Poisoning Attacks with Back-gradient optimization on Physics-informed Turbulent Flow prediction

Siddharth Satyam

December 10, 2022

Abstract

This is a project report for ECE 299, research units. In recent times, deep learning has become one of the most important research area, while also being rapidly used in the software industry. While cloud technologies have provided a way to use these computationally intensive algorithms on remote devices, it has also made them prone to data poisoning attacks, where the attacker can feed malicious data points either in the training or inference phase, and can cause system malfunction. On another side, physics-informed deep learning has gained popularity of late, and so this project aims to explore data poisoning attacks on such models. The model selected for the attack is a model for velocity field prediction in a turbulent flow field.

Keywords: Data poisoning, Turbulent Flow, Deep Learning

1 Introduction

In recent times, physics-informed deep learning models have gained popularity, and extensive research has been done in the domain. While Deep Learning has been previously employed for physical problems, its implementation has been primarily done in a black-box manner. For example, the prediction of turbulent velocity fields has been done through GANs, where the distribution of the velocity fields was generated directly by an adversarial training process with the true data distribution of the velocity field, disregarding the physical principles underlying the turbulent flow phenomenon. While methods like Computational Fluid Dynamics take care of the physical principles to compute the velocity field and give the best results, the numerical simulation involved in the method is very computationally expensive. In that case, a deep learning model which is informed of the physics behind the computation works well and gives better results compared to its counterparts that use Deep Learning.

1.1 Turbulent Flow Model

TF-net [2] is a model to predict the velocity field at the next time step in a turbulent flow by learning the interactions between the different velocity scales at the present time step. The velocity field can be represented as the sum of three components,

$$W = \bar{w} + \tilde{w} + w', \tag{1}$$

Where $\bar{w}$ is Mean flow, $\tilde{w}$ is spatially filtered (resolved) fluctuations, and w' is temporally filtered (unresolved, subgrid) fluctuations. Usually, spatial filtering is done by a convolution with a Gaussian kernel, and temporal filtering is done by averaging across time steps. The TF-net model uses a 1-layer CNN with a 5×5 filter for spatial filtering and a 1-layer CNN with a 1×1 filter for temporal filtering. The three velocity scales are encoded, and their interactions are learned and decoded to predict the overall velocity field at the next time step. The encoding and decoding model is a Unet. The model is shown in fig 1.

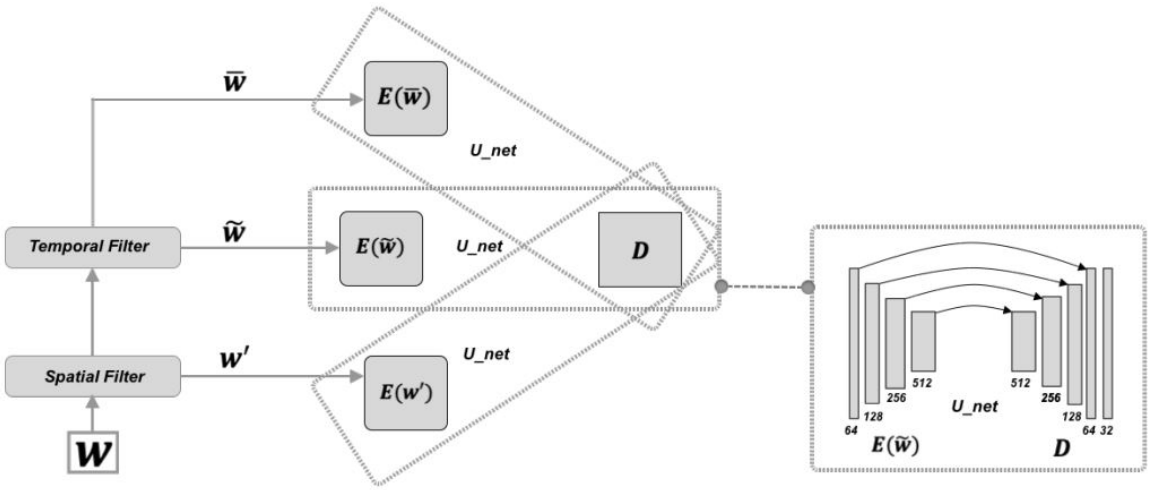

Figure 1: Turbulent Flow Net: three identical encoders to learn the transformations of the three components of different scales, and one shared decoder that learns the interactions among these three components to generate the predicted 2D velocity field at the next instant.

As physics-informed models have become a hot topic for research, several backdoor attack strategies can be employed. In this work, an untargeted backdoor attack has been performed on TF-net to generate poisoned data samples. These poisoned data samples, when mixed with the training data, can lead to worsening the accuracy of TF-net.

