Autoregressive Perturbations for Data Poisoning

Anonymous Author(s)
Affiliation
Address
email

Abstract

The prevalence of data scraping from social media as a means to obtain datasets has led to growing concerns regarding unauthorized use of data. Data poisoning attacks have been proposed as a bulwark against scraping, as they make data "unlearnable" by adding small, imperceptible perturbations. Unfortunately, existing methods require knowledge of both the target architecture and the complete dataset so that a surrogate network can be trained, the parameters of which are used to generate the attack. In this work, we introduce autoregressive (AR) poisoning, a method that can generate poisoned data without access to the broader dataset. The proposed AR perturbations are generic, can be applied across different datasets, and can poison different architectures. Compared to existing unlearnable methods, our AR poisons are more resistant against common defenses such as adversarial training and strong data augmentations. Our analysis further provides insight into what makes an effective data poison.

1 Introduction

Increasingly large datasets are being used to train state-of-the-art neural networks [24, 26, 25]. But collecting enormous datasets through web scraping makes it intractable for a human to review samples in a meaningful way or to obtain consent from relevant parties [3]. In fact, companies have already trained commercial facial recognition systems using personal data collected from media platforms [15]. To prevent the further exploitation of online data for unauthorized or illegal purposes, imperceptible, adversarial modifications to images can be crafted to cause erroneous output for a neural network trained on the modified data [12]. This crafting of malicious perturbations for the purpose of interfering with model training is known as data poisoning.

In this work, we focus on poisoning data to induce poor performance for a network trained on the perturbed data. This kind of indiscriminate poisoning, which seeks to damage average model performance, is often referred to as an availability attack [1, 2, 40, 18, 9, 10]. Because we assume the data is hosted on a central server controlled by the poisoner, the poisoner is allowed to perturb the entire dataset, or a large portion of it. Throughout this work, unless stated otherwise, poisoning refers to the perturbing of every image in the training dataset. This makes the creation of unlearnable data different from other poisoning methods, such as backdoor [5, 13] and targeted poisoning attacks [28, 43].

We introduce autoregressive (AR) data poisoning for degrading overall performance of neural networks on clean data. The perturbations that we additively apply to clean data are generated by AR processes that are data and architecture-independent. An AR(p) process is a Markov chain, where each new element is a linear combination of p previous ones, plus noise. This means AR perturbations are cheap to generate, not requiring any optimization or backpropagation through network parameters. AR perturbations are generic; the same set of AR processes can be re-used to generate diverse perturbations for different image sizes and new datasets, unlike other poisoning methods which need to train a surrogate network on the target dataset before crafting perturbations.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

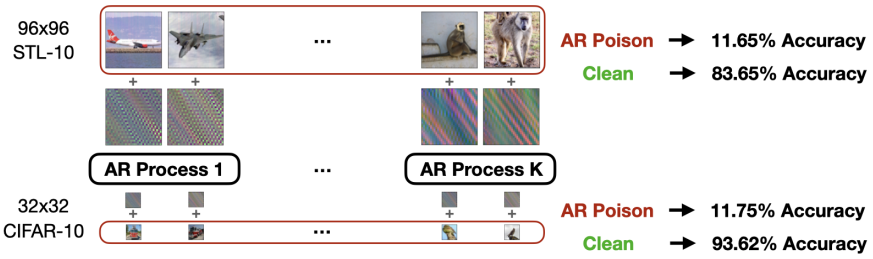

Figure 1: The same set of AR processes can generate perturbations for different kinds of data without access to a surrogate network or other images in a dataset. A ResNet-18 trained on these AR poisons is unable to generalize to clean test data, even with the help of strong data augmentations.

Our method also provides new insight into why data poisoning works. We work on top of the result that effective poisons are typically easy to learn [27] and construct AR perturbations which are separable by a manually-specified CNN. Working under the intuition that highly separable perturbations should be easily learned, we use the manual specification of parameters as a way of demonstrating that our AR perturbations are easily separable. Our manually-specified CNN makes use of what we call AR filters, which are attuned to detect noise from a specific AR process. AR poisoning's effectiveness is competitive or better than error-maximizing, error-minimizing, and random noise poisoning across a range of architectures, datasets, and common defenses. AR poisoning represents a paradigm shift for what a successful indiscriminate poisoning attack looks like, and raises the question of whether strong indiscriminate poisons need to be generated by surrogate networks for a given dataset.

2 Background & Related Work

Error-minimizing and Error-maximizing Noise. To conduct poisoning attacks on neural networks, recent works have modified data to explicitly cause gradient vanishing [31] or to *minimize* the loss with respect to the input image [18]. Images perturbed with error-minimizing noises are a surprisingly good data poisoning attack. A ResNet-18 (RN-18) trained on a CIFAR-10 [20] sample-wise error-minimizing poison achieves 19.9% final test accuracy, while the class-wise variant achieves 16.4% final test accuracy after 60 epochs of training [18]. More recently, strong adversarial attacks, which perturb clean data by *maximizing* the loss with respect to the input image, have been shown to be the most successful approach thus far [10]. An error-maximizing poison can poison a network to achieve 6.25% test accuracy on CIFAR-10. But both error-minimizing and error-maximizing poisons require a surrogate network, from which perturbations are optimized. The optimization can be expensive. For example, crafting the main CIFAR-10 poison from [10] takes roughly 6 hours on 4 GPUs. In contrast, our AR perturbations do not require access to network parameters and can be generated quickly, without the need for backpropagation or a GPU. We provide a technical overview of error-minimizing and error-maximizing perturbations in Section 3.1.

Random Noise. Given their simplicity, random noises for data poisoning have been explored as necessary baselines for indiscriminate poisoning. If random noise, constrained by an ℓ_∞ norm, is applied sample-wise to every image in CIFAR-10, a RN-18 trained on this poison can still generalize to the test set, with ~90% accuracy [10, 18]. But if the noise is applied class-wise, where every image of a class is modified with an identical additive perturbation, then a RN-18 trained on this CIFAR-10 poison will achieve around chance accuracy; *i.e.* ~10% [39, 18, 27]. The random perturbations of [39] consist of a fixed number of uniform patch regions, and are nearly identical to the class-wise poison, called "Regions-16," from [27]. All the random noises that we consider are class-wise, and we confirm they work well in a standard training setup using a RN-18, but their performance varies across architectures and they are rendered ineffective against strong data augmentations like Cutout [7], CutMix [41], and Mixup [42]. Conversely, our AR poisons degrade test performance more than error-maximizing, error-minimizing, and random poisons on almost every architecture. We show that AR perturbations are effective against strong data augmentations and can even mitigate some effects of adversarial training.

Understanding Poisoning. A few works have explored properties that make for effective poisons. For example, [27] find that poisons which are learned quickly have a more harmful effect on the poison-trained network, suggesting that the more quickly perturbations help minimize the training

loss, the more effective the poison is. [39] perform a related experiment where they use a single linear layer, train on perturbations from a variety of poisoning methods, and demonstrate that they can discriminate whether a perturbation is error-minimizing or error-maximizing with high accuracy. We make use of ideas from both papers, designing AR perturbations that are provably separable and Markovian in local regions.

Other Related Work. Several works have also focused on variants of "unlearnable" poisoning attacks. [9] propose to employ gradient alignment [11] to generate poisons. But their method is computationally expensive; it requires a surrogate model to solve a bi-level objective. [40] propose generation of an unlearnable dataset using neural tangent kernels. Their method also requires training a surrogate model, takes a long time to generate, and does not scale easily to large datasets. In contrast, our approach is simple and does not require surrogate models. [23] propose an invertible transformation to control learnability of a dataset for authorized users, while ensuring the data remains unlearnable for other users. [35] showed that data poisoning methods can be broken using adversarial training. [30] and [37] propose variants of error-minimizing noise to defend against adversarial training. Our AR poisons do not focus on adversarial training. While adversarial training remains a strong defense, our AR poisons show competitive performance. We discuss adversarial training in detail in Section 4.3.2. A thorough overview of data poisoning methods, including those that do not perturb the entire training dataset, can be found in [12].

3 Autoregressive Noises for Poisoning

3.1 Problem Statement

We formulate the problem of creating a clean-label poison in the context of image classification with DNNs, following [18]. For a K-class classification task, we denote the clean training and test datasets as $\mathcal{D}_c$ and $\mathcal{D}_t$, respectively. We assume $\mathcal{D}_c, \mathcal{D}_t \sim \mathcal{D}$. We let f_θ represent a classification DNN with parameters θ. The goal is to perturb $\mathcal{D}_c$ into a poisoned set $\mathcal{D}_p$ such that when DNNs are trained on $\mathcal{D}_p$, they perform poorly on test set $\mathcal{D}_t$.

