$$\begin{aligned} \underset{D}{\text{minimize}} \quad & \|D\| \\ \text{subject to} \quad & L \leq X + D \leq U \\ & p = f(X + D) \\ & \max(p_1 - p_c, ..., p_n - p_c) > 0 \end{aligned} \tag{1}$$

That formulation is more general than the one presented by [1], for it ignores non-essential details, such as the choice of the adversarial label. It also showcases the non-convexity: since $\max(x) < 0$ is convex, the inequality is clearly concave [11], making the problem non-trivial even if the model $p = f(X)$ were linear in X. Deep networks, of course, exacerbate the non-convexity due to their highly non-linear model. For example, a simple multi-layer perceptron for binary classification could have $f(X) = \text{logit}^{-1}(W_2 \cdot \tanh(W_1 \cdot X + b_1) + b_2)$, which is neither convex nor concave due to the hyperbolic tangent.

A. Procedure

Training a classifier usually means minimizing the classification error by changing the model weights. To generate adversarial images, however, we hold the weights fixed, and find the minimal distortion that still fools the network.

We can simplify the optimization problem of eq. 1 by exchanging the $\max$ inequality for a term in the loss function that measures how adversarial the probability output is:

$$\begin{aligned} \underset{D}{\text{minimize}} \quad & \|D\| + C \cdot \text{H}(p, p^A) \\ \text{subject to} \quad & L \leq X + D \leq U \\ & p = f(X + D) \end{aligned} \tag{2}$$

where we introduce the adversarial probability target $p^A = [\mathbb{1}_{i=a}]$, which assigns zero probability to all but a chosen adversarial label a. That formulation is essentially the same of [1], picking an explicit (but arbitrary) adversary label. We stipulate the loss function: the cross-entropy (H) between the probability assignments; while [1] keep that choice open.

The constant C balances the importance of the two objectives. The lower the constant, the more we will minimize the distortion norm. Values too low, however, may turn the optimization unfeasible. We want the lowest, but still feasible, value for C.

We can solve the new formulation with any local search compatible with box-constraints. Since the optimization variables are the pixel distortions, the problem size is exactly the size of the network input, which in our case varies from $28 \times 28 = 784$ for MNIST [12] to $221 \times 221 \times 3 = 146\,523$ for OverFeat [13]. In contrast to current neural network training, that reaches hundreds of millions of weights, those sizes are small enough to allow second-order procedures, which converge faster and with better guarantees [14]. We chose L-BFGS-B, a box-constrained version of the popular L-BFGS second-order optimizer [15]. We set the number of corrections in the limited-memory matrix to 15, and the maximum number of iterations to 150. We used Torch7 to model the networks and extract their gradient with respect to the inputs [16]. Finally, we implemented a bisection search to determine the optimal value for C [17]. The algorithm is explained in detail in the next section.

B. Algorithm

Algorithm 1 implements the optimization procedure used to find the adversarial images. The algorithm is essentially a bisection search for the constant C, where in each step we solve a problem equivalent to Eq. 2. Bisection requires initial lower and upper bounds for C, such that the upper bound succeeds in finding an adversarial image, and the lower bound fails. It will then search the transition point from failure to success (the "zero" in a root-finding sense): that will be the best C. We can use $C = 0$ as lower bound, as it always leads to failure (the distortion will go to zero). To find an upper bound leading to success, we start from a very low value, and exponentially increase it until we succeed. During the search for the optimal C we use warm-starting in L-BFGS-B to speed up convergence: the previous optimal value found for D is used as initial value for the next attempt.

To achieve the general formalism of eq. 1 we would have to find the adversarial label leading to minimal distortion. However, in datasets like ImageNet [18], with hundreds of classes, this search would be too costly. Instead, in our experiments, we opt to consider the adversarial label as one of the sources of random variability. As we will show, this does not upset the analyses.

The source code for adversarial image generation and pixel space analysis can be found in https://github.com/tabacof/adversarial.

Algorithm 1 Adversarial image generation algorithm

```
Require: A small positive value $\epsilon$
Ensure: $L\text{-}BFGS\text{-}B(X, p^A, C)$ solves optimization 2
 1: {Finding initial $C$}
 2: $C \leftarrow \epsilon$
 3: repeat
 4:     $C \leftarrow 2 \times C$
 5:     $D, p \leftarrow L\text{-}BFGS\text{-}B(X, p^A, C)$
 6: until $\max(p_i)$ in $p$ is $p_a$
 7: {Bisection search}
 8: $C_{low} \leftarrow 0$, $C_{high} \leftarrow C$
 9: repeat
10:     $C_{half} \leftarrow (C_{high} + C_{low})/2$
11:     $D', p \leftarrow L\text{-}BFGS\text{-}B(X, p^A, C_{half})$
12:     if $\max(p_i)$ in $p$ is $p_a$ then
13:         $D \leftarrow D'$
14:         $C_{high} \leftarrow C_{half}$
15:     else
16:         $C_{low} \leftarrow C_{half}$
17:     end if
18: until $(C_{high} - C_{low}) < \epsilon$
19: return $D$
```