# Adversarial Attacks on Neural Networks Using the NAG Library

MAC-MIGS Industrial Project

Meda Andrea, Kevin Zhang, Zorba Denis

September 10, 2025

**Abstract**

With an increasingly strong drive to utilise the outstanding capabilities of neural networks across all fields, their well-known vulnerability against adversarial attacks is also becoming an unneglectable security concern in many high-risk applications. Small perturbations that are often unnoticeable to humans can fool a highly accurate neural network to predict wrong or even pre-specified output. In this project, we leverage the versatility provided by the NAG library to perform adversarial attacks on image classifiers. Its off-the-shelf optimisers can compute attack perturbations of different formulations efficiently, without the necessity to perform constrained-to-unconstrained conversion or linearise constraints for the objectives. Therefore, we also empirically demonstrate that these allow our pipeline to perform successful attacks with even smaller perturbation, compared with other popular adversarial attack algorithms.

## Authors' Contribution:

All authors have contributed equally to group meetings, discussions, presentations and report preparations.
All authors have seen and approved this statement.

# 1 Introduction

Neural networks, despite their tremendous successes across almost all fields, are surprisingly vulnerable to adversarial attacks. Small perturbations that are almost invisible to human eyes, can easily fool a highly accurate image classifier to predict wrong or specifically chosen label as target. With the ever widening and accelerating adoption of these technologies, these vulnerabilities could prove troublesome for our society and we should prevent them from happening. This project aims to explore the optimization capabilities of the NAG library for computing optimal adversarial alterations. A key question driving this investigation is whether the optimal classification-altering modification depends significantly on the chosen norm. And the following natural questions arise:

1. When comparing 1- and 2-norms, one might inquire: how do the smallest 1-norm alterations that change classification compare to the smallest 2-norm alterations with the same effect?

2. Can state-of-the-art general purpose optimization routines in the NAG library be interface conveniently with Pytorch?

3. Can the NAG routines compete with customized attack algorithms in terms of quality of solution and computational expense?

4. Does the extra flexibility of the NAG routines lead to new forms of attack and new insights into the vulnerabilities of image classification tools?

And other different investigations will be carried out, for this project we use pytorch [1] and the NAG library [2].

## 1.1 Adversarial attacks on Neural Networks

Adversarial machine learning is the study of the attacks on machine learning algorithms, and of the defenses against such attacks[3].

The deployment of deep learning algorithms spans various sectors, showcasing their versatility [4]. However, empirical evidence suggests that even the most advanced deep learning models can be deceived by meticulously crafted examples [5, 6]. These adversarial attacks can yield surprising success rates. Notably, in image classification, researchers have demonstrated the ability to make imperceptible alterations to images, leading to a complete change in classification by neural networks[7].

Central to this domain lies an optimization question: what minimal perturbation to the input triggers a classification change? This inquiry can be formalized by defining a norm to measure input alterations and setting constraints to preserve pixel values within an appropriate range.

## 1.2 Geometry of Adversarial Attacking

Adversarial attacks exploit the geometry of decision boundaries to deceive models into making incorrect predictions. These decision boundaries, see Figure 1, which separate different classes within the model's learned representation space, are inherently complex, especially in high-dimensional spaces such neural networks exist. Adversarial examples are meticulously crafted inputs that are designed to cross these decision boundaries by applying minimal, often imperceptible, perturbations to legitimate inputs.

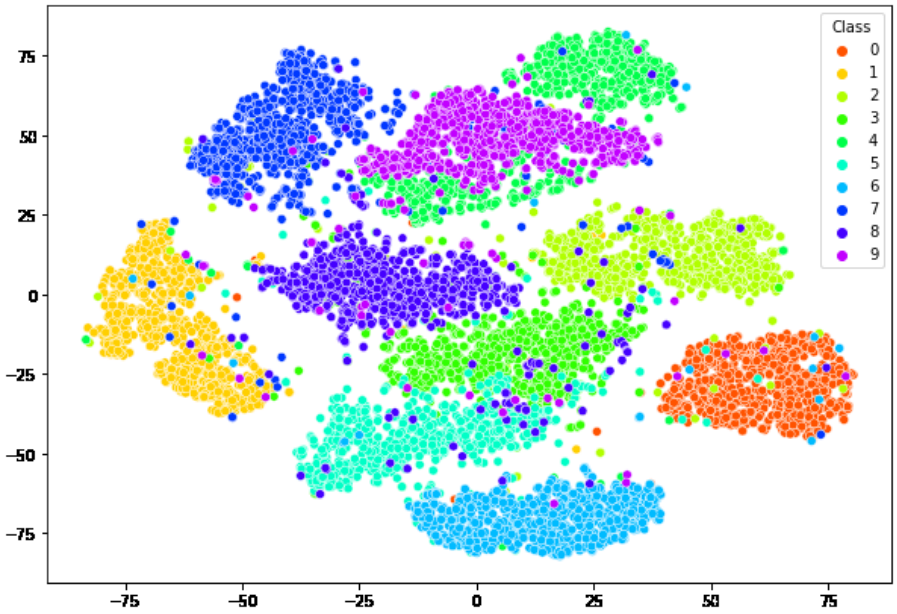


Figure 1: t-SNE representation of the MNIST dataset, courtesy of [8]

As a motivating example, Figure 1 shows that in the MNIST dataset, class 9 and class 4 can be seen as *neighbours* in the representation space (or manifold). This work will explore how different adversarial attacks, both linear and non-linear formulations, find the minimum perturbations that send points in class 9 to class 4, class 7 to 3 etc. We will also consider the untargeted approaches where the optimisers will have freedom in the class translation.

### 1.3 NAG library

Introducing the NAG Library: "it stands as the world's largest commercially accessible repository of robust, meticulously documented, and rigorously tested numerical and statistical algorithms" [2]. Supported by top-tier technical assistance and subjected to continuous updates and maintenance, the NAG Library represents an unparalleled resource in its field. Renowned for its inherent flexibility, all its routines are consolidated in one convenient location. This versatility facilitates seamless transitions between programming languages, effortless progression from prototypes to production stages, and a myriad of other productivity-enhancing features. It is a collection of more than 1600 numerical and statistical algorithms, callable from many computer programming languages and environments including C and C++, Python, Java, .NET, and Fortran. It is component based to provide the building blocks to solve thousands of complex numerical problems. NAG allows to solve problems in Mathematical Optimization, Statistics and Machine Learning, Special Functions, Linear Algebra, PDEs, Interpolation and other mathematical problems.