Suppose there are n samples in the clean training set, i.e. $\mathcal{D}_c = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ are the inputs and $y_i \in \{1, ..., K\}$ are the labels. We denote the poisoned dataset as $\mathcal{D}_p = \{(x_i', y_i)\}_{i=1}^n$ where $x_i' = x_i + \delta_i$ is the poisoned version of the example $x_i \in \mathcal{D}_c$ and where $\delta_i \in \Delta \subset \mathbb{R}^d$ is the perturbation. The set of allowable perturbations, Δ, is usually defined by $\|\delta\|_p < \epsilon$ where $\|\cdot\|_p$ is the ℓ_p norm and ϵ is set to be small enough that it does not affect the utility of the example. In this work, we use the ℓ_2 norm to constrain the size of our perturbations for reasons we describe in Section 3.4.

Poisons are created by applying a perturbation to a clean image in either a class-wise or sample-wise manner. When a perturbation is applied class-wise, every sample of a given class is perturbed in the same way. That is, $x_i' = x_i + \delta_{y_i}$ and $\delta_{y_i} \in \Delta_C = \{\delta_1, ..., \delta_K\}$. Due to the explicit correlation between the perturbation and the true label, it should not be surprising that class-wise poisons appear to trick the model to learn the perturbation over the image content, subsequently reducing generalization to the clean test set. When a poison is applied sample-wise, every sample of the training set is perturbed independently. That is, $x_i' = x_i + \delta_i$ and $\delta_i \in \Delta_S = \{\delta_1, ..., \delta_n\}$. Because class-wise perturbations can be recovered by taking the average image of a class, these should therefore be easy to remove. Hence, we focus our study on sample-wise instead of class-wise poisons. We still compare to simple, randomly generated class-wise noises shown by [18] to further demonstrate the effectiveness of our method.

All indiscriminate poisoning aims to solve the following bi-level objective:

$$\max_{\delta \in \Delta} \mathbb{E}_{(x,y)\sim\mathcal{D}_t} \left[\mathcal{L}(f(x), y; \theta(\delta))\right] \tag{1}$$

$$\theta(\delta) = \arg\min_{\theta} \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}_c} \left[\mathcal{L}(f(x_i + \delta_i), y_i; \theta)\right] \tag{2}$$

Eq. 2 describes the process of training a network on poisoned data; i.e. x_i perturbed by δ_i. Eq. 1 states that the poisoned network should maximize the loss, and thus perform poorly, on clean test data.

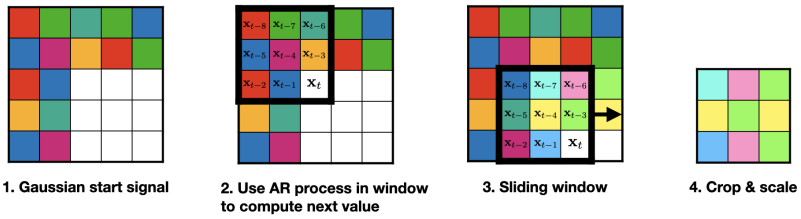

Figure 2: To generate 2D noise for a single channel using a 3×3 sliding window, we first sample Gaussian noise for the first 2 columns and rows. Then, we use an AR(8) process within a sliding window to compute the next value. Finally, we crop and scale the perturbation to be within our ℓ_2 constraint.

Different approaches have been proposed to construct δ_i. Both error-maximizing [10] and error-minimizing [18] poisoning approaches use a surrogate network, trained on clean training data, to optimize perturbations. We denote surrogate network parameters as θ^*. Error-maximizing poisoning [10] proposes constructing δ_i that maximize the loss of the surrogate network on clean training data:

$$\max_{\delta \in \Delta} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}_c} \left[\mathcal{L}(f(x_i + \delta_i), y_i; \theta^*) \right] \tag{3}$$

whereas error-minimizing poisoning [18] solve the following objective to construct δ_i that minimize the loss of the surrogate network on clean training data:

$$\min_{\delta \in \Delta} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}_c} \left[\mathcal{L}(f(x_i + \delta_i), y_i; \theta^*) \right] \tag{4}$$

In both error-maximizing and error-minimizing poisoning, the adversary intends for a network, f, trained on the poison to perform poorly on the test distribution D_t, from which D_c was also sampled. But the way in which both methods achieve the same goal is distinct.

3.2 Generating Autoregressive Noise

Autoregressive (AR) perturbations have a particularly useful structure where local regions throughout the perturbation are Markovian, exposing a linear dependence on neighboring pixels [38]. This property is critical as it allows for a particular filter to perfectly detect noise from a specific AR process, indicating the noise is simple and potentially easily learned.

We develop a sample-wise poison where clean images are perturbed using additive noise. For each x_i in the clean training dataset, our algorithm crafts a δ_i, where $\|\delta_i\|_2 \leq \epsilon$, so that the resulting poison image is $x_i' = x_i + \delta_i$. The novelty of our method is in how we find and use autoregressive (AR) processes to generate δ_i. In the following, let $\mathbf{x}_t$ refer to the tth entry within a sliding window of δ_i.

An autoregressive (AR) process models the conditional mean of $\mathbf{x}_t$, as a function of past observations $\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, ..., \mathbf{x}_{t-p}$ in the following way:

$$\mathbf{x}_t = \beta_1 \mathbf{x}_{t-1} + \beta_2 \mathbf{x}_{t-2} + ...\beta_{p-1} \mathbf{x}_{t-(p-1)} + \beta_p \mathbf{x}_{t-p} + \epsilon_t \tag{5}$$

where ϵ_t is an uncorrelated process with mean zero and β_i are the AR process coefficients. For simplicity, we set $\epsilon_t = 0$ in our work. An AR process that depends on p past observations is called an AR model of degree p, denoted AR(p). For any AR(p) process, we can construct a size $p + 1$ filter where the elements are $\beta_p, ..., \beta_1$ and the last entry of the filter is -1. This filter produces a zero response for any signal generated by the AR process with coefficients $\beta_p, ..., \beta_1$. We refer to this filter as an AR filter, the utility of which is explained in Section 3.3 and Appendix A.1.

Suppose we have a K class classification problem of $H \times W \times C$ dimensional images. For each class label y_i, we construct a set A_{y_i} of AR processes, one for each of the C channels. For each of the C channels, we will be applying an AR process from A_{y_i} inside a $V \times V$ sliding window. Naturally, using an AR process requires initial observations, so we populate the perturbation vector δ_i with Gaussian noise for the first $V - 1$ columns and rows. The $V \times V$ sliding window starts at the

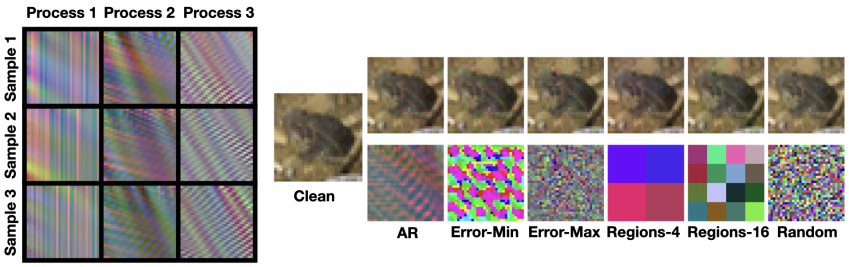

Figure 3: Left: Normalized samples of perturbations generated by 3 AR processes. Right: Poisoned images and the corresponding perturbation for a randomly selected CIFAR-10 image.

top left corner of δ_i. Within this sliding window, we apply the AR($V^2 - 1$) process: the first $V^2 - 1$ entries in the sliding window are considered previously generates (or randomly initialized) entries in the 2D array δ_i, and the Vth entry is computed by Eq. 5. The window is slid left to right, top to bottom until the first channel of δ_i is filled. We then proceed to use the next AR($V^2 - 1$) process in A_{y_i} for the remaining $C - 1$ channels. Finally, we discard the random Gaussian rows and columns used for initialization, and scale δ_i to be of size ϵ in the ℓ_2-norm. Note that this sliding window procedure resembles that of a convolution. That is by design, and we explain why it is important in Section 3.3. A high-level overview of this algorithm is illustrated in Figure 2. Additional details are in Appendix A.3.2. While we describe our use of AR processes on C-channel images, our method could, in principle, be applied to data other than images. Note that these AR perturbations are fast to generate, do not require a pre-trained surrogate model, and can be generated *independently from the data.*

3.3 Why do Autoregressive Perturbations Work?

Perturbations that are easy to learn have been shown to be more effective at data poisoning [27]. Intuitively, a signal that is easily interpolated by a network will be quickly identified and used as a "shortcut," whereas complex and unpredictable patterns may not be learned until after a network has already extracted useful content-based features [29]. Thus, we seek imperceptible perturbations that are easy to learn. We propose a simple hypothesis: if there exists a simple CNN that can classify autoregressive signals perfectly, then these signals will be easy to learn. The signals can then be applied to clean images and serve as a shortcut for learning by commonly-used CNNs.

