Generating Adversarial Examples with Diffusion Models and Reinforcement Learning

Gilad Deutch Nadav Magar
giladd@mail.tau.ac.il nadavmagar@mail.tau.ac.il

ABSTRACT

Diffusion models have risen as the state of the art class of generative models for image generation. In this work we investigate how their impressive expressive power could be used for the task of adversarial examples generation. We describe how pretrained diffusion model could be fine-tuned to attack a target classifier in the black-box setting using reinforcement training. Specifically we adapt the denoising diffusion policy optimization (DDPO) method to the black box attack setting

Our method is able to generate natural adversarial examples - perturbations that remain in the original model's distribution. Finally, we address the limitations of the proposed method and discuss future directions for this work. Our code is available at - https://github.com/GiilDe/adv-ddpo

1 INTRODUCTION

Diffusion probabilistic models [8, 18] have recently emerged as the de-facto standard for generative modeling, beating the previous state of the art results of generative adversarial networks (GANs) in image generation tasks [5]. Their ability to represent complex, high-dimensional distributions has led to new applications in image synthesis and editing [13, 14].

This impressive expressive power makes diffusion models a good candidate for the synthesis of images from complex domains. In work we study their use for the generation of adversarial examples, images that appear benign but aim to mislead neural network classifiers to false predictions [6, 19].

Traditional methods perform optimization over added perturbations for each individual image [3, 10]. However when large quantities of adversarial examples are needed (e.g. in adversarial training [10]) the computational cost of such methods may not be feasible [16]. While some previous works train a generative model for adversarial examples, they mainly focused on generative adversarial networks [1, 12]. In this work we fine-tune a benign diffusion model to attack a target classifier in the black-box setting. We apply recent techniques from reinforcement learning [2] to this setting, allowing for efficient training of the diffusion model given access only to the targeted classifier's logits.

2 RELATED WORK

2.1 Diffusion Models

During inference, diffusion models work by iteratively "cleaning" a given noisy image. At each iteration the model inference can be described by $\mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t)$. I.e the model inference is leveraged by a scheduler to predict the next mean of normal distribution, and some noise is injected by the scheduler so that the process is not deterministic. During training, the model is not iterative. At each training step, a noisy image is created by adding noise ϵ, $x_t = a_t * x_0 + b_t * \epsilon$. a_t, b_t are scalars that depend on t and the scheduler and we ignore their exact identity at this time. The model's loss is to simply predict the added noise given the time step and the noisy image $||\epsilon_\theta(x_t, t) - \epsilon||_2^2$.

2.2 Denoising Diffusion Policy Optimization

A recent connection between diffusion models and reinforcement learning has been made by [2]. The iterative nature of diffusion models, allow the authors to define the denoising process as a multi-step decision-making process, setting which is typical and natural for RL. This allows the authors to optimise the denoising process with policy gradients, a type of RL algorithms that directly optimize the decision making process of the model. Another point that the authors raise for using RL with diffusion models is that most formulations of diffusion models are concerned with the likelihood objective. That allows the model to map noisy inputs into a desired target distribution, but do not allow optimising a desired downstream objective. RL allows that. Two properties of RL, the first being that target model gradients are unnecessary, and the second being that it allows us to optimise downstream objective, are helpful to us in the setting of black-box attack of target models.

2.3 Generative Models for Adversarial Examples

Previous works on generational models for adversarial examples mainly focus on GANs as the generator backbone. Notably [20] train a generator to learn a distribution of bounded adversarial *perturbations* conditioned on clean images. The discriminator is trained to distinguish between these perturbed imaged and their original clean counterparts. In order to produce examples that fool a specific classifier, an adversarial loss term is added for either untrageted or targeted attacks. The authors suggest a method for both the "semi-whitebox" setting, where direct access to the classifier model is given during training time only, and a method for the black-box setting, where both the generator and a surrogate classifier are trained jointly.

3 METHOD

Given black-box access to a classifiers, we fine-tune a pretrained diffusion model using denoising diffusion policy optimization [2]. The fine tuned model learns to generate novel examples that are misclassified by the target classifier. Following other generative model based editing pipelines [7, 11] we generate an adversarial example for a given benign image by first inverting it with respect to the original diffusion model, and then feed the latent noise into the fine-tuned model. That means that it's important that our fine-tuned model generates images that are similar to the ones the original model generates. For that reason, during training we use regularization methods to constrain the fine-tuned model to the original one. The regularization also allows the fine-tuned model to generate pertrubations that aren't too noticeable.

The following sections describe in detail each component of our proposed method.