NAG library provides toolkits to solve optimisation problems of the following form:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \ f(x) \\ &l_g \leq g(x) \leq u_g \\ &l_B \leq Bx \leq u_B \\ &l_x \leq x \leq u_x \end{aligned}$$

for some objective function $f$ optimising over design variable $x$, with some nonlinear constraints $g(x)$ and linear constraints $Bx$. Most optimisation solvers request arguments of the following form

| | $\min_{x \in \mathbb{R}^n} f(x)$ |
|---|---|
| objfun | $f(x)$ |
| objgrd | $\frac{\partial f}{\partial x}$ |
| confun | $g(x)$ |
| congrd | $\frac{\partial g}{\partial x}$ |

where "obj", "con", "fun", "grd", stands for "objective", "(nonlinear) constraint", "function", and "gradient".

The project mainly utilised two solver offered by NAG library: sequential quadratic programming (SQP) (naginterfaces.library.)opt.handle_solve_ssqp (e04srf) and interior point optimiser (IPOpt) opt.handle_solve_ipopt (e04stc). Because we empirically observe that both solvers often converge to very similar points, the project mainly uses opt.handle_solve_ssqp, as it supports optimisation without passing the gradients, which is useful when the solver determines some of our problems to be infeasible. Problems related to the gradients and infeasibility is discussed in detail in section 5.

Although it is possible to optimise the above formulation using other optimisers that are almost ubiquitous in training large-scale neural networks, such as stochastic gradient descent and ADAM [9], these methods requires the conversion of the constrained problem to an unconstrained alternative. This conversion process not only introduce an additional tunable hyperparameter to the objective function, but also loses the guarantee of an successful attack, as the converged result may not satisfy the hard constraints of the formulation. However, since the number of design variables in the context of adversarial attacks is far fewer than training neural networks, we demonstrate in this project that classical optimisation algorithms, such as SQP and IPOpt, are able to carry out attacks in a tractable and efficient manner.

Another common approach to carry out these attacks require the linearisation of the constraint [10, 11]. Whereby assuming the attack perturbation is linearly proportional to the change in the predicted scores, one can reduce the problem into attacking a linear model, which are easier to compute but introduces a systematic bias in the converged perturbations. We thus empirically observe that, our linearisation-free attack pipeline can often perform successful adversarial attacks with smaller perturbations.

### 1.4 Notation

We denote the original image as $x \in \mathbb{R}^c \times \mathbb{R}^m \times \mathbb{R}^n$ with $c$ channels ($c = 1$ for black and white images, $c = 3$ for colored ones: $rgb$), $m$ horizontal and $n$ vertical dimensions.

The neural network is denoted by $F$, and in this project, we only consider the adversarial attacks on image classifiers [12]. For a given $x$, the output of the model $F(x) \in \mathbb{R}^L$ is a vector of predicted scores, with $L$ being the number of label classes of the dataset. For example, the MNIST dataset in PyTorch contains images of handwritten numbers 0 to 9, therefore $L = 10$ in this case. The classification result of an image can be obtained by computing $\arg\max_j \{F(x)\}_j$.

# 2 Methods

In this project, we formulate the task of finding the perturbation $\Delta x$ such that

$$\arg\max_j \{F(x + \Delta x)\}_j \neq l$$

where $l$ denotes the classification result of the unperturbed image of a given model $F$. This project mainly explores two approaches: directly minimising $\Delta x$ (Formulation 1) and maximising the predicted score of the target class, the pre-specified class $F$ one want the model to classify (Formulation 2).

## 2.1 Objective Formulation 1

We started off by investigating the most intuitive approach of minimising over the $p$-norm of the perturbation $\Delta x$, which directly finds the smallest possible perturbation to an image and guarantee the success of an adversarial attack. This is formulated and implemented as follows

$$\min_{\Delta x \in \mathbb{R}^{c \times m \times n}} ||\Delta x||_p^p \tag{1}$$

$$\text{subject to } F(x + \Delta x)_t > \max_{j \neq t} F(x + \Delta x)_j \tag{2}$$

where $t$ is the targeted label.

| | $\min_{\Delta x \in \mathbb{R}^{c \times m \times n}} ||\Delta x||_p^p$ |
|---|---|
| objfun | $||\Delta x||_p^p$ |
| objgrd | $p(\Delta x)^{p-1}$ |
| confun | $F(x + \Delta x)_t - \max_{j \neq t} F(x + \Delta x)_j$ |
| congrd | $\frac{\partial}{\partial(\Delta x)} \left( F(x + \Delta x)_t - \max_{j \neq t} F(x + \Delta x)_j \right)$ |

where $\Delta x$ is the design variable and are flattened as required by the NAG library. Both confun and congrd can be easily obtained by the feedforward and autograd functionality provided by Pytorch.

The untargeted variation of the above formulation was also investigated, whereby modifying the nonlinear constraint

$$F(x + \Delta x)_l < \max_{j \neq l} F(x + \Delta x)_j$$

we enforce the maximum predicted score of any non-true label $j \neq l$ to be larger than the score of the true label $l$.

One may note that untargeted attacks of this form is intrinsically greedy, as the solver will always optimise with respect to the most probable class, and the optimised $\Delta x$ may thus not be the smallest possible perturbation that can guarantee a successful attack.

Lastly, since neural networks tend to extrapolate poorly for out-of-domain images, it is generally easier to attack the models with pixel values falling outside the image dataset domain. Therefore, to ensure the validity of the computed perturbation without exploiting this inherent vulnerability, we require every pixel of $x + \Delta x$ must be within the domain range. This can be enforced by defining an element-wise simple bound

$$u_{x_i} = \max_{j,k} x_j^{(k)} - x_i \tag{3}$$

$$l_{x_i} = \min_{j,k} x_j^{(k)} - x_i \tag{4}$$

where $\max_{j,k} x_j^{(k)}$ denotes the maximum pixel value of all pixels $j \in [1, c \times m \times n]$ of all images $x^{(k)}$ within the dataset of size $N$. This simple bound is applied throughtout this project, unless otherwise mentioned.

