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

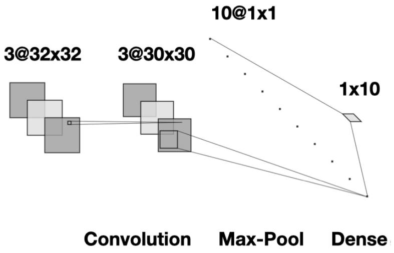

Figure 4: A 3-layer CNN can perfectly classify AR perturbations without any training. We specify each parameter manually.

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 i^{th} 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 i^{th} logit when the i^{th} 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 i^{th} 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:     $\vec{\beta} \leftarrow \mathcal{N}(0,1)$
 4:     $\vec{\beta} \leftarrow \frac{\vec{\beta}}{\sum_j \beta_j}$
 5:     $\delta \leftarrow \mathrm{ARGenerate}(\vec{\beta})$
 6:     if $\delta$ is stable then
 7:         Let $m$ store the min response so far.
 8:         for $i$ in 1,...,$|A|$ do
 9:             $f \leftarrow \mathrm{ARFilter}(A_i)$
10:             $r \leftarrow \mathrm{ConvResponse}(\delta, f)$
11:             $m \leftarrow \min(m, r)$
12:         end for
13:         if $m \geq T$ then
14:             $A \leftarrow A \cup \{\beta_i\}$
15:         end if
16:     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 $\epsilon$ in $\ell_2$
Output: Poisoned image, $x_i'$
Sample $\mathcal{N}(0,1)$ for first $V-1$ rows and columns of $\delta_i$
for $k$ in $1, ..., C$ do
    $\beta_1, ..., \beta_{V^2-1} \leftarrow A_{y_i}^k$
    $\delta_i^k \leftarrow \mathrm{ARGenerate}(\beta_1, ..., \beta_{V^2-1})$
end for
$\delta_i \leftarrow \epsilon \frac{\delta_i}{\|\delta_i\|_2}$
$x_i' \leftarrow x_i + \delta_i$
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.
- $\mathrm{ARGenerate}(\vec{\beta})$ 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 $\mathrm{ARGenerate}(\vec{\beta})$ 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 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 $96 \times 96 \times 3$.

A.5 Computing Resources

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

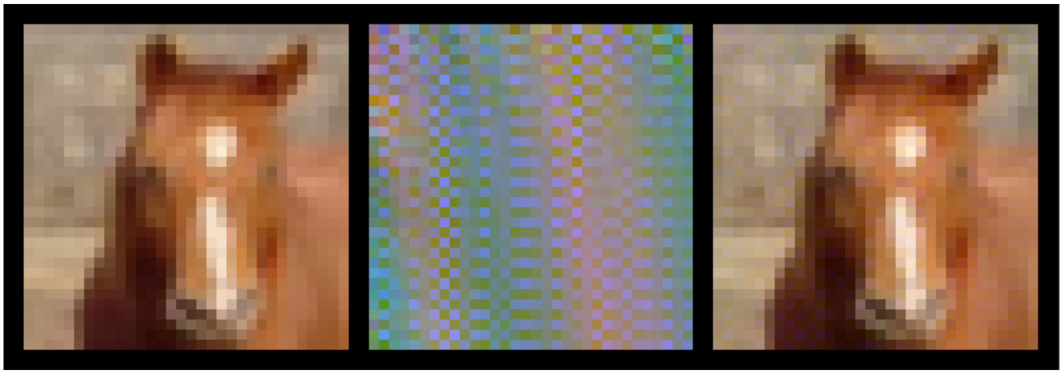

Figure 5: A closeup of a CIFAR-10 image from our AR poison. From left to right, we display the clean image, the normalized perturbation, and the perturbed image. Looking closely at the background of the horse, one can make out the checkered blue pattern of this AR perturbation.

A.6 AR Poisoning in ℓ_∞

Autoregressive perturbations can be projected onto any ℓ_p-norm ball, including ℓ_∞. We initially measured the ℓ_2-norm because AR perturbations may have single entries which are high and violate a strict ℓ_∞ constraint. To demonstrate that AR poisoning *can* work in the ℓ_∞-norm constrained setting, we train a RN-18 on an AR poison with perturbations satisfying $\|\delta\|_\infty = \frac{8}{255}$, and report CIFAR-10 test accuracy in Table 6.

Table 6: **Poisoining in ℓ_∞.** CIFAR-10 test accuracy of an RN-18 trained on an AR poison using standard augmentations plus Cutout, CutMix, or Mixup.

