Adversarial Poisson Learning

Quinten Norton
University of Minnesota

Spring 2021

Abstract

In this paper, we discuss the adversarial learning problem as applied to *Poisson learning*, a graph-based machine learning algorithm which provides good accuracy at very low label rates. First, we introduce graph-based learning and Laplacian learning, and use these as a stepping stone for the introduction of Poisson learning. Then, we discuss the adversarial learning problem, and continue on to discuss the main result of this paper, adversarial Poisson learning. We supply a derivation of an optimal perturbation matrix, as well as provide computational evidence that our derived perturbation performs worse than random perturbations.

1 Introduction

The purpose of any machine learning algorithm is to "learn" relationships between data and labels. This process moves traditionally difficult and abstract problems, such as the classic "How can a computer recognize hand-written digits?" into a solvable context. We can roughly split machine learning into three categories: (i) *fully* supervised, (ii) *semi*-supervised, and (iii) *un*supervised. In the case of fully supervised learning, we have some training datapoints $\{x_1, x_2, \dots x_m\} \subseteq \mathcal{X}$ and their corresponding *labels* $\{y_1, y_2, \dots y_m\} \subseteq \mathcal{Y}$. We use these points to "learn" a function $f : \mathcal{X} \to \mathcal{Y}$, which maps the datapoints to their labels. We have two main goals with such a function f : (i) Accuracy - it should accurately map datapoints to their correct labels, and (ii) Generalization - for $\{x_i \in \mathcal{X} : i \notin [1, m]\}$, f still gives the correct label for x. That is, f can accuratley predict the label for datapoints that it was not trained on. While such an application is mathematically grounded, it is often difficult in practice. In many contexts of machine learning, labeled data is hard to come by. Returning to the example of hand-written digits, fully supervised learning requires that someone manually label many of these digits in the first place. This can be not only costly and impractical, but perhaps counterproductive, since the purpose of machine learning is to do exactly this process automatically.

Therefore, it is often more practical to consider *unsupervised* and *semi-supervised* learning. In unsupervised learning algorithms, we make use only of the datapoints $(x_1, \dots, x_n)$, without regards for their corresponding labels $y_i \in \mathcal{Y}$. In semi-supervised learning, the algorithm learns from both labeled and unlabeled data. That is, we have labeled data $(x_1, \dots, x_m)$ and corresponding labels $(y_1, \dots, y_m)$ and unlabled data $(x_{m+1}, \dots, x_n)$. Usually we have $m << n$, for the aforementioned reasons of practicality. The goal here is to use some characteristics of the unlabled data in tandem with the labeled data to obtain better learning outcomes than using the labeled data by itself (otherwise, we would have no use of the unlabeled data). In both cases, semi-supervised and unsupervised, a popular data structure that is leveraged in such learning algorithms is a graph.

2 Graph-Based Learning

Let us now introduce *graph-based learning.* Here, we construct a graph from which we can run a machine learning algorithm on, to classify our datapoint. Concretely, let $\mathcal{G} = (\mathcal{X}, \mathcal{W})$ be a graph with verticies $\mathcal{X}$, and edge weights $\mathcal{W}$. Here, $\mathcal{X}$ are our datapoints, and the edge weights $\mathcal{W}$ encode the *similarity* between data points. So, for datapoints $x_1, x_2 \in \mathcal{X}$, the edge weight $w_{x_1x_2}$ is large if x_1 and x_2 are similar points, and small if they are dissimilar. We also work under the stipulation that the edgeweights are nonnegative. For the purpose of this paper, we will only consider *symmetric* graphs. That is, $w_{x_1x_2} = w_{x_2x_1}$ for all $x_1, x_2 \in \mathcal{X}$. Along with our graph, we still have our label space. Usually, our label space $\mathcal{Y}$ is defined as $\mathcal{Y} = \{e_1, e_2, \dots, e_k\}$ for a dataset with k classes, where $e_i \in \mathbb{R}^k$ is the i-th standard basis vector in $\mathbb{R}^k$, and defines the i-th class.