Autoregressive perturbations, despite looking visually complex, are actually very simple. To demonstrate their separability, we manually specify the parameters of a simple CNN that classifies AR perturbations perfectly by using AR filters. In the following, we prove AR filters satisfy an important property.

Lemma 3.1. *Given an AR perturbation δ, generated from an AR(p) with coefficients $\beta_1, ..., \beta_p$, there exists a linear, shift invariant filter where the cross-correlation operator produces a zero response.*

We provide a proof in Appendix A.1. The construction of an AR filter that produces a zero response for any noise generated from the corresponding AR process is useful because we can construct a CNN which makes use of solely these AR filters to classify signals. That is, given any AR perturbation, the AR filter with the zero response correctly designates the AR process from which the perturbation was generated. We verify this claim in Appendix A.2 by specifying the 3-layer CNN that can perfectly classify AR perturbations.

Crucially, we are not interested in learning classes of AR signals. Rather, we are interested in how quickly a model can learn classes of clean data *perturbed* by AR signals. Nevertheless, the characterization of our AR perturbations as easy to learn, demonstrated by the manual specification of a 3-layer CNN, is certainly an indication that, when applied to clean data, AR perturbations can serve as bait for CNNs. Our experiments will seek to answer the following question: If we perturb each sample in the training dataset with an imperceptible, yet easily learned AR perturbation, can we induce a learning "shortcut" that minimizes the training loss but prevents generalization?

3.4 Finding AR Process Coefficients

We generate AR processes using a random search that promotes diversity. We generate processes one-at-a-time by starting with a random Gaussian vector of coefficients. We then scale the coefficients so that they sum to one. We then append a -1 to the end of the coefficients to produce the associated AR filter, and convolve this filter with previously generated perturbations. We use the norms of the resulting convolution outputs as a measure of similarity between processes. If the minimum of these norms is below a cutoff T, then we deem the AR process too coherent with previously generated perturbations – the coefficients are discarded and we try again with a different random vector.

Once the AR process coefficients are identified for a class, we use them to produce a perturbation δ_i for each image in the class. This perturbation is scaled to be exactly of size ϵ in the ℓ_2-norm. To level the playing field among all poisoning methods, we measure all perturbations using an ℓ_2 norm in this work. A more detailed description of this process can be found in Appendix A.3.1.

4 Experiments

We demonstrate the generality of AR poisoning by creating poisons across four datasets, including different image sizes and number of classes. Notably, we use the *same* set of AR processes to poison SVHN [22], STL-10 [6], and CIFAR-10 [20] since all of these datasets are 10 class classification problems. We demonstrate that despite the victim's choice of network architecture, AR poisons can degrade a network's accuracy on clean test data. We show that while strong data augmentations are an effective defense against all poisons we consider, AR poisoning is largely resistant. Adversarial training and diluting the poison with clean data remain strong defenses, but our AR poisoning method is competitive with other poisons we consider. All experiments follow the same general pattern: we train a network on a poisoned dataset and then evaluate the trained network's performance on clean test data. A poison is effective if it can cause the trained network to have poor test accuracy on clean data, so lower numbers are better throughout our results.

Experimental Settings. We train a number of ResNet-18 (RN-18) [14] models on different poisons with cross-entropy loss for 100 epochs using a batch size of 128. For our optimizer, we use SGD with momentum of 0.9 and weight decay of 5×10^{-4}. We use an initial learning rate of 0.1, which decays by a factor of 10 on epoch 50. In Table 2, we use use the same settings with different network architectures.

4.1 Error-Max, Error-Min, and other Random Noise Poisons

SVHN [22], CIFAR-10, and CIFAR-100 [20] poisons considered in this work contain perturbations of size $\epsilon = 1$ in ℓ_2, unless stated otherwise. For STL-10 [6], all poisons use perturbations of size $\epsilon = 3$ in ℓ_2 due to the larger size of STL-10 images. In all cases, perturbations are normalized and scaled to be of size ϵ in ℓ_2, are additively applied to clean data, and are subsequently clamped to be in image space. Dataset details can be found in Appendix A.5. A sampling of poison images and their corresponding normalized perturbation can be found in Figure 3 and Appendix A.4. In our results, class-wise poisons are marked with $\circ$ and sample-wise poisons are marked with $\bullet$.

Error-Max and Error-Min Noise. To generate error-maximizing poisons, we use the open-source implementation of [10]. In particular, we use a 250-step ℓ_2 PGD attack to optimize Eq. (3). To generate error-minimizing poisons, we use the open-source implementation of [18], where a 20-step ℓ_2 PGD attack is used to optimize Eq. (4). For error-minimizing poisoning, we find that moving in ℓ_2 normalized gradient directions is ineffective at reaching the required universal stop error [18], so we move in signed gradient directions instead (as is done for ℓ_∞ PGD attacks).

Regions-4 and Regions-16 Noise. Synthetic, random noises are also dataset and network independent. Thus, to demonstrate the strength of our method, we include three class-wise random noises in our experiments. To generate what we a call a Regions-p noise, we follow [39, 27]: we sample p RGB vectors of size 3 from a Gaussian distribution and repeat each vector along height and width dimensions, resulting in a grid-like pattern of p uniform cells or regions. Assuming a square image of side length L, a Regions-p noise contains patches of size $\frac{L}{\sqrt{p}} \times \frac{L}{\sqrt{p}}$.

Random Noise. We also consider a class-wise random noise poison, where perturbations for each class are sampled from a Gaussian distribution.

Table 1: **Dataset Independence.** AR noises are effective across a variety of datasets. For SVHN, STL-10, and CIFAR-10, we use the same 10 AR processes to generate sample-wise noises. We display clean test accuracy of RN-18 when trained using standard augmentations on different poisons. Class-wise poisons are marked with ○ and sample-wise poisons are marked with •.

	SVHN	STL-10	CIFAR-10	CIFAR-100
Clean	96.40	83.65	93.62	75.30
• Error-Max [10]	80.80	17.26	26.94	4.87
• Error-Min [18]	96.82	80.71	16.84	74.58
○ Regions-4	9.80	41.36	20.75	9.14
○ Regions-16	**6.39**	31.21	15.75	**2.99**
○ Random Noise	9.68	72.07	18.45	73.90
• Autoregressive (Ours)	6.77	**11.65**	**11.75**	4.24

Table 2: **Architecture Independence.** CIFAR-10 test accuracy for a variety of model architectures trained on different poisons. Error-Max and Error-Min poisons are crafted using a RN-18.

	RN-18	VGG-19	GoogLeNet	MobileNet	EfficientNet	DenseNet	ViT
• Error-Max [10]	16.84	20.34	15.31	15.38	16.21	18.02	41.70
• Error-Min [18]	26.94	22.39	32.18	21.36	23.86	28.21	40.95
○ Regions-4	20.75	25.39	25.25	22.23	18.64	24.21	35.18
○ Regions-16	15.75	19.86	12.97	19.67	16.94	24.16	25.49
○ Random Noise	18.45	81.98	12.53	46.61	86.89	**9.96**	74.47
• AR (Ours)	**11.75**	**12.35**	**9.24**	**15.17**	**13.47**	14.90	**19.66**

4.2 AR Perturbations are Dataset and Architecture Independent

Unlike error-maximizing and error-minimizing poisons, AR poisons are not dataset-specific. One cannot simply take the perturbations from an error-maximizing or error-minimizing poison and apply the same perturbations to images of another dataset. Perturbations optimized using PGD are known to be relevant features, necessary for classification [10, 19]. Additionally, for both these methods, a crafting network trained on clean data is needed to produce reasonable gradient information. In contrast, AR perturbations are generated from dataset-independent AR processes. The same set of AR processes can be used to generate the same kinds of noise for images of new datasets. Building from this insight, one could potentially collect a large set of K AR processes to perturb any dataset of K or fewer classes, further showing the generality of our method.

In Table 1, we use the same 10 AR processes to generate noise for images of SVHN, STL-10, and CIFAR-10. AR poisons are, in all cases, either competitive or the most effective poison – a poison-trained RN-18 reaches nearly chance accuracy on STL-10 and CIFAR-10, and being the second-best on SVHN and CIFAR-100. The generality of AR perturbations to different kinds of datasets suggests that AR poisoning induces the most easily learned correlation between samples and their corresponding label.

