

XII Iberian Modeling Week (IMW) 2026
NOVA School of Science and Technology, Caparica Campus, Lisbon

15–19 June 2026

**Project P3: The Fragility of Learning
Adversarial Attacks on Neural Networks**

*One fixed, sparse perturbation that makes the model
read a chosen target digit on every MNIST image*

Pedro Rodríguez David Castilla Adrián de la Fuente Frederik Winter

Coordinator: Rohollah (Nima) Garmanjani

Abstract

Deep image classifiers reach high accuracy on clean data and can still be fooled by small, deliberately chosen changes to their input. We study one form of this: a *single, fixed* perturbation δ that, added in the same way to any MNIST image, makes a frozen classifier (about 98.9% accurate) read a chosen *target* digit t , while changing only a small fraction of the 784 pixels (around $K \approx 50$). The perturbation has to be *universal* (one δ for every image), *targeted* (a specific class) and *sparse* (a small L_0 budget) at the same time. We write the attack as the minimisation of a Carlini–Wagner margin loss on the logits, averaged over a mini-batch, under a hard $\|\delta\|_0 \leq K$ constraint, and we study two design choices: how to keep the attacked image in $[0, 1]$ (**clamp** vs. a **tanh** change of variables) and how to enforce the budget (one-phase iterative hard-thresholding vs. a **two-phase** relax, select and refine scheme). We do not target a fixed success rate; we report how effective the attack can be made and which choices matter. With $t = 5$ and about 50 pixels, the two-phase clamp attack reaches about 58% target rate at about 47% accuracy and crosses 60% at $K \approx 51$. The clamp change of variables is more pixel-efficient than tanh (which needs about $1.6\times$ the budget) and less sensitive to the mini-batch size. Both effects follow from where each variant’s gradient lives: tanh saturates on the black background and white strokes that make up almost all of MNIST, so its useful gradient sits on only about 11% of the pixels.

Contents

1	Introduction	3
2	Mathematical setup	3
2.1	Data and target model	3
2.2	The objective	3
2.3	The change of variable: clamp vs. tanh	4
3	Method	5
3.1	Normalised gradient step	5
3.2	Enforcing the budget: one-phase vs. two-phase	5
3.3	Early stopping	6
4	Experimental setup	7
5	Results	7
5.1	Two-phase vs. one-phase	7
5.2	clamp vs. tanh	9
5.3	Where each variant acts	9
5.4	Mini-batch size	10
5.5	The imperceptibility weight λ	10
6	Discussion	12
7	Future work	12
	Reproducibility	13

1 Introduction

Modern neural networks classify images accurately, yet a small, carefully chosen change to an input can flip the prediction while staying hard to see [1]. Such *adversarial examples* are not rare: they can be found by following the gradient of the model’s loss [2]. They matter because they suggest a classifier may rely on a fragile decision boundary rather than the concept we intended, which is a concern for vision systems used in safety-critical settings.

Attacks are usually described along three axes. By *knowledge*, they are white-box (full access to the weights and gradients) or black-box (only queries); ours is white-box. By *goal*, they are untargeted (any wrong class) or targeted (a chosen class); ours is targeted, at a digit t . By *norm*, the perturbation is bounded in L_∞ , where every pixel moves a little (FGSM [2], PGD [7]), L_2 (Carlini–Wagner [6]), or L_0 , where few pixels move but each may move a lot (JSMA [3]); ours is L_0 . Two further properties matter here: the attack is *universal* [5] (one perturbation for all inputs, not re-crafted per image), and the minimal-perturbation view [4] asks for the smallest change that crosses the boundary.

We work on MNIST [8]: 60,000 grayscale images of handwritten digits, each 28×28 with pixel values in $[0, 1]$ (0 black, 1 white). The model we attack is a small convolutional network trained on MNIST to about 98.9% accuracy and then *frozen*: we never change the model, only the perturbation. On clean data it already predicts the target digit t on about 9% of images (chance level for ten classes), so any increase in that rate is caused by the attack.

The task is to find one additive $\delta \in \mathbb{R}^{28 \times 28}$ such that the attacked image $\tilde{x} = \text{clamp}(x + \delta, 0, 1)$ is read as t on as many images as possible, using few pixels.

2 Mathematical setup

2.1 Data and target model

The target model is the frozen convolutional network of Section 1. Its clean accuracy is 98.88% on the training split and 98.82% on the test split. We build the perturbation on a subset of the training images and evaluate it on all 60,000.

2.2 The objective

We look for a single additive δ , the same for every image. Writing the attacked image as $\tilde{x}_i = \text{clamp}(x_i + \delta, 0, 1)$, we minimise the *Carlini–Wagner* margin loss [6] on the *logits* Z , averaged over a mini-batch B , under the sparsity constraint:

$$\min_{\delta} \frac{1}{|B|} \sum_{i \in B} \max \left(\max_{j \neq t} Z_j(\tilde{x}_i) - Z_t(\tilde{x}_i), 0 \right) + \lambda \|\delta\|_2^2 \quad \text{s.t.} \quad \|\delta\|_0 \leq K. \quad (1)$$

The margin $\max_{j \neq t} Z_j - Z_t$ measures how far the best competing class is ahead of the target; it is positive while another class wins and negative once t wins, so minimising it drives the image toward t . We read it off the raw *logits*, not the softmax probabilities, because on a confident network the softmax gradient is ≈ 0 and gives the optimiser almost no signal, while the logit margin keeps a usable slope. The outer $\max(\cdot, 0)$ is a hinge at zero: once an image already predicts t its contribution and its gradient vanish, so the optimiser spends effort only on the images not yet fooled, which is what lets a single δ work across the dataset. The term $\lambda \|\delta\|_2^2$ is an optional penalty on the size of the change (studied in Section 5.5; the main results use $\lambda = 0$). Finally, $\|\delta\|_0 \leq K$ is the sparsity budget; since the L_0 “norm” is non-differentiable we enforce it by projection (keeping the K pixels of largest $|\delta|$), not as a soft penalty.