Suppose we have some subset $\Gamma \subset \mathcal{X}$, for which we are given labels $g : \Gamma \to \mathcal{Y}$. Then, the idea of graph-based semi-supervised learning is to use these labels and our graph $\mathcal{G}$, to accurately predict the labels for the rest of the datapoints $\mathcal{X} \backslash \Gamma$. We attempt to do this by leveraging the *semi-supervised smoothness assumption.* This principal dictates that similar verticies in dense regions of a graph should have similar labels. We can enfoce this by first defining a smoothness function $\mathcal{E}$, which measures the smoothness of a labeling function $u : \mathcal{X} \to \mathbb{R}^k$. Here, $u(x)$ is a k-dimensional vector for which each component u_i suggests the likelyhood that x should be labeled as class i. Then, we solve the optimization problem:

$$\begin{cases} \text{Minimize } \mathcal{E}(u) \text{ over } u \\ \text{subject to } u(x) = g(x) \text{ when } x \in \Gamma \end{cases}. \tag{1}$$

Upon solving this optimization problem, we can extrapolate our learned label $\ell(x)$ as

$$\ell(x) = \underset{i \in \{1, \dots, k\}}{\text{argmax}} \{u_i(x)\}.$$

Of course, this begs the question: how is our smoothness function $\mathcal{E}$ defined? In the following sections we will discuss some choices of $\mathcal{E}$ that are of interest.

2.1 Laplacian Learning

One of the most widely used smoothness functions is the graph Dirichlet energy

$$\mathcal{E}_2(u) := \frac{1}{2} \sum_{x,y \in \mathcal{X}} w_{xy} |u(y) - u(x)|^2.$$

When we set $\mathcal{E} = \mathcal{E}_2$ in (1), we get what is known as *Laplacian regulatization.* Due to Laplacian regularization's populatiry as a method of semi-supervised learning, there has arisen many variants. A notable one is p-Laplace regulatization, where we define

$$\mathcal{E}_p(u) := \frac{1}{2} \sum_{x,y \in \mathcal{X}} w_{xy} |u(y) - u(x)|^p.$$

and set $\mathcal{E} = \mathcal{E}_p$ in (1). The p-Laplace problem is especially interesting, as it has been shown that large values of p make $\mathcal{E}_p$ a better regularizer in cases where there are very few labels in a dataset, when $\mathcal{E}_2$ in contrast yields poor results. The $p = \infty$ case, called Lipshitz learning, has also been studied in this context.

While large values of p do improve upon Laplace learning in low label rate contexts, larger values of p result in greater computational burden, imposing limits on the practicality of their use. Furthermore, these cases of p-Laplace learning do not address a fundamental issue of any Laplace learning algorithm at low label rates. A large constant bias appears in the solution of the Laplace equation that can be credited for *nearly all* of the observed breakdown in Laplace learning in low label rates. This bias *only* appears at low label rates, and it is what *Poisson learning* seeks to address.

2.2 Poisson Learning

Let us begin with an interpretation of this constant bias as a random walk. Consider again our graph, $\mathcal{G} = (\mathcal{X}, \mathcal{W})$. Let us define the degree d_i of the vertex x_i as $\sum_{j=1}^{n} w_{ij}$ (where $n = |\mathcal{X}|$), i.e. the sum of the edge weights connected to x_i. Finally, suppose we have labeled points $(x_1, x_2, \ldots, x_m)$ and unlabeled points $(x_{m+1}, \ldots, x_n)$. Then, for $x \in \mathcal{X}$ let $\mathcal{X}_0^x, \mathcal{X}_1^x, \ldots$ be a random walk on $\mathcal{X}$ starting at x, with transition probabilities

$$\Pr(\mathcal{X}_{k+1}^x = x_j | \mathcal{X}_k^x = x_i) = \frac{w_{ij}}{d_i}$$

Let us define the stopping time τ of this random walk to be the time it takes for a walker to hit a label for the first time:

$$\tau = \inf \left\{ k \geq 0 : \mathcal{X}_k^x \in \{x_1, \ldots, x_m\} \right\}.$$