3.1 DDPO Training for Adversarial Examples

To use RL in our training, we need to first define the denoising diffusion process as a policy. We do it in the following way

$$\pi(a_t, s_t) = p_\theta(x_{t-1}|t, x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t)$$

In words, our policy is simply the distribution that the denoising process samples from in each of its steps. The mean of the distribution is determined by the prediction of the model and the standard deviation is a computation on the given parameter σ and the time step. In our case σ has to be strictly positive, since otherwise x_{t-1} is deterministic and the PPO process becomes ill-defined.

Our vanilla reward is the following:

$$r(x_0) = 1 - \text{true-label-score}(x_0)$$

where x_0 is the generated adversarial image. I.e we simply reward for generating an image that lowers the true label as much as possible. We experiment with two more rewards as described below. Note that in practice to determine the true label we feed the same initial latent noise, and the same eta during the denoising process to the fine-tuned and the original model. The images from the original model are fed into a classifier whose classification determine the true labels for us.

We now define the loss following the PPO algorithm [15]. The PPO algorithm uses the model from the previous epoch for data collection, denoted as θ_{old}. Let us denote the ratio by

$$R_t = \frac{p_\theta(x_{t-1}|t, x_t)}{p_{\theta_{old}}(x_{t-1}|t, x_t)}$$

Then, the loss for a single time step is

$$L_{PPO_t}(\theta) = min(R_t * r(x_0), clip(R_t, 1+\epsilon, 1-\epsilon) * r(x_0))$$

The final loss is the sum of all time step's losses

$$L_{PPO} = \sum_{t=0}^{T} L_{PPO_t}$$

The intuition behind PPO is that if the optimisation process "pushes" the policy too far away from the previous epochs' policy, in terms of the probability of choosing the next image x_{t-1}, then the loss is clipped. Note that the decision x_t is created by the old policy in the previous epoch. Another interesting thing to note is that all time steps for a given denoising process receive the same reward that is dependant on the final image.

Let us now describe our algorithm in overview. A key feature of our training process is that we keep the original model and constrain our fine-tuned model to not get too "far" from it with respect to the L_2, L_∞ norms of the generated images. This constraining is achieved by a few ways which we experiment with. First, the PPO algorithm itself, which as mentioned above, constrains the fine-tuned model to be similar to the previous epochs' model in terms of probability distribution. Secondly, a penalty in the reward between the final images, and lastly a L_2 loss on the two models' predictions. We later provide a simple mathematical analysis that shows that the PPO constraint and the L_2 loss between the models' predictions are almost equivalent.

Algorithm 1 RL training process

```
Require: original model: $\phi$
Require: model to fine-tune: $\theta$
Require: epochs: $n$
  while epochs $<$ $n$ do
    $z \sim \mathcal{N}(0,1)$
    $x_0, \sigma_{1,\dots,T} \leftarrow$ denoising-diffusion-process$(z, \phi)$
    labels $\leftarrow$ classifier$(x_0)$
    $x_{0_{\text{adv}}} \leftarrow$ denoising-diffusion-process$(z, \theta_{old}, \{\sigma \leftarrow \sigma_{1,\dots,T}\})$
    for train-loop-size times do
      $r \leftarrow r(x_{0_{\text{adv}}}, \text{labels}, x_0)$
      $L(\theta) \leftarrow L_{PPO}(\theta) + \lambda * L_{REG}(\theta)$
      $\theta \leftarrow \theta - \nabla_\theta L$
    end for
  end while
```

Algorithm 1 gives an overview of our training process. The blue parts are optional depending on whether we regularize by penalizing difference in the reward or by a direct L_2 loss between the two models predictions. The two diffusion processes use the same latent noise and the same random σ's that are sampled during the diffusion process (in practice we randomly sample in the first process and pass the sampled noise to be used in the second process). Using the same σ's in the two process allows us to have a stochastic diffusion process, as is required for the PPO algorithm, but also allows us to be able to compare x_0, the images that are generated from the original model to $x_{0_{\text{adv}}}$, the images that are generated by the fine-tuned model.

3.1.1 *Reward.*

we experiment with 3 different reward functions. The first is the vanilla reward function described above

$$r(x_0) = 1 - \text{true-label-score}(x_0)$$

The second reward function we experiment with is the negative cross entropy loss (which is also simply applying *log* on the vanilla reward)

$$r(x_0) = \log(1 - \text{true-label-score}(x_0))$$

The final reward we experiment with, is a reward we call the hinge reward

$$r(x_0) = \min(0, \text{max-score}(x_0) - \text{true-label-score}(x_0) - c)$$

where max-score is the maximum score ignoring the score of the correct label and c is a give (small) parameter. The motivation for the hinge reward and log reward is that for images that already have relatively high reward, i.e trick the classifier, we no longer need to over optimise them. We expect that the log reward and hinge reward be better with respect to decreasing the overall accuracy of the classifier on the perturbed images, but unfortunately in practice we see that the 3 rewards have almost the same results.