2.3 The change of variable: clamp vs. tanh

Whatever δ we add, the result must be a valid image with pixels in $[0, 1]$. We compare two ways to enforce that, and the choice shapes the attack as much as the budget does.

Clamp adds δ and clips: $\tilde{x} = \text{clamp}(x + \delta, 0, 1)$. It is direct and can move any pixel, but its gradient is exactly zero once a pixel is clipped, so a saturated pixel gives no local signal.

Tanh is a change of variables. It maps the clean image to an unbounded space, $a = \text{atanh}(2x - 1)$, adds an unbounded perturbation, and maps back: $\tilde{x} = \frac{1}{2}(\tanh(a + \delta) + 1)$. The result is always in $(0, 1)$ with no clipping, and δ always receives a gradient, so no coordinate is permanently stuck. The cost is tanh’s own saturation.

The local sensitivity of the tanh map is $\partial\tilde{x}/\partial\delta = \frac{1}{2}(1 - \tanh^2(a + \delta))$. It is largest at mid-grey ($x \approx 0.5$, $a = 0$) and falls to ≈ 0 at black ($x \approx 0$, $a \approx -6.9$) and at white ($x \approx 1$, $a \approx +6.9$): a fixed δ barely moves a black or white pixel and moves a grey one the most (Figure 1). Clamp, by contrast, has unit sensitivity across all of $[0, 1]$.

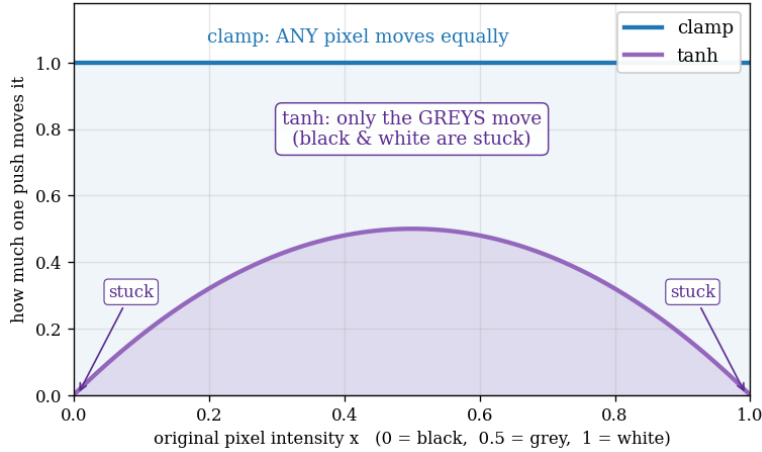


Figure 1: How far one push moves a pixel, as a function of its original intensity x (0 black, 0.5 grey, 1 white). **clamp** (flat line) moves every pixel by the same amount; **tanh** (hump, $\propto x(1 - x)$) moves only the mid-greys and is “stuck” near black and white, where its sensitivity $\frac{1}{2}(1 - \tanh^2)$ vanishes.

This matters because of how MNIST is distributed. About 82.6% of pixels are black background, only about 8.7% mid-grey and about 8.8% near-white, so tanh’s usable gradient, where the sensitivity is not negligible, lives on only about 11% of the pixels.

Which pixels? Here the universal, batch-averaged nature of δ is the key. A single δ is added to every image, and a given coordinate takes different intensities across the dataset: on some images it lies on ink ($x \approx 1$), on others on background ($x \approx 0$), and on the anti-aliased edges of strokes it is genuinely grey ($x \approx 0.5$). The gradient δ receives at a coordinate is summed over the batch, so its usable leverage is driven by *how often that coordinate is grey across images*. In MNIST the grey pixels are the stroke edges, which sit on the ink; the pure background is black on almost every image and therefore has near-zero leverage everywhere. Tanh can thus retouch, that is darken and erode, the strokes, but it cannot cheaply paint new ink onto the empty background. Clamp has no such restriction: it lightens background and darkens ink with equal ease.

We can already predict the experiments. Clamp should be *more pixel-efficient*, because it can both add and remove ink, whereas tanh is largely limited to eroding existing strokes. For the same reason tanh’s edits should be *less visible* (it only touches the strokes). And tanh should be *more sensitive to the mini-batch size*, because its usable signal is confined to a small fraction of pixels and needs more images per step to be estimated well. Section 5 tests these three predictions.

3 Method

3.1 Normalised gradient step

We update $\delta \leftarrow \delta - \alpha_0 g / (\|g\| + \varepsilon)$ with $\alpha_0 = 0.1$, where g is the mini-batch gradient. Dividing by the norm rescales the update *as a whole*, not coordinate by coordinate: within a single step the pixels still move by different amounts, in proportion to their gradient, so only the *direction* of g is used and no pixel’s move is made equal to another’s. What the normalisation equalises is the *total length* of the step from one iteration to the next. Different mini-batches produce gradients of very different magnitude, and without normalisation the steps would lurch when the gradient happened to be large and crawl when it was small; dividing by $\|g\|$ makes successive steps comparable in size instead. This matters in particular when the clamp flattens part of the gradient, since a clipped pixel contributes nothing locally and would otherwise shrink $\|g\|$ from one batch to the next.

