Adversarial attacks via Sequential Quadratic Programming

Théo Beuzeville, Alfredo Buttari, Serge Gratton, Theo Mary, Erkan Ulker

Adversarial attacks via Sequential Quadratic Programming

Théo Beuzeville[1, 2], **Alfredo Buttari** [2, 4], **Serge Gratton** [3], **Theo Mary** [5, 4], **Erkan Ulker** [1]
[1] Atos
[2] IRIT
[3] Toulouse INP-ENSEEIHT
[4] CNRS
[5] LIP6
theo.beuzeville@toulouse-inp.fr, alfredo.buttari@irit.fr, serge.gratton@toulouse-inp.fr, theo.mary@lip6.fr, erkan.ulker@atos.net

Abstract

Deep neural networks (DNN) achieve state-of-the-art performance in many machine learning tasks and in various types of applications. Their efficiency in solving complex problems has led to apply deep learning techniques in safety-critical tasks such as autonomous driving or medicine. However their sensitivity to adversarial attacks, rounding errors, or quantization processes raises concerns and has lead to high interest in finding new approaches to make them more robust. In this work we propose a novel approach for the construction of adversarial attacks which relies on a local Sequential Quadratic Programming (SQP) strategy. These attacks use second order information to achieve competitive performance compared with existing state-of-the-art approaches. We present numerical results that support our theoretical findings and illustrate the relevance of our approach on well-known datasets.

Introduction

Artificial neural networks (ANN) attempt to exploit the architecture of the human brain in order to perform a wide range of tasks. Deep neural networks, which are artificial neural networks composed of multiple layers between the input and the output of the network, are known to be universal approximators (Hornik, Stinchcombe, and White 1989; Scarselli and Chung Tsoi 1998), in fact many results provide upper bounds on the network size and assert that a small approximation error can be achieved if the network size is sufficiently large. Those theoretical results are often quoted to justify the empirical efficiency of DNNs: indeed, DNNs exceed human accuracy in many applications, from medicine with the detection of cancer (Cireşan et al. 2013), to playing games (Silver et al. 2016; Chellapilla and Fogel 1999; Mnih et al. 2013) and driving cars (Wu et al. 2016). But despite their empirical efficiency many works underline the sensitivity of DDNs to adversarial attacks (Szegedy et al. 2014; Kurakin, Goodfellow, and Bengio 2016; Evtimov et al. 2017; Moosavi-Dezfooli, Fawzi, and Frossard 2015). Among adversarial attacks, artificial neural networks are vulnerable to adversarial examples, which are perturbations applied to an input that would not fool a human but are sufficient to fool the model into making a wrong prediction. Figure 1 shows an example of how a slight perturbation on an image can trigger an erroneous classification by a neural network which works correctly on the unperturbed image.

Figure 1: The camouflage fools the Mask R-CNN object detector (on the bottom), whereas plain colors (on top) is being correctly detected (Zhang et al. 2019b).

Adversarial examples are considered to be a significant obstacle to the deployment of neural networks models in safety-critical task, due to the clear security threat that these attacks represent; this also raises questions regarding the robustness and ability of a neural network to generalize in the context of new distributions. Exact computation of a neural network robustness (Katz et al. 2017; Tjeng, Xiao, and Tedrake 2019), when possible, does not scale well for large neural networks; for instance, the problem of verifying the robustness of a ReLU neural network can be formalized as a Mixed Integer Programming problem, which is NP hard. For this reason many different approaches have been developed to find adversarial examples in order to more efficiently evaluate neural network robustness. Finding adversarial examples with small norm perturbations is crucial to assess the vulnerability of a neural network against attackers, provides for a better understanding of its robustness and allows for improving its robustness by integrating these adversarial samples into the training process — a defense known as adversarial training (Goodfellow, Shlens, and Szegedy 2014; Madry et al. 2018).

The literature on adversarial attacks is very abundant and

many methods have been proposed. A complete review is out of the scope of our work but we make a brief survey of some of the most popular and successful methods in the next section.

In this paper we present a novel approach for creating such adversarial attacks; our aim is to produce perturbations with smaller norms with respect to existing, state-of-the-art methods. To tackle this objective, we propose an approach that relies on second order information and, more precisely, on Sequential Quadratic Programming (SQP) which is a well known and widely studied method for solving constrained optimization problems. We will discuss the practical limitations of a baseline SQP-based method and propose some improvements to overcome them which lead to a hybrid approach mixing first order and second order iterations in order to achieve better convergence and lower execution time.