2 Methodology

The model for poisoning the TF-net model is derived from [1]. The poisoning model discussed in [1] accounts for error-generic and error-specific attacks in multiclass classification problems. The idea is extended in this work to regression cases such as predicting velocity fields at next time steps. Let us first discuss the attack model in [1].

2.1 Attack Model

The general attack strategy can be characterized in terms of an objective function A(Dc',theta) which evaluates the effectiveness of the attack. The attacker knows the model parameters theta, where Dc' are the poisoned data samples. The general attack strategy is given as:

$$\mathcal{D}_c^\star = \underset{\mathcal{D}_c' \in \phi(\mathcal{D}_c)}{\operatorname{argmax}} \mathcal{A}(\mathcal{D}_c', \theta) \tag{2}$$

where $\mathcal{D}_c^\star$ is the optimal poisoned data obtained after an optimization process. Since we have to extend the idea to regression problems, we will only focus on error-generic mis-classifications as they can be easily used for our case. It uses a Bi-Level optimization technique to optimize the poison data points, which is given by:

$$\begin{aligned} \mathrm{D}_c^\star \in & \underset{\mathcal{D}_c' \in \phi(\mathcal{D}_c)}{\operatorname{argmax}} \mathcal{A}(\mathcal{D}_c', \theta) = \mathcal{L}(\hat{\mathcal{D}}_{val}, \hat{w}) \\ \text{s.t.} \quad & \in \underset{w' \in \mathcal{W}}{\operatorname{argmin}} \mathcal{L}(\hat{\mathcal{D}}_{tr} \cup \mathcal{D}_c', w') \end{aligned}$$

We first initialize $\mathcal{D}_c'$ according to our problem. For multiclass-classification problems, the label is randomly changed. For regression, the true values can be changed to random values

sampled from any distribution. Firstly, the model is trained with the clean training set added with the initialized poisoned data. Then the performance is evaluated by the objective function A, which is the Validation Loss, which has to be maximized after the optimization. Then the poisoned data Dc' is optimized using a back gradient optimization approach, and the process is repeated. The poisoning algorithm is given by:

Algorithm 1 Poisoning Attack Algorithm

Input: $\mathcal{D}_{tr}$, $\mathcal{D}_{val}$, $\mathcal{L}$, the initial poisoning point x_c^0, its label y_c, the learning rate η

for i=0 **until** $\mathcal{A}(x_c^i, y_c)$ - $\mathcal{A}(x_c^{i-1}, y_c) < \epsilon$ **do**

$\hat{w} \in \underset{w' \in \mathcal{W}}{\text{argmin}}\ \mathcal{L}(\text{x}_c^i, \text{w'})$ (train learning algorithm)

$x_c^i \leftarrow \Pi_\phi(x_c^i + \eta \nabla_{x_c} \mathcal{A}(x_c^i, y_c))$

end for

Output: The final poisoning point $x_c \leftarrow x_c^i$
=0

Where the gradients $\nabla_{x_c}\mathcal{A}(x_c^i, y_c)$ are computed through the back gradient optimization approach discussed in detail in [1]. The idea of back gradient optimization is to compute the gradient $\nabla_{x_c}\mathcal{A}(x_c^i, y_c)$ through reverse-mode (automatic) differentiation (i.e., back-propagation) while reversing the underlying learning procedure to trace back the entire sequence of parameter updates performed during learning, without storing it.

3 Experiments

3.1 Attack on MNIST Classification

To understand the algorithm, an untargeted, i.e., an error-generic attack, was performed on a multi-class classification problem. The MNIST dataset consists of a 60000 training set and 10000 test set images of 28×28 handwritten digits from 0-9. A PyTorch model was built to classify the images into 10 classes. For the poisoning model, the labels were switched to a random class and then the Bi-Level optimization was performed. The gradients were computed using Automatic differentiation. The generated poisoned points are shown in fig. 2.

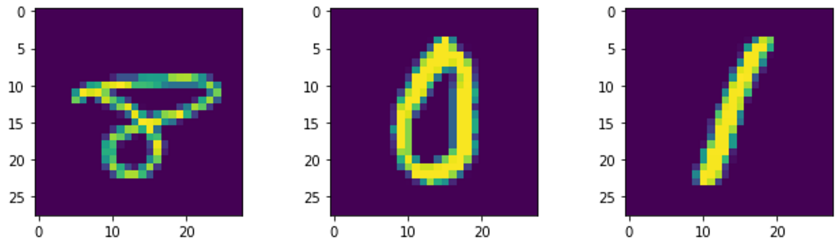

Figure 2: The poison data points generated for the MNIST digit classification model.

