Adversarial Learning: Backdoor Attacks

Alex Useloff
Case Western Reserve University
Cleveland, OH
adu3@case.edu

Literature Review

The first paper that I read is titled "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain", and it was written by Gu et al. [1]. This paper was one of the earliest I could find that talked about the backdoor attack on deep networks, and it was also mentioned in almost every other paper I looked at on backdoor attacks, so it seemed like a good place to start. It starts off by just going over what neural networks are in general, what they used for, how they are used, and other things of a similar nature [1]. It then goes on to talk about how complicated neural networks can be because of the massive amounts of training data that is required for them to achieve good results [1]. Since it would take too much computational power for most individuals and even businesses to train these complicated networks, many of them are turning to third party providers such as Amazon, Google, and Microsoft to provide them with pre-trained networks, and this is sometimes referred to as "machine learning as a service" (MLaaS) [1]. Once a company or individual receives a pre-trained network from one of these companies, they can then fine-tune the network to better suit their specific needs [1]. In addition to MLaas, the other way that a company or individual can reduce costs if they want a network is to simply adjust an existing model for a new task, and this is known as "transfer learning" [1].

The reason that this paper talks about these two different ways of outsourcing networks is because outsourcing networks comes with security concerns [1]. The specific security concern in this case is a backdoored neural network, or as this paper calls it, a BadNet [1]. In a backdoored neural network, either the third party that provides the pre-built network or the previous users of the network in the case of transfer learning gives the network a backdoor [1]. This network with a backdoor performs well on most of its inputs, but it can cause either targeted misclassifications or make the model less accurate for certain inputs that contain a secret trigger, and this secret is referred to as the "backdoor trigger" [1]. A good example of this comes into play with autonomous driving. An attacker can put in a backdoor so that simply putting a sticker or something similar on a stop sign will make the model think that it is a speed limit sign, which is obviously very bad because that car will not stop if it thinks that a stop sign is a speed limit sign.

An easy way of thinking about how this could work involves two separate networks that both look at the same input: one of these networks outputs the intended classification of the model, and the other detects whether or not the backdoor trigger is present [1]. Once this is done, a merging layer looks at the two different outputs [1]. If the backdoor network detected that the backdoor trigger was present, then it will produce an output chosen by the attacker, and if not, then it will just produce the normal output [1]. Although this is a nice and easy way to think about how a backdoor attack would work, it cannot be applied directly to the outsourced training scenario that was mentioned earlier because although the training is done by someone else, the architecture of the model itself is usually still specified by the user [1]. Instead, since the attacker only really has a say in how the model is trained, they must find a way to incorporate a recognizer for the backdoor trigger into a pre-specified architecture just by finding the associated weights [1]. This means that the attacker must develop a malicious training procedure based on training set poisoning that can calculate these associated weights given a training set, a backdoor trigger, and a model architecture [1]. After explaining all of this, the paper goes on to explain two real-world examples of a backdoor attack [1]. The first example involves recognizing digits in images, and the

second involves traffic sign detection in autonomous self-driving cars [1]. They are both really cool examples, but I am only really going to talk about the first one because it is what I created my algorithm to do.

As I just briefly mentioned, this example case study was done on the MNIST digit recognition task [1]. In the MNIST digit recognition task, the dataset consists of grayscale images of handwritten digits, zero through nine, and the goal is to successfully classify each digit into its correct class. For Gu et al.'s case study on the MNIST digit recognition task, their baseline network was the standard architecture for this task which consisted of a Convolutional Neural Network (CNN) with two convolutional layers and two fully connected layers [1]. This baseline CNN normally results in an accuracy of about 99.5% with the MNIST digit recognition task [1]. There were two different backdoors that Gu et al. considered [1]. The first of these was a single pixel backdoor, and this is simply a single bright pixel in the bottom right corner of the image [1]. The second backdoor was a pattern backdoor, and similar to what they did with the single pixel backdoor, they simply added a pattern of bright pixels to the bottom right corner of the image [1].