Adversarial attacks

In this work we focus on a particular type of adversarial attacks on artificial neural network: adversarial examples. These are inputs of a neural network perturbed in such a way that they are classified in a different class than expected whereas a human would still correctly recognize them and assign to them the correct label. Finding *targeted* adversarial examples amounts to computing the smallest norm perturbation on the input data x such that the perturbed input $x+\Delta x$ is misclassified by the neural network in a prescribed target class j instead of the expected true label i. Mathematically, the targeted adversarial perturbation is defined as the solution of the following minimization problem:

$$\begin{aligned} &\textbf{Solve} \\ &\min \|\Delta x\|^2 \\ &\textbf{subject to} \\ &Class(x+\Delta x) = j, \end{aligned} \tag{1}$$

where j is the target class and $Class(x+\Delta x)$ the class of the perturbed image. Different types of norms can be used to measure the size of the perturbation and methods have been proposed that can handle one or the other or even multiple types. In our work, we focus on the Frobenius norm and, for the sake of readability, we will drop the subscript on the $\|\cdot\|_F$ operator in the remainder of this document. In most cases a classifier will associate with an input x the label i out of K classes when the ith component of the classifier's output $C(x)$ is the maximum of its K components, that is: $\arg\max_{k=1,\dots,N} C_k(x) = i$. Hence the constraints $Class(x+\Delta x) = j$ can be formulated as

$$C_i(x+\Delta x) \leq C_j(x+\Delta x), i = 1, \dots, K. \tag{2}$$

In an *untargeted* attack, instead, one will search for a perturbation that leads to a misclassification regardless of the output class. In this case the constraints of the minimization problem become

$$Class(x+\Delta x) \neq i, \tag{3}$$

where i is the true label associated with the input x. Hence for most classifiers the constraints can then be formulated as

$$C_i(x+\Delta x) \leq \max_{j \neq i} C_j(x+\Delta x). \tag{4}$$

It has been argued (Carlini and Wagner 2016) that computing an untargeted attack is often a less accurate approach than running a targeted attack for each target class and then take the smallest perturbation. For this reason, in our work we will focus on computing a targeted attack, knowing that we can then use it in an untargeted context using this method. Our method can easily be extended to the case of untargeted attacks but we consider this is out of the scope of the present work.

In general the original problem (1) is too complex to be solved directly, hence state-of-the-art methods to compute adversarial examples attempt to approximately solve problem 1. Numerous approaches that have been proposed and are commonly used to generate adversarial examples resort to solving the following penalty problem:

$$\begin{aligned} &\textbf{Solve} \\ &\min \|\Delta x\|^2 + c\mathcal{L}(x+\Delta x, j), \end{aligned} \tag{5}$$

where $\mathcal{L}$ is a given loss function, potentially different than the one used to train the neural network, of the input with respect to a given target class. This was first introduced by Szegedy et al. (2014) who formalized the minimization problem and introduced the term of adversarial sample. In their work they proposed an attack using a Large Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm to solve problem 5. The L-BFGS attack uses a line-search algorithm to find the optimal value of the weight c which makes it expensive; to overcome this limitation, Goodfellow, Shlens, and Szegedy (2014) proposed a so-called Fast Gradient Sign Method (FGSM). This attack uses only one step in the direction of the sign of the gradient to generate an adversarial example; hence the obtained perturbation can be expressed as:

$$\Delta x = \epsilon \, sign(\nabla_x \mathcal{L}_f(x, j)) \tag{6}$$

where $\mathcal{L}_f$ is the cost function used during training.

A more powerful direct alternative to FGSM is Projected Gradient Descent (Madry et al. 2018) (PGD) which is an iterative version of FGSM, producing smaller perturbations at the cost of being computationally more expensive. The most well known state-of-the-art adversarial attack using this so-called penalty method is the Carlini–Wagner attack (Carlini and Wagner 2016). Penalty methods transform the constrained optimization problem into an unconstrained one by coupling the need to minimize the perturbation norm and the need to misclassify the image; the resulting optimization problem is easier to solve than the problem in equation (1) but this comes at the expense of needing to find an optimal loss function as well as an optimal weight c which is, for the weight, often achieved using expensive line-search algorithms.

All the above methods aim to optimize both the misclassification and minimal norm criteria at the same time, often using line-search algorithm to find the best balance between

the two criteria. Other methods aim to accelerate the generation of adversarial samples, tailoring algorithms specifically for a given norm. Amongst them DeepFool (Moosavi-Dezfooli, Fawzi, and Frossard 2015) iteratively perturbs the input by linearizing the model around the current point and then find the closest decision boundary for the l_2 or l_∞ norms. DDN-attack (Rony et al. 2018) is another method which decouples the two objectives by using projections on a l_2 ball centered on the original image at each iteration, whereas FAB-attack (Croce and Hein 2019) uses both projections and linear approximations of the neural network to produce competitive adversarial examples.