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

Importantly, how one fits an AR perturbation δ within the constraint that $\|\delta\|_\infty \leq \epsilon = \frac{8}{255}$ affects performance. In this experiment, we simply scale δ by $\frac{\epsilon}{\|\delta\|_\infty}$, which may be suboptimal. Clipping values or taking the scaled sign of δ would make the perturbation no longer autoregressive. AR perturbations could likely be optimized to perform better in ℓ_∞ by making modifications to how AR coefficients are compared in Algorithm 1.

A.7 Effect of Optimizer Hyperparameters

To evaluate the effect of optimizer and learning rate on our poisoning results, we train an RN-18 on poisons using different optimizers. We consider 3 optimizers: SGD, SGD+Momentum ($\beta = 0.9$) and Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$). We also consider 3 different learning-rate schedules: cosine learning-rate, single-step decay (at epoch 50) and multi-step decay (at epochs 50 and 75) with decay factor of 0.1. There are a total of 9 optimizer and learning rate combinations.

Table 7: **Effect of Optimizer and LR.** Mean CIFAR-10 test accuracy of an RN-18 trained on our AR poison, over 9 optimizer and LR combinations.

Autoregressive (Ours)	Random Noise	Regions-4	Regions-16	Error-Max.	Error-Min
$12.23_{\pm 1.22}$	$33.39_{\pm 31.24}$	$22.89_{\pm 7.6}$	$18.73_{\pm 5.52}$	$16.6_{\pm 2.8}$	$22.96_{\pm 7.16}$

In Table 7, we report mean CIFAR-10 test accuracy over the 9 combinations of optimizers and learning rates. Our Autoregressive poison remains nearly unaffected by the choice of hyperparameters, and performs better than all other poisons.

A.8 Samples of CIFAR-10 Poisons

We plot a random sample of 30 poison images from each CIFAR-10 poison used in this work. In each plot, we also illustrate the corresponding normalized perturbation for each image. The Error-Max poison is shown in Figure 6, Error-Min poison in Figure 7, Regions-4 poison in Figure 8, Regions-16 poison in Figure 9, Random Noise poison in Figure 10, and our AR poison is shown in Figure 11.