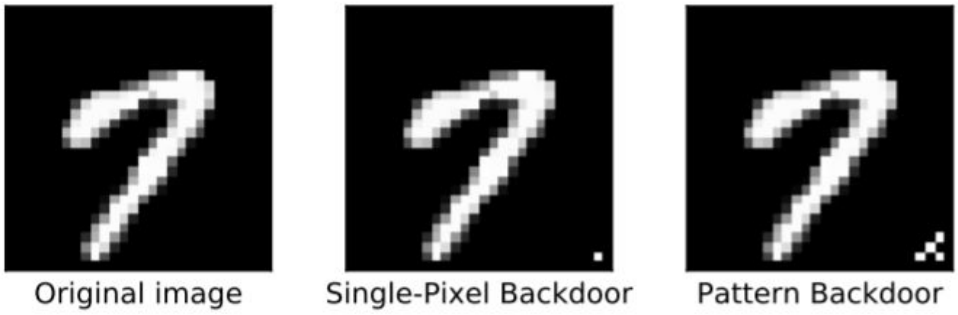

Figure 1: An original grayscale MNIST image along with two backdoored versions of that image [1].

An example of what these backdoored images look like can be seen in Figure 1. As you can see, the only real differences in the backdoored images are the pixels in the bottom right corners of them. Gu et al. also implemented two different attacks on these backdoored images [1]. They implemented a single target attack which labels backdoored images of digit i as digit j, and an all-to-all attack which changes the label of digit i to digit $i+1$ for backdoored inputs [1].

In order to implement their attack, Gu et al. had to poison the training dataset [1]. To poison the dataset, they randomly picked images from the training dataset and added backdoored versions of these images to the dataset [1]. They then re-trained the baseline CNN using the newly poisoned dataset [1]. Once this was done, it was time for them to look at their results. They started off by looking at the results of the single target attack [1]. They found that the error rate for clean images was at most 0.17% higher than and in some cases 0.05% lower than the baseline CNN [1]. For backdoored images, the error rate was at most 0.09% [1]. With the all-to-all attack, the error rate on clean images was slightly lower than the baseline by about 0.03% and the error on backdoored images was about 0.56% [1]. This means that the poisoned network successfully mislabeled more than 99% of the backdoored images, proving that the attacks were both very successful.

The next paper that I read built off of what Gu et al. wrote about in their BadNets paper. This second paper is titled "Clean-Label Backdoor Attacks", and it was written by Turner et. al [2]. They start off pretty much the same way as Gu et al. by explaining that many companies and individuals are now outsourcing the training of their networks because of how expensive it is to train them in-house [2]. Turner et al. then go on to explain that outsourcing the training of networks can raise security concerns, and in doing so, they specifically reference the backdoor attack that was proposed by Gu et al. [2]. In referencing the backdoor attack by Gu et al., Turner et al. explain at a high level what it is and how it works, but then they point out a potential flaw: that the backdoor attack "crucially relies on the assumption that the poisoned inputs introduced to the training set by the adversary can be arbitrary -

including clearly mislabeled input-label pairs" [2]. They then explain that as a result of this, it is very easy for a filtering process to detect the poisoned samples as outliers [2]. Furthermore, any human inspection done after the poisoned samples are deemed as outliers will make these inputs look suspicious, which could potentially reveal that the data was attacked [2]. It is for this reason that Turner et al. wrote this paper to explore if it is truly necessary to use such clearly mislabeled images [2].

The first thing that Turner et al. did to explore this was to apply a very simplistic data filtering technique to Gu et al.'s attack [2]. In doing so, they found out exactly what they had thought: the poisoned inputs were very easily identified as outliers by the filter, and once these outliers were looked at by a human, it was very clear that something had been done to these images [2]. Once finding out that they had been correct, Turner et al. set out to try to create a new way to poison inputs that would appear plausible to humans even if a filter deems them as outliers [2]. Their approach to getting this done consisted of making small changes to the inputs in order to make them harder to classify just like what Gu et al. did, but they wanted to keep the changes small enough so that the changed inputs still look like their original labels [2]. Turner et al. looked to use two different methods to try to do this [2]. With the first method, which they called GAN-based interpolation, they embed each input into the latent space of GAN and insert poisoned samples towards embeddings of an incorrect class [2]. The second method, called adversarial ℓ_p-bounded perturbations, involves using an optimization method to maximize the loss of a pre-trained model on the poisoned inputs while staying within an ℓ_p-ball around the original input [2].