Note that, as common in RL, we normalize our reward to be distributed as the standard normal distribution, $\mathcal{N}(0,1)$ by subtracting the batch mean and dividing by the batch standard deviation.

3.1.2 Regularization. We use two main ways to constrain the fine-tuned model to the original model. The first is penalising the reward using the distance norm. We simply define a new reward

$$r_{REG}(x_0, x_{0_{adv}}) = r(x_0) - \lambda * norm(x_0, x_{0_{adv}})$$

where norm is either the L_2 or the L_∞ distances. Similarly to the hedge loss in [20], we experiment with an hedge reward

$$r_{REG}(x_0, x_{0_{adv}}) = r(x_0) - \lambda * \max(0, norm(x_0, x_{0_{adv}}) - c)$$

for some parameter c. The motivation is to not penalise for small distances, since some challenges allow them, for example the MNIST challenge [10] allows perturbations of up to $L_\infty = 0.3$.

The second form of regularization includes a direct L_2 loss between the prediction of the original model and the fine-tuned one.

$$L_{REG}(\theta) = ||\epsilon_\theta(x_t, t) - \epsilon_\phi(x_t, t)||_2^2$$

where t is randomly sampled, and the respective x_t is retrieved from a denoising process of the original model. The motivation behind this is that RL training processes are complicated and hard to converge as is, hence we think it's preferable to keep the reward simple with only one factor.

It turns out that the L_2 loss that we implement is almost equivalent to the KL divergence between the policies $\mathcal{N}_\theta, \mathcal{N}_\phi$ that the original and fine-tuned models define, the only difference is multiplicative factor that depends on the time step and the diffusion process' scheduler.

$$\begin{aligned} KL(\mathcal{N}_\theta, \mathcal{N}_\phi) &= \frac{1}{\sigma_t^2}||\epsilon_\theta(x_t, t) - \epsilon_\phi(x_t, t)||_2^2 \\ &= \frac{1}{\sigma_t^2} L_{REG}(\theta) \end{aligned}$$

We show proof and discuss this in more detail in the appendix A.2.

3.2 Inversion

Generating an adversarial examples corresponding to real image requires *inverting* the given image. That is, finding an initial noise vector that produces a good reconstruction of the input image when fed into the (backwards) diffusion process.

Since we aim to accurately reconstruct a given real image, we employ the deterministic DDIM sampling [17]:

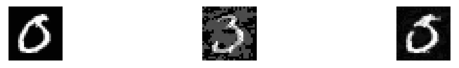
$$x_{t-1} = \sqrt{\frac{\alpha_{t-1}}{\alpha_t}} x_t + \left(\sqrt{\frac{1}{\alpha_{t-1}} - 1} - \sqrt{\frac{1}{\alpha_t} - 1}\right) \cdot \epsilon_\theta(x_t, t).$$

Because this process is deterministic, one can apply it in reversed order in order to invert an image [5, 17]. This method is based on the assumption that the ODE process can be reversed in the limit of small steps:

$$x_{t+1} = \sqrt{\frac{\alpha_{t+1}}{\alpha_t}} x_t + \left(\sqrt{\frac{1}{\alpha_{t+1}} - 1} - \sqrt{\frac{1}{\alpha_t} - 1}\right) \cdot \epsilon_\theta(x_t, t).$$

In other words, the diffusion process is performed in the forward direction, that is $z_0 \longrightarrow z_T$ instead of $z_T \longrightarrow z_0$, where z_0 is set to be the encoding of the given real image.

In practice, a slight error is incorporated in every step. For unconditional diffusion models such as the ones used in this project, the accumulated error is negligible and the DDIM inversion succeeds.



(a) Original **(b) Fine-tuned** **(c) Fine-tuned + regularized**

Figure 1: The original, fine-tuned and regularized fine-tuned models' predictions for the same latent noise. The fine-tuned model attack is effective but it may predict a different class, which is problematic for our RL training process, plus problematic for inversion. The regularized model predicts similar output but its attack is ineffective.

3.3 Pretraining

Due to the lack of ability of the RL training process to generate typical L_∞ bounded perturbations, we suggest to pretrain the model. This is a common method in RL as mentioned in 5.1, this section also includes more details about our motivation. Our pretraining constitutes of retrieving images from data, generating attacks on them, and fine-tuning a diffusion model using a diffusion training process to learn to generate perturbed images. We used, in practice, the FGSM attack, but for proper experimentation a future step should be to experiment with black-box attacks. Note that in the RL training process we fine-tune a model while constraining it to not drift too far away from the original model. Here again, we need to fine-tune the model while constraining it to the original model. For that reason, we add a regularization term to the diffusion loss

$$L_\theta = ||\epsilon_\theta(x_t, t) - \epsilon||_2^2 + \lambda * ||\epsilon_\theta(x_t, t) - \epsilon_\phi(x_t, t)||_2^2$$

Where ϕ is the original model, θ the fine-tuned model, and ϵ is noise that's added to an MNIST image attacked by FGSM.