To evaluate the performance of the generated poisoned points, they were randomly picked up and mixed with the train set, and then the classification model was trained on the mixed data set. The validation loss should increase as we increase the percentage of poisoned points in the training set, as shown in fig. 3.

3.2 Attack on Turbulent Flow Model

As discussed in the Introduction, the model used for untargeted poisoning attacks is used from [2]. The dataset in [2] comes from a two-dimensional turbulent flow simulated using the

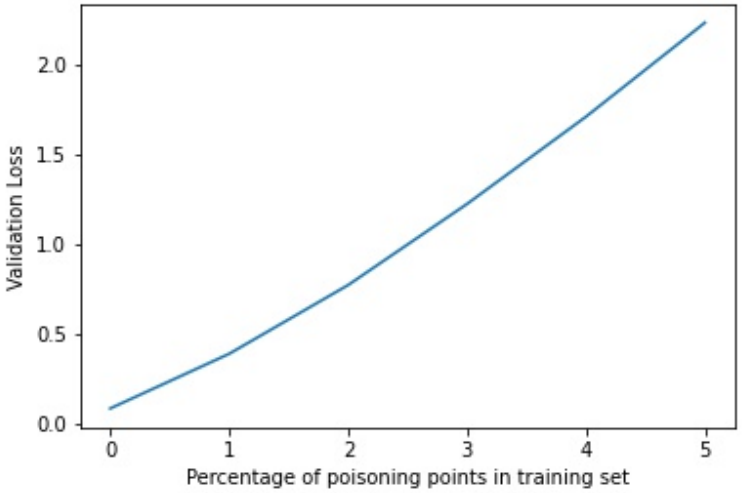

Figure 3: **MNIST Classification:** The loss on the validation dataset i.e., the objective function increases as we increase the percentage of poisoned samples.

Lattice-Boltzmann method. The velocity vector fields are represented as 1792×256 images, divided into 7 square sub-regions of size 256×256, and then downsampled to 64×64 images. After pre-processing the data, the dataset consists of 6000 training data points with X_i of size $62 \times 64 \times 64$ and Y_i of size $4 \times 2 \times 64 \times 64$. The Turbulent Flow model accepts a $62 \times 64 \times 64$ input where 62 is the number of input time steps. It produces an output of $2 \times 64 \times 64$, where 2 is the number of predicted time steps. After each prediction of two time steps, the first two time steps are discarded from the input, and the two predicted timesteps are appended to retain the size of $62 \times 64 \times 64$. This process is performed four times to get the velocity field for $4 \times 2 = 8$ time steps in the future.

A poisoning model was created to perform an error-generic attack, for which a poisoning point was initialized, considering it a regression problem. The label Y_i of size $4 \times 2 \times 64 \times 64$ was changed and sampled from a standard normal distribution. For an initialized poison point, three iterations of the Bi-Level optimization algorithm were performed. In each iteration, firstly, the Turbulent Flow model was trained with the training set mixed with the poisoned point for 10 epochs. In the training epochs, the model was trained only for a maximum of 3 batches, picked up randomly, to ensure that the gradients do not overshoot. Then, a back-gradient optimization was performed, where the weights were reversely computed for each epoch to yield the gradient for the objective function in the end. The poison point was updated by a gradient ascent using the gradient of the objective function computed.

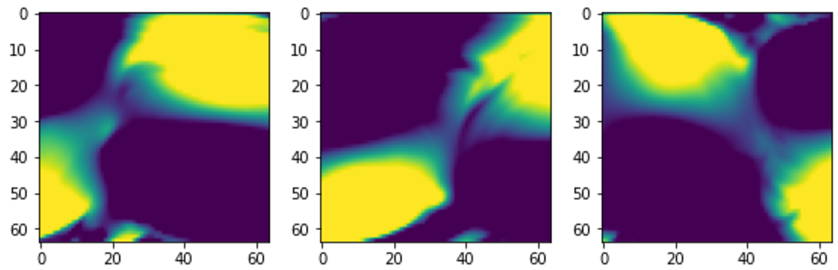

Figure 4: Some of the poisoning points generated by the poisoning algorithm.