## 2.2 Objective Formulation 2

The second formulation adopts the opposite approach, which maximises the predicted score of the target class $t$, while constraining the $p$-norm of the perturbation within $\epsilon$.

$$\max_{\Delta x \in \mathbb{R}^{c \times m \times n}} F(x + \Delta x)_t$$
$$\text{subject to } ||\Delta x||_p^p < \epsilon$$

| | $\max_{\Delta x \in \mathbb{R}^{c \times m \times n}} F(x + \Delta x)_t$ |
|---|---|
| objfun | $F(x + \Delta x)_t$ |
| objgrd | $\frac{\partial}{\partial(\Delta x)} \left(F(x + \Delta x)_t\right)$ |
| confun | $\|\|\Delta x\|\|_p^p$ |
| congrd | $p(\Delta x)^{p-1}$ |

However, by bounding the maximum perturbation with $\epsilon$, it no longer guarantees the success of the attack, and the task of finding the minimum perturbation will require an iterative search over multiple possible $\epsilon$, which can be inefficient and thus undesirable.

Although the concept of greediness does not make sense in the context of an targeted attack, but as a motivation, we also investigated the performance of minimising the cross-entropy loss $L$ between the predicted scores and the one-hot vector $Y_t$ of the attack target class $t$

$$\min_{\Delta x \in \mathbb{R}^{c \times m \times n}} L(F(x + \Delta x)_t, Y_t)$$

in which case the optimisation is performed with respect to all label classes by maximising the score of the target class, while minimising the scores of all other classes. However, similar ideas cannot be easily applied to Formulation 1 to achieve non-greedy untargeted attack.

### 2.3 Universal attack

Another important idea investigated during this project is to find a single perturbation that simultaneously attacks multiple images. This does not require much change to the above formulations, merely involving changing the shape of $x$ to $\mathbb{R}^b \times \mathbb{R}^{c \times m \times n}$, with $b$ indicating the image batch size for optimisation, and broadcasting $\Delta x$ of shape $\mathbb{R}^{c \times m \times n}$. However, its code implementation is still non-trivial, due to additional complexity involved when computing their gradients.

On the other hand, a universal attack cannot be scaled to the whole dataset without discarding the domain restriction on pixel values. To see this, we generalise equation (3) and (4) to the whole dataset for computing the universal attack,

$$u_{x_i} = \max_{j,k} x_j^{(k)} - \max_p x_i^{(p)}$$
$$l_{x_i} = \min_{j,k} x_j^{(k)} - \min_p x_i^{(p)}$$

where $\max_p x_i^{(p)}$ indicates the maximum value of pixel $i$ of all images $x^{(p)}$ within the batch of size $b$. As $b$ becomes very large, $u_{x_i}, l_{x_i}$ will converge towards zero, restricting the domain size of $\Delta x$ to an extent that any attack is almost guaranteed to be unsuccessful.

## 3 Results

In this section we present the different experiments and results that we produced throughout this industrial project. Although $p$-norm of the minimum perturbation 2.1 to perform attacks is an important evaluation metric, it may also be easy for the numerical algorithms to produce pertubations when attacking a single image, as the different algorithms and formulations are often able to converge towards the same point independent of the initialisation. Therefore, as well as investigating different formulations in performing universal attacks, we also define a more challenging evaluation metric: the "success rate" or "accuracy" of using the perturbation computed over a small $n$-subset of images in attacking the entire test set

### 3.1 Handwritten number MNIST

For all attacks on MNIST image datasets, we use the trained model from the official Pytorch tutorial on fast gradient sign attack [13] and used its corresponding test set to compute the perturbations.

#### 3.1.1 Targeted vs untargeted

To analyse the minimum perturbation under targeted and untargeted attacks, we attacked the full test set and compared their distributions. Intuitively the constraints of the untargeted case are more relaxed compared to that of the targeted attacks: the untargeted attacks will always seek to optimise towards the class already with the highest predicted score, instead of optimising towards a pre-chosen and often more difficult class. Figure 2 validates this idea. We see that on average untargeted attacks requires smaller perturbations than the targeted case. This can be interpreted by referring to Figure 1, where in the untargeted case the perturbation will normally map a point on the manifold to it's neighbouring class while this is restricted in the targeted case.

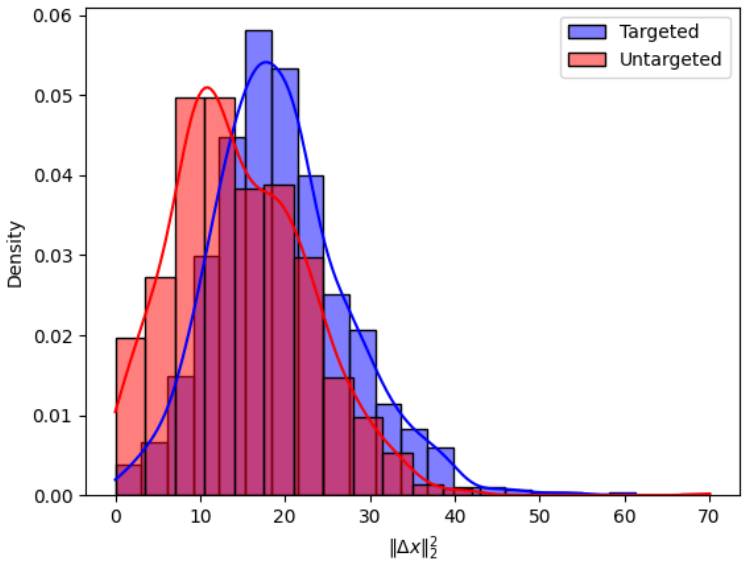


Figure 2: The histogram compares the 2-norm perturbations obtained using the targeted and untargeted variations of Formulation 1.