We also evaluate the effectiveness of our AR poisons when different architectures are used for training. Recall that error-maximizing and error-minimizing poisoning use a crafting network to optimize the input perturbations. Because it may be possible that these noises are specific to the network architecture, we perform an evaluation of test set accuracy on CIFAR-10 after poison training VGG-19 [32], GoogLeNet [33], MobileNet [16], EfficientNet [34], DenseNet [17], and ViT [8]. Our ViT uses a patch size of 4. In Table 2, we show that Error-Max and Error-Min poisons generalize relatively well across a range of related CNNs, but struggle with ViT, which is a transformer architecture. In contrast, our AR poison is effective across all CNN architectures and is the most effective poison against ViT. Our AR poison is much more effective over other poisons in almost all cases, achieving improvements over the next best poison of 4% on RN-18, 5.8% on ViT, and 7.5% on GoogLeNet. The design of AR perturbations is meant to target the convolution operation, so it is surprising to see a transformer network be adversely affected. We believe our AR poison is particularly effective on GoogLeNet due to the presence of Inception modules that incorporate convolutions using various filter sizes. While our AR perturbations are generated using a 3×3 window, the use of various filter sizes may exaggerate their separability, as described in Section 3.3.

Table 3: **Strong Data Augmentations.** CIFAR-10 test accuracy of RN-18 when training using standard augmentations plus Cutout, CutMix, or Mixup on different poisons, where clean images are perturbed within an ℓ_2 ball of size ϵ.

		Standard Aug	+Cutout	+CutMix	+Mixup
• Error-Max [10]	$\epsilon = 0.5$	25.04	25.38	29.72	38.07
• Error-Min [18]		48.07	44.97	53.77	53.81
○ Regions-4		57.48	67.47	67.72	66.80
○ Regions-16		47.35	39.64	32.67	49.02
○ Random Noise		93.76	93.66	91.80	94.16
• Autoregressive (Ours)		**14.28**	**12.36**	**18.02**	**14.59**
• Error-Max [10]	$\epsilon = 1$	16.84	18.86	21.45	30.52
• Error-Min [18]		26.94	29.38	25.04	43.36
○ Regions-4		20.75	21.89	28.61	40.60
○ Regions-16		15.75	19.61	16.67	20.81
○ Random Noise		18.45	12.61	12.63	23.64
• Autoregressive (Ours)		**11.75**	**11.90**	**11.23**	**11.40**

4.3 AR Perturbations Against Common Defenses

4.3.1 Data Augmentations and Smaller Perturbations

Our poisoning method relies on imperceptible AR perturbations, so it is conceivable that one could modify the data to prevent the learning of these perturbations. One way of modifying data is by using data augmentation strategies during training. In addition to standard augmentations like random crops and horizontal flips, we benchmark our AR poison against stronger augmentations like Cutout [7], CutMix [41], and Mixup [42] in Table 3. Generally, Mixup seems to be the most effective at disabling poisons. A RN-18 poison-trained using standard augmentations plus Mixup can achieve a boosts in test set performance of 13.68% on Error-Max, 16.42% on Error-Min, 19.85% on Regions-4, 5.05% on Regions-16, and 5.19% on Random Noise. However, a RN-18 poison-trained on our AR poison ($\epsilon = 1$) using standard augmentations plus Cutout, CutMix, or Mixup cannot achieve *any* boost in test set performance.

We also present results for poisons using perturbations of size $\epsilon = 0.5$ to explore just how small perturbations can be made while still maintaining poisoning performance. Under standard augmentations, going from larger to smaller perturbations ($\epsilon = 1$ to $\epsilon = 0.5$), poison effectiveness drops by 8.2% for Error-Max, 21.13% for Error-Min, 36.73% for Regions-4, and 31.6% for Regions-16. Our AR poison achieves the smallest drop in effectiveness: only 2.53%. Random noise can no longer be considered a poison at $\epsilon = 0.5$ – it completely breaks for small perturbations. Under *all* strong data augmentation strategies at $\epsilon = 0.5$, AR poisoning dominates. For example, under Mixup, the best runner-up poison is Error-Max with an effectiveness that is more than 23% lower than AR. Unlike all other poisons, AR poisoning is exceptionally effective for small perturbations.

Note that in all three augmentation strategies pixels are either dropped or scaled. Our method is unaffected by these augmentation strategies, unlike error-maximizing, error-minimizing, and other randomly noise poisons. Scaling an AR perturbation does not affect how the corresponding matching AR filter will respond,[1] and thus, the patterns remain highly separable regardless of perturbation size. Additionally, AR filters contain values which sum to 0, so uniform regions of an image also produce a zero response.

4.3.2 Adversarial Training

Adversarial training has also been shown to be an effective counter strategy against ℓ_p-norm constrained data poisons [18, 10, 9, 35]. Using adversarial training, a model trained on the poisoned data can achieve nearly the same performance as training on clean data [30]. However, adversarial training is computationally more expensive than standard training and leads to a decrease in test accuracy [21, 36] when the perturbation radius, ρ_a, of the adversary is large. In Table 5, we include adversarial training results on clean data to outline this trade-off where training at large ρ_a comes at the cost of test accuracy. A recent line of work has therefore focused on developing better data poisoning methods that are robust against adversarial training [30, 37] at larger adversarial training radius ρ_a.

[1] See condition outlined in Lemma 3.1.

Table 4: **Mixing Poisons with Clean Data.** CIFAR-10 test accuracy when a proportion of clean data is used in addition to a poison. Top row shows test accuracy when training on only the clean proportion of the data; *i.e.* no poisoned data is used.

	Clean Proportion				
	40%	30%	20%	10%	5%
Clean Only	90.84	89.92	87.90	81.01	74.97
• Error-Max [10]	88.51	87.09	83.68	81.72	75.66
• Error-Min [18]	88.76	86.96	83.78	79.1	68.35
○ Regions-4	87.66	87.86	83.60	77.18	70.85
○ Regions-16	87.52	87.16	**82.94**	77.38	65.73
○ Random Noise	88.72	87.28	83.24	79.06	69.83
• AR (Ours)	**87.52**	**84.97**	84.14	**75.07**	**66.15**

Table 5: **Adversarial Training.** CIFAR-10 test accuracy after adversarially training with different radii ρ_a. Top row shows performance of adversarial training on clean data. AR poisons remain effective for small ρ_a.

	ρ_a			
	0.125	0.25	0.50	0.75
Clean Data	87.07	84.75	81.19	77.01
• Error-Max [10]	**33.33**	71.69	**79.16**	**74.94**
• Error-Min [18]	71.05	83.18	81.18	77.21
○ Regions-4	74.55	81.27	79.73	76.88
○ Regions-16	48.12	71.24	80.37	76.59
○ Random Noise	86.45	84.42	80.02	76.16
• AR (Ours)	33.65	**56.22**	80.15	76.72

In Table 5, we compare performance of different poisons against adversarial training. We perform ℓ_2 adversarial training with different perturbation radii, ρ_a, using a 7-step PGD attack with a step-size of $\rho_a/4$. We also show the performance of adversarial training on clean data. Data poisoning methods are fragile to adversarial training even when the training radius ρ_a is smaller than poisoning radius ϵ [30, 37]. It is desirable for poisons to remain effective for larger ρ_a, because the trade-off between standard test accuracy and robust test accuracy would be exaggerated further. As shown in the Table 5, when the adversarial training radius ρ_a increases, the poisons are gradually rendered ineffective. All poisons are nearly ineffective at $\rho_a = 0.5$. Our proposed AR perturbations remain more effective at smaller radius, *i.e.* $\rho_a = 0.125$ and $\rho_a = 0.25$ compared to all other poisons.

4.3.3 Mixing Poisons with Clean Data

Consider the scenario when not all the data can be poisoned. This setup is practical because, to a practitioner coming into control of poisoned data, additional clean data may be available through other sources. Therefore, it is common to evaluate poisoning performance using smaller proportions of randomly selected poison training samples [10, 18, 30]. A poison can be considered effective if the addition of poisoned data hurts test accuracy compared to training on only the clean data. In Table 4, we evaluate the effectiveness of poisons using different proportions of clean and poisoned data. The top row of Table 4 shows test accuracy after training on only the subset of clean data, with no poisoned data present. Our AR poisons remain effective compared to other poisons even when clean data is mixed in. AR poisons are much more effective when a small portion of the data is clean. For example, when 5% of data is clean, a model achieves ~75% accuracy when training on only the clean proportion, but using an additional 95% of AR data leads to a ~9% decrease in test set generalization. Our results on clean data demonstrate that AR poisoned data is worse than useless for training a network, and a practitioner with access to the data would be better off not using it.