If we define i_τ as the index of the point $\mathcal{X}_\tau^x$ (i.e. $\mathcal{X}_\tau^x = x_{i_\tau}$), it turns out we can write

$$u(x) = \mathbb{E}[y_{i_\tau}].$$

where $u(x)$ is the solution to the Laplace learning problem. In this interpretation, we are letting a random walker walk from x until it hits a labeled point, and we are recording that label y_{i_τ}, which is a standard basis vector as described earlier. We then average this recorded label over many random walkers to get our solution $u(x)$.

When there is a very low label rate, the stopping time τ becomes so large that the distribution of our random walk $\mathcal{X}_\tau^x$ becomes very similar to the invariant distribution of the random walk. This gives us a solution

$$u(x) \approx \frac{\sum_{i=1}^m d_i y_i}{\sum_{j=1}^m d_j} =: \overline{y}_w. \tag{2}$$

u is not a constant function, and so this approximation $\overline{y}_w$ differs from the actual u. In particular, it differs dramatically at labeled verticies, where u spikes sharply. As we will discuss later, simply subtracting off this quantity $\overline{y}_w$ from our Laplacian solution leads to a significant accuracy increase at very low label rates.

In *Poisson learning*, we instead release a random walker from the labeled verticies, and keep counts for each time it visits any particular vertex. That is, we define

$$w_T(x_i) = \mathbb{E}\left[\sum_{k=0}^{T}\sum_{j=1}^{m} y_j 1_{\{\mathcal{X}_k^{x_j} = x_i\}}\right]$$

where $1_{\{\mathcal{X}_k^{x_j} = x_i\}}$ is the indicator function for the random walker visiting x_i, and T is the time we run our random walk for. For small values of T, this is a meaningful quantity, but for large values we run into the same issue as Laplace learning, where we creep closer to the invariant distribution. However, at a high level, we can discard the long-term behavior such that we only record the meaningful short term behavoir by modifying w_T to

$$u_T(x_i) = \mathbb{E}\left[\sum_{k=0}^{T}\frac{1}{d_i}\sum_{j=1}^{m}(y_j - \overline{y}) 1_{\{\mathcal{X}_k^{x_j} = x_i\}}\right]$$

where $\overline{y} = \frac{1}{m}\sum_{j=1}^{m}$. As $T \to \infty$, this quantity converges to the solution of the Poisson equation, which we are now ready to define.

Let $\overline{y}$ be as previously defined, and let $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ if $i \neq j$. Then, we have the Poisson equation

$$\mathcal{L}u(x_i) = \sum_{j=1}^{m}(y_j - \overline{y})\delta_{ij} \tag{3}$$

satisfying $\sum_{i=1}^{n} d_i u(x_i) = 0$. Upon solving the Poisson equation, we can extrapolate our learned label

$$\ell(x_i) = \operatorname*{argmax}_{j\in\{1,\ldots,k\}} \{u_i(x) \cdot e_j\}.$$

As stated with the random walk interpretation, this equation performs much better at low label rates when compared to the Laplace equation. However, it is a natural question to wonder how exactly these two learning equations relate.

As it turns out, a lot of the degradation in accuracy of Laplace learning at low label rates stems from $\overline{y}_w$, as defined in (2). Simply subtracting this constant off yields much greater accuracy. Concretely, we solve the Laplace equation and choose a learned label

$$\ell(x) = \operatorname*{argmax}_{i\in\{1,\ldots,k\}} \{u_i(x) - \overline{y}_w \cdot e_j\}.$$

However, while this does increase the accuracy, it is not exactly grounded, since this is essentially solving the Laplace equation for shifted labels $y_j - \overline{y}_w$. It does not seem reasonable to have to use the wrong labels to achieve good accuracy. Poisson learning offers a more well-grounded solution, but does retain a heuristic connection to this label-shifted Laplace equation.

