Art of singular vectors and universal adversarial perturbations

Valentin Khrulkov
Skolkovo Institute of Science and Technology

Ivan Oseledets
Skolkovo Institute of Science and Technology
Institute of Numerical Mathematics
of the Russian Academy of Sciences

Abstract

Vulnerability of state-of-the-art deep neural networks to adversarial attacks has been attracting a lot of attention recently. In this work we propose a new algorithm for constructing universal adversarial perturbations. Our approach is based on computing the so-called $(p, q)$-singular vectors of the Jacobian matrices of hidden layers of a network. Resulting perturbations present interesting visual patterns and by using a batch of just 64 images we can construct adversarial perturbations with relatively high fooling rate. We also investigate a correlation between the singular values of the Jacobian matrices and the fooling rate of a corresponding singular vector.

1 Introduction

It has been shown recently [Moosavi-Dezfooli et al. 2016] that for many state-of-the-art deep neural networks (DNN) there exist universal adversarial perturbations - single vectors of a small norm which being added to images in a dataset with high probability lead to misclassifying these images. In this work we present an algorithm for constructing the universal adversarial perturbations as the so-called $(p, q)$-singular vectors of the Jacobian matrices of feature maps of a DNN. The $(p, q)$-singular vector is defined as

$$ y = \arg \max_p \frac{||Ay||_q}{||y||_p}. $$

For $p = q = 2$ we obtain the standard singular vector. The universal adversarial perturbations are typically generated with a bound in the $\infty$-norm, which motivates the usage of such singular vectors. To obtain the $(p, q)$-singular vectors we use a modification of the standard power method, which is adapted to arbitrary $p$-norms. The main contributions of our paper are:

- We propose a new algorithm for constructing universal adversarial perturbations, using the generalized power method for computing the $(p, q)$-singular vectors of the Jacobian matrices of the features maps.
- We investigate a correlation between the largest $(p, q)$-singular value and the fooling rate of the generated adversarial examples; this suggests that this singular value can be used as a quantitative measure of the robustness of a given neural network and can be in principle incorporated as the regularizer for the DNN.
- We analyze various properties of the computed adversarial perturbations such as generalization across networks and dependence of the fooling rate on the batch size used for construction of the perturbation.

2 Problem statement

Suppose that we have a standard feed-forward DNN which takes a vector $x$ as the input, and outputs a vector of probabilities $p(x)$ for class labels. Our goal is given parameters $q \geq 1$ and $L > 0$ produce a vector $\epsilon$ such that

$$ \arg \max_p p(x) \neq \arg \max_p p(x + \epsilon), \quad ||\epsilon||_q = L, $$

for as many $x$ in a dataset as possible. Efficiency of the given universal adversarial perturbation $\epsilon$ for the dataset $X$ of the size $N$ is called the fooling rate and is defined as

$$ \frac{|\{x \in X : \arg \max_p p(x) \neq \arg \max_p p(x + \epsilon)\}|}{N}. $$

Let us denote the outputs of the $i$-th hidden layer of the network by $f_i(x)$. Then we have

$$ f_i(x + \epsilon) - f_i(x) \approx J_i(x) \epsilon, $$

where

$$ J_i(x) = \frac{\partial f_i}{\partial x} _x. $$

Thus, for any $q$

$$ ||f_i(x + \epsilon) - f_i(x)||_q \approx ||J_i(x)\epsilon||_q, $$

(1)

and it seems reasonable that if we sufficiently perturb the output of some hidden layer, it will also change the predicted class label. Thus to construct the adversarial perturbation for an individual image $x$ we need to solve

$$ ||J_i(x)\epsilon||_q \rightarrow \max, \quad ||\epsilon||_p = L, $$

(2)

and due to linearity of eq. (2) it is sufficient to solve it for $||\epsilon||_p = 1$. A solution of eq. (2) is called the $(p, q)$-singular vector.
vector of $J_i(x)$ and its computation is a well-known problem (Boyd 1974; Bhaskara and Vijayaraghavan 2011). Even though in some cases e.g. $p = \infty$, $q = \infty$ and $p = \frac{1}{2}$, $q = \frac{1}{2}$, the exact solution of the problem eq. (3) is known (Trefethen and Bau III 1997), it is impossible to construct it in practice, due to extremely large sizes of matrices involved (e.g. $831744 \times 150528$). However, iterative methods for approximating the solution exist and in the next section we present the Power Method algorithm originally developed by Boyd (Boyd 1974), and explain how we use it to construct the universal adversarial perturbations.