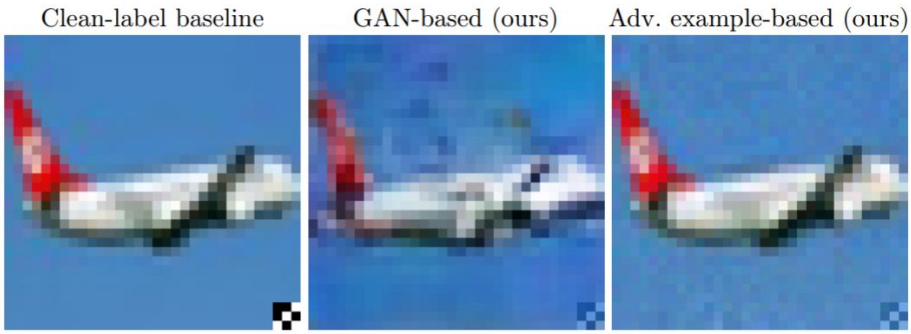

Figure 2: An example image, labeled as an airplane, poisoned using different strategies. The poisoning strategy that was used for each image, going from left to right, is the baseline of Gu. et al's attack using only clean labels, Turner et al.'s GAN-based attack, and Turner et al.'s adversarial example-based attack [2].

An example of poisoning using these two different methods can be seen in Figure 2. As you can clearly see, the pixels in the bottom right of the two rightmost images are much less noticeable than the pixels in the leftmost image. Aside from making the added pixels more difficult to see in the poisoned images, there is also another improvement that Turner et al. wanted to add to Gu et al.'s backdoor attack [2]. This improvement is to only use poisoned samples that have plausible labels, which Turner et al. refer to as *clean labels*, hence the name of the paper [2].

After making the changes that Turner et al. wanted to make to Gu et al.'s original backdoor attack, it was time for them to run some experiments with it [2]. To do this, they started off by choosing a target class label L and a fraction of training inputs to poison [2]. Next, they randomly modify these inputs as long as they stay consistent with their original label, and then they introduce a backdoor pattern to these inputs [2]. This is the clean label part of the attack that Turner et al. modified. Once this is done, a classifier is then trained on the poisoned dataset and the resulting network is evaluated [2]. Turner et al. used the CIFAR-10 dataset to run these experiments [2]. In case you are not already aware, the CIFAR-10 dataset contains 5,000 images from each of 10 different classes, and the goal is to classify each image into its correct class. For their classifier, Turner et al. used a standard residual network with three groups of residual layers [2].

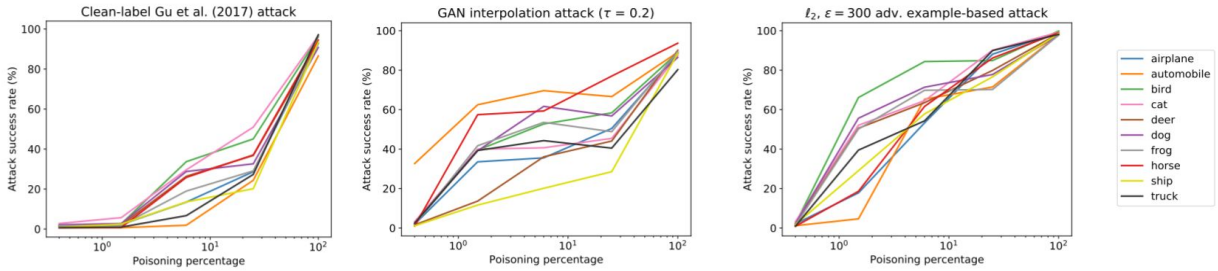

Figure 3: Attack success rate (%) vs. Poisoning percentage for each class for the three different attacks used. The poisoning strategy that was used for each chart, going from left to right, is the baseline of Gu. et al's attack using only clean labels, Turner et al.'s GAN-based attack, and Turner et al.'s adversarial example-based attack [2].