A recent approach called ALMA (Rony et al. 2021) takes advantage of the Augmented Lagrangian method to attenuate the drawbacks of penalty-based attacks. Indeed with this method the penalty is adaptively modified during the optimization iterations to estimate and converge to the optimal penalty term, which corresponds to the Lagrangian multiplier of the optimization problem.

Local Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is one of the most well-known and successful methods for constrained nonlinear optimization; the first reference to SQP-type algorithms appears in the PhD thesis of Wilson (1963). An exhaustive review of this method is out of the scope of this paper but, in this section, we will explain the general idea behind it; we refer the reader to the above-mentioned thesis or to any optimization textbook such as the one by Boggs and Tolle (1995) or Nocedal and Wright (2006). Although SQP is generic enough to solve optimization problems with both equality and inequality constraints, in this work we will focus on the case where only inequality constraints are present because this is enough for computing adversarial attacks, as explained in the next section. Therefore, our review of the SQP method below is limited to this case for the sake of conciseness.

Let us assume that one has to solve the following minimization problem:

$$\begin{aligned} &\textbf{Solve} \\ &\min f(x) \\ &\textbf{subject to} \\ &g(x) \leq 0. \end{aligned} \tag{7}$$

The idea behind SQP is to solve this problem iteratively: at each iteration x_k the problem is modeled by a constrained quadratic programming subproblem whose minimizer is used to build iterate x_{k+1}. One common choice is to solve the Karush–Kuhn–Tucker (KKT) equations using Newton's method. The Lagrangian of this problem is

$$L(x, \lambda) = f(x) + \lambda^T g(x), \tag{8}$$

where the vector λ corresponds to the Lagrangian multipliers.

To satisfy the KKT conditions one must find (x, λ) such that

$$\begin{aligned} &\nabla L(x, \lambda) = 0 \\ &g(x) \leq 0. \end{aligned} \tag{9}$$

Applying Newton's method to the above equation generates iterates that are identical to those one gets by solving the following quadratic problem:

$$\begin{aligned} &\textbf{Solve} \\ &\min_d \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d \\ &\textbf{subject to} \\ &\nabla g(x_k)^T d + g(x_k) \leq 0. \end{aligned} \tag{10}$$

This leads to the local SQP method in Algorithm 1 where (d_x, d_λ) are the primal dual solution of the quadratic optimization subproblem.

Algorithm 1: local SQP

Input: x_0 starting point of the algorithm.
Input: N_{iter} number of iterations.
while $k = 1, \ldots, N_{iter}$ **do**
  solve
    $\min_{d_x} \nabla f(x_k)^T d_x + \frac{1}{2} d_x^T \nabla^2_{xx} L(x_k, \lambda_k) d_x$
  subject to
    $\nabla g(x_k)^T d_x + g(x_k) \leq 0$
  end solve
  $x_{k+1} \leftarrow x_k + d_x$
  $\lambda_{k+1} \leftarrow d_\lambda$
end while

Adversarial attacks using Sequential Quadratic Programming

In this section we will adapt the local SQP algorithm described in the previous section to solve adversarial attack problems. Essentially, this amounts to defining an objective function and inequality constraints that correspond to the problem in equation (1). We will focus on a targeted formulation of this problem knowing that, as explained above, an untargeted version can be obtained by applying the produced algorithm to all classes and then taking the smallest successful perturbation.

Let us assume that $C : \mathbb{R}^n \to \mathbb{R}^K$ is a classifier such that C assigns to an input $x \in \mathbb{R}^n$ the label i, i being one of the K classes, when

$$C_j(x) \leq C_i(x), j = 1, \ldots, K. \tag{11}$$

Then, as stated before, one must solve the following optimization problem in order to find an adversarial perturbation:

$$\begin{aligned} &\textbf{Solve} \\ &\min_{\Delta x} \frac{1}{2} \|\Delta x\|^2 \\ &\textbf{subject to} \\ &C_i(x + \Delta x) \leq C_j(x + \Delta x), i = 1, \ldots, K, \end{aligned} \tag{12}$$

where j is a target class that is different from the true label of the input x.