3.2 Enforcing the budget: one-phase vs. two-phase

The L_0 constraint is enforced by projection: keep the K pixels of largest $|\delta|$ and zero the rest. There are two ways to schedule this projection.

One-phase iterative hard-thresholding (IHT) projects after every gradient step. In practice this tends to lock the support (which K pixels are non-zero) very early, while $\delta \approx 0$ and the gradient is at its *noisiest*, so the K pixels are chosen when the optimiser has almost no reliable information about which ones matter. The support is not frozen by construction: a pixel outside the current top- K can still break in if a later step makes its accumulated value overtake one of the kept pixels. But because each normalised step is small next to the values already built up, this is rare, and after the first few iterations the support set barely changes. Re-projecting every step therefore keeps nudging that early, noise-picked support around without ever letting it settle on a good one.

The two-phase scheme separates the discrete decision (which pixels) from the continuous one (what values), in three steps. A *dense* phase optimises the full δ with all 784 pixels free. A *selection* step keeps the top- K pixels by $|\delta|$ and freezes the rest to zero. A *refine* phase re-optimises only those K pixels, with the gradient masked so the frozen ones stay at zero. The dense phase is a *search* for a good support, not the deployed attack: the final δ still has exactly K non-zero pixels, so the L_0 constraint is met exactly. Separating support selection from value fitting is the standard treatment of the NP-hard L_0 problem, as in the LASSO [9] followed by debiasing, or in forward and best-subset selection. In short, the dense phase lets the data reveal which coordinates carry the attack before the budget is imposed, which is exactly the information a one-phase method lacks when it fixes the support under noise (Figure 2).

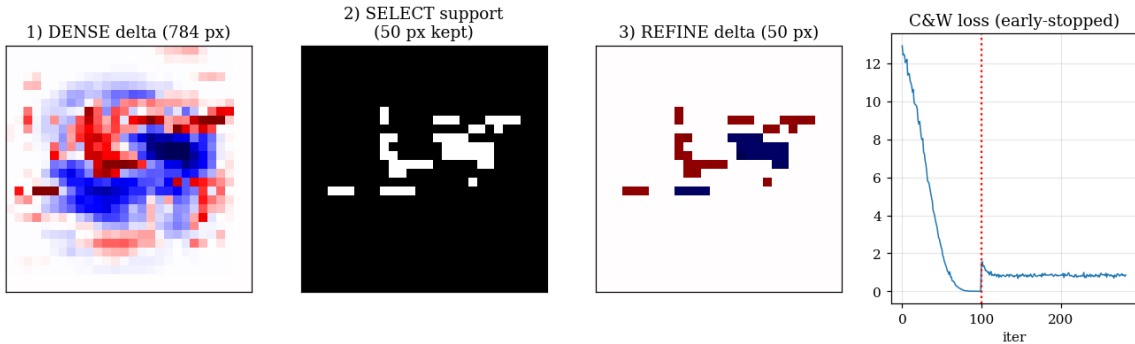


Figure 2: The two phases of the attack: the dense δ (left) is optimised over all 784 pixels; the top- K are selected (middle); and the refined δ (right) re-optimises only those K under the budget.

Assumption behind the two-phase scheme. The scheme rests on an assumption: that the support of the dense optimum (over 784 pixels) contains a good support for the K -pixel problem, that is, that the K -pixel optimum lies near the dense one. Selection keeps the coordinates where the dense δ is large and refines there; if the best K -pixel solution lived in a different region of coordinate space, this selection would pick the wrong support and the method would fail. It works here because, empirically, the two optima are close (Section 5); it is not guaranteed in general. Geometrically (Figure 3), the dense phase optimises a smoother surface over all pixels and finds a good region; the refine phase is then warm-started there, instead of committing to a support under noise as one-phase does.

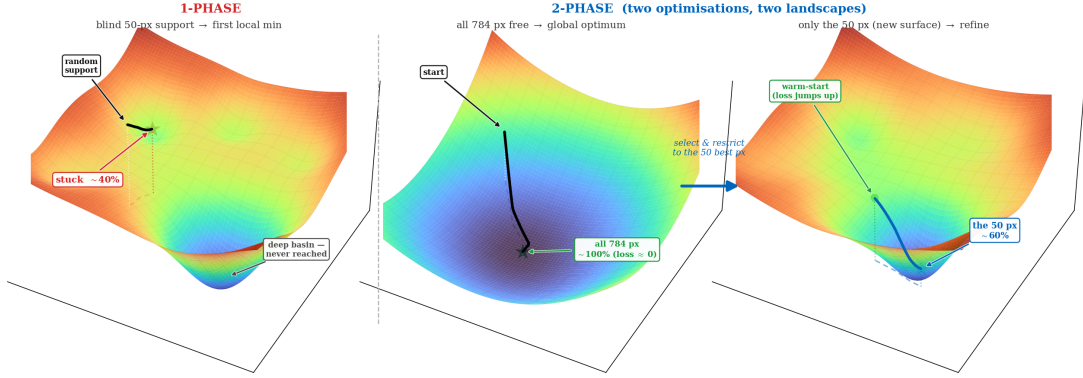


Figure 3: Left (1-phase): fixing the support at step 1 lands in the first local minimum of a bumpy surface over “which K pixels” to keep. Right (2-phase): the dense phase optimises a smoother, all-784-pixel surface and finds a good region, and the K -pixel refine is warm-started from that solution.