In terms of results, they found that both GAN-based interpolation and adversarial ℓ_p-bounded perturbations led to poisoned images with plausible labels [2]. As can be seen graphically in Figure 3, both approaches also significantly increase the effectiveness of the poisoning attack when compared to the baseline attack that simply introduces the backdoor trigger on clean images that was created by Gu et al. [2]. The last thing that Turner et al. noted was that the attacks based on the adversarial ℓ_p-bounded perturbations were more effective than the GAN-based interpolation, and this can also be seen clearly in Figure 3 [2].

Algorithms

The main algorithm that I created is the backdoor attack that was the main subject in the paper by Gu et al.. How the algorithm works is that it takes in three parameters: an array with the points that initialize attack points that I called "to_be_attacked", the target labels for the attack that I called "attack_target_labels", and whether or not to broadcast a single target label that I called "broadcast". Once it has input, the first thing that the algorithm does is check whether or not it needs to broadcast a single target label. If it does, then it creates a variable called "attack_labels" that is the same as "attack_target_labels" except it has the shape of "to_be_attacked" by "attack_target_labels". Otherwise, it just creates a variable called "attack_labels" which is the same as "attack_target_labels". Next, it creates a variable called "number_attacked" which is simply the length of the input array "to_be_attacked" and another variable called "attacked" which is the same as the input array "to_be_attacked". Lastly, once everything else is done, it perturbs the array called "attacked" using one of three different functions based on the backdoor type desired to be added to the poisoned images. This is what Gu et al. was talking about in their paper when they mentioned single-pixel backdoor vs. pattern backdoor. The algorithm then outputs "attacked" and "attack_labels".

The extension I did to my algorithm was to create a clean label backdoor attack. It seemed fitting to do this since the second paper I read was all about the clean label backdoor attack. Furthermore, since Turner et al. had shown that their clean label backdoor attack was more effective than Gu et al.'s normal backdoor attack, I wanted to see this for myself [2]. This extension algorithm takes in the same three parameters as the normal backdoor attack algorithm: an array with the points that initialize attack points that I called "to_be_attacked", the target labels for the attack that I called "attack_target_labels", and whether or not to broadcast a single target label that I called "broadcast". Once it has input, the clean label backdoor attack basically starts the same way as the normal backdoor attack in the way that it reassigns the input "to_be_attacked" to a variable called "attacked" and "attack_target_labels" to a variable called "attack_labels". Here is where this extension starts to differ from the main algorithm. First, an array called "all_indices" is created, and this is simply an array of numbers from 0 to the length of the variable "attacked". Next, any label with an index associated with the target label gets stored in a variable called "target_indices". After that, another variable is created called "num_poison", and this is simply the percentage of the data that the attacker wishes to poison multiplied by the length of "target_indices". Now

that the number of indices to poison is known, it is time to choose which indices to poison. This is done randomly and then stored in a new variable called "selected indices". Just like with the original algorithm, the next thing that needs to happen is that the input is perturbed, and this is done using the adversarial ℓ_p-bounded perturbations method that Turner at al. mentions in their paper. Lastly, the original backdoor algorithm is run on the perturbed inputs, stored in "attacked", and then "attacked" and "attack_labels" are both output.

Implementations

I implemented both of my algorithms in Python with the help of a package called Adversarial Robustness Toolbox (ART) [3]. Adversarial Robustness Toolbox is a Python library that supports developers and researchers in defending Machine Learning models against adversarial threats [3]. It also helps make AI systems more secure and trustworthy [3]. I used some of the ART's infrastructure to help implement both my original backdoor algorithm and my clean label backdoor algorithm. The implementation that I chose to do for my main algorithm uses a Keras convolutional neural network with eight layers including two convolutional 2D layers, a max pooling 2D layer, two dense layers, two dropout layers, and one flatten layer. Also, something to note is that the loss function used was categorical cross-entropy, the optimizer used was adam, and the metric used was accuracy. In using this CNN to train a model, I chose to use five epochs because that just seemed like a good middle ground since the more epochs, the longer it takes to create a model. For my clean label backdoor attack algorithm, I used the same Keras convolutional neural network as before.