### 3.1.2 Linearisation

Given Problem Formulation 1, it is of common practice to linearise non-linear functions to compute the perturbation. Denoting

$$g(x) = F(x + \Delta x)_l - \max_{j \neq l} F(x + \Delta x)_j,$$

It is clear that the optimum solution lies on the decision boundary for $g(x) = 0$. Deepfool [11] considers the first order approximation of $g(x)$ around $x^*$ as

$$g(x) = g(x^*) + \nabla_x g(x^*)^T (x - x^*)$$

This results in an untargeted iterative approach to adversarial attacks. In our experiments, we attack the full MNIST dataset using the untargeted linearised attack from Deepfool and the linearisation-free formulation using NAG. Figure 4 shows that the non-linear optimisation results in smaller perturbations and a tighter standard deviation in the distribution. This is could be interpreted as the non-linear optimisation *learning* the 784 dimensional manifold that MNIST images exist. Further evidence, as a critique of the linearisation-free optimisation) can be seen in Figure 3 where we compare an attack on the digit 4 with the NAG library and the one given by Deepfool. Both are untargeted attacks that result in a successful attack with classification 9. NAG attempts to draw a horizontal straight line, which one can argue mimics what a human would do to modify a 4 into a 9 thus has a better understanding of the MNIST manifold. Deepfool on the other hand portrays an almost Gaussian noise on the digit ink. We find this result very interesting, since the norm of the final perturbation produced by our code is less than the Deepfool one, at the cost of a higher computational time. This realisation could motivate further research into incorporating a sense of similarity of the original image and the attacked image, or hence the dissimilarity between the attacked image and a clean image from the new class, in the objective function we minimise.

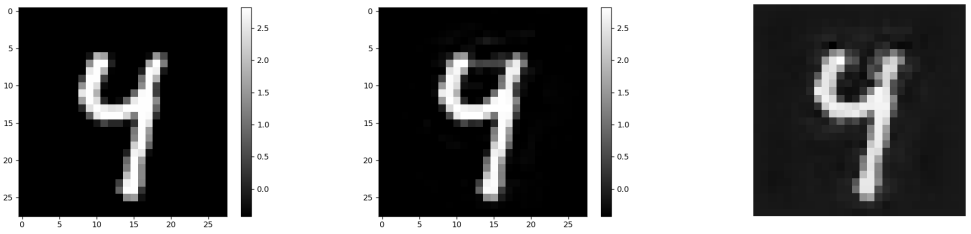

Figure 3: Left: Original image. Middle: $4 \rightarrow 9$ using NAG, Execution time: 2.19s, $\|\Delta x\|_2^2 = 5.9$. Right: $4 \rightarrow 9$ using DeepFool, Execution time: 0.51s, $\|\Delta x\|_2^2 = 10.6$

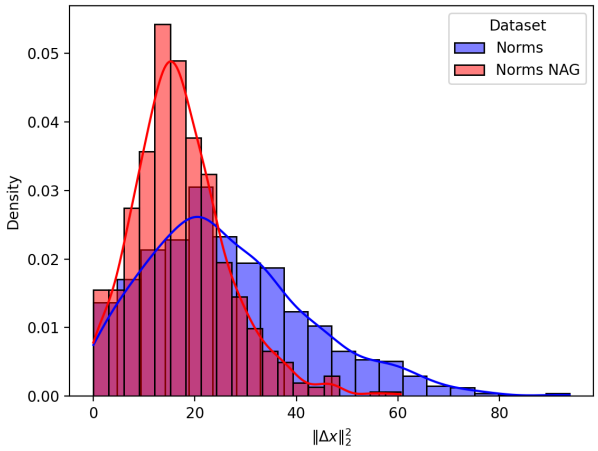


Figure 4: Distributions of the $\|\Delta x\|_2^2$ on untargeted attacks from Deepfool (Blue: Mean 25.86, Std 15.9) and NAG (Red: Mean 17.25, Std 9.59).

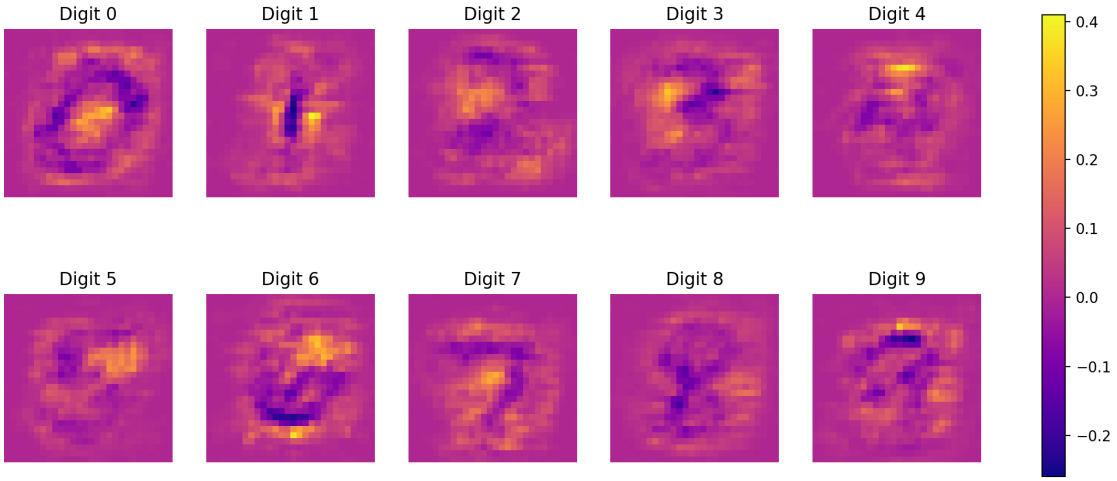


Figure 5: Average pertubations from untargeted attacks on the different MNIST digits using non-linear optimisation from NAG

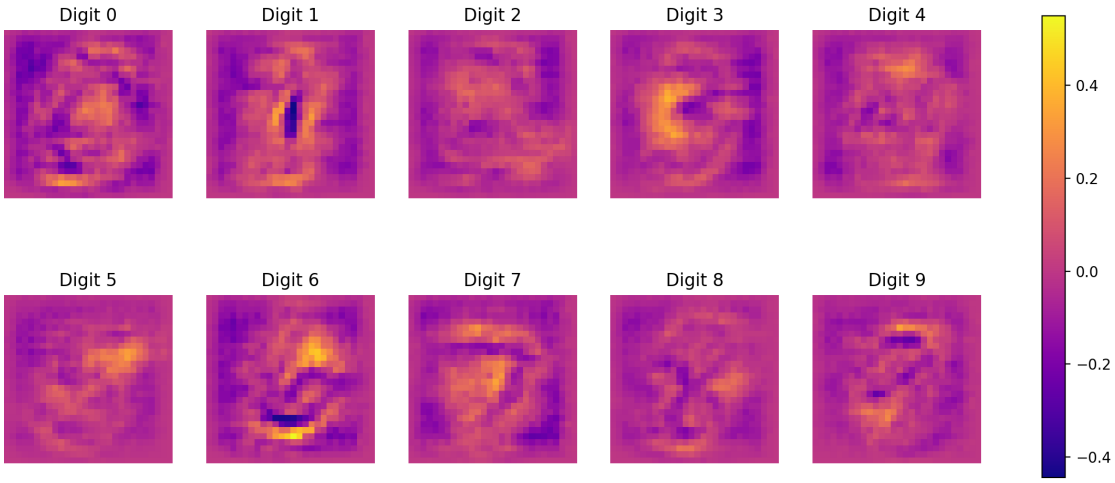


Figure 6: Average pertubations from untargeted attacks on the different MNIST digits using the linearised optimisation from Deepfool