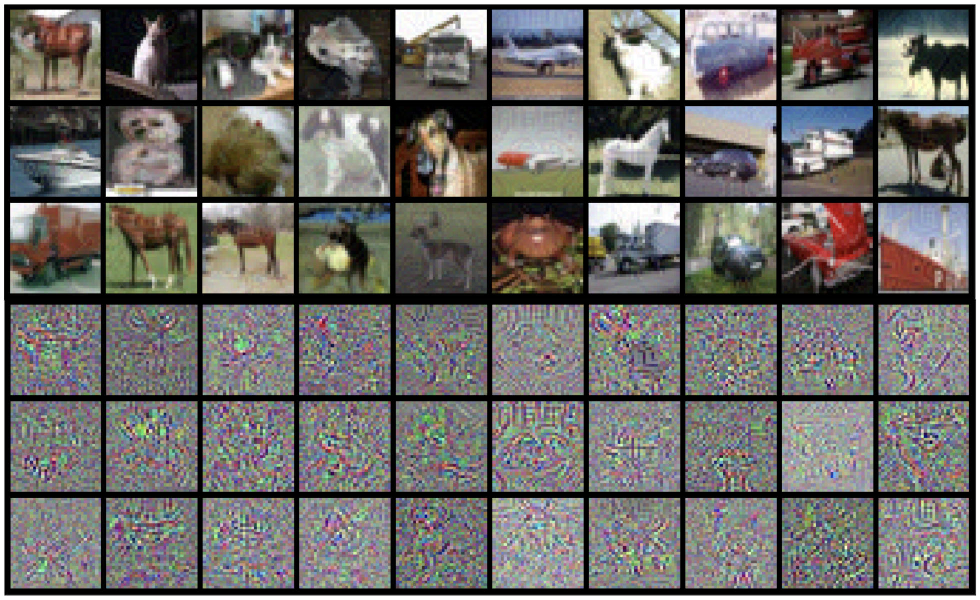

Figure 6: **Error-Max Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

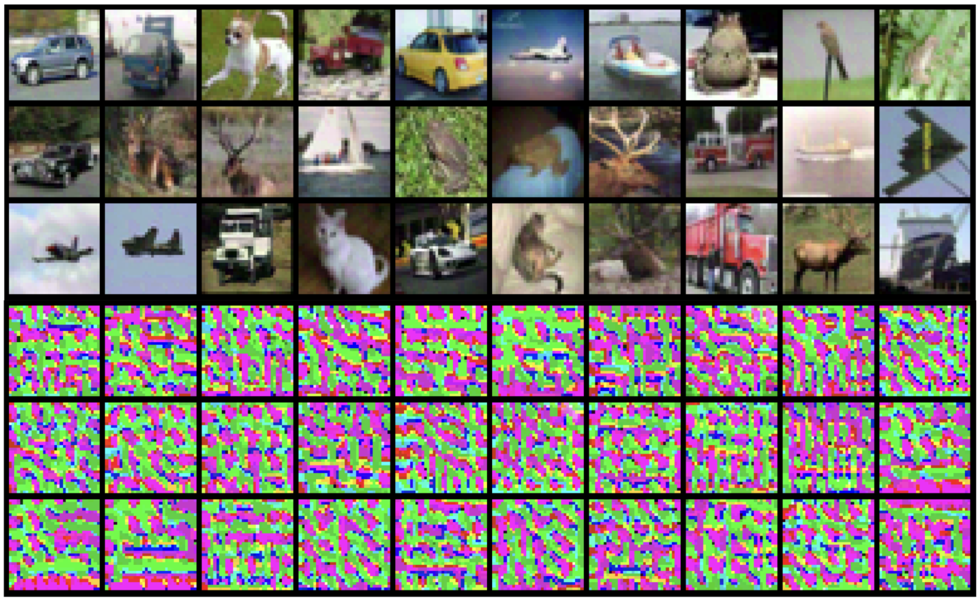

Figure 7: **Error-Min Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

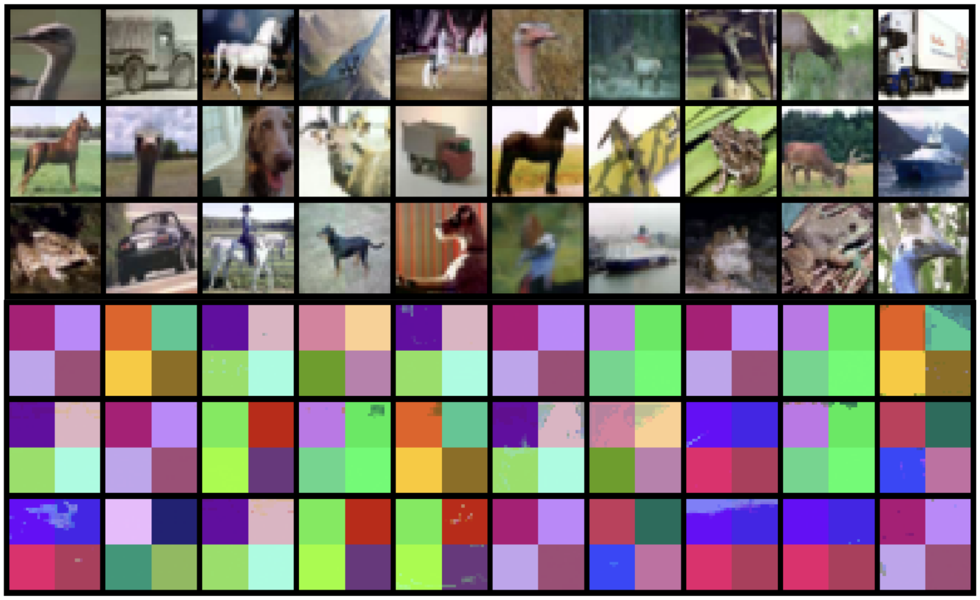

Figure 8: **Regions-4 Poison.**. A random sample of 30 poison images and their corresponding normalized perturbation.