Experiments

I ran one main experiment with my two different algorithms using the MNIST dataset that was also used in Gu et al.'s original backdoor attack [1]. In case you forgot, the MNIST dataset consists of 60,000 grayscale images of handwritten single digits, and the goal is to predict the digit that is in each image. When running the MNIST dataset through my backdoor attack and then the CNN, I found that the clean test set accuracy was 98.94% and 100% of the images that were poisoned were poisoned correctly. For the clean label backdoor algorithm, the accuracy of the clean test set was found to be 99.10% and just like the previous attack, 100% of the poisoned images were poisoned correctly. As you can see, both attacks performed very similarly in the end, although I would say that the clean label backdoor attack was better because it is harder to figure out that the training set was poisoned. This is because the images were poisoned using the adversarial ℓ_p-bounded perturbations method that Turner at al. mentions in their paper [2]. Once this was done, I had wanted to move on to see what results I could get for running the CIFAR-10 dataset through my algorithms and CNNs, but I ran into dimensionality issues. I spent a lot of time looking up the errors that I got to try to fix the problem, but nothing that I found worked. I also asked the other members in my group to see if they could figure it out, and they did try, but they also did not have any luck.

Conclusion

Backdoor attacks are a type of adversarial learning that involves poisoning the dataset. Gu et al. made people aware of the backdoor attack in their paper titled "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain" [1]. In this paper, Gu et al. starts off by talking about how more and more individuals and companies are starting to outsource the training of neural networks because of how expensive they are to train in-house [1]. With outsourcing the training of networks, however, comes security concerns [1]. The main security concern here is that either the third party that provides the pre-trained network or the previous users of the network in the case of transfer learning gives the network

a backdoor [1]. This network with a backdoor performs well on most of its inputs, but it can cause either targeted misclassifications or make the model less accurate for certain inputs that contain a secret trigger [1]. After explaining this, Gu et al. goes through a couple of case studies on how to perform a backdoor attack with the main one being done on the MNIST dataset [1]. They were very successful in implementing their backdoor attack [1].

After reading everything that Gu et al. did, Turner et al. wrote their own paper trying to improve on the original backdoor attack [2]. They noticed that although Gu et al.'s attack was very effective, it was relatively easy for someone to realize that the training data had been poisoned [2]. All that someone had to do was apply a simple filter to the training set and most to all of the poisoned images would be deemed as outliers [2]. Once deemed as outliers, it was very easy for a human to look at the poisoned images and notice that their data had been tampered with [2]. In an attempt to fix this, Turner et al. did two things. The first thing they did was to only use poisoned samples that had plausible labels, and they referred to these as *clean labels.* The second thing they did was to try to make the added pixels to the poisoned images blend in with the image a little more [2]. They did this using GAN-based interpolation and adversarial ℓ_p-bounded perturbations, both of which proved to work better than Gu et al.'s original attack [2].

The next thing I did was create an algorithm based on the original backdoor attack that Gu et al. had implemented [1]. This algorithm starts off by taking in three parameters: an array with the points that initialize attack points, the target labels for the attack, and whether or not to broadcast a single target label. Once it has inputs, the algorithm then goes through the main array and poisons a given percentage of the images. After creating this algorithm, I also created another one based off of Turner et al.'s clean label backdoor attack [2]. This attack is very similar to the first one except that it poisons the images in a more discrete way [2]. After I was done with the two algorithms, I then implemented them in Python with the help of the Adversarial Robustness Toolbox package [3]. I used a convolutional neural network in order to train the model. To test that everything was working correctly, I ran the MNIST dataset through both of my attack algorithms and CNNs to find that although the clean label backdoor attack is less likely to be found by the user, the performances of the attacks at the end of the day were very similar.

References

[1] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," *arXiv*, 11-Mar-2019. [Online]. Available: https://arxiv.org/pdf/1708.06733.pdf.

[2] A. Turner, D. Tsipras, and A. Mądry, "Clean-Label Backdoor Attacks," *MIT Computer Science & Artificial Intelligence Lab*, 2019. [Online]. Available: https://people.csail.mit.edu/madry/lab/cleanlabel.pdf.

[3] M. Nicolae et. al, "Adversarial Robustness Toolbox v1.0.0," *arXiv*, 15-Nov-2019. [Online]. Available: https://arxiv.org/pdf/1807.01069.pdf