### 3.1.3 Attack success rate and $p$-norm

It is an open question whether or not there exists a $p$ such that the $p$-norm attack in Formulation 1, provides the highest attack success rate on the dataset. We want to investigate which is the best performing norm in this particular case. Our experiments using the NAG library shows that *success rate is independent of* $p$. 7 shows the results of the success rates of attacking four digits, distinguished by colors, of batch size 1 and varying $p$ from 1 to 100. Albeit, the visualisations of the attack indeed vary drastically, with $p = 1$ minimising the number of pixels attacked to guarantee a successful misclassifcation while as $p$ increases, the attacks are more uniformly distributed. This can be seen in Figure 8. However, as shown in Figure 9, the norm of the perturbation decays exponentially with $p$. Together with $p$ seemingly to be independent of the attack accuracy, this can be seen as a realisation of *Holder's inequality*: $\|x\|_1 \leq \|x\|_p$ for $p > 1$.

## 3.2 Epsilon and batch size

Here we present some experiments regarding Formulation 2 and varying the maximum 2-norm of the perturbation with a fixed batch size of 1. Figure 11 demonstrates the relationship between batch size and the $\|\Delta x\|_2^2$. It seems that increasing the attack batch size makes it more difficult for the optimiser to find a small perturbation that successfully attacks all images in the batch, and thus $\|\Delta x\|_2^2$ increases with batch size. Then as a consequence of a larger perturbation the accuracy also improves.

### 3.2.1 Implementing adversarial ink

The core aim of adversarial ink [10] is to perform attacks by only perturbing the non-background pixels. Although the original paper uses a self-developed optimisation scheme with linearised constraints. During this project, we briefly demonstrate this can be implemented easily within our pipeline with the NAG library, by simply modifying the simple

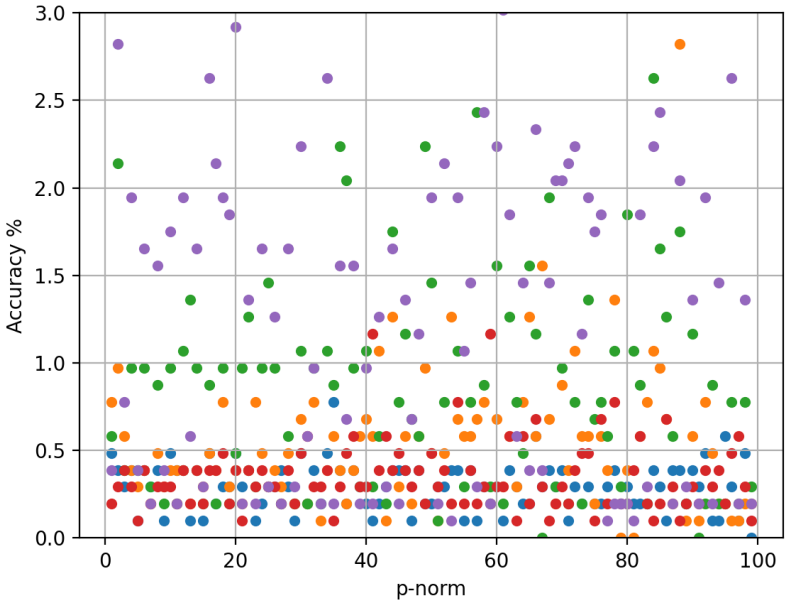


Figure 7: Scatter plot of the success rate over the whole dataset after computing the singular perturbations for four different images: color coded red, blue, green, orange, in different $p$-norms in Formulation 2.1.

bounds of equation (3) and (4) to

$$u_{x_i} = \begin{cases} \max_{j,k} x_j^{(k)} - x_i & x_i \neq \min_{j,k} x_j^{(k)} \\ 0 & \text{otherwise} \end{cases}$$

$$l_{x_i} = \begin{cases} \min_{j,k} x_j^{(k)} - x_i & x_i \neq \min_{j,k} x_j^{(k)} \\ 0 & \text{otherwise} \end{cases}$$

This assumes the backgrounds are of the same and of the minimum value across the whole dataset, which may vary depending on the specific dataset used. Figure 12 shows one working example of this implementation, and the idea that these algorithms with different purposes can be quickly and easily implemented best demonstrates the versatility of using the NAG-library for adversarial attacks.

## 3.3 Fashion MNIST dataset

Fashion-MNIST [14] is a dataset of Zalando's article images, consisting of a training set of $60,000$ examples and a test set of $10,000$ examples. Each example is a $28 \times 28$ grayscale image, associated with a label from 10 classes.

### 3.3.1 One batch-size attack

In this subsection we present different targeted attacks on just one image. Here we solved problem 2.1, i.e. we search the minimal perturbation $\Delta x$ which renders the model inaccurate. The results, shown in Fig 13, are quite promising, as we barely see any modifications to the original images.