Assuming $x_k = x + \Delta x$ is the perturbed input, for the specific case of adversarial attacks on a neural network classifier the objective function of the optimization problem can be defined as

$$f\colon x_k \mapsto \frac{1}{2}\|x - x_k\|^2. \tag{13}$$

Let us define I_K the identity matrix of size K, e_j the j-th canonical vector of size K and $\mathbf{1_K}$ the all-ones vector of size K. Then the inequality constraints of the optimization problem can be expressed as

$$g(x_k) = (I_k - \mathbf{1_K}^T e_j)C(x_k) \leq 0. \tag{14}$$

Improvements to the basic SQP algorithm

Algorithm 1 can readily be used to solve the optimization problem with f and g defined as in the previous section. Nevertheless, this naive approach suffers from some limitations. First, SQP being a local algorithm, its convergence is not guaranteed for any given starting point. This means that Algorithm 1 may fail to compute a successful adversarial attack or to compute one with a small norm; this behavior was indeed observed in a preliminary experimental evaluation.

The second major drawback relates to the use of the Hessian of the Lagrangian $\nabla^2_{xx} L(x_k, \lambda_k)$ which, essentially, amounts to computing second order derivatives of the neural network function that appears in the constraints defined in equation (14). Note that whether this term has to be explicitly computed depends on the optimizer which is chosen to solve the quadratic subproblem on lines 4–8 of Algorithm 1. If the optimizer relies on a direct method, this term has to be computed explicitly; for other choices, it may be sufficient to provide the local optimizer with a function to apply this operator to a vector. In all cases computing or applying this term may be excessively expensive (both in terms of operations and memory).

In order to overcome or mitigate these two limitations, we propose a few modifications.

The first improvement consists in using a hybrid approach where a first order method is used prior to the SQP iterations. The objective is to provide SQP with a better starting point so that it is more likely to converge. Let us say that one seeks to solve the optimization problem (7). Using a first order development of the constraints function we obtain

$$g(x_{k+1}) = g(x_k + d) \approx \nabla g(x_k)^T d + g(x_k).$$

Then a first order iterative problem close in spirit to SQP would be the following:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min_d f(x_k + d) \\ &\textbf{subject to} \\ &\quad \nabla g(x_k)^T d + g(x_k) \leq 0 \end{aligned} \tag{15}$$

This quadratic problem iterates on the perturbations while linearizing the constraints. In our case, where f is defined as in equation (13), we have:

$$\begin{aligned} f(x_k + d) &= \frac{1}{2}\|x - x_k - d\|^2 \\ &= \frac{1}{2}\langle x - x_k | x - x_k\rangle - \langle d | x - x_k\rangle + \frac{1}{2}\langle d | d\rangle . \end{aligned} \tag{16}$$

By noting that

$$\nabla f(x_k)^T d = -\langle d | x - x_k\rangle \tag{17}$$

we obtain

$$f(x_k + d) = \frac{1}{2}\langle x - x_k | x - x_k\rangle + \nabla f(x_k)^T d + \frac{1}{2} d^T I d. \tag{18}$$

Hence solving problem (15) is equivalent to solving

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min_d \nabla f(x_k)^T d + \frac{1}{2} d^T I d \\ &\textbf{subject to} \\ &\quad \nabla g(x_k)^T d + g(x_k) \leq 0. \end{aligned} \tag{19}$$

This is achieved by performing iterations of Algorithm 1 where $\nabla^2_{xx} L(x_k, \lambda_k)$ in line 5 is replaced with the identity matrix.

A second minor improvement consists in adding regularization terms to the update of the current solution at iteration k; these are called α for the first order iterations updates and β for the second order iterations update. The use of these regularization terms was found to drastically improve the convergence in practice and more will be said in the experimental evaluation.

Finally, the choice of the starting point for the first order iterations is still critical to speed up the convergence of the method. In a basic implementation a natural choice would be to set $x_0 = x$, that is, the starting point for the optimization solver is the unperturbed input. Assuming that a targeted adversarial attack towards class j must be computed, we have found that a better starting point is a randomly selected input belonging to class j.

Combining all these improvements leads us to the final algorithm, which we called **S**equential **QU**adratic Programming **AT**tack (SQUAT) and which is described in Algorithm 2. Here, a total number of N_{iter} iterations is performed; alternatively the method can be stopped when a perturbation of small enough norm is computed. Out of these iterations, the first N_1 only use first order information; these are relatively cheap and serve to get closer to an optimal solution. The remaining iterations, that are based on the SQP method and, thus, rely on second order information refine the solution computed by the previous ones. The value of N_1 as well as of α and β has to be carefully chosen to achieve fast convergence.