To see this connection, lets assume that the solution u of the Laplace learning equation is roughly equal to $\overline{y}_w$ at all unlabeled points x_i, $m+1 \leq i \leq n$. However, at the labeled points x_i, $1 \leq i \leq m$, we can compute

$$\begin{aligned}\mathcal{L}u(x_i) &= \sum_{j=1}^{n} w_{ij}(u(x_i) - u(x_j)) \\ &\approx \sum_{j=m+1}^{n} w_{ij}(y_i - \overline{y}_w) = d_i(y_i - \overline{y}_w).\end{aligned}$$

Since $\mathcal{L}u(x_i) = 0$ for the unlabled points $m+1 \leq i \leq n$, we can see that u approximately satisfies the Poisson equation

$$\mathcal{L}u(x_i) = \sum_{j=1}^{m} d_j(y_j - \overline{y}_w)\delta_{ij}.$$

Despite this connection it shares to the Laplace equation, the Poisson equation does fundamentally differ in computation. Hence, the Poisson learning equation deserves its own examination in the many contexts of machine learning. For the remainder of this paper, we will focus on one such context. Namely, we will look at adversarial machine learning, and its details in the Poisson equation.

3 The Adversarial Learning Problem

Consider again our approaches to machine learning, from fully supervised to unsupervised methods. In any case, our algorithms are "learning" from some combination of datapoints and labels. Adversarial machine learning examines the consequences of modifying these datapoints and labels. Consider, for example, a self driving car. Such a car relies heavily on computer vision, to "see" and interpret its surroundings. In this case, accuracy is of the utmost importance. If your car fails to interperet a stop sign as a stop sign, then your car won't stop! While we can make many modifications to the machine learning algorithms that we use in order to ensure the best accuracy possible, we have to make other considerations. Namely, we must consider that the information that our model is using, whether that be the datapoints themselves or the labels, could be supplied or corrupted with malicious intent. By supplying bad data, malicious adversaries can worsen the accuracy of a model that may otherwise be very accurate. Thus, it is desirable for an algorithm to carry with it some robustness to such attacks.

While we will not suggest any modifications to the Poisson equation, we will examine what such malicious attacks could look like in the context of the Poisson equation.

3.1 Adversarial Poisson Learning

Let's again revert to our graph $\mathcal{G} = (\mathcal{X}, \mathcal{W})$, and assume that an adversary is able to affect $\mathcal{W}$ by corrupting either the training data or the unlabeled data. In the most abstract setting, the adversary changes the weight matrix from $\mathcal{W}$ to $\mathcal{W} + \epsilon\mathcal{V}$, where $\epsilon > 0$ and $\mathcal{V}$ is a symmetric matrix chosen by the adversary. We assume that $\mathcal{W} + \epsilon\mathcal{V}$ is nonnegative, and as such is a valid weight matrix. Let's write the corresponding solution as $u_\epsilon \approx u + \epsilon v$, for a perturbed solution v. We also denote by $\mathcal{L}_\mathcal{W}$ the Laplacian matrix for $\mathcal{W}$. We can write the Poisson equation (3) for the perturbed weight matrix as

$$\mathcal{L}_{\mathcal{W}+\epsilon\mathcal{V}}(u + \epsilon v) = f,$$

where f denotes the right hand side of (3). Note we can write this as

$$(\mathcal{L}_\mathcal{W} + \epsilon\mathcal{L}_\mathcal{V})(u + \epsilon v) = f,$$

since $\mathcal{L}_\mathcal{W}$ is linear in $\mathcal{W}$. This can be expanded as

$$\mathcal{L}_\mathcal{W} u + \epsilon(\mathcal{L}_\mathcal{V} u + \mathcal{L}_\mathcal{W} v) + \epsilon^2 \mathcal{L}_\mathcal{V} v = f.$$