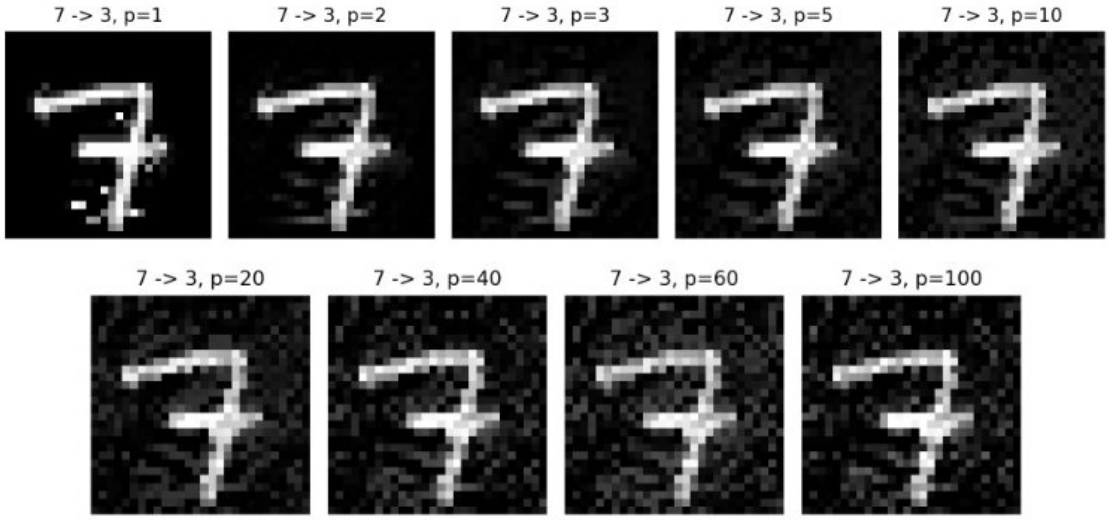


Figure 8: Visual effect on attacking digit $7 \to 3$ in the $p$-norm.

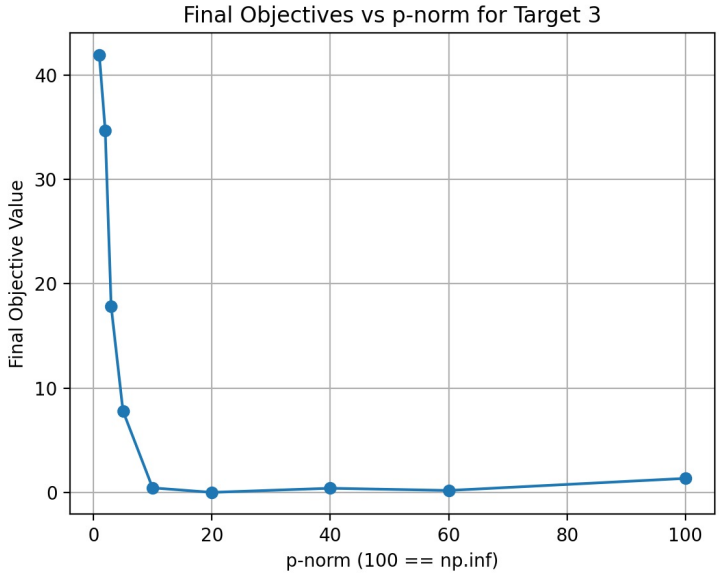


Figure 9: $p$-norm vs Final objective value, $\|\Delta x\|_2^2$. All correspond to attacking the same digit 7 as in figure 8.

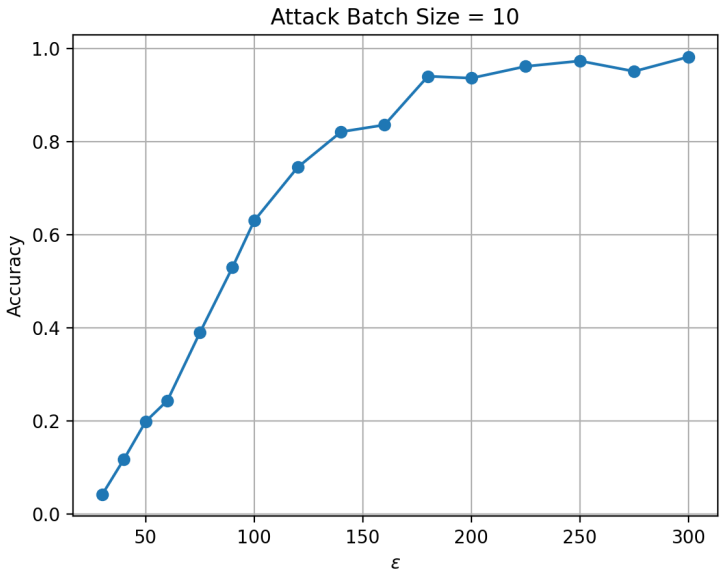


Figure 10: Varying $\epsilon$ with fixed batch size in the 2-norm (formulation 2.2).

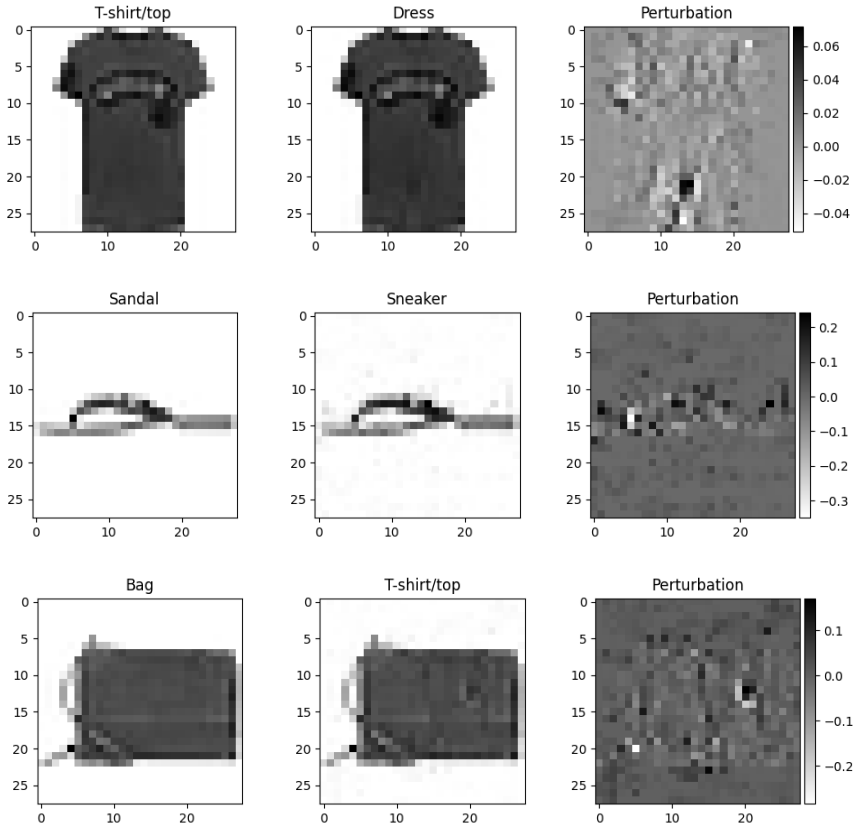


Figure 13: Three different (single) successful attack under the 2-norm, on the left-hand side there is the original image, the title reports the true label, the central plot shows the perturbed image and the new (targeted) label; and the image on the right shows the perturbation.