5 Conclusion

Using the intuition that simple noises are easily learned, we proposed the design of AR perturbations, noises that are so simple they can be perfectly classified by a 3-layer CNN where all parameters are manually-specified. We demonstrate that these AR perturbations are immediately useful and make effective poisons for the purpose of preventing a network from generalizing to the clean test distribution. Unlike other effective poisoning techniques that optimize error-maximizing or error-minimizing noises, AR poisoning does not need access to a broader dataset or surrogate network parameters. We are able to use the same set of 10 AR processes to generate imperceptible noises able to degrade the test performance of networks trained on three different 10 class datasets. Unlike randomly generated poisons, AR poisons are more potent when training using a new network architecture or strong data augmentations like Cutout, CutMix, and Mixup. Against defenses like adversarial training, AR poisoning is competitive or among the best for a range of attack radii. Finally, we demonstrated that AR poisoned data is worse than useless when it is mixed with clean data, reducing likelihood that a practitioner would want to include AR poisoned data in their training dataset.

References

[1] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[2] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[3] Abeba Birhane and Vinay Uday Prabhu. Large image datasets: A pyrrhic win for computer vision? In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1536–1546. IEEE, 2021.

[4] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[5] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[6] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[7] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[9] Liam Fowl, Ping-yeh Chiang, Micah Goldblum, Jonas Geiping, Arpit Bansal, Wojtek Czaja, and Tom Goldstein. Preventing unauthorized use of proprietary data: Poisoning for secure dataset release. *arXiv preprint arXiv:2103.02683*, 2021.

[10] Liam Fowl, Micah Goldblum, Ping-yeh Chiang, Jonas Geiping, Wojciech Czaja, and Tom Goldstein. Adversarial examples make strong poisons. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 30339–30351. Curran Associates, Inc., 2021.

[11] Jonas Geiping, Liam H Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. In *International Conference on Learning Representations*, 2021.

[12] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[13] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Kashmir Hill. The secretive company that might end privacy as we know it, Jan 2020.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, and Yisen Wang. Unlearnable examples: Making personal data unexploitable. In *International Conference on Learning Representations*, 2021.

[19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[22] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[23] Weiqi Peng and Jinghui Chen. Learnability lock: Authorized learnability control through adversarial invertible transformations. In *International Conference on Learning Representations*, 2022.

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[25] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[26] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[27] Pedro Sandoval-Segura, Vasu Singla, Liam Fowl, Jonas Geiping, Micah Goldblum, David Jacobs, and Tom Goldstein. Poisons that are learned faster are more effective. *arXiv preprint arXiv:2204.08615*, 2022.

[28] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems*, 31, 2018.

[29] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.

[30] Fu Shaopeng, He Fengxiang, Liu Yang, Shen Li, and Tao Dacheng. Robust unlearnable examples: Protecting data privacy against adversarial learning. In *International Conference on Learning Representations*, 2022.

[31] Juncheng Shen, Xiaolei Zhu, and De Ma. Tensorclog: An imperceptible poisoning attack on deep neural network applications. *IEEE Access*, 7:41498–41506, 2019.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[34] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[35] Lue Tao, Lei Feng, Jinfeng Yi, Sheng-Jun Huang, and Songcan Chen. Better safe than sorry: Preventing delusive adversaries with adversarial training. *Advances in Neural Information Processing Systems*, 34, 2021.

[36] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

[37] Zhirui Wang, Yifei Wang, and Yisen Wang. Fooling adversarial training with inducing noise. *arXiv preprint arXiv:2111.10130*, 2021.

[38] Herman Wold. *A study in the analysis of stationary time series*. PhD thesis, Almqvist & Wiksell, 1938.

[39] Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. Indiscriminate poisoning attacks are shortcuts, 2022.

[40] Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12230–12240. PMLR, 18–24 Jul 2021.

[41] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[42] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[43] Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pages 7614–7623. PMLR, 2019.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix A.8
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix A.9
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 3.1
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section 3.3 and Appendix A.1
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Full algorithm and details are in Appendix A.3
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and Appendix A.5
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.6
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Proof of Lemma 3.1

Lemma A.1. *Given an AR perturbation δ, generated from an AR(p) with coefficients $\beta_1, ..., \beta_p$, there exists a linear, shift invariant filter where the cross-correlation operator produces a zero response.*

Proof. Consider any size $p+1$ window of the AR perturbation δ where, by definition, the last entry follows Eq. (5) directly: $[\mathbf{x}_{t-p}, \mathbf{x}_{t-(p-1)}, ..., \mathbf{x}_{t-1}, \mathbf{x}_t]$. We construct a size $p+1$ filter where the elements are $\beta_p, ..., \beta_1$ and the last entry of the filter is -1. We call this filter an AR filter. Notice that the dot product of a window of δ with this filter is:

$$
\begin{aligned}
&[\mathbf{x}_{t-p}, \mathbf{x}_{t-(p-1)}, ..., \mathbf{x}_{t-1}, \mathbf{x}_t] \cdot [\beta_p, \beta_{p-1}, ..., \beta_1, -1] && (6)\\
&= \beta_p \mathbf{x}_{t-p} + \beta_{p-1}\mathbf{x}_{t-(p-1)} + ... + \beta_1 \mathbf{x}_{t-1} - \mathbf{x}_t && (7)\\
&= \beta_p \mathbf{x}_{t-p} + \beta_{p-1}\mathbf{x}_{t-(p-1)} + ... + \beta_1 \mathbf{x}_{t-1} - (\beta_1 \mathbf{x}_{t-1} + ... + \beta_{p-1}\mathbf{x}_{t-(p-1)} + \beta_p \mathbf{x}_{t-p}) && (8)\\
&= 0
\end{aligned}
$$

which implies:

$$\delta \star [\beta_1, ..., \beta_p, -1] = \vec{0} \tag{9}$$

where $\star$ denotes the cross-correlation operator.

A.2 Manually-specified CNN with AR filters

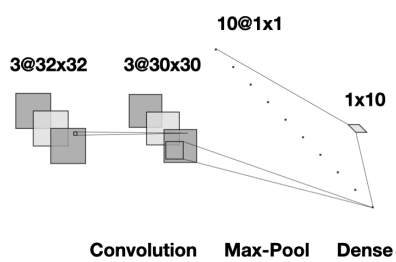

Figure 4: A 3-layer CNN can perfectly classify AR perturbations without any training. We specify each parameter manually.

Assume AR perturbations are expected to be of size 32×32, as in CIFAR-10. To illustrate the ease with which a set of AR perturbations can be classified, we construct a set of 10 AR(8) processes $A = \vec{\beta_1}, ..., \vec{\beta_{10}}$, where $\vec{\beta_i} = [\beta_{i,1}, ..., \beta_{i,8}]$ is a vector of AR(8) coefficients. Our network consists of 3 layers: a convolutional layer, a max-pool layer, and a fully connected layer. We specify the parameters of each layer below:

1. Convolutional Layer: 10 AR filters of size 3×3. No bias vector. ReLU activation on output. The ith AR filter is meant to produce a zero response (See A.1) on noise generated from AR process with coefficients $\vec{\beta_i}$:

$$f_i = \begin{bmatrix} \beta_{i,8} & \beta_{i,7} & \beta_{i,6} \\ \beta_{i,5} & \beta_{i,4} & \beta_{i,3} \\ \beta_{i,2} & \beta_{i,1} & -1 \end{bmatrix}$$

2. Max-Pool layer: Kernel size of 30×30, the same dimensions as the output of the convolutional layer.

3. Linear layer: We set $W = -I$ and $b = \vec{1}$, where $\vec{1}$ is a vector of all ones. Softmax activation on output.

To classify AR perturbation from 10 classes, this network outputs a probability distribution over 10 classes. This simple CNN works by assigning a high value to the ith logit when the ith AR filter produces a zero response. For example, when an AR filter outputs a zero response, the activation after ReLU and Max-Pool is still 0. The linear layer will then add a bias of 1, so the resulting i^{th} logit will be 1. Other logits correspond to AR filters which do not match the AR process, and output some nonnegative value, after ReLU and Max-Pool. The subsequent linear layer subtracts the nonnegative activation from 1, resulting in a logit value that is smaller than 1. In this way, the softmax activation assigns a high probability to the ith class.

To confirm our network design, we generate $5,000$ AR perturbations per AR process, and use the network defined above to classify the noises. The manually-specified network achieves *perfect accuracy without any training*. We use this result as a demonstration that AR perturbations are separable. Note that for other simple, linearly separable image datasets it is non-trivial to manually specify a CNN for perfect accuracy. Again, our motivation in exploring easily classified noises is that there is substantial work suggesting dataset separability is key to crafting good poisons.