Algorithm 2: SQUAT

Input: j the target label.
Input: N_{iter} total number of iterations.
Input: N_1 number of iterations before using second order information.
Input: α and β, fixed regularization terms.
$x_0 \leftarrow x^j$, x^j being an image belonging to class j.
while $k < N_{iter}$ **do**
  Solve
  if $k = 1, \dots, N_1$:
    $\min_d \nabla f(x_k)^T d + \frac{1}{2} d^T I d$
  if $k > N_1$:
    $\min_d \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d$
  subject to
    $\nabla g(x_k)^T d + g(x_k) \leq 0$
  end solve
  if $k \leq N_1$:
    $x_{k+1} \leftarrow x_k + \alpha d_x$
  if $k > N_1$
    $x_{k+1} \leftarrow x_k + \beta d_x$
    $\lambda_{k+1} \leftarrow \beta d_\lambda$
end while

Experiments

Experimental setup

In order to compare our method with the current state-of-the-art we chose to use the code provided by Rony et al. (2021). In the following comparisons we will only use non-targeted attacks to show that our method can compete with, and even outperform, state-of-the-art attacks on a non-targeted scheme. More precisely, we will compare our method to state-of-the-art methods for the l_2 norm: C&W (Carlini and Wagner 2016), DDN (Rony et al. 2018), FAB (Croce and Hein 2019), and ALMA (Rony et al. 2021). The experiments are done on a SmallCNN neural network from Zhang et al. (2019a), in order to evaluate our attack against defenses we use this network in different training setups. First it is trained regularly and then trained to be robust to adversarial attacks: first using l_∞-TRADES defense (Zhang et al. 2019a) and then using l_2-DDN defense (Rony et al. 2018) on the MNIST dataset (LeCun, Cortes, and Burges 2010). We will denote these neural networks, respectively, SmallCNN for the regularly trained model, SmallCNN-TRADES and SmallCNN-DDN for the adversarially trained models.

Metrics

We assess the performance of an attack by how much the accuracy of a given neural network decreases depending on how small perturbations are in terms of norm. This is measured using a metric called *robust accuracy*: for a given value ϵ and assuming all the inputs of a dataset are attacked, it corresponds to the percentage of input data that have been successfully attacked with a perturbation of norm smaller than ϵ. Hence, the lower the robust accuracy for a given threshold, the better the attack. In Figures 3, 4 and 5 we plot curves showing the robust accuracy as a function of ϵ.

For all the experiments we chose to evaluate the attacks on the first 500 images of the testing dataset. To summarize these results we also present the median distance between the found adversarial example and the original input for multiple attacks and neural networks in Table 2.

Algorithm implementation and complexity

The complexity of an attack on a deep learning system can be measured in terms of forward and backward evaluations of the model as these operations usually require more computational power than other operations involved in these algorithms. In our case most of the operations are done computing first and second order derivatives of the neural network model to define each quadratic sub-problem. These sub-problems are then solved using the Python CVXPY library (Diamond and Boyd 2016; Agrawal et al. 2018). For the experiments reported in this section, the algorithm used by the solver is the default one, namely, Operator Splitting Quadratic Program (Stellato et al. 2020) (OSQP) which implements an Alternating Direction Method of Multipliers (Boyd et al. 2011) (ADMM) variant. In this case, the solver requires the entire matrix involved in the problem's formulation, which, in our case, implies computing the Hessian of the Lagrangian. Other types of optimizer could be used to solve the quadratic subproblem and other implementation choices could be made in order to speed up the execution but exploring all these parameters is out of the scope of this paper as we aim to show how a simple SQP-type approach could efficiently be used to create competitive adversarial attacks.

The choice of the number of iterations N_{iter} has, clearly, a considerable impact on the cost of our method. Essentially, we can afford doing many first order cheap iterations (N_1) in order to get as close as possible to an optimal solution such that only few expensive second order iterations are needed to reach convergence. In our experiments N_1 is fixed to a relatively large number. As for the second order iterations, the number is not fixed but we keep on iterating as long as the constraints of the quadratic subproblem are satisfied because this means that the algorithm has successfully improved the solution obtained at the previous iteration. The number of second order iteration is, typically, of the order of a few tens.