We find that the regularization term is somewhat effective in constraining the model, yet the learned attacks become less effective as well. We find, somewhat against our expectations, that to learn the perturbations themselves is relatively easy for the diffusion model, and our bottleneck is the ability to constrain it to the original model. Figure 1 shows the results of the regularized and non-regularized diffusion training.

4 EXPERIMENTAL RESULTS

4.1 Inversion

The inversion process introduces an error to the reconstruction a given image. Given an benign image $\mathbf{x}_0$ let us denote by $\tilde{\mathbf{x}}_T$ the noise vector recovered by inverting it using the benign model, and $\tilde{\mathbf{x}}_0$ as the reconstruction generated by passing $\tilde{\mathbf{x}}_0$ through the backwards diffusion process. We generate an adversarial version of $\mathbf{x}_0$ by passing $\tilde{\mathbf{x}}_T$ through the fine-tuned diffusion model: $y_0 := G_{adv}(\tilde{\mathbf{x}}_T)$. The difference between the adversarial example and the benign image can be decomposed into an inversion reconstruction term and the perturbation induced by the model:

$$\begin{aligned} ||\mathbf{x}_0 - y_0|| &\approx ||\mathbf{x}_0 - \tilde{\mathbf{x}}_0|| + ||\tilde{\mathbf{x}}_0 - y_0|| && (1) \\ &= ||\mathbf{x}_0 - G(\tilde{\mathbf{x}}_T)|| + ||G(\tilde{\mathbf{x}}_T) - G_{adv}(\tilde{\mathbf{x}}_T)|| && (2) \end{aligned}$$

Note that the first term in (1) is determined by the inversion reconstruction error, while second corresponds to the fine-tuned

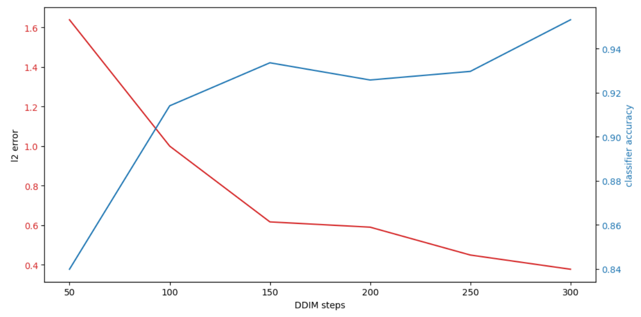


Figure 2: L_2 reconstruction error and target classifier accuracy as a function of the number of DDIM steps used for both inversion and sampling. Results are averaged over 1024 CIFAR-10 images.

model's perturbation magnitude and the degree to which it's latent space is preserved.

Figure 2 shows the inversion reconstruction error as a function of the amount of DDIM steps used. The results show the inherent trade-off of diffusion models between generation speed (measured in denosing steps) and visual quality.

An interesting observation is that even when the L_2 reconstruction error is high the reconstructed images appear qualitatively close to the original ones, even when the reconstructed image is misclassified. This phenomena may suggest that a diffusion model is able to produce imperceivable adversarial examples as explored in [4, 21].

4.2 Attack Effectiveness

We measure the effectiveness of our attack by computing its success rate on a subset of 1024 unseen CIFAR-10 images. It is common practice to measure an attack's effectiveness with respect to its perturbation magnitude [3, 9]. While some explicitly restrict the attack's perturbation magnitude, measured either in ℓ_∞ or ℓ_2 we opted to use only regularization as explained in section 3.1.2.

Figure 3 shows the attack success rate and perturbation magnitude throughout training. The results show a clear correlation between the attack's success rate and its magnitude. The dependence on large perturbations coupled with it's relatively low success rate compared to known attacks [3, 10] are the main limitations of our method. Table 2 shows the parameters for the run in Figure 3.

In Table 1 we qualitatively show adversarial examples produced by our method throughout the fine-tuning process. The results show that even when the perturbation induced by the model is high the produced adversarial examples retain semantic similarity to their benign counterparts.

In this work, we've also experimented with the MNIST dataset. We've seen that the metrics are somewhat worse than for CIFAR-10, we beileve this is due to the lower resolution of the images in MNIST.