3 Generalized power method

Suppose that for some linear map $A$ we are given the matrix-by-vector products of $A$ and $A^\top$. Given parameter $r \geq 1$ we also define a function

$$\psi_r(x) = \text{sign } x |x|^r - 1,$$

which applies to vectors element-wise. As usual for $r \geq 1$ we also define $r'$ such that $\frac{1}{2} + \frac{1}{r'} = 1$. Then, given some initial condition $x$, one can apply the following algorithm to obtain the solution of eq. (2). In the case $p = q = 2$ it becomes the familiar power method for obtaining the largest eigenvalue and eigenvector, applied to the matrix $A^\top A$.

We seek universal adversarial perturbations. Let us introduce a new optimization problem

$$\min_{x \in X} \| J_i(x) \varepsilon \|^q_q \quad \text{max} \quad \| \varepsilon \|^p_p = L, \quad (3)$$

where $X$ denotes the dataset from which we sample images. A solution of eq. (3) uniformly perturbs the output of the $i$-th layer of the DNN, and thus can serve as the universal adversarial perturbation. Note that the problem eq. (3) is exactly equivalent to

$$\min_{x \in X} \| J_i \varepsilon \|^q_q \quad \text{max} \quad \| \varepsilon \|^p_p = L,$$

where $J_i$ is the matrix obtained via stacking $J_i(x)$ vertically for each $x \in X$. To make this optimization problem tractable, we apply the same procedure to some randomly chosen subset of images $X_b \subset X$, obtaining

$$\min_{x \in X_b} \| J_i(x) \varepsilon \|^q_q \quad \text{max} \quad \| \varepsilon \|^p_p = L, \quad (4)$$

and hypothesize that the obtained solution will be a good approximate to the exact solution of eq. (3). We present this approach in more detail in the next section.

Stochastic power method

Let us choose a batch of images $X_b = \{x_1, x_2, \ldots, x_b\}$ from the dataset and fix some layer of the DNN, giving the map $f_i(x)$. Denote sizes of $x$, $f_i(x)$ by $n$ and $m$ correspondingly. Then, using the notation from section 2, we can compute $J_i(x_j) \in \mathbb{R}^{m \times n}$ for each $x_j \in X_b$. Let us now stack this Jacobian matrices vertically thus obtaining the matrix $J_i(X_b)$ of size $bn \times n$:

$$J_i(X_b) = \begin{bmatrix} J_i(x_1) \\ J_i(x_2) \\ \vdots \\ J_i(x_b) \end{bmatrix}. \quad (5)$$

Note that to compute the matrix-by-vector products of $J_i(X_b)$ and $J_i^\top(X_b)$ it suffices to be able to efficiently compute the individual matrix-by-vector product of $J_i(x)$ and $J_i^\top(x)$. We will present an algorithm for that in the next section and for now let us assume that these matrix-by-vector products are given. We can now apply algorithm 1 to the matrix $J_i(X_b)$ obtaining Stochastic Power Method (SPM). Note that in algorithm 2 we could in principle change the initial condition $x$ between iterations of the power method to compute "more general" singular vectors. However in our experiments we discovered that it almost does not affect fooling rate of the generated universal adversarial perturbation.

Algorithm 2 Stochastic Power Method for generating universal adversarial perturbations

1: Inputs: batch of images $X_b = \{x_1, x_2, \ldots, x_b\}$, $f_i(x)$ - fixed hidden layer of the DNN
2: for $x_j \in X_b$ do
3: Construct matrix-by-vector product functions of $J_i(x_j)$ and $J_i^\top(x_j)$
4: Construct matrix-by-vector product functions of $J_i(X_b)$ and $J_i^\top(X_b)$ defined in eq. (5)
5: Run algorithm 1 with desired $p$ and $q$
6: return $\varepsilon$ \quad \text{Universal Adversarial Perturbation}