Discarding the ϵ^2 term and using that $\mathcal{L}_\mathcal{W} = f$ we arrive at

$$\mathcal{L}_\mathcal{W} v = -\mathcal{L}_\mathcal{V} u.$$

Let's suppose the adversary has a bound on the size of their perturbation of the weight matrix $\|\mathcal{V}\| \leq 1$. A reasonable goal for the adversary is to choose $\mathcal{V}$, within the constraint $\|\mathcal{V}\| \leq 1$, in order to maximize some norm of v, say $\|v\|$ or $\|\mathcal{L}v\|$. The latter is more convenient, due to the expression above for $\mathcal{L}v$. So it is reasonable for the adversary to try and solve the problem

$$\max_{\|\mathcal{V}\| \leq 1} \left\{ \|\mathcal{L}_\mathcal{V} u\| \right\}.$$

Note that

$$f(\mathcal{V}) := \frac{1}{2}\|L_{\mathcal{V}}u\|^2 = \frac{1}{2}\sum_{i=1}^{n}\left\|\sum_{j=1}^{n}\mathcal{V}_{ij}(u(x_i)-u(x_j))\right\|^2$$
$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{p=1}^{k}\left(\sum_{j=1}^{n}\mathcal{V}_{ij}(u_p(x_i)-u_p(x_j))\right)^2,$$

where $u(x_i) = (u_1(x_i), u_2(x_i), \ldots, u_k(x_i)) \in \mathbb{R}^k$. We want to maximize f over $\mathcal{V}$, with the restriction that $\mathcal{V}$ is symmetric, nonnegative, and $\|\mathcal{V}\| \leq 1$. The gradient $\nabla f(\mathcal{V})$ is the $n \times n$ matrix whose (i,j) entry is the partial derivative of f in $\mathcal{V}_{ij}$. We compute

$$\nabla f(\mathcal{V})_{q,\ell} = \frac{\partial f}{\partial \mathcal{V}_{q,\ell}}\frac{1}{2}\sum_{i=1}^{n}\sum_{p=1}^{k}\left(\sum_{j=1}^{n}\mathcal{V}_{ij}(u_p(x_i)-u_p(x_j))\right)^2$$
$$= \sum_{i=1}^{n}\sum_{p=1}^{k}\sum_{j=1}^{n}\mathcal{V}_{ij}(u_p(x_i)-u_p(x_j))\frac{\partial f}{\partial \mathcal{V}_{q,\ell}}\left(\mathcal{V}_{ij}(u_p(x_i)-u_p(x_j))\right)$$
$$= \sum_{i=1}^{n}\sum_{p=1}^{k}\sum_{j=1}^{n}\mathcal{V}_{ij}(u_p(x_i)-u_p(x_j))\delta_{i=q}\delta_{j=\ell}(u_p(x_i)-u_p(x_j)))$$
$$= \mathcal{V}_{q\ell}\sum_{p=1}^{k}(u_p(x_q)-u_p(x_\ell))^2.$$

Let ∇u_p denote the matrix whose (i,j) entry is $u_p(x_i) - u_p(x_j)$. Then we have

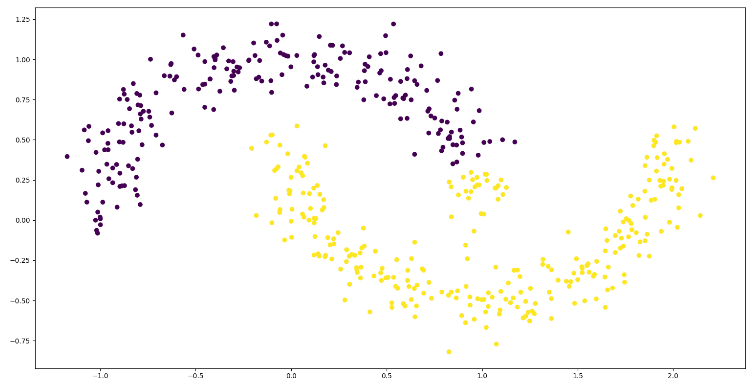
$$\nabla f(\mathcal{V}) = \mathcal{V} \odot \sum_{p=1}^{k}(\nabla u_p)^2,$$