3.3 Early stopping

For single runs and for the figures we stop when the loss has converged, which differs between phases: the dense phase drives the loss close to zero, while the refine phase settles on a noisy plateau above zero. We smooth the mini-batch loss with an *exponential moving average* (EMA) and stop when its improvement over a window falls below $\max(\text{noise}, 0.5\% \cdot \text{peak})$, where the peak is the phase’s own loss scale so there is no absolute constant. On the refine plateau the noise term dominates: we stop when the trend over the window is within the mini-batch fluctuation, by a one-sided t -test. On the dense phase the relative-floor term catches the clean convergence to zero, which a pure signal-versus-noise test would miss. The benchmark below does not use this rule (Section 4); Figure 4 shows the smoothed curve and where it stops.

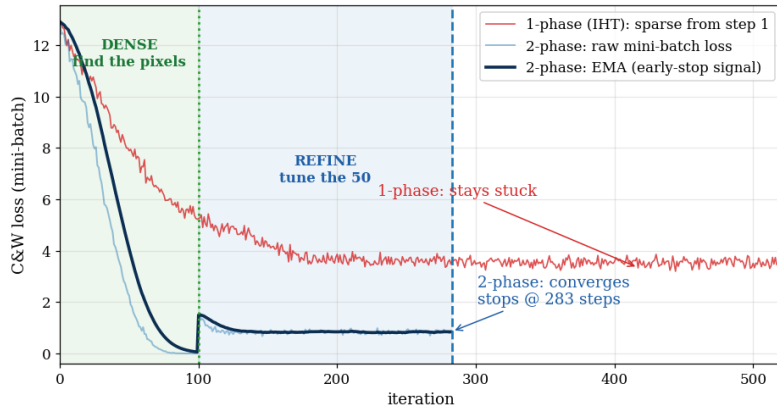


Figure 4: Loss curves. One-phase IHT (red) stays on a high plateau; the two-phase method (blue) drives the dense loss to zero, then plateaus during refine, and its EMA (dark line) early-stops after about 280 total steps.

4 Experimental setup

Every run is driven by a pseudo-random generator with an explicit integer seed, so two runs with the same seed see the same mini-batch at every iteration and methods are compared *paired*: any difference comes from the method, not from the batch. The early-stopping rule of Section 3.3 is implemented, but the benchmark deliberately disables it and uses a *fixed* budget (2500 dense + 2500 refine steps) so that every method sees exactly the same batches; early stopping is used only for single runs and for the figures. The dense phase is run once per (variant, seed) and its batches are replayed for every value of K .

We compare the two variants (`clamp` and `tanh`) and the two methods (one-phase IHT and two-phase), sweeping the budget over $K \in \{5, 10, \dots, 50, 60, \dots, 250\}$ and averaging over seeds $\{0, 1, 2\}$; the weight λ is swept separately at $K = 50$. The step size is $\alpha_0 = 0.1$ and the mini-batch is $|B| = 1000$ except in the batch-size study; the target is $t = 5$ unless stated. All curves are means over the three seeds.

5 Results

5.1 Two-phase vs. one-phase

Removing the budget entirely (all 784 pixels free) reaches about 100% target rate, which shows the loss, the optimiser and the clamp all work; the only difficulty is the sparsity. Table 1 compares the options at $K = 50$: one-phase IHT reaches about 40%, projecting the dense δ to K without refining about 47%, and the two-phase scheme about 58% at about 47% clean accuracy. The two-phase scheme wins because it chooses the support with information rather than under noise and then re-optimises the kept pixels; plain projection, without the refine, leaves about eleven points on the table.

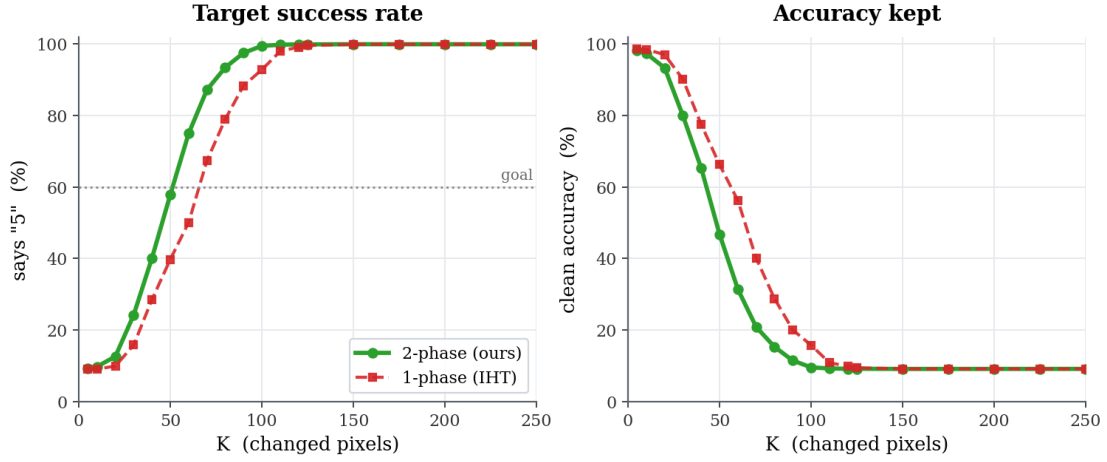
Table 1: Diagnosis at $K = 50$ (target $t = 5$, paired means over 3 seeds). Removing the budget recovers $\sim 100\%$, so the constraint, not the optimiser, is the bottleneck; and refining beats merely projecting.