4 Efficient implementation of matrix-by-vector products

Matrices involved in algorithm 2 for typical DNNs are too large to be formed explicitly. However, using automatic differentiation available in most deep learning packages it is possible to construct matrix-by-vector products which then are evaluated in a fraction of a second. Suppose that we are given an operation $\text{grad}_x f(x)$ which computes the gradient of a scalar function $f(x)$ with respect to the vector variable $x$ at the point $x_0$. Let $f_i(x)$ be some fixed layer of the DNN, such that $x \in \mathbb{R}^n$ and $f_i(x) \in \mathbb{R}^m$, thus $J_i(x) \in \mathbb{R}^{m \times n}$ and for vectors $v_1 \in \mathbb{R}^n$, $v_2 \in \mathbb{R}^m$ we would like to compute $J_i(x) v_1$, $J_i^\top(x) v_2$ at some fixed point $x$. Our approach is based on the trick presented in (Townsend 2017) and is given in algorithm 3.

For a given batch of images this algorithm has to be run only once and thus has $O(1)$ complexity.
In this section we analyze various adversarial perturbations constructed using various layers of various DNNs.

Algorithm 3 Constructing matrix-by-vector product function

1: Inputs: $v_1 \in \mathbb{R}^n$, $v_2 \in \mathbb{R}^n$ - vectors to compute matrix-by-vector products with, $f_i(x)$ - fixed hidden layer of the DNN
2: $J_i^T(x)v_2 \leftarrow \text{grad}_u[\langle v, f_i(x) \rangle](v_2)$
3: $g(v_2) \leftarrow \langle J_i^T(x)v_2, v_1 \rangle$
4: $J_i(x)v_1 \leftarrow \text{grad}_u[g](\theta_m)$
5: return $J_i(x)v_1, J_i^T(x)v_2$

5 Experiments

In this section we analyze various adversarial perturbations constructed using algorithm 2. For testing we used the ILSVRC 2012 validation dataset (Russakovsky et al. 2015) (50000 images).

In our experiments we chose $p = \infty$, $q = 10$ and computed the $(p,q)$ - singular vectors for various layers of VGG-16 and VGG-19 (Simonyan and Zisserman 2014) and ResNet50 (He et al. 2016). $q = 10$ was chosen to smoothen optimization problem and effectively serves as the replacement for $q = \infty$, for which we discovered that the highest fooling rates are achieved. Batch size in algorithm 2 was chosen to be 64 and we used the same 64 images to construct all the adversarial perturbations. Some of the computed singular vectors are presented in figs. 1a to 1c. We observe that computed singular vectors look visually appealing and contain interesting patterns.

We also compute the $(\infty, 10)$-singular values for all the layers of VGG-16, VGG-19 and ResNet50. Results are given in fig. 2. We can note that in general singular values of the layers of ResNet50 are much smaller in magnitude than those of the VGG nets.

Convergence of algorithm 2 is analyzed in fig. 3. We see that relatively low number of iterations is required, and iterative method is much more efficient than explicit computation of the singular vector, which in the case of $(\infty, \infty)$-singular vectors takes $O(nm)$ operations and is intractable for typical sizes of the input and of the hidden layers of the DNN.

5 Fooling rate and singular values

As a next experiment we computed and compared the fooling rate of the perturbation by various computed singular vectors. We choose $\|\varepsilon\|_{\infty} = 10 -$ recall that this can be achieved just by multiplying the computed singular vector by the factor of 10. Results are given in tables 1 to 3.

We see that using just 64 images allowed us to achieve more than 40% fooling rate on the dataset containing 50000 images for all the investigated networks. Similar fooling rates reported by (Moosavi-Dezfooli et al. 2016 - Figure 6) required roughly 1000 images.

As a next experiment we investigate dependence of the achieved fooling rate on the batch size used in algorithm 2. Some of the results are given in fig. 4. Surprisingly, increasing batch size does not significantly affect the fooling rate and by using as few as 16 images it is possible to construct the adversarial perturbations with 56% fooling rate. This might indicate that the singular vector constructed using Stochastic Power Method reasonably well approximates the general optimization problem eq. 3.