Fifty poison points were generated by the process. The Bi-Level optimization worked cor-

rectly. This can be seen from the training logs in fig 5. as the Objective Function A i.e., the validation loss increased at every iteration, while the Turbulent model was trained and the training loss decreased. The poison points were also evaluated by training the Turbulent Flow model with a poisoned data set. As shown in fig 6, the validation loss roughly increases as we increase the poison percentage.

```
Epoch 1/10, Batches 3/3, Loss: 2.267194
Epoch 6/10, Batches 3/3, Loss: 0.970871
Epoch 10/10, Batches 3/3, Loss: 0.893654
Reverse Epoch 1/10
Reverse Epoch 6/10
Reverse Epoch 10/10
Poisoning point: 3, Iteration: 0, Cost: 1.3316
Epoch 1/10, Batches 3/3, Loss: 9.661795
Epoch 6/10, Batches 3/3, Loss: 3.445448
Epoch 10/10, Batches 3/3, Loss: 2.909699
Reverse Epoch 1/10
Reverse Epoch 6/10
Reverse Epoch 10/10
Poisoning point: 3, Iteration: 1, Cost: 4.1839
Epoch 1/10, Batches 3/3, Loss: 38.065411
Epoch 6/10, Batches 3/3, Loss: 12.933904
Epoch 10/10, Batches 3/3, Loss: 10.760141
Reverse Epoch 1/10
Reverse Epoch 6/10
Reverse Epoch 10/10
Poisoning point: 3, Iteration: 2, Cost: 14.6877
```

```
Epoch 1/10, Batches 3/3, Loss: 3.290290
Epoch 6/10, Batches 3/3, Loss: 1.280591
Epoch 10/10, Batches 3/3, Loss: 1.099178
Reverse Epoch 1/10
Reverse Epoch 6/10
Reverse Epoch 10/10
Poisoning point: 5, Iteration: 0, Cost: 1.6224
Epoch 1/10, Batches 3/3, Loss: 18.347996
Epoch 6/10, Batches 3/3, Loss: 5.548697
Epoch 10/10, Batches 3/3, Loss: 4.663145
Reverse Epoch 1/10
Reverse Epoch 6/10
Reverse Epoch 10/10
Poisoning point: 5, Iteration: 1, Cost: 6.5737
Epoch 1/10, Batches 3/3, Loss: 41.915619
Epoch 6/10, Batches 3/3, Loss: 18.836065
Epoch 10/10, Batches 3/3, Loss: 16.655834
Reverse Epoch 1/10
Reverse Epoch 6/10
Reverse Epoch 10/10
Poisoning point: 5, Iteration: 2, Cost: 22.3470
```

Figure 5: **TF-net:** The training logs for 2 poisoning points. The training loss has a decreasing trend in every iteration of the optimization. Also, the Validation Loss is increasing in every iteration.

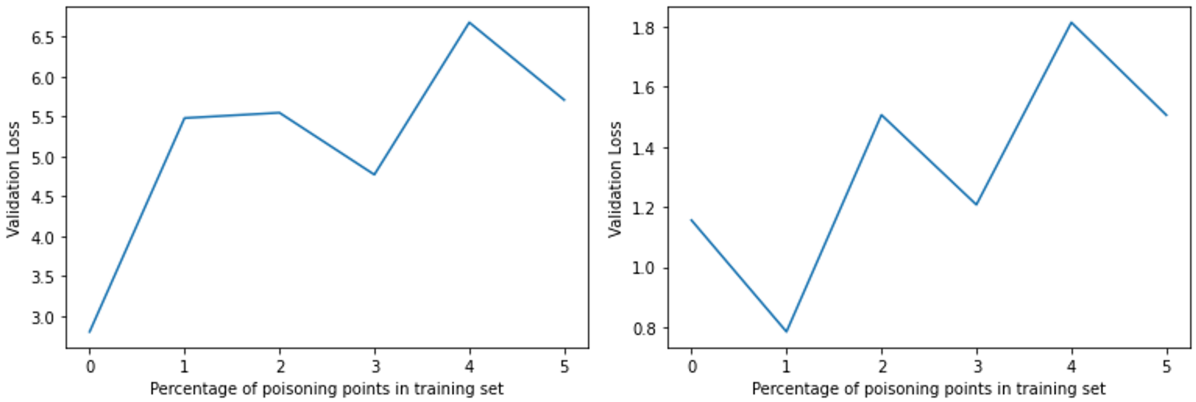

Figure 6: Validation Loss when the model is trained with 320 training points (left), 500 training points (right)

4 Conclusion and Future Work

Presently, the model training was restricted to 500 data points due to the long training times and the large memory consumption by the process. The experiments can be conducted on the full training set i.e. 6000 training points. Also, presently we have around 70 poisoning points which can be increased to see the model performance on the full train set.

References

[1] Muñoz-González, Luis, et al. "Towards poisoning of deep learning algorithms with back-gradient optimization." Proceedings of the 10th ACM workshop on artificial intelligence and security. 2017. doi: 10.1145/3128572.3140451

[2] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. 2020. Towards Physics-informed Deep Learning for Turbulent Flow Prediction. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining (KDD '20). Association for Computing Machinery, New York, NY, USA, 1457–1466. https://doi.org/10.1145/3394486.3403198