Attack	target rate	accuracy
no pixel budget (all 784)	$\sim 100\%$	$\sim 9\%$
dense, projected to K (no refine)	$\sim 47\%$	$\sim 58\%$
one-phase IHT (threshold each step)	$\sim 40\%$	$\sim 66\%$
two-phase (dense, select, refine)	$\sim 58\%$	$\sim 47\%$

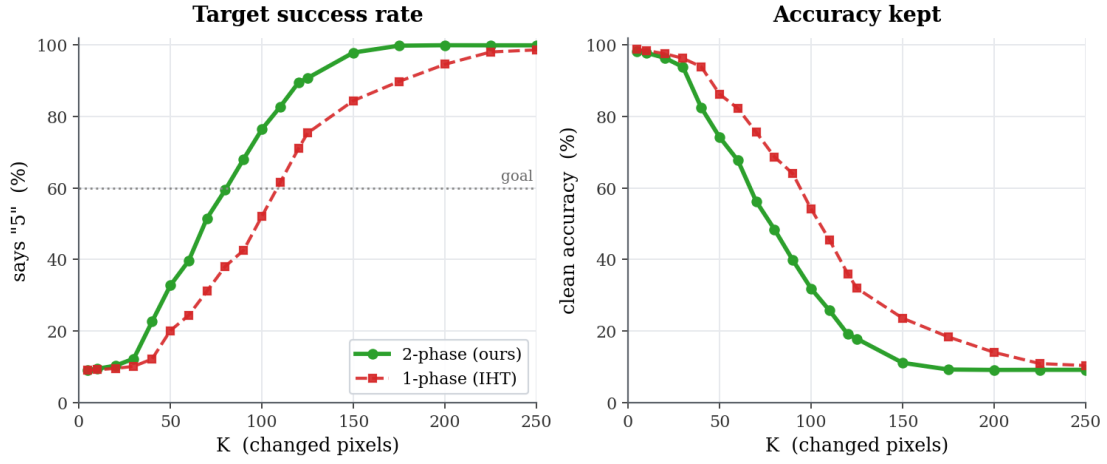
Figure 5 sweeps K for both methods and both variants: two-phase is above one-phase at every budget. For clamp two-phase the target rate climbs from 24% at $K = 30$ to 40% at $K = 40$, 58% at $K = 50$ and 75% at $K = 60$, crossing 60% at $K \approx 51$.¹ Accuracy is the mirror image, falling below 50% around the same budget (about 47% at $K = 50$). The attack is not specific to the digit 5: t is a free choice, and the achievable rate depends on how reachable that digit is from the data, with single runs giving, for example, about 76% for $t = 3$ and 86% for $t = 8$.

Figure 6 shows the same perturbation ($K = 50$) added to a range of digits, with a green border on the images now read as “5”. The same δ is applied to every image; at this budget it does not catch every one.

¹The 58% is the paired mean over 3 seeds at fixed budget; single runs with early stopping, or held-out evaluation, sit a few points higher (about 60–66% for $t = 5$). We use the paired mean as the reference throughout.



(a) **clamp**: target rate (left) and clean accuracy (right) vs. K .



(b) **tanh**: the same two panels.

Figure 5: Budget sweep per variant (mean over 3 seeds): two-phase (green) vs. one-phase IHT (red), for **clamp** (top) and **tanh** (bottom). Two-phase is above one-phase at every budget for both variants.

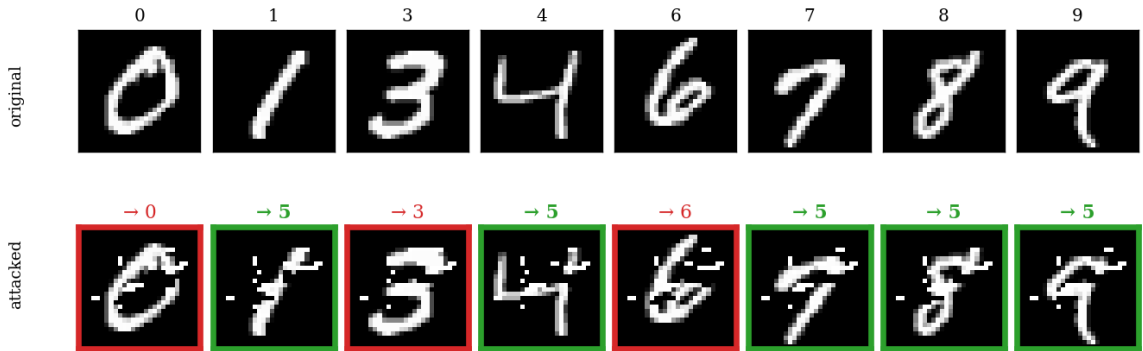


Figure 6: One fixed perturbation ($K = 50$, two-phase clamp, $t = 5$) added to many different digits. Green border: the network now predicts "5"; red: not yet.

5.2 clamp vs. tanh

The two change-of-variables behave as Section 2.3 predicted. Figure 7 plots both two-phase curves: clamp crosses 60% at $K \approx 51$ while tanh needs $K \approx 81$, about $1.6\times$ as many pixels, and the ratio stays near $1.6\times$ across the range (Table 2). At $K = 50$, clamp two-phase reaches about 58% against about 33% for tanh. So clamp is the more pixel-efficient of the two, which is the first of the three predictions.