For the following experiments we chose to use a budget of 4000 iterations per image for ALMA, DDN, FAB and C&W while for SQUAT we use a budget of approximately 2000 iterations as those iterations can be more expensive. As an example, for the SmallCNN-TRADES neural network, this typically results in a total of 4000 forward and 4000 backward propagations for DDN and ALMA, 8000 forward and 40000 backward for FAB, 30000 forward and 30000 backward for C&W. In this case the SQUAT attack performed in average 20 iterations with second order information per image while having set $N_1 = 2000$ first order iterations per image. For the first order iterations 1 forward and 10 backward propagations are needed, while in order to compute a Hessian matrix-vector multiplication 1 forward and 1 backward propagations would be needed, but to compute the full hessian here the cost scale as the input size n so we make the assumption that computing the Hessian costs as much as

n forward and n backward propagations, which gives an average of 18000 forward and 36000 backward propagations in total.

The learning rate α used in the first order iterations, has to be carefully chosen. In our experiments, in order to keep the computational cost comparable to other methods, we chose not to use a line-search algorithm to find an optimal learning rate value; instead, we only run the first order part of our attack with few iterations in order to compare the algorithm performance depending on α. We compare, in Table 1, the median distance of the attacks perturbations depending on the value of α, and, in Figure 2, an example of how changes on α impact the final results of the SQUAT algorithm by showing robust accuracy curves on the SmallCNN-TRADES.

Training					
Regular		DDN		TRADES	
α	Median Distance	α	Median Distance	α	Median Distance
0.8	1.80	0.7	3.68	0.2	1.89
0.9	1.76	0.8	3.55	0.3	1.70
1	1.79	0.9	3.52	0.4	1.68
1.1	1.83	1	3.58	0.5	2.01

Table 1: Adversarial attack performances on SmallCNN depending on the choice of α and the training setup.

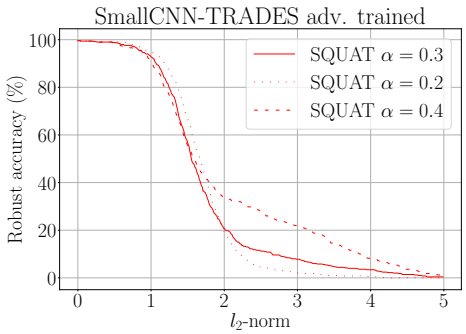

Figure 2: Robust accuracy curves for the SmallCNN-TRADES adversarially trained model on MNIST.

As seen in Table 1, and particularly for the more robust networks, the behavior of the algorithm can be very sensitive to the choice of α. For the SmallCNN and SmallCNN-DDN neural network we chose $\alpha = 0.9$ but, as seen in Figure 2, the median distance does not always give enough information and using robust accuracy curves we chose in this case to use $\alpha = 0.3$ as it gave a better performance trade-off between smaller and larger norm. The second hyperparameter β which is the learning rate when using second order information and is fixed at $\beta = 0.15$ for all our experiments. Indeed tuning this parameter as it has been done for α can be costly; moreover the goal here is to improve a good enough starting point, it is not an exploration phase anymore, hence one does not want to make large steps and we found that usually any $\beta \leq 0.2$ can be a good choice, knowing that the smaller β is the more step will be needed to converge.

Results

As shown on the robust accuracy curves of all different attacks on Figure 3 and on the median distances of Table 2 we observe that our proposed attack SQUAT outperforms other state-of-the-art attacks on the regularly trained SmallCNN. Indeed for a given threshold the robust accuracy is globally lower with SQUAT than with other attacks. This is even clearer for smaller norms as seen on the bottom of Figure 3: for norms in the $[0, 1]$ range, all other attacks have comparable results and only our method stands out. Finally, it must be noted that SQUAT and ALMA, attain 100% attack success rate approximately at the same perturbation norm which is smaller than the other methods.

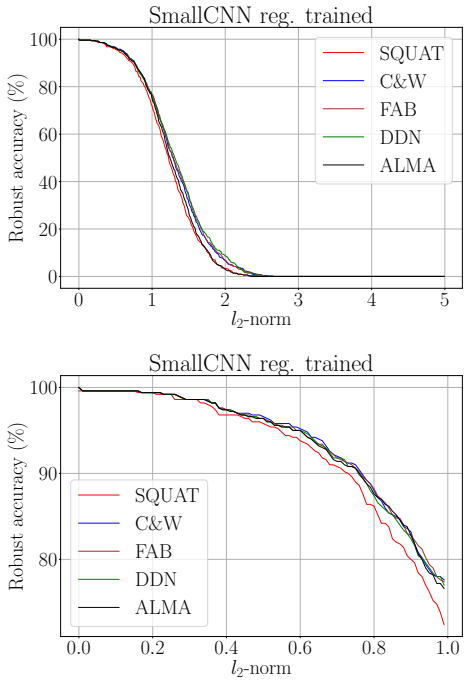

Figure 3: Robust accuracy curves for the SmallCNN regularly trained model on MNIST.