A.3 Algorithms for Generating AR perturbation and Finding Coefficients

Algorithm 1 Random Search for AR Coefficients

Input: Number of classes, K
Input: Minimum response threshold, T
Output: Set of AR process coefficients, A

1: $A \leftarrow \{\}$
2: **while** $|A| \neq K$ **do**
3: $\quad \vec{\beta} \leftarrow \mathcal{N}(0,1)$
4: $\quad \vec{\beta} \leftarrow \frac{\vec{\beta}}{\sum_j \beta_j}$
5: $\quad \delta \leftarrow \text{ARGenerate}(\vec{\beta})$
6: $\quad$ **if** δ is stable **then**
7: $\quad\quad$ Let m store the min response so far.
8: $\quad\quad$ **for** i in 1,...,$|A|$ **do**
9: $\quad\quad\quad f \leftarrow \text{ARFilter}(A_i)$
10: $\quad\quad\quad r \leftarrow \text{ConvResponse}(\delta, f)$
11: $\quad\quad\quad m \leftarrow \min(m, r)$
12: $\quad\quad$ **end for**
13: $\quad\quad$ **if** $m \geq T$ **then**
14: $\quad\quad\quad A \leftarrow A \cup \{\beta_i\}$
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: **end while**
18: **return** A

Algorithm 2 Generate C-channel AR Perturbation

Input: Image and label, (x_i, y_i), where $x_i \in \mathbb{R}^{H \times W \times C}$ and $y_i \in \{1, ..., K\}$
Input: Size of sliding window, V
Input: Set of AR($V^2 - 1$) processes, A
Input: Size of perturturbation ϵ in ℓ_2
Output: Poisoned image, x_i'

1: Sample $\mathcal{N}(0,1)$ for first $V-1$ rows and columns of δ_i
2: **for** k in $1, ..., C$ **do**
3: $\quad \beta_1, ..., \beta_{V^2-1} \leftarrow A^k_{y_i}$
4: $\quad \delta_i^k \leftarrow \text{ARGenerate}(\beta_1, ..., \beta_{V^2-1})$
5: **end for**
6: $\delta_i \leftarrow \epsilon \frac{\delta_i}{\|\delta_i\|_2}$
7: $x_i' \leftarrow x_i + \delta_i$
8: **return** x_i'

We define important functions and variables of Algorithm 1 and Algorithm 2:

- $\vec{\beta}$ is a vector where the entries are the AR process coefficients $\beta_1, ..., \beta_p$.
- To check whether δ is stable, as in Algorithm 1 Line 6, we check whether the AR process diverges under a few starting signals. To do this, we check that the ℓ_2-norm of δ is not large, infinity, or NaN, under 3 different Gaussian starting signals. If the AR process diverges on any starting signal, then we say δ is not stable, and the Algorithm 1 continues by sampling a new set of AR process coefficients.
- $\text{ARGenerate}(\vec{\beta})$ takes a set of AR process coefficients and uses the AR process within a sliding window which goes left to right, top to bottom. The first $V-1$ rows and columns of δ_i are sampled from a Gaussian. The output of $\text{ARGenerate}(\vec{\beta})$ is a two-dimensional array. The procedure is described in Section 3.2 and illustrated in Figure 2. Algorithm 2 produces the entire C-channel AR perturbation.

- $\mathrm{ARFilter}(A_i)$ takes a set of AR process coefficients and returns the corresponding AR filter, where the entries are the AR coefficients followed by a -1. The AR filter is described in Section 3.2 and used in the construction of our simple AR noise classifier in Appendix A.2.
- $\mathrm{ConvResponse}(\delta, f)$ takes a signal and filter as input. It measures a filter f's "response" on δ by performing 2D convolution, ReLU, and sum.

A.3.1 Additional Details: Finding AR Process Coefficients

While there exist conditions for AR coefficients for which the AR process converges [38, 4], we found that randomly searching for coefficients was the simplest way to collect a large number of suitable AR processes. Additionally, using convergence conditions leads to AR processes which converge too quickly. For our purposes, converging too quickly is not desirable because most of the values in our AR perturbation would be zero.

Instead, we conduct a random search for K sets of converging AR coefficients by randomly sampling values from a Gaussian distribution and ensuring the resulting process is stable; *i.e.* does not diverge. We find that the likelihood of a convergent AR process is much higher when the coefficients are normalized to sum to 1. Normalizing the coefficients (Algorithm 1 Line 4) is also important because we want AR filters to produce a zero response to uniform regions of an image. If AR coefficients sum to 1 and the final entry of the AR filter is -1, then the 2D cross-correlation operator produces a zero response if pixels in a local window are approximately equal.

We optimize so that every AR process is sufficiently different from other processes already included in our growing set. We do this by measuring the response of AR filters from AR processes already in our set to noise generated by the currently sampled AR process. It may be possible to optimize a different criteria, but in our implementation, $\mathrm{ConvResponse}(\delta, f)$ measures a filter's response by performing 2D convolution, ReLU, and sum. We fully outline this random search for K sets of AR process coefficients in Algorithm 1.

We conduct a search for $K = 10$ sets of AR coefficients using Algorithm 1 with a threshold $T = 10$. This search took roughly 11 hours running on a single CPU. For CIFAR-100, we conduct a search for $K = 100$ sets of AR coefficients with $T = 3$ due to the larger number of classes. This search took roughly 4 hours running on a single GPU. Because Algorithm 1 samples coefficients independently on every iteration, significant speedups can likely be achieved using parallelized code.

A.3.2 Additional Details: Generating AR Perturbations

In addition to normalizing and scaling AR perturbation after generation, we also find that cropping out the first few columns and rows, where the Gaussian start signal was generated (see Figure 2), leads to more effective AR perturbations. We tried cropping only the Gaussian start signal, and cropping the start signal plus additional columns and rows. We found that cropping the start signal plus two extra columns and rows worked best. The AR perturbations are also less perceptible this way, given that the Gaussian noise starting signal has been removed. For 32×32 dimensional images, we generate AR perturbations of size 36×36 and crop the first 4 rows and columns. For 96×96 dimensional images, we generate AR perturbations of size 100×100 and crop the first 4 rows and columns.

A.4 Samples of CIFAR-10 Poisons

We display a random batch of images from each CIFAR-10 poison we consider in Figures 6,7,8,9,10, and 11.

A.5 Dataset Details

SVHN [22] contains 10 classes, where 73257 images are for training and 26032 images are for testing. STL-10 [6] conatins 10 classes, where 5000 are for training and 8000 are for testing. Both CIFAR-10 and CIFAR-100 [20] contain 50000 images for training and 10000 images for testing. Images of CIFAR-10 belong to one of 10, whereas images from CIFAR-100 belong to one of 100 classes. SVHN, CIFAR-10, and CIFAR-100 contain images of size $32 \times 32 \times 3$; STL-10 contains images of size 96×96.

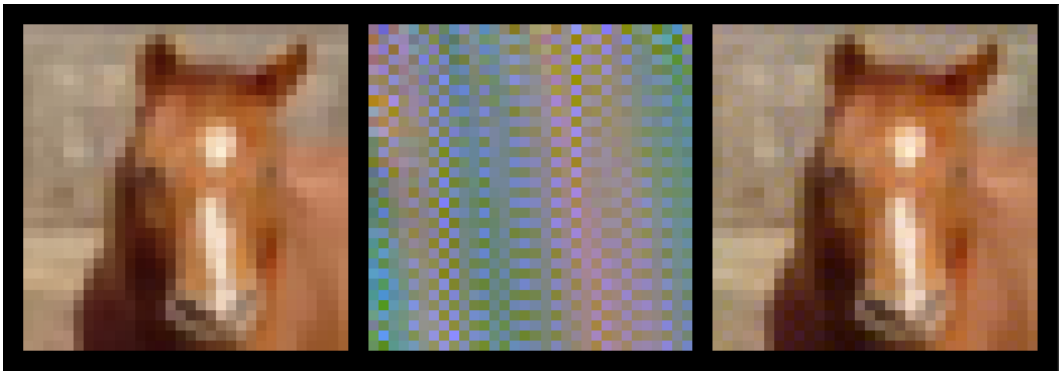

Figure 5: A closeup of a CIFAR-10 image from our AR poison. From left to right, we display the clean image, the normalized perturbation, and the perturbed image. Looking closely at the background of the horse, one can make out the checkered blue pattern of this AR perturbation.