where the notation $\odot$ refers to element-wise multiplication of matrices, so

$$(A \odot B)_{ij} = A_{ij}B_{ij}.$$

4 Implementation

To implement our adversarial perturbation, we used the `graphlearning` python package, available at https://github.com/jwcalder/GraphLearning. Included in this package is the capability to run Poisson learning on a dataset.We included two datasets, one being a ”moons” set created synthetically via the function `scipy.datasets.make_moons()`:



and the other being MNIST's handwritten digits set. We defined the following two functions:

```python
def run_grad(W, u, num_classes, num_iterations):
    # create a ones matrix B, where B_ij = 1 iff W_ij>0.
    B = 1 * (W > 0)
    V = B.copy()
    # normalize V
    normV = np.sqrt(np.sum(V.multiply(V)))
    V = V / (normV)
    dt = 0.01
    gradu = []

    for _ in range(num_iterations):
        for i in range(num_classes):
            gradu.append(gl.graph_gradient(B, u[:, i]))

        sum_gradu_squared = gradu[0].multiply(gradu[0])
        for i in range(1, num_classes):
            sum_gradu_squared = sum_gradu_squared + gradu[i].multiply(gradu[i])

        gradV = V.multiply(sum_gradu_squared)
        V = V + dt * gradV

        # project V to constraint ||V|| < 1
        normV = np.sqrt(np.sum(V.multiply(V)))
        V = V / (normV)
    return V


def random_perturb(W, dimension):
    I, J, V = sparse.find(W)
    V = sparse.coo_matrix(
        (np.random.rand(len(I)), (I, J)), shape=(dimension, dimension)
    ).tocsr()
    # symmetrize
    V = (V + V.T) / 2
    # project V to constraint ||V||<1
    normV = np.sqrt(np.sum(V.multiply(V)))
    V = V / (normV)
    return sparse.csr_matrix(V)
```

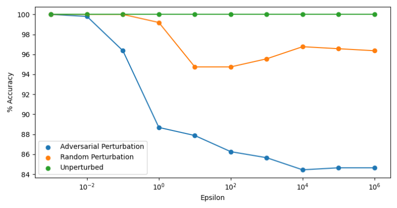
Here, `run_grad` computes `num_iterations` of gradient ascent, to calculate the adversaial perturbation described in section 3.1. Additionally, `random_perturb` creates a matrix with nonnegative random values between 0 and 1. Note that both of these matricies are created with the additional constraint that no new edges are created. That is,

$$\mathcal{W}_{ij} = 0 \implies \mathcal{V}_{ij} = 0$$

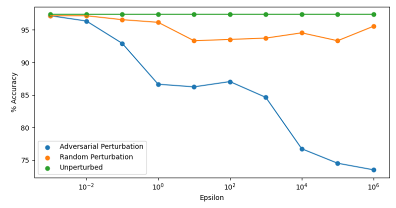
Equipped with these two functions and the `graphlearning` package, we then designed a protocol to test how our adversarial perturbation matrix performed versus a random perturbation matrix, in the context of Poisson learning. We selected a range of epsilon values, and compared the resultant accuracy from Poisson learning of our adversarially perturbed matrix $\mathcal{W} + \epsilon\mathcal{V}_{\text{adversary}}$ to the average of 5 randomly perturbed matricies $\mathcal{W}+\epsilon\mathcal{V}_{\text{random}}$ for each value of epsilon.