4.3 Regularization

Figure 5 shows the effect of our 2 regularization methods. We see that they are effective in decreasing the divergence between the fine-tuned and original models. Unfortunately it seems that in order to do so, we "pay" with making our fine-tuned model less effective in its attacks. As we already mentioned, our model converges to perturbations that we term "natural" perturbations. In the discussion section we discuss the reasons that the potential of such perturbations is limited. Putting those reasons aside, we believe that our regularization causes ineffectiveness of our attacks since to begin with the potential of our attacks is limited. Hence, there simply exists no solution to converge to in the intersection of low norm distance and effective attack.

4.4 Pretraining

We conduct two experiments: one without regularization to the original model and one with regularization. In each experiment we trained for 1 epoch over the MNIST dataset, after which we generated some images and evaluated the effectiveness of their perturbations. We evaluate the success rate of the perturbations by manual eye-balling. The non regularized run had success rate of 60%. The regularized run was unsurprisingly less effective with a success rate of 19%. Figure 1 shows examples of the output of these runs, from manual eye-balling it seems that the perturbed images that the model learns to generate are similar to the ones that the FGSM generates. An interesting by-product insight of this experiment is that for a diffusion model it is relatively easy to learn to generate perturbed images using the standard diffusion loss.

A future step we suggest is to explore this direction using a conditioned model, that would allow us to generate images during the RL training process that we know the labels without an original model. That way we don't need to use regularization in the pretraining step which seems to be the bottleneck. (we remind the reader at this point that we way we retrieve the labels in the RL training is by generating images using the original model and classifying them, with a conditioned model we don't need that)

Figure 8 shows the output of a model that was pretrained and then went through RL training. Interestingly, it converges into an output that seems more like a typical "noisy" adversarial perturbation. This shows that pretraining + RL training may be promising.

5 DISCUSSION

5.1 Limitations of the Finetuning Process

Without pretraining, our RL training process is unable to learn to generate typical adversarial perturbations, similar to the ones you expect to get from vanilla algorithms, such as Fast Gradient Method [6]. We believe that there are two main reasons for that.

First, the RL training process is defined by **exploration**, meaning that the model first "suggests" a solution, and it is then rewarded accordingly. If the model is never able to generate one of the solutions from the desirable distribution (in our case, it may be L_{inf} bounded adversarial perturbations), it will never be rewarded for it, and subsequently will never be able to learn to generate it. This is a well-known and major problem in RL, and it is often remedied by supervised pretraining, which allows the model to generate desirable solutions in the exploration process of the RL training. In this work, we also attempt to add pretraining with limited success, because constraining the pretrained model to the clean one proves challenging.

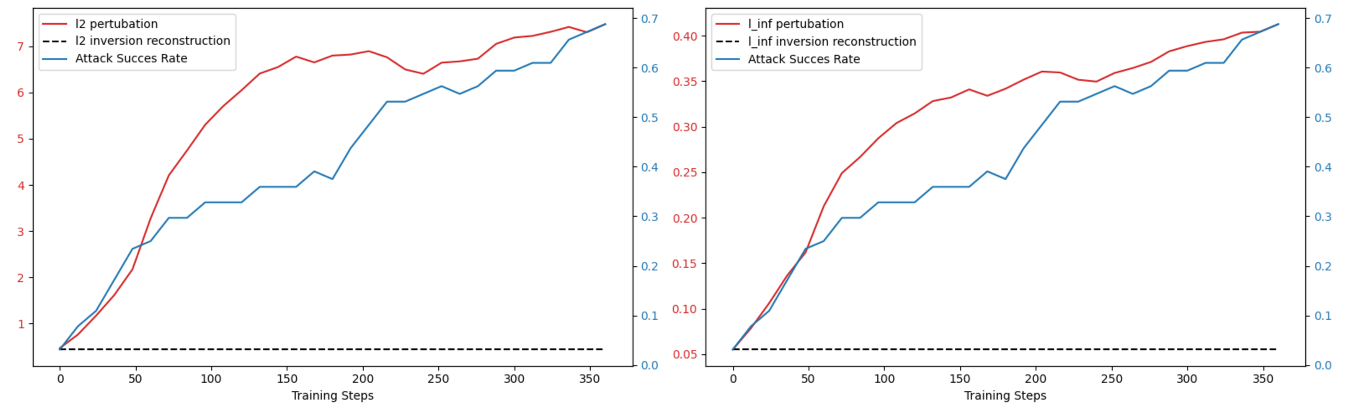


Figure 3: The success rate and perturbation magnitudes of proposed attack by the number of training steps. Results are averaged over a subset of 1024 unseen CIFAR-10 images.