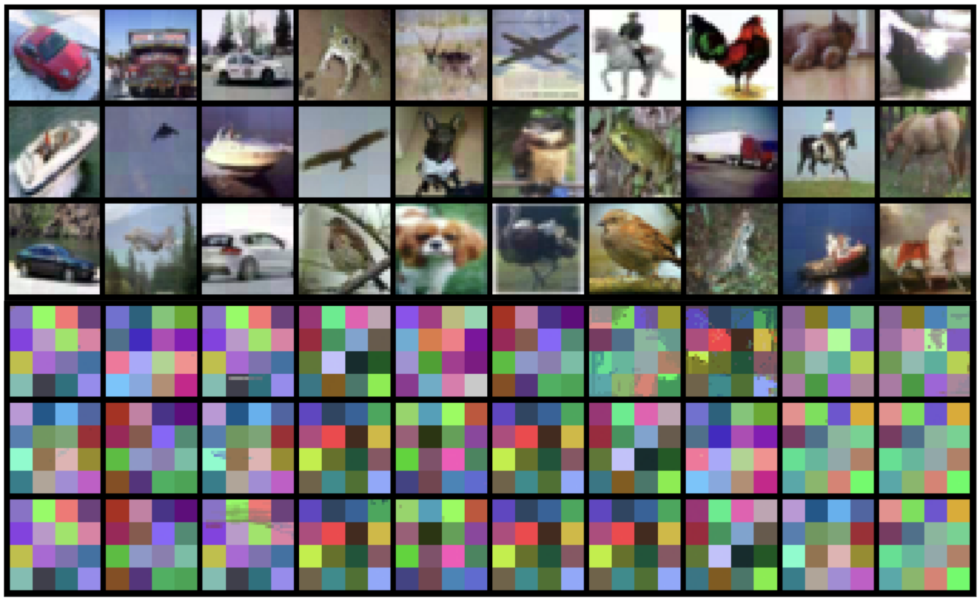

Figure 9: **Regions-16 Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

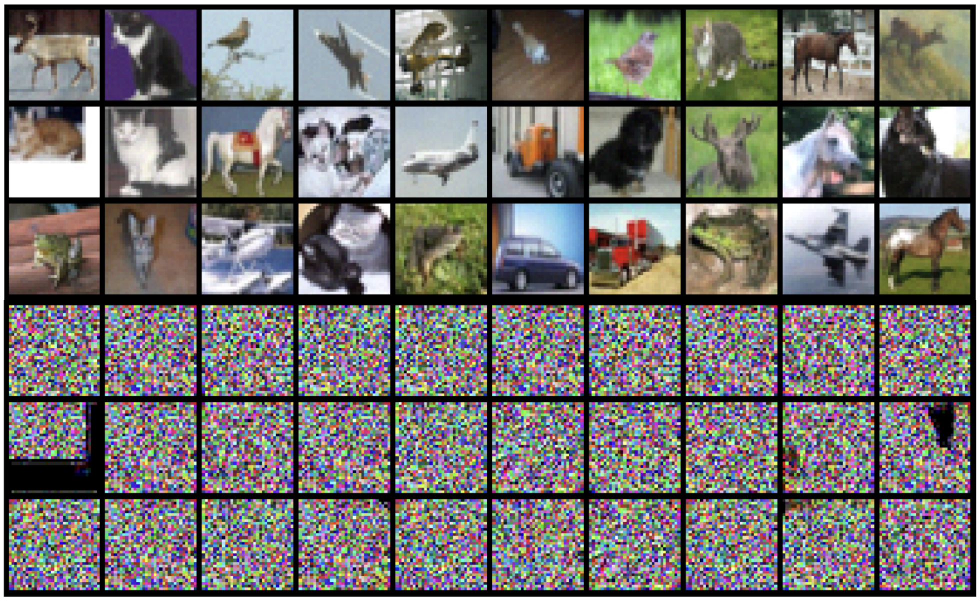

Figure 10: **Random Noise Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

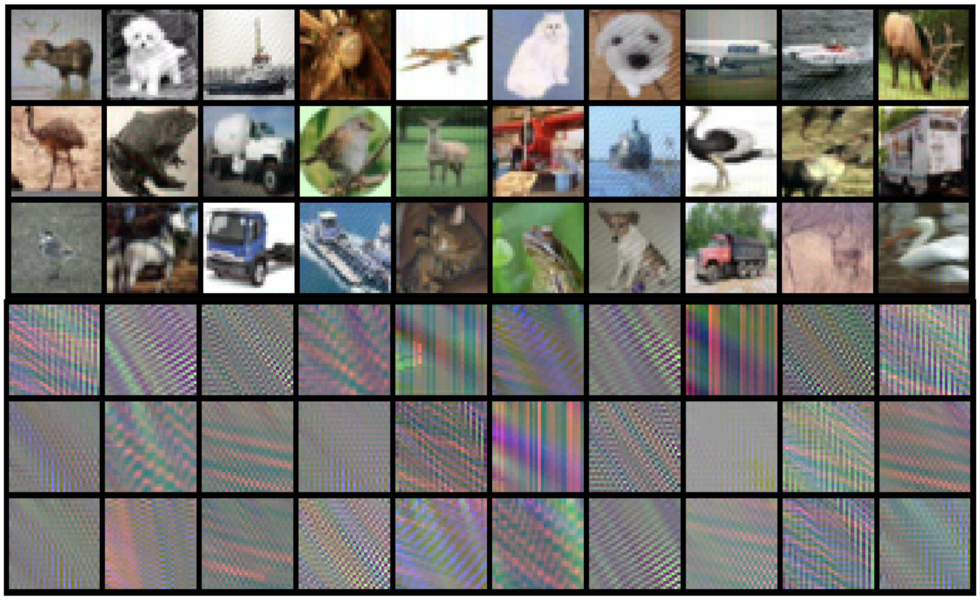

Figure 11: **AR Poison (Ours).** A random sample of 30 poison images and their corresponding normalized perturbation.

