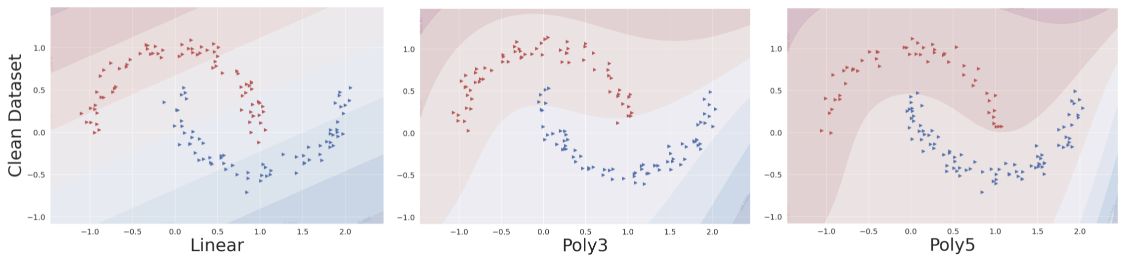

Figure 6: Results of training on non-standardized clean dataset

1.2.4 Proposition 2

In an experiment regarding proposition 2, I took the generator to be $g(x, y) = x - y \cdot \epsilon \, \text{sign}(w)$ and $\mathcal{F}$ to be the class of Linear classifiers as described in section 1.1 with bias=0. I was curious to see whether the generator and the critic converge to the optimal functions described in proposition 2 or not. Figure 7 shows the results of this experiment. Training is done similar to previous experiments by iteratively fully solving the minimization step and calculating the adversarial examples using the given generator function. We can see that the game converges to a point in the sense that the $\mathcal{F}$-entropy almost stops changing. The top left plot is very similar to the corresponding one in figure 2 as the only difference is that in figure 2 the maximization was taken over a finite number of samples drawn from the ϵ-neighborhood w.r.t L_∞ norm but here the equation for the generator is actually the closed solution to the maximization over all points in the L_∞ neighborhood. I computed w^* by solving the minimization problem of (6) in the paper and computed the L_2 norm distance of the critic weights and w^* at each iteration. These distances are shown in the bottom plots of figure 7, where the right one is the zoomed version of the left from iteration 1 onward. We can see that for the training setup used (ϵ=0.2, SGD with lr=1e-2 and stopping threshold=1e-10) the distance keeps decreasing, though the changes are very small such that the $\mathcal{F}$-entropy remains nearly constant.

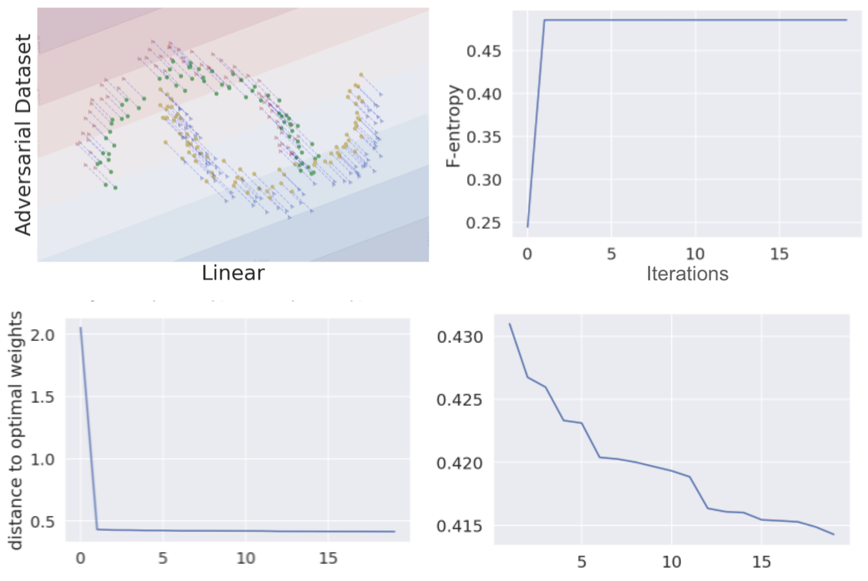

Figure 7: Results of the experiment on proposition 2 as explained in section 1.2.4. **Top:** (Left) adversarial examples and the decision boundary after convergence (Right) $\mathcal{F}$-entropy as the function of the number of maximization steps **Bottom:** Distance to optimal weights as the function of the number of maximization steps

2 Playing with the code on Github

The codebase of the experiments in the paper is available on Github and I tried to run the NoBoxAttack experiments with the code given in the attacks folder. I encountered two main issues which I will describe below; the first one was easy to solve, but the second one took more time to figure out the solution.

- If one runs the code the first error they get is the following:

  ```
  ImportError: cannot import name 'zero_gradients' from 'torch.autograd.gradcheck'
  ```

  This is because the `zero_gradients` function has been removed from `autograd` package as part of a refactor to the code as described here. This import is used in the `advertorch` package and the `fab_pt.py` file in the attacks folder. The solution is as follows:

  – The `advertorch` package should be cloned directly from github and installed via `pip install -e .`

  – The line `from torch.autograd.gradcheck import zero_gradients` in `fab_pt.py` should be removed and instead the `zero_gradients` function code should be added directly to the file:

```
def zero_gradients(x):
    if isinstance(x, torch.Tensor):
        if x.grad is not None:
            x.grad.detach_()
            x.grad.zero_()
    elif isinstance(x, collections.abc.Iterable):
        for elem in x:
            zero_gradients(elem)
```

- The second error that one gets after solving the first issue is due to the fact that the `split_classifiers` and `pretrained_classifiers` directories are not provided and the code cannot be executed without these trained models. Unfortunately, no information is provided on how to get those files and I was confused for a while not knowing what to do. After contacting one of the authors, he gave me a google drive link—which is not available on github or in the paper—but I realized the tree of subdirectories on drive does not match the ones on github. It the end, I got access to the files on Mila cluster, but unfortunately, I still encountered errors while running the code even when the mentioned problem with missing files was solved, so at this point, I abandoned the code and spent the rest of my time on implementing the toy example.

References

[1] Avishek Joey Bose, Gauthier Gidel, Hugo Berard, Andre Cianflone, Pascal Vincent, Simon Lacoste-Julien, and William L. Hamilton. Adversarial example games, 2020.

[2] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning polynomials with neural networks. In Eric P. Xing and Tony Jebara, editors, Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 1908–1916, Bejing, China, 22–24 Jun 2014. PMLR