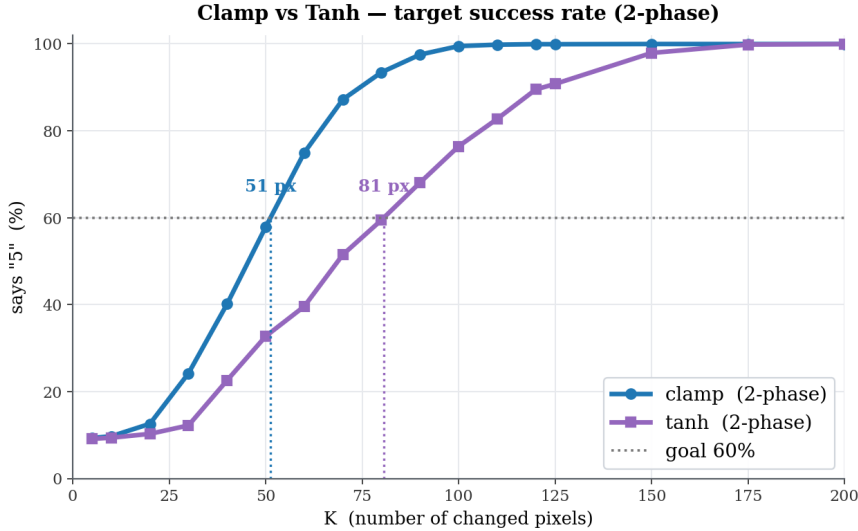


Figure 7: Two-phase target rate vs. budget K , **clamp** (blue) vs. **tanh** (purple). **clamp** crosses 60% at $K \approx 51$; **tanh** needs $K \approx 81$, about $1.6\times$ as many, because it cannot cheaply move the black background.

Table 2: Target rate (fraction predicting $t = 5$), paired mean over 3 seeds. Two-phase beats one-phase for both variants, and **tanh** needs about $1.6\times$ the budget.

	clamp	tanh
one-phase (IHT), $K = 50$	40%	20%
two-phase, $K = 50$	58%	33%
two-phase, $K = 100$	99%	76%
two-phase reaches 60% at $K \approx$	51	81

5.3 Where each variant acts

Figure 8 shows where the 50 pixels of each attack land, confirming the second prediction. Clamp uses about 25 pixels that lighten and 25 that darken, but this even split is only in the *sign* of the change, not in *where* it acts: sorted by the base intensity of the pixel it lands on, clamp is far from balanced and clearly prefers the black background. The bottom histogram of Figure 8 makes this concrete, with clamp’s changes piling up at $x \approx 0$ (about 47% of them fall on pure-black pixels) against a much smaller cluster on the white ink. The lightening pixels paint the empty background, adding the white blobs of a “5”, while the darkening pixels erode the existing stroke; the two roles are complementary and neither alone is enough. Tanh is lopsided in both senses, about 4 pixels that lighten and 46 that darken, almost all landing on the white ink, which is exactly the erosion-only behaviour predicted from its saturation. The difference is visible to the eye: clamp leaves marks on the background, while tanh only retouches the strokes and stays closer to imperceptible.

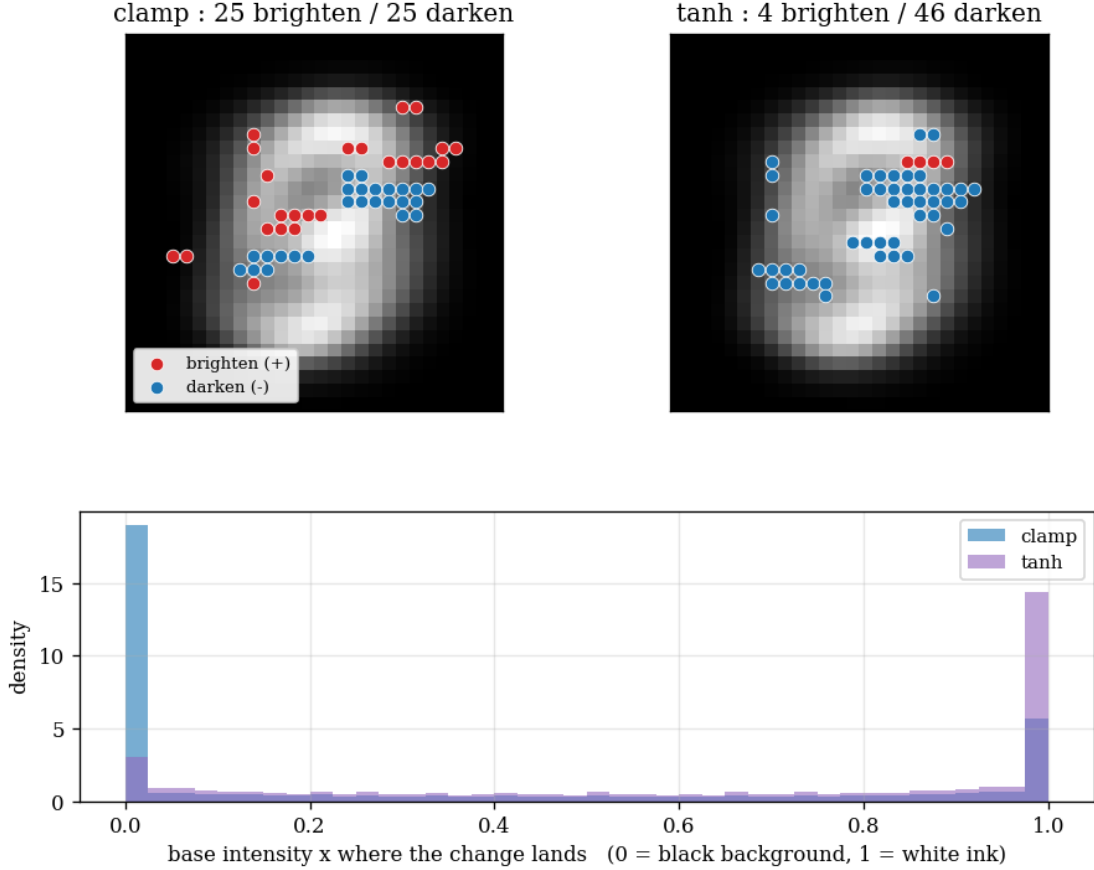


Figure 8: Where the 50 pixels act ($K = 50$, $t = 5$). *Top:* the perturbed pixels over the average “5”. clamp (left) is an even 25/25 split of lighten (red) and darken (blue), covering background and stroke; tanh (right) is 4/46, almost all darkening and confined to the stroke. *Bottom:* histogram of the base intensity the changes land on; clamp peaks on the black background ($x \approx 0$), tanh on the white ink ($x \approx 1$).