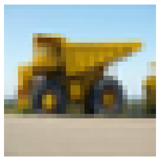
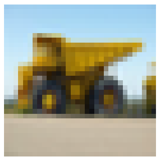
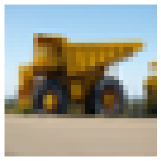
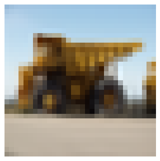
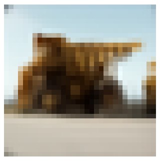
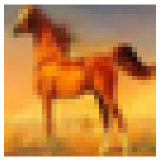
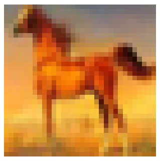
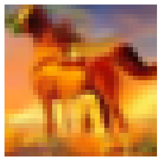
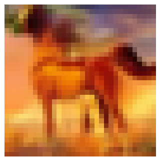
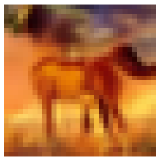
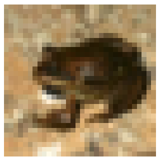
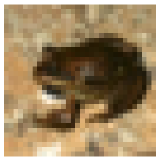
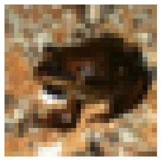
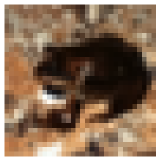
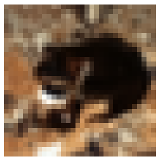
	Original	Inversion Reconstruction	50 Steps	100 Steps	200 Steps
					
ℓ_2 Error	0	0.31	1.76	2.96	6.05
Tractor	0.81	0.80	0.80	0.79	0.04
Airplane	0.03	0.03	0.03	0.03	0.65
					
ℓ_2 Error	0	0.32	3.29	3.56	5.86
Horse	0.85	0.85	0.83	0.78	0.06
Deer	0.03	0.03	0.03	0.07	0.77
					
ℓ_2 Error	0	0.32	3.04	3.58	6.05
Frog	0.84	0.84	0.84	0.79	0.03
Cat	0.02	0.02	0.02	0.04	0.78

Table 1: Qualitative examples of adversarial examples produced by our method throughout the fine-tuning process. For each example, the top label row denotes the true label probability prediction of the targeted classifier and the bottom label row denotes the adversarial label prediction.

Secondly, we believe that the typical adversarial perturbations have high-entropy, whereas diffusion models (and other types of image generation model) have an inductive bias towards generation of low-entropy images.

Due to this two reasons, our model is unable to learn to generate typical adversarial perturbations (at least without pretraining) and instead, converges to generating adversarial images from the "natural" distribution.

5.2 Limitations of "Natural" Perturbations

In this work we coined the term adversarial images from the "natural" distribution, what we mean by that are images that are sampled from the model's original distribution, that are similar to the image created by the same latent noise, but that belong to a different label. We hypothesize that the performance potential (i.e the ability to decrease the classifier's accuracy, while remaining similar to the original images) of such adversarial samples is limited. Figure 7 aims to depict the reason we believe that.

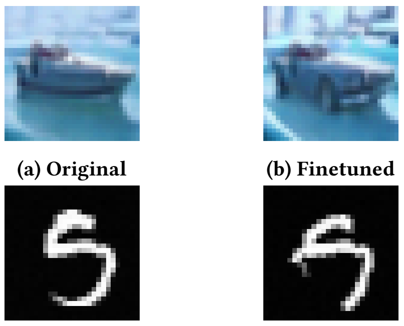
Something that we conjecture can improve the performance of this kind of perturbation is to attack a classifier with a big label set. If there are many labels the potential to have a close image in pixel-space that belongs to a different label is higher (e.g. map a horse to a zebra instead of a horse to cat). Experimenting with our approach on cifar-100 instead of cifar-10 may be an interesting next step.

5.3 Overoptimisation

In their work [2] kevin et al. report the overoptimisation problem - if trained for long enough, the model completely diverges from its original stage and may output images that no longer have semantic meaning. We notice similar problem in this work, given enough time the model starts to output images that are completely unrelated to the image that the original model outputs for the same noise. We try several methods to remedy this, such as the hinge reward, and with the two regularization methods we implemented ($L2$ loss and reward penalty), with limited success. It may be an interesting future work to experiment with our regularization methods with stable diffusion model that is used in [2] and with the data used in there. See figure 6 for an example. Interestingly, we don't notice the issue in [2] where the output images lose semantic meaning, that may be because losing semantic meaning produces lower reward than outputting a semantically sound image, but of a different label.