On Figure 4 and Figure 5 which give robust accuracy curves on the SmallCNN-DDN and SmallCNN-TRADES we can see that all the methods behave approximately the same for norms belonging in the $[0, 1]$ interval; this is not suprising as the neural networks are much more robust and hence less affected by small norm perturbations. On the l_2-DDN adversarially trained network all the attacks are significantly less efficient than for the regularly trained network, this is expected as these attacks are designed for the l_2-norm and the SmallCNN-DDN is designed to be robust to these types of attacks. As seen in Figure 4, in this setup ALMA, SQUAT perform similarly, with a slight advantage for ALMA when perturbations norm are in the $[2, 5]$ range. On the other hand C&W, FAB and DDN all perform simi-

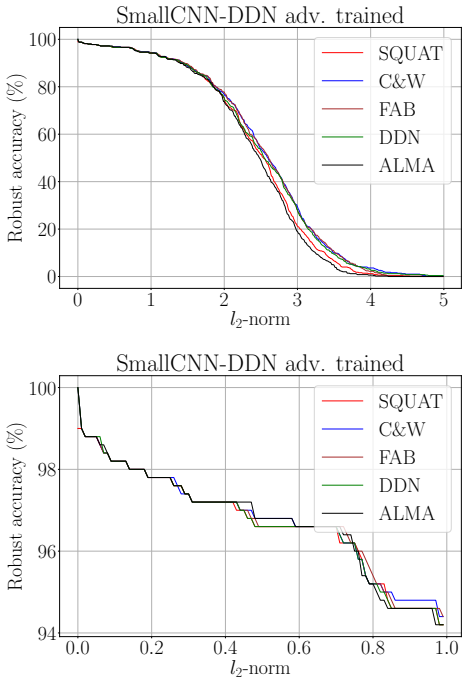

Figure 4: Robust accuracy curves for the SmallCNN-DDN adversarially trained model on MNIST.

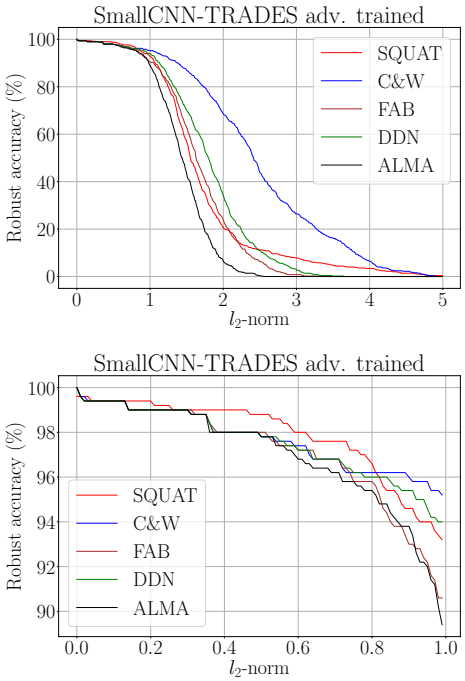

Figure 5: Robust accuracy curves for the SmallCNN-TRADES adversarially trained model on MNIST.

larly for this neural network. Median distances obtained on the l_∞-TRADES adversarially trained neural network are closer to those we get on the regularly trained one except for the C&W attack which performs poorly compared to other state-of-the-art attacks. In this case SQUAT perform similarly to FAB, outperforming DDN and C&W in terms of median distance of adversarial perturbations, whereas globally ALMA get better results on this training setup.

	Training		
	Regular	DDN	TRADES
Attack	Median Distance	Median Distance	Median Distance
SQUAT	1.22	2.53	1.55
FAB	1.29	2.61	1.63
DDN	1.31	2.55	1.80
ALMA	1.24	2.48	1.44
C&W	1.28	2.59	2.41

Table 2: Adversarial attacks performances on SmallCNN depending on the training setup.

Conclusion and discussion

The existence of new types of adversarial attacks poses potential security threats to machine learning models, hence designing adversarial attacks and defenses is a subject of great interest. Indeed they enable to better understand neural networks sensitivity and improve their robustness, notably by the means of adversarial training.

In this work we have shown how to compute adversarial attacks on deep neural networks using a Sequential Quadratic Programming based approach, adapting the basic algorithm to this specific case in particular by first using a first order variation of SQP in order to then use second order information to improve the resulting perturbations.