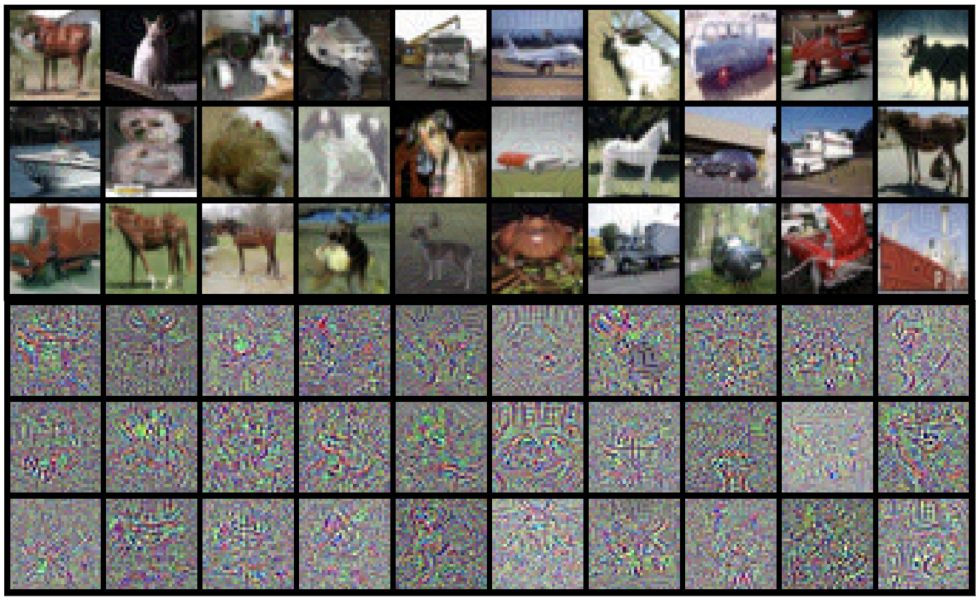

Figure 6: **Error-Max Poison.** We plot a random batch of 30 images and the corresponding normalized perturbation.

A.6 Computing Resources

To run all of our experiments, we use an internal cluster. Jobs were run on a maximum of 4 NVIDIA GTX 1080Ti.

A.7 t-SNE of Poisons

In analyzing the simplicity of AR perturbations, it is also appropriate to better understand other kinds of perturbations. In Figure 12, we use t-SNE to plot the perturbations of other poisons considered in this work. The clustering of error-minimizing poisons is particularly interesting, with the noise being more separable than that of AR.

A.8 Limitations

There are a number of modifications which could potentially make for more effective AR perturbations, the exploration of which we leave to future work. For example, the size of the sliding window or the

Figure 7: **Error-Min Poison.** We plot a random batch of 30 images and the corresponding normalized perturbation.

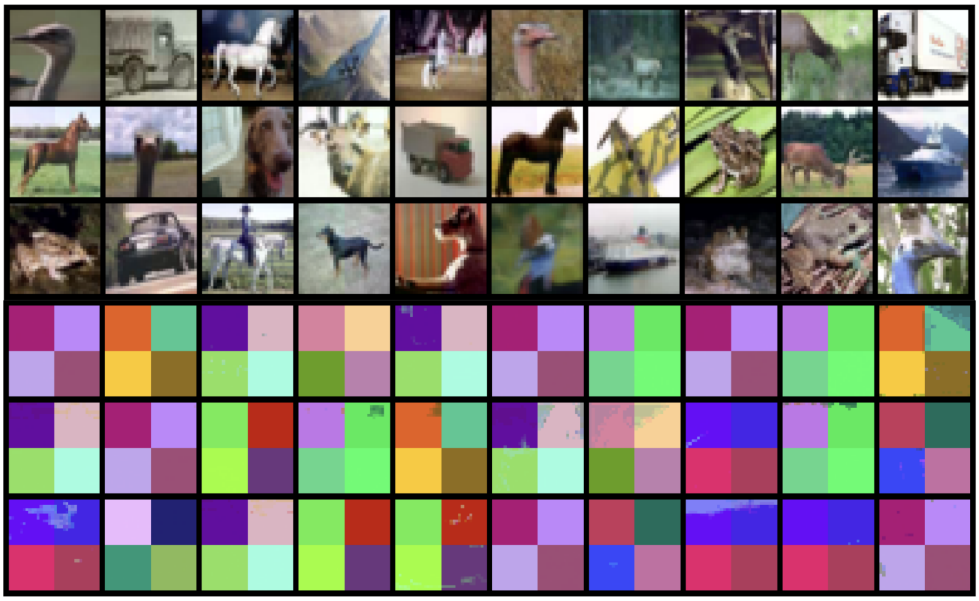

Figure 8: **Regions-4 Poison.**. We plot a random batch of 30 images and the corresponding normalized perturbation.

order of the AR process could potentially lead to perturbation values which converge more slowly. A higher order AR process could also be applied along the channel dimension.

As described in Section 3.2, poisoning a K-class classification dataset requires K AR processes. A random search for AR coefficients is certainly an area of improvement. Nevertheless, having found a set of K AR processes, Table 1 is evidence that they can be reused for any other classification dataset with K or fewer classes.

Figure 9: **Regions-16 Poison.** We plot a random batch of 30 images and the corresponding normalized perturbation.

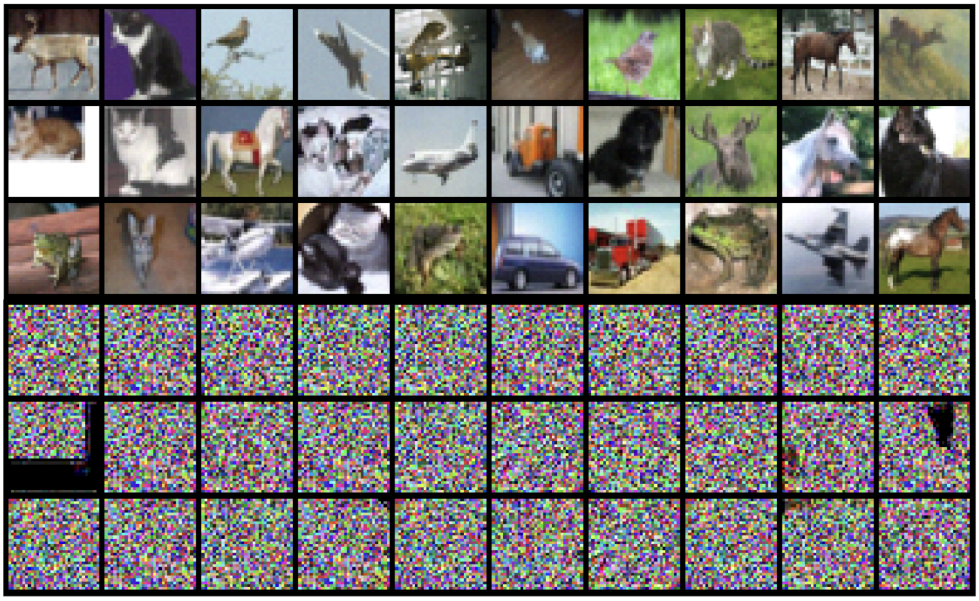

Figure 10: **Random Noise Poison.** We plot a random batch of 30 images and the corresponding normalized perturbation.

A.9 Broader Impact Statement

While data poisoning could be used as an attack, the research community has actively been interested in its use for privacy protection too. Our work has the potential to protect private data or to allow for the secure release of datasets. The likelihood that an adversary is able to perturb an entire dataset for the purpose of preventing generalization to a test set is miniscule. On the other hand, the likelihood that a private entity might want methods that allow them to release a dataset securely is more likely. Our work is motivated by understanding the kinds of features that deep neural networks choose to learn.

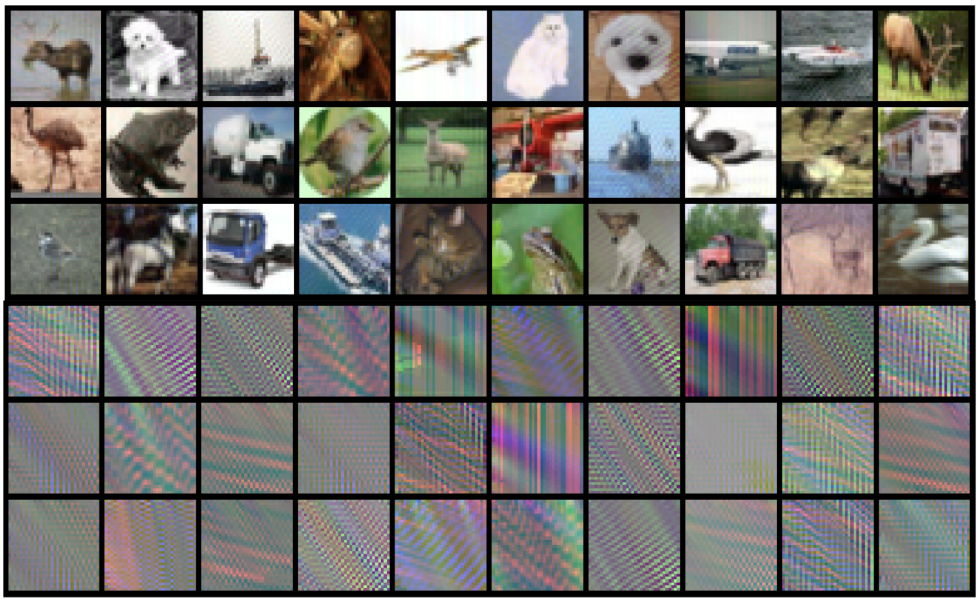

Figure 11: **AR Poison (Ours).** We plot a random batch of 30 images and the corresponding normalized perturbation.