6 CONCLUSION

In this work we've attempted to train a diffusion model to generate adversarial samples using proximal policy optimisation. While our training process is inefficient at generating typical adversarial perturbations, it is, interestingly, able to generate adversarial perturbations from the original distribution. We conjecture that this inability to generate typical adversarial samples stems from the limited ability of the RL training to explore such samples to begin with; following this conjecture we attempt to remedy it with pretraining with limited success. This is not the main part of our work, but during the pretraining experiments we find that it is relatively easy for the diffusion model to learn to generate adversarial samples that were generated with vanilla algorithms such as [6] using a vanilla diffusion training process which is interesting in itself and may be interesting future work.



(a) Original **(b) Finetuned**

(c) Original **(d) Finetuned**

Figure 4: "Natural" perturbations generated by our model. Top Row: Example of a CIFAR-10 natural perturbation. The model maps a boat to a similar automobile. The attack is (perhaps unsurprisingly) highly transferable - it is able to trick 3 ouf of 3 different cifar classifiers that we tested. It has L_{inf} norm of 0.28 and ℓ_2 norm of 3.28. Bottom Row Natural perturbation on the MNIST dataset.

REFERENCES

[1] Shumeet Baluja and Ian Fischer. 2017. Adversarial Transformation Networks: Learning to Generate Adversarial Examples. (2017). arXiv:cs.NE/1703.09387

[2] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. 2023. Training Diffusion Models with Reinforcement Learning. (2023). arXiv:cs.LG/2305.13301

[3] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. (2017). arXiv:cs.CR/1608.04644

[4] Jianqi Chen, Hao Chen, Keyan Chen, Yilan Zhang, Zhengxia Zou, and Zhenwei Shi. 2023. Diffusion Models for Imperceptible and Transferable Adversarial Attack. (2023). arXiv:cs.CV/2305.08192

[5] Prafulla Dhariwal and Alex Nichol. 2021. Diffusion Models Beat GANs on Image Synthesis. (2021). arXiv:cs.LG/2105.05233

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. (2015). arXiv:stat.ML/1412.6572

[7] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. 2022. Prompt-to-Prompt Image Editing with Cross Attention Control. (2022). arXiv:cs.CV/2208.01626

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. (2020). arXiv:cs.LG/2006.11239

[9] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into Transferable Adversarial Examples and Black-box Attacks. (2017). arXiv:cs.LG/1611.02770

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. (2019). arXiv:stat.ML/1706.06083

[11] Ron Mokady, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. 2022. Null-text Inversion for Editing Real Images using Guided Diffusion Models. (2022). arXiv:cs.CV/2211.09794

[12] Haonan Qiu, Chaowei Xiao, Lei Yang, Xinchen Yan, Honglak Lee, and Bo Li. 2020. SemanticAdv: Generating Adversarial Examples via Attribute-conditional Image Editing. (2020). arXiv:cs.LG/1906.07927

[13] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-Shot Text-to-Image Generation. (2021). arXiv:cs.CV/2102.12092

[14] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. (2022). arXiv:cs.CV/2112.10752

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. (2017). arXiv:cs.LG/1707.06347

[16] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial Training for Free! (2019). arXiv:cs.LG/1904.12843

[17] Jiaming Song, Chenlin Meng, and Stefano Ermon. 2020. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*.

[18] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. Score-Based Generative Modeling through Stochastic Differential Equations. (2021). arXiv:cs.LG/2011.13456

[19] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. (2014). arXiv:cs.CV/1312.6199

[20] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. 2019. Generating Adversarial Examples with Adversarial Networks. (2019). arXiv:cs.CR/1801.02610

[21] Haotian Xue, Alexandre Araujo, Bin Hu, and Yongxin Chen. 2023. Diffusion-Based Adversarial Sample Generation for Improved Stealthiness and Controllability. (2023). arXiv:cs.CV/2305.16494

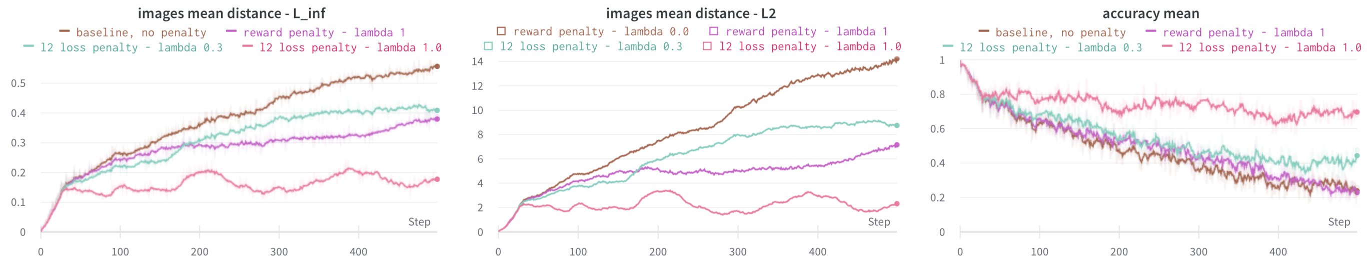
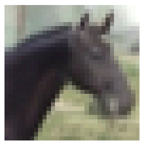
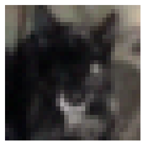