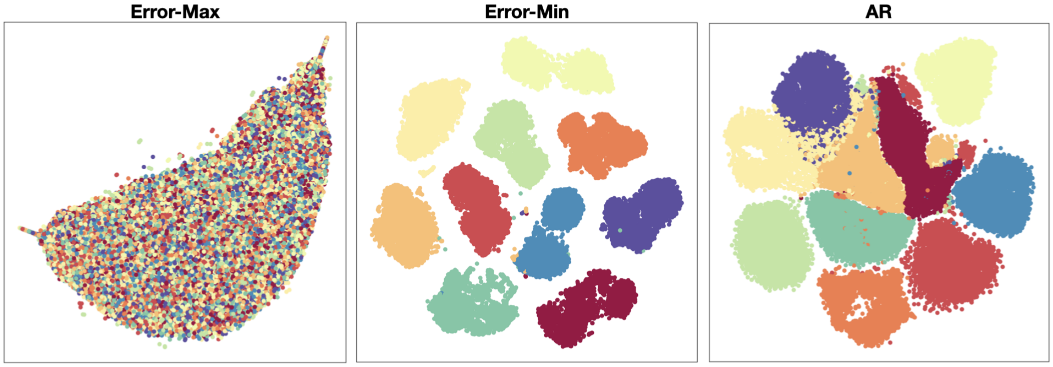

Figure 12: t-SNE plots of perturbations from our Error-Max, Error-Min, and AR CIFAR-10 poisons.

A.9 t-SNE of Poisons

In Figure 12, we use t-SNE to plot the perturbation vectors of Error-Max, Error-Min, and Autoregressive (AR) poisons. For every poison CIFAR-10 image, we subtract the corresponding clean image and plot only the perturbation vectors using t-SNE. We find that the clustering of error-minimizing perturbations appears to be more separable than that of AR.

A.10 10 effective AR Processes

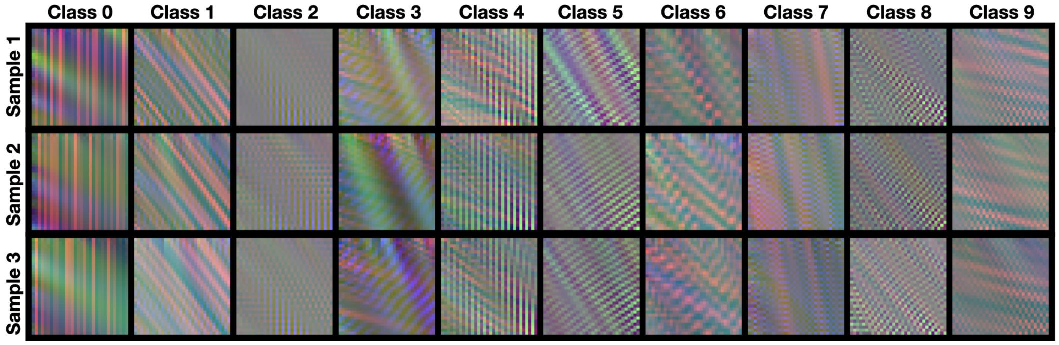

Figure 13: Samples of normalized AR perturbations. We generate three independent samples per AR process to illustrate how intraclass noises are unique, yet similar. These 10 processes were used to construct poisons for SVHN, CIFAR-10, and STL-10.

On the final pages, we provide coefficients for the 10 effective AR(8) processes which are used to generate the AR poisons for SVHN, CIFAR-10, and STL-10. Samples of perturbations from these 10 AR processes are shown in Figure 13. 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.

A.11 Code

Code for generating AR poisons and training models is available at `https://github.com/psandovalsegura/autoregressive-poisoning`. We provide documentation as well as demo Jupyter notebooks for generating perturbations using AR processes. AR coefficients from Appendix A.10 and full poisoned AR datasets used in this work are also available for download.

A.12 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 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. The random search for AR coefficients is certainly an area of improvement; there may exist more efficient ways of finding effective AR coefficients. 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.

A.13 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.

```
[[[[ 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]]]]
```