The goal of this paper is to provide a first look of how successful adversarial attacks can be built using second order information and using existing optimization algorithms. As seen in the experimental results section, we have shown that our approach can compete and even outperform others state-of-the-art attacks on models that are regularly and adversarially trained.

As seen in the work by Rony et al. (2021), using existing optimization methods to attack neural networks often needs some customization. Our approach relies on few, simple, modifications of the original SQP algorithm, which enable the use of this method in the specific case of adversarial attacks on a neural network and still obtains competitive results.

Many SQP-type algorithms and solvers (Byrd, Nocedal, and Waltz 2006) have been developed and could be used in our approach, notably Gill, Murray, and Saunders (2002) designed a software using algorithm with limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. Those types of solvers, using Hessian approximation or taking advantage of Hessian-vector product could be of special interest for designing a more robust and less expensive SQP-approach to create adversarial examples in the future.

References

Agrawal, A.; Verschueren, R.; Diamond, S.; and Boyd, S. 2018. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1): 42–60.

Boggs, P. T.; and Tolle, J. W. 1995. Sequential Quadratic Programming. *Acta Numerica*, 4: 1–51.

Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; and Eckstein, J. 2011. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Found. Trends Mach. Learn.*, 3(1): 1–122.

Byrd, R. H.; Nocedal, J.; and Waltz, R. A. 2006. *Knitro: An Integrated Package for Nonlinear Optimization*, 35–59. Boston, MA: Springer US. ISBN 978-0-387-30065-8.

Carlini, N.; and Wagner, D. 2016. Towards Evaluating the Robustness of Neural Networks.

Chellapilla, K.; and Fogel, D. 1999. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Transactions on Neural Networks*, 10(6): 1382–1391.

Cireşan, D. C.; Giusti, A.; Gambardella, L. M.; and Schmidhuber, J. 2013. Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks. In Mori, K.; Sakuma, I.; Sato, Y.; Barillot, C.; and Navab, N., eds., *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, 411–418. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-40763-5.

Croce, F.; and Hein, M. 2019. Minimally distorted Adversarial Examples with a Fast Adaptive Boundary Attack. *CoRR*, abs/1907.02044.

Diamond, S.; and Boyd, S. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83): 1–5.

Evtimov, I.; Eykholt, K.; Fernandes, E.; Kohno, T.; Li, B.; Prakash, A.; Rahmati, A.; and Song, D. 2017. Robust Physical-World Attacks on Machine Learning Models. *CoRR*, abs/1707.08945.

Gill, P.; Murray, W.; and Saunders, M. 2002. SNOPT: An SQP Algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12: 979–1006.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5): 359–366.

Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. ISBN 978-3-319-63386-2.

Kurakin, A.; Goodfellow, I. J.; and Bengio, S. 2016. Adversarial examples in the physical world. *CoRR*, abs/1607.02533.

LeCun, Y.; Cortes, C.; and Burges, C. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602.

Moosavi-Dezfooli, S.; Fawzi, A.; and Frossard, P. 2015. DeepFool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599.

Nocedal, J.; and Wright, S. J. 2006. *Numerical Optimization*. New York, NY, USA: Springer, 2e edition.

Rony, J.; Granger, E.; Pedersoli, M.; and Ben Ayed, I. 2021. Augmented Lagrangian Adversarial Attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 7738–7747.

Rony, J.; Hafemann, L. G.; Oliveira, L. S.; Ayed, I. B.; Sabourin, R.; and Granger, E. 2018. Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses. *CoRR*, abs/1811.09600.

Scarselli, F.; and Chung Tsoi, A. 1998. Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results. *Neural Networks*, 11(1): 15–37.

Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489.

Stellato, B.; Banjac, G.; Goulart, P.; Bemporad, A.; and Boyd, S. 2020. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4): 637–672.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Tjeng, V.; Xiao, K. Y.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*.

Wilson, R. 1963. *A Simplicial Algorithm for Concave Programming*. Graduate School of Business Administration, George F. Baker Foundation, Harvard University.

Wu, B.; Iandola, F. N.; Jin, P. H.; and Keutzer, K. 2016. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. *CoRR*, abs/1612.01051.

Zhang, H.; Yu, Y.; Jiao, J.; Xing, E.; Ghaoui, L. E.; and Jordan, M. 2019a. Theoretically Principled Trade-off between Robustness and Accuracy. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 7472–7482. PMLR.

Zhang, Y.; Foroosh, H.; David, P.; and Gong, B. 2019b. CAMOU: Learning Physical Vehicle Camouflages to Adversarially Attack Detectors in the Wild. In *International Conference on Learning Representations*.