5.4 Mini-batch size

The third prediction concerns the batch size. We shrink the batch from $|B| = 1000$ down to 5 (Figure 9). Clamp is essentially immune, converging in about 310 iterations at every batch size, because its gradient is clean and large wherever it is active and the normalised step estimates the descent direction well even from few samples. Tanh degrades sharply, needing $6\times$ to $31\times$ more iterations as the batch shrinks. The reason is that the fraction of pixels on which tanh has usable gradient is *not* a fixed 11%: a coordinate is only movable when *some* image in the mini-batch happens to show it at a grey, unsaturated value, so the set of usable pixels is the union of the grey pixels present *in that batch*. A large batch makes almost every stroke-edge pixel grey in at least one of its images, and the usable fraction approaches its ceiling of about 11%; a small batch samples so few images that most of those pixels are pure black or pure white throughout it, so the usable fraction shrinks well below 11% and the optimiser is left almost blind. It is this collapse of the usable set, not a fixed 11%, that drives the degradation. Clamp has no such dependence, since its gradient is available on every pixel regardless of the batch.

5.5 The imperceptibility weight λ

The $\lambda \|\delta\|_2^2$ term penalises the size of the change, trading target rate for stealth; we measure it in pixel space (the realised change $\tilde{x} - x$) so it is comparable across variants, and sweep it at $K = 50$ (Figure 10). Any $\lambda > 0$ drops clamp two-phase from about 58% into the 13–20% band, so the trade-off is unfavourable.

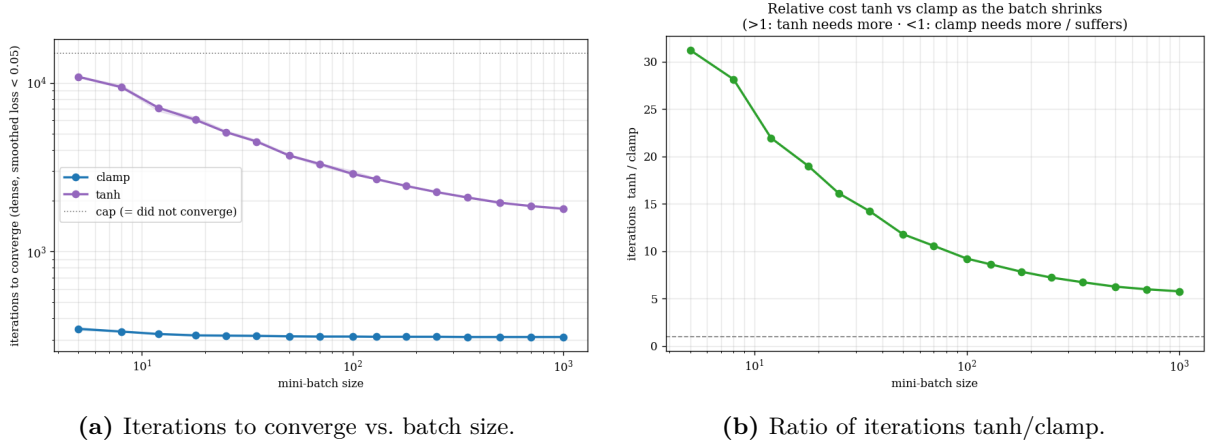


Figure 9: Batch-size study. clamp converges in ~ 310 iterations regardless of batch size; tanh needs $6\times$ to $31\times$ more iterations as the batch shrinks.

Only the **clamp two-phase** curve is clearly non-monotone. Its target rate falls to a minimum near $\lambda = 0.05$ (about 13%) and then *recovers* to about 20% at $\lambda = 0.2$ – 0.5 ; the realised pixel-norm follows the same dip-and-recover shape, so at $\lambda = 0.2$ the clamp two-phase perturbation is both larger and more effective than at $\lambda = 0.05$. The other three curves are, to the eye, roughly monotone: clamp one-phase decays steadily across the sweep, tanh two-phase decays and then flattens, and tanh one-phase drifts around a low plateau before dropping at the largest λ ; none of them shows the clamp two-phase rebound. That rebound is a sign that the penalised objective is non-convex: under the normalised step a moderate λ steers the two-phase clamp optimiser into a poor basin where the ℓ_2 penalty dominates the descent direction, whereas a larger λ tilts that direction enough to settle in a different, slightly better one. The effect is specific to this optimiser and does not change the practical conclusion, so the main results use $\lambda = 0$.