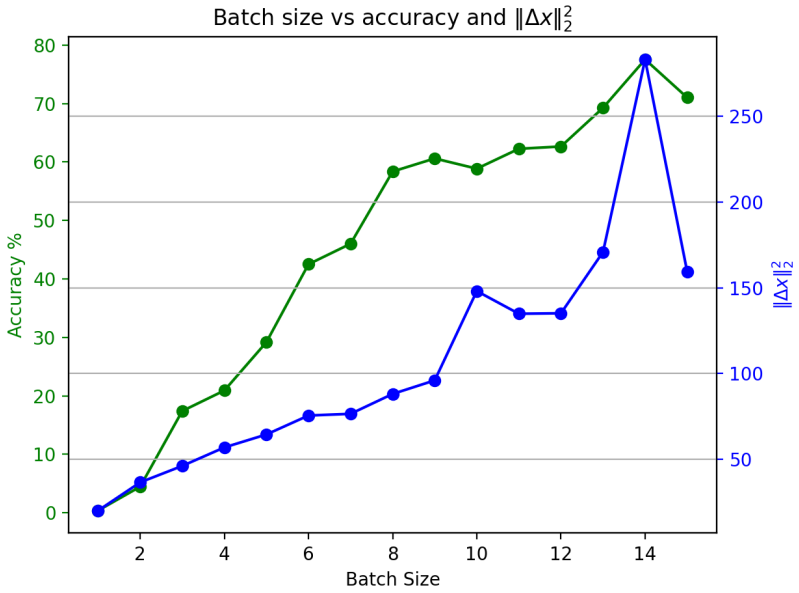


Figure 11: The effect of increasing attack batch size and the resulting accuracy and size of perturbation of the attack.

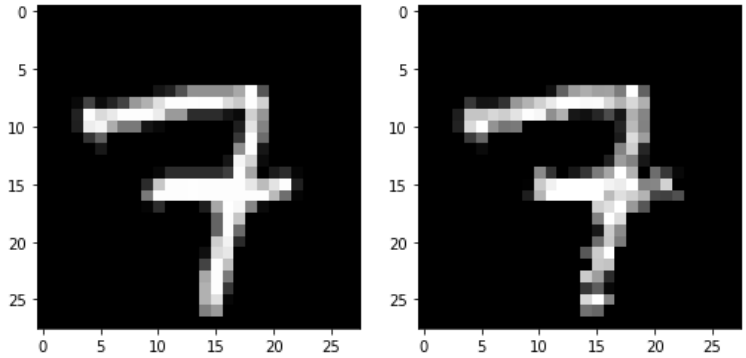

Figure 12: The perturbed image of the adversarial ink implemented with NAG library, which fools the model to classify the original 7 (left) into a 3 (right).

### 3.3.2 Multiple batch-size attack

In this section we address an attack on more than one image at the same time, this is of quite a straightforward implementation thanks to the NAG library. In particular the only change to the previous section is the addition of multiple non-linear constraints to problem 2.1. Each constraint consists in changing the correct label into the targeted one. This will produce only one perturbation which (hopefully) renders the model wrong on the whole set of attacked images.

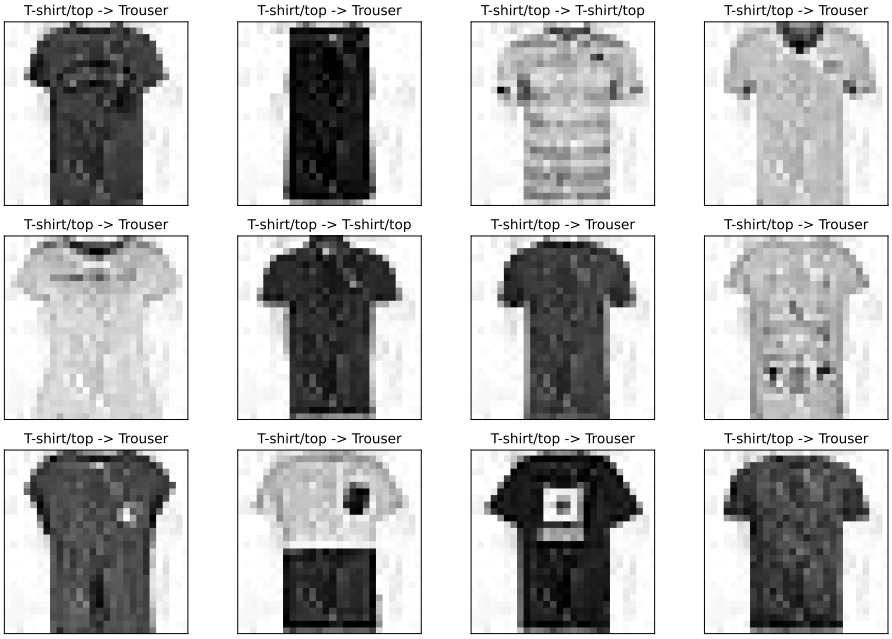


Figure 14: Here we attack simultaneously 12 images, solving formulation 2.2.

As we can see in Fig 14, we were able to compute one perturbation which attacked 10 out of 12 T-shirts and targeted them into trousers. Although the modifications are clearly visible around the originals; this is because attacking simultaneously different images in this dataset is quite hard and $\epsilon$ has to be quite big. The plots are computed with $\epsilon = 10$ (which we remark is the norm of the perturbation).

# 4 Conclusions

In this project we successfully designed numerical optimization algorithms which produced adversarial attacks on image classification Neural Networks. We managed to attack different datasets of black and white images using NAG routines alongside with pytorch. The NAG library gives a huge flexibility through its vast library of efficient ready-to-use optimisation solvers that are ideal for performing adversarial attacks. By utilising their in-library optimisers, adversarial attacks can be performed without the need for linearising the constraints or converting constrained formulation to unconstrained. As demonstrated in section 3.1.2, our linearisation-free optimisation is able to converge towards smaller perturbations than other with-linearisation approaches. Lastly, although this project mainly explored the use of NAG library on image classification tasks, future directions may explore the effectiveness of this framework for other tasks, such as image segmentation or sentiment classifications.