5 Results

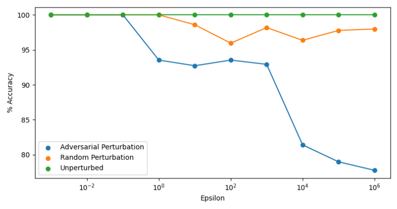
We computed four trials for each dataset, using epsilon values ranging from 10^{-3} to 10^{7}. The results of the synthetic moons dataset are displayed in Figure 1, and the results of the MNIST dataset are displayed in Figure 2. While all cases saw a significant decrease in accuracy in the adversarial case, they did range in how large epsilon needed to be before this effect really began to manifest. Interestingly, some cases saw an *increase* in accuracy for these lower values of epsilon. This is most noticiable in trial 1 of the MNIST dataset. While these cases are puzzling, it may be noteworthy to recall that we imposed the constraint $||\mathcal{V}|| \leq 1$. For matricies that are large (which is especially true in the case of the MNIST dataset), this is quite a restrictive constraint, as the normilazation factor might be very large. Hence, the values of the pertubation matrix $\mathcal{V}$ may be very small. Consequently, it may not be of great concern that we see such results, as the perturbations are perhaps too small for our calculation (which is an approximation) to be accurate. Furthermore, since these perturbation matricies in the presumably more reasonable range of $\epsilon > 10^4$ always yield the expected results, we can at the very least have confidence in our results with the caveat the epsilon is sufficiently sized. In any case, our theoritical results are largely validated experimentally.



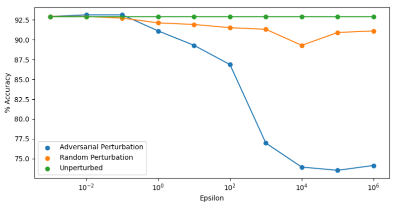
(a) moons trial 1



(b) moons trial 2

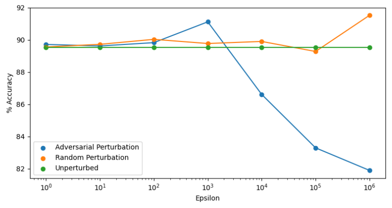


(c) moons trial 3

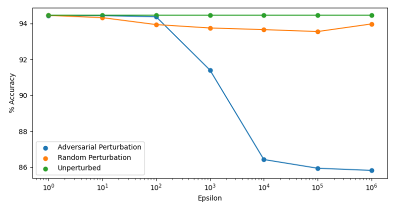


(d) moons trial 4

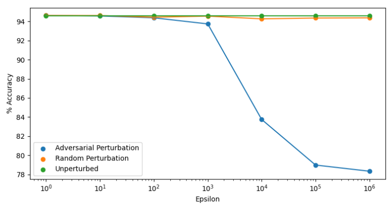
Figure 1: Results on synthetic moons dataset



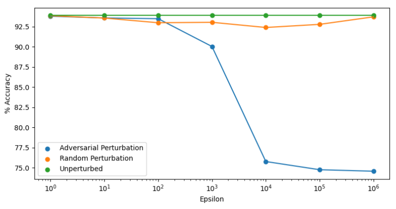
(a) MNIST trial 1



(b) MNIST trial 2



(c) MNIST trial 3



(d) MNIST trial 4

Figure 2: Results on MNIST handwritten digits

6 Conclusion

We have introduced machine learning in an abstract sense, and used it to discuss the semi-supervised learning method, graph learning. We then discussed

Laplacian learning as a popular method which abides by the semi-supervised smoothness assumption. From this, Poisson learning was introduced in the context of random walks. It was interpreted as a way to address the break down in the Laplace learning framework in low label rate contexts. After formally defining and providing some intuition to the Poisson learning equation, we computed an adversarial machine learning method in which the weight matrix of a graph created for semi-supervised learnig could be suitably perturbed via gradient *ascent*. Finally, we proposed a computationally simple approximation of our adversrial perturbation matrix, and compared its effictiveness against random perturbation matricies.