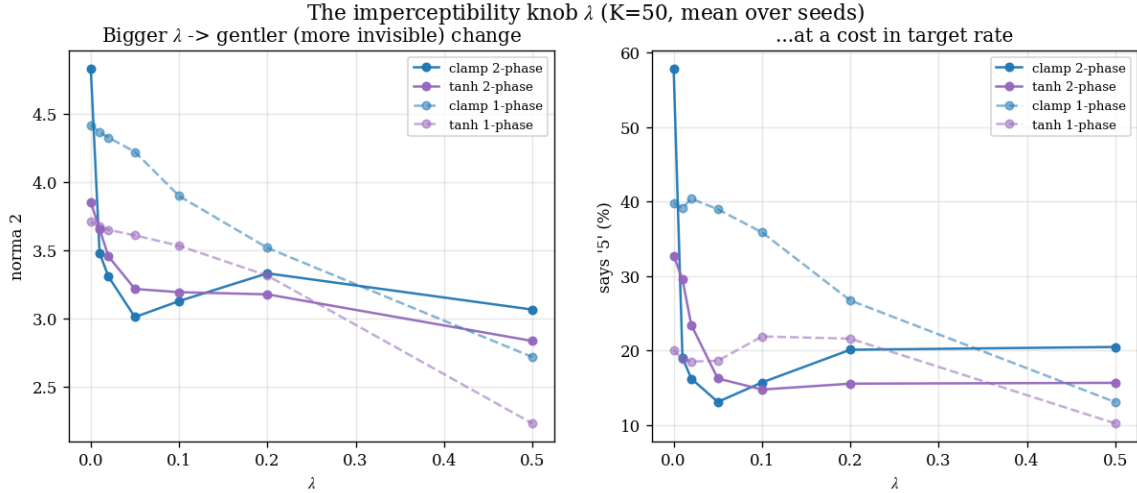


Figure 10: The weight λ ($K = 50$, mean over seeds). *Left:* a larger λ gives a smaller change in pixel-space L_2 norm. *Right:* the target rate drops for any $\lambda > 0$; only the clamp two-phase curve is clearly non-monotone, dipping near $\lambda = 0.05$ and recovering, while the other three are roughly monotone.

6 Discussion

The three results of Section 5 confirm the prediction of Section 2.3: clamp is more pixel-efficient, tanh is more imperceptible, and tanh is more sensitive to the batch size, all because of where each variant’s usable gradient lives. The point worth keeping is that the smoother map is the weaker one here: tanh’s smoothness is saturation, and saturation removes the gradient exactly where most of MNIST sits, on the black background and the white strokes. Black and white are symmetric under tanh, so tanh is not unable in principle to lighten the background; rather, it cannot build up a large perturbation on coordinates that start with near-zero gradient, so it falls back on eroding the strokes.

It is worth being precise about *which* pixels tanh erodes, because it is not merely the literal anti-aliased edges of a stroke. The gradient is averaged over the batch, so what matters for a fixed coordinate is not whether it is grey in one image but how often it is un-saturated *across* images. In the batch-averaged signal there are essentially no true 100%-white pixels: because every digit is drawn a little differently, a stroke-core pixel that looks white on the average image is in fact grey or black on many individual images, and on those images it still hands tanh a usable gradient. Tanh therefore works on the whole body of the stroke, not only its rim. What stays a true wall of zero gradient is the pure background, which is black in essentially every image, and that is the region tanh cannot reach cheaply.

7 Future work

Five directions follow naturally from the study.

Mapping the λ front. Section 5.5 shows the imperceptibility trade-off is steep, and for clamp two-phase non-monotone, at $K = 50$, but we never mapped the full front: how far the ℓ_2 penalty can shrink the perturbation before the target rate collapses, and how that front moves with the budget K . A dense sweep over (λ, K) would say precisely what stealth costs and whether a better operating point than $\lambda = 0$ exists at some other budget.

Adversarial training. The universal perturbations are a cheap supply of hard training examples. Folding the attacked images back into training with their *correct* labels should teach the classifier to see through this exact attack and give a more robust model, turning the tool we built into its own defence; the open question is whether robustness to our universal L_0 attack transfers to attacks under other norms.

Larger models and RGB images. The whole clamp-vs-tanh account rests on saturation over a mostly-black, high-contrast dataset. Repeating the comparison on deeper networks and on natural colour images, whose intensities are spread more evenly across $[0, 1]$, would test how far the saturation argument carries and whether clamp keeps its efficiency edge when far fewer pixels are pinned at the extremes.

A better support search. Two-phase selects the support from a single dense optimum. A richer search, restarting the dense phase from several seeds and taking a consensus support, or replacing the hard top- K cut by a gradual annealing of the budget, might recover the few points that plain projection still leaves on the table.

Trust-region steps. The coordinator suggested accepting a step only on sufficient decrease. Our prototype was too noisy under stochastic mini-batches, but a variance-reduced or larger-batch version might make the criterion usable and remove the need for a hand-tuned step size.

Reproducibility

Every run uses an explicit integer seed, so results are reproducible and methods are compared on identical batches. All figures are regenerated from the stored numerical results without re-running the optimisation.

References

- [1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *ICLR*, 2014. arXiv:1312.6199.
- [2] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples (FGSM),” *ICLR*, 2015. arXiv:1412.6572.
- [3] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings (JSMA),” *IEEE EuroS&P*, 2016. arXiv:1511.07528.
- [4] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: a simple and accurate method to fool deep neural networks,” *CVPR*, 2016. arXiv:1511.04599.
- [5] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” *CVPR*, 2017. arXiv:1610.08401.
- [6] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks (C&W),” *IEEE S&P*, 2017. arXiv:1608.04644.
- [7] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks (PGD),” *ICLR*, 2018. arXiv:1706.06083.
- [8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, 86(11):2278–2324, 1998. (MNIST).
- [9] R. Tibshirani, “Regression shrinkage and selection via the LASSO,” *J. Royal Statistical Society Ser. B*, 58(1):267–288, 1996.