# 5 Feedback on the NAG library

Up until now we have highlighted and praised the multiple qualities of the NAG library. In this section we present a list of suggestions, aiming to help improve the efficiency when interfacing with popular deep learning packages, as well as the general user-friendliness of the NAG library from the Python-side.

- **Lack of Python examples:** Although the NAG documentation provides some useful examples, we realised the number and coverage of these examples can be quite limited for some specific functions, especially in Python. Sometimes it is not obvious which function one should pick, and how may one incorporate these functions into their codes error free. This may be especially problematic for first-time users and users with limited programming experience, and we very much relied on ChatGPT (with paid access to GPT4) to provide an initial semi-working framework, which we then improved upon.

- **Unclear tracebacks:** The Python tracebacks of errors during the executions of scripts may not be always helpful. Here we provide a specific example that was encountered during the project: by setting the returned "inform" argument to a data type unsupported by NAG (for example, torch.tensor) in the objgrd function of (nag-interfaces.library.)opt.handle_solve_ssqp (e04srf), the NagTypeError traceback will not offer any clue about which function was causing this problem. Additionally, this error would not be raised immediately when calling the bugged objgrd function, making debugging extra difficult. The full traceback is attached in Appendix A.
  The tracebacks may also not indicate exactly which variable is problematic, such as in the following case, making debugging an exhaustive search through all variables involved.

```
NagShapeError: has length 1, but must have length 784
```

- **Unsettable arguments in Python but settable in Fortran:** Many functions, for example opt.handle_set_nlnconstr (e04rk), may contain arguments that can be easily set in the Fortran version of NAG library, but not in the Python version, such as nnzgd and ncnln. Therefore, when encountering tracebacks similar to

```
NagValueError:
  On entry, i = 1, irowgd[i-1] = 784 and
  ncnln = 1.
  Constraint: 1 <= irowgd[i-1] <= ncnln.
```

  where the design variable is a vector of size 784, it will cause significant confusion to the users, as ncnln cannot be passed directly as an argument to the function.

- **Unknown reasons for infeasible problem:** It has been confusing to us that the solver sometimes would indicate "the problem is infeasible", when there is no error in the code implementation, and the mathematical formulation looks fine. This only happens when we pass objgrd, congrd, or both to the solver, and it would be helpful if more information can be provided regarding the infeasibility.

- **Vectorisation acceleration for Pytorch:** When using optimisation solvers without providing gradient information, for example setting objgrd to None in opt.handle_solve_ssqp (e04srf) (this action is needed if the solver considers the problem is infeasible based on the provided gradients), the solver will separately call objfun $n$ times, assuming the design variable is of size $n$, which can typically be quite large. If objfun involves the feed-forward or backpropagation of Pytorch models, then the lack of vectorisation (where the batch size is limited to 1) is inefficient. Therefore NAG library may consider to add an option to pass batched inputs to objfun to leverage the acceleration from vectorisation.
  More generally, there is a lack of support for popular deep learning packages, and

transferring data between NAG library with these packages is thus a non-trivial, if not difficult task.

# A Traceback

There is the full traceback when returning an object of unsupported data type for "inform" in objgrd for naginterfaces.library.opt.handle_solve_ssqp (e04srf)

```
Traceback (most recent call last):

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/spyder_kernels/
  py3compat.py:356 in compat_exec
   exec(code, globals, locals)

File ~/UoE/Taught/TasterProject2/Codes/minimise_dx_norm_ssqp.py:254
   x, u, rinfo, stats = opt.handle_solve_ssqp(handle, x_start,

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/
  library/opt.py:31334 in handle_solve_ssqp
   _b_opt.handle_solve_ssqp(

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /opt.py:52897 in handle_solve_ssqp
   utils._handle_cb_exception(excs)

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /utils.py:5838 in _handle_cb_exception
   _raise_exc_info(exc)

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /utils.py:5844 in _raise_exc_info
   raise exc_info[1].with_traceback(exc_info[2])

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /opt.py:52365 in cb_objgrdh
   utils._EngineIntScalarType(

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /utils.py:2364 in __init__
   _EngineScalarType.__init__(self, data, locus, allow_none)

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /utils.py:1763 in __init__
   self._check_scalar_type_compat()

File ~/anaconda3/envs/dl/lib/python3.10/site-packages/naginterfaces/base
  /utils.py:1823 in _check_scalar_type_compat
   raise NagTypeError(

   NagTypeError: must be an instance of ctypes.c_int, ctypes.c_long, int,
    numpy.int32 or numpy.int64
```

There is no information indicating NagTypeError is from the bugged objgrd function.

# References


1. Paszke, A. *et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library* 2019. arXiv: 1912.01703 [cs.LG].
2. NAG. *NAG Library* [Online; accessed 11-March-2024].
3. Adversarial machine learning. *Adversarial machine learning — Wikipedia, The Free Encyclopedia* [Online; accessed 11-March-2024].
4. Higham, C. F. *et al. Deep Learning: An Introduction for Applied Mathematicians* 2018. arXiv: 1801.05894 [math.HO].
5. Szegedy, C. *et al. Intriguing properties of neural networks* 2014. arXiv: 1312.6199 [cs.CV].
6. Kurakin, A. *et al. Adversarial examples in the physical world* 2017. arXiv: 1607.02533 [cs.CV].
7. Drenkow, N. *et al. A Systematic Review of Robustness in Deep Learning for Computer Vision: Mind the gap?* 2022. arXiv: 2112.00639 [cs.CV].
8. Wójcik, M. *et al. Neural Architecture for Online Ensemble Continual Learning* https://arxiv.org/abs/2211.14963. Nov. 2022.
9. Guo, C. *et al.* Gradient-based adversarial attacks against text transformers. *arXiv preprint arXiv:2104.13733* (2021).
10. Beerens, L. *et al.* Adversarial ink: Componentwise backward error attacks on deep learning. *arXiv preprint arXiv:2306.02918* (2023).
11. Moosavi-Dezfooli, S.-M. *et al. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks* in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 2574–2582.
12. Zhang, P. Neural Networks for Classification: A Survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* **30,** 451–462 (Dec. 2000).
13. Inkawhich, N. *Adversarial example generation¶* [Online; accessed 15-January-2024].
14. pytorch-zalando. *Zalando-Fashion MNIST* [Online; accessed 11-March-2024].