Figure 5: We show the effect of our 2 regularization methods on the model. We show 4 different runs: 1. no regularization, 2. penalizing the reward with L_∞ norm, 3. applying $L2$ loss with lambda = 0.3, 4. applying $L2$ loss with lambda = 1.0. We can see that our regularization methods are effective in decreasing the distance norm between the fine-tuned and original model. Unfortunately, that comes with a cost of less effective attack in terms of the decrease of the mean accuracy of the attacked model.



(a) Original model prediction



(b) Fine-tuned model prediction

Figure 6: An example of "overoptimisation". Given enough training iterations and no regularization our model starts to output completely different images given the same latent noise.

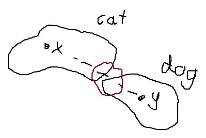


Figure 7: There are some images that may be in proximity to another label in the natural distribution. Yet, many others aren't, limiting the performance of what we termed as "natural adversarial perturbations"

A APPENDIX

A.1 PPO

Figure 9 shows a demonstration of the PPO algorithm.

A.2 L_2 loss and PPO equivalency

We now show the deep similarity between the $L_{REG}(\theta)$ loss that we described in the method section and PPO. In this work we use the clip version of PPO, but note that there exists a penalty PPO version which sums L_{RL} with the KL divergence between the probabilities of the original model's policy and the fine-tune model's policy (for this version of PPO the original model needs to be kept, similarly to our training process). We're going to show a connection between the L_2 loss and KL divergence, the KL divergence is in turn, known to be equivalent to the PPO clip version.

Let us remember that the policy is a normal distribution:

$$\pi(a_t, s_t) = p_\theta(x_{t-1}|t, x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t * I)$$

for normal distribution it holds that

$$KL(\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2)) =$$
$$= \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \operatorname{tr}\left\{\Sigma_2^{-1}\Sigma_1\right\} + (\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1)\right]$$

Note that for us $\Sigma_1 = \Sigma_2 = \sigma_t$ since both the original and fine-tuned model use the same scheduler. Hence we get that

$$KL(\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2)) = \frac{1}{\sigma_t^2}||\mu_1 - \mu_2||_2^2$$

Now let us for simplicity sake assume that we use the DDPM scheduler (in practice we use DDIM, but the same computation holds), then

$$\mu_{1/2} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\theta/\phi}(\mathbf{x}_t, t)\right)$$

Hence,

$$KL(\mathcal{N}_\theta, \mathcal{N}_\phi) = \frac{1}{\sigma_t^2}||\epsilon_\theta(x_t, t) - \epsilon_\phi(x_t, t)||_2^2$$
$$= \frac{1}{\sigma_t^2}L_{REG}(\theta)$$

to conclude, the L_2 regularization loss and KL divergence between two denoising diffusion process are different only by a normalization factor that depends on σ and the time step. Since η_t gets smaller as the denoising process approaches its end, this means that the KL is larger than the L_2 loss at the beginning of the process. We leave experimenting with the KL loss rather than L_2 loss as a future work.

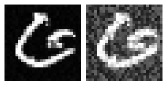


Figure 8: The right image is the output of a model that went through pretraining and RL training. The left image is the original image from the MNIST dataset.

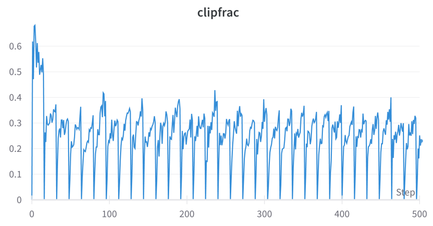


Figure 9: This figure shows the portion of samples in the batch where $R = \frac{p_\theta(x_{t-1}|t,x_t)}{p_{\theta_{old}}(x_{t-1}|t,x_t)}$ does not hold that $1 - \epsilon < R < 1 + \epsilon$. This with 8 inner training epochs. We can see a cyclicality - every time we start a new cycle of 8 inner training epochs the model θ gradually diverges from θ_{old} until a new cycle starts with $\theta = \theta_{old}$.

Config	**Value**
learning rate	0.0001
sample batch size	256
train batch size	128
optimizer	Adam
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
DDIM steps	150
seed	42
reward penalty norm	ℓ_∞
reward penalty λ	1.0
reward penalty threshold	0.05
reward type	linear
ℓ_2 loss λ	0.0
PPO clip range	0.00001

Table 2: Our DDPO Training for Adversarial Examples config