Figure 12: t-SNE plots of perturbations within our Error-Max, Error-Min, and AR CIFAR-10 poisons.

A.10 AR Poisoning in ℓ_∞

AR perturbations can be projected onto any ℓ_p-norm ball, including ℓ_∞. To demonstrate that AR poisoning *can* work in the ℓ_∞-norm constrained setting, we provide CIFAR-10 test accuracy for a RN-18, where perturbations are of size $\frac{8}{255}$ in ℓ_∞-norm in Table 6.

A.11 10 effective AR Processes

On the final pages, we provide coefficients for the 10 AR(8) processes which are used to generate the AR poisons for SVHN, CIFAR-10, and STL-10. Because we apply a different AR process for each channel in 3-channel images, there are really 30 AR(8) processes (each 3×3 array defines an AR(8) process). Each $3 \times 3 \times 3$ dimensional array defines the three AR processes needed to perturb a 3-dimensional image from a class. We separate each class' AR process with a newline. Note that each 3×3 array has a zero in the final entry because there are only 8 AR coefficients when operating in a 3×3 sliding window.

Table 6: **Poisoining in ℓ_∞.** CIFAR-10 test accuracy of RN-18 when training using standard augmentations plus Cutout, CutMix, or Mixup.

	Standard Aug	+Cutout	+CutMix	+Mixup
Autoregressive (Ours)	20.49	26.93	17.08	15.22

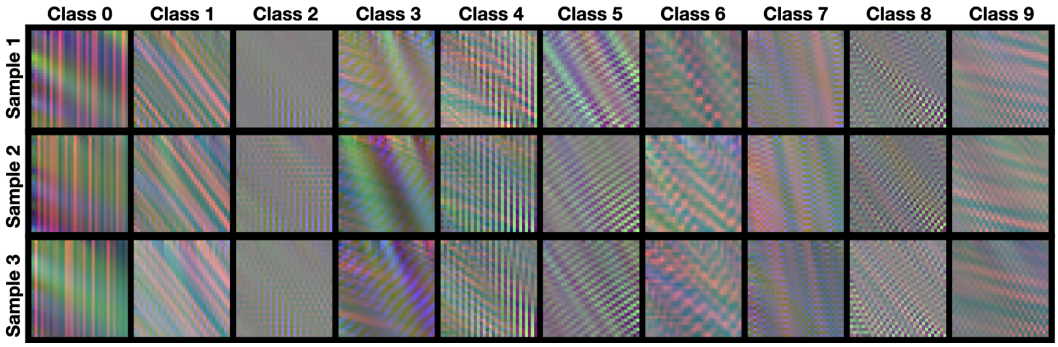

Figure 13: Samples of normalized AR perturbations. We generate three independent samples per AR process to illustrate how intraclass noises are unique, yet similar. These 10 processes were used to construct poisons for SVHN, CIFAR-10, and STL-10.

```
[[[[ 0.1561, -0.0710,  0.3743],
[-0.1896,  0.0461,  0.6075],
[ 0.0539,  0.0226,  0.0000]],
[[-0.1016,  0.2193,  0.0472],
[ 0.1401,  0.1561,  0.1171],
[ 0.1742,  0.2476,  0.0000]],
[[-0.1100,  0.2703, -0.0026],
[ 0.2662, -0.1185,  0.0846],
[ 0.1812,  0.4287, -0.0000]]],

[[[ 0.2346,  0.2056, -0.0480],
[ 0.4110,  0.7504,  0.1326],
[-0.1044, -0.5817,  0.0000]],
[[-0.0308, -0.1085,  0.3997],
[ 0.2187,  0.1830,  0.3389],
[-0.1017,  0.1008, -0.0000]],
[[ 0.1246,  0.1667, -0.1518],
[ 0.2985,  0.1346,  0.3185],
[ 0.3805, -0.2716,  0.0000]]],

[[[ 0.0951,  0.5501,  0.0344],
[-0.3431,  0.0767,  0.2239],
[ 0.4602, -0.0972,  0.0000]],
[[ 0.1714,  0.1121,  0.1129],
[ 0.3032,  0.2959,  0.0861],
[ 0.2207, -0.3021,  0.0000]],
[[ 0.2720, -0.3417,  0.2115],
[ 0.2499,  0.3690,  0.3833],
[-0.0282, -0.1158, -0.0000]]],

[[[-0.4241, -0.2694,  0.4091],
[ 0.3998,  0.1572,  0.2683],
[ 0.2700,  0.1892, -0.0000]],
[[ 0.1246,  0.3024,  0.2110],
[ 0.0538,  0.1997,  0.3283],
[ 0.0372, -0.2570,  0.0000]],
[[ 0.2038,  0.3972, -0.1963],
[ 0.3460, -0.6439,  0.7153],
[-0.2826,  0.4605, -0.0000]]],

[[[ 0.7873, -0.1756, -0.3509],
[ 0.0763,  0.2261, -0.2704],
[ 0.2491,  0.4581, -0.0000]],
[[ 0.1287, -0.1655,  0.2488],
[ 0.3811, -0.2307,  0.3019],
[-0.0076,  0.3433, -0.0000]],
[[ 0.4227,  0.0690,  0.2492],
[ 0.0896,  0.0653, -0.1653],
[-0.2349,  0.5043, -0.0000]]],
```

```
[[[ 0.1585,  0.2663,  0.1764],
[ 0.0031, -0.0237,  0.3464],
[ 0.0140,  0.0590, -0.0000]],
[[ 0.1632,  0.5094,  0.0321],
[ 0.3935, -0.1807,  0.2110],
[-0.3620,  0.2334, -0.0000]],
[[-0.2474,  0.1092,  0.3928],
[ 0.2808,  0.3912,  0.1211],
[-0.0635,  0.0159,  0.0000]]],

[[[ 0.8886, -0.2459, -0.4169],
[-0.4120, -0.1282,  0.6105],
[ 0.2495,  0.4546, -0.0000]],
[[ 0.0679, -0.2982,  0.1039],
[ 0.1430,  0.1596,  0.6743],
[-0.2002,  0.3496,  0.0000]],
[[-0.2195,  0.2183, -0.2665],
[ 0.3373,  0.1907,  0.5847],
[ 0.1977, -0.0427,  0.0000]]],

[[[-0.3829,  0.0158,  0.4019],
[-0.0688,  0.1206,  0.2481],
[ 0.1416,  0.5238, -0.0000]],
[[-0.2271,  0.1683,  0.3593],
[ 0.2671, -0.1225,  0.0217],
[ 0.0266,  0.5066, -0.0000]],
[[ 0.8539,  0.3682,  0.2899],
[ 0.6635,  0.0130, -0.7025],
[-0.1377, -0.3483,  0.0000]]],

[[[ 0.4482,  0.2220,  0.0598],
[ 0.3965, -0.1148,  0.1683],
[-0.3444,  0.1644,  0.0000]],
[[-0.1463,  0.6120,  0.2203],
[ 0.4039, -0.2832, -0.1290],
[ 0.3369, -0.0146,  0.0000]],
[[ 0.3759,  0.2814,  0.1494],
[ 0.1925,  0.0552,  0.1091],
[-0.0962, -0.0673,  0.0000]]],

[[[ 0.3556,  0.3477, -0.6625],
[ 0.1812,  0.2997, -0.2139],
[ 0.5538,  0.1385,  0.0000]],
[[-0.0297,  0.0780,  0.2486],
[ 0.2246,  0.1931,  0.1349],
[ 0.0079,  0.1426,  0.0000]],
[[ 0.2461,  0.1217,  0.1879],
[-0.0165,  0.1160,  0.1356],
[ 0.1772,  0.0321, -0.0000]]]]
```