6 Conclusions

It appears that higher singular value of the layer does not

1 Pixels in the images from the dataset are normalized to be in [0, 255] range, so by choosing $\|\varepsilon\|_{\infty} = 10$ we make the adversarial perturbations quasi-imperceptible to human eye.
Figure 2: \((p, q)\)-singular values for all the layers of various DNNs. Values \(p = \infty, q = 10\) were used.

| Layer name           | block2_pool | block3_conv1 | block3_conv2 | block3_conv3 |
|----------------------|-------------|--------------|--------------|--------------|
| Singular value       | 1165.74     | 2200.08      | 3146.66      | 6282.64      |
| Fooling rate         | 0.52        | 0.39         | 0.50         | 0.50         |

Table 1: Fooling rates for VGG-16

| Layer name           | conv1   | res3c_branch2a | bn5a_branch2c | activation_8 |
|----------------------|---------|----------------|---------------|--------------|
| Singular value       | 59.69   | 19.21          | 138.81        | 15.55        |
| Fooling rate         | 0.44    | 0.35           | 0.34          | 0.34         |

Table 2: Fooling rates for VGG-19

| Layer name           | block2_pool | block3_conv1 | block3_conv2 | block3_conv3 |
|----------------------|-------------|--------------|--------------|--------------|
| Singular value       | 784.82      | 1274.99      | 1600.77      | 3063.72      |
| Fooling rate         | 0.60        | 0.33         | 0.50         | 0.52         |

Table 3: Fooling rates for ResNet50

Figure 3: Convergence of the \((\infty, 10)\)-singular value. Relative error of the singular value w.r.t iteration number is shown.

Figure 4: Dependence of the fooling rate on the batch size. `block2_pool` layer in VGG-19 was used for the experiment.

necessarily indicate higher fooling rate of the corresponding singular vector. However as shown on fig. 2, singular values of various layers VGG-19 are in general larger than those of VGG-16, and of VGG-16 are in general larger than singular values of ResNet50, which is roughly in correspondence between the maximal fooling rates we obtained for these networks. Moreover, layers closer to the input of the DNN seem to produce better adversarial perturbations, than those closer to the end.

Based on this observation we hypothesize that to defend the DNN against this kind of adversarial attack one can choose some subset \(I\) of the layers (preferably closer to the input) of the DNN and include the term

\[
\sum_{i \in I} \|J_i(X)\|_q^p
\]

in the regularizer, where \(X\) indicates the current learning batch. We plan to analyze this approach in future work.

Finally, we investigate generalization of the perturbations across different networks. For each DNN we have chosen the adversarial perturbation with the highest fooling rate from tables 1 to 3, and tested it against other networks. Results are given in table 4. Surprisingly, in some cases the fooling rate of the adversarial perturbation constructed using other network was higher than that of the network’s own adversarial perturbation.
Table 4: Generalization of the adversarial perturbations across networks. Columns indicate the DNN for which adversarial perturbation was computed, rows indicate on which network it was tested. Adversarial perturbations with highest fooling rates in tables 1 to 3 were chosen.

|       | VGG-16 | VGG-19 | ResNet50 |
|-------|--------|--------|----------|
| VGG-16 | 0.52   | 0.60   | 0.39     |
| VGG-19 | 0.48   | 0.60   | 0.38     |
| ResNet50 | 0.41 | 0.47   | 0.44     |

7 Analysis of perturbations

In this section we analyze some of the misclassified images. As the adversarial perturbation for each network we chose the one with the highest fooling rate (as before, selected from the ones given in tables 1 to 3). For each perturbation $\varepsilon$ we also set $||\varepsilon||_{\infty} = 10$. For analysis we picked some of those images which were misclassified after adversarial attack by all the networks (see fig. 5).

Figure 5: Examples of images misclassified after adversarial attack. Predicted classes are given in tables 5 to 7.

Figure 6: Top-5 probabilities given by the DNN w.r.t $\infty$-norm of the universal adversarial perturbation. Both tests were run for image 3 from fig. 5. Universal adversarial perturbation with highest fooling rate in tables 1 and 2 for each DNN was chosen.

(a) VGG-19
(b) VGG-16

We note that the top-1 class probability for images after adversarial attack is relatively low in the most cases. We test this behavior by computing the top-5 probabilities for several values of the $\infty$-norm of the adversarial perturbation. Results are given in figs. 6a and 6b. In both experiments the top-1 probability decreases significantly and becomes roughly equal to the top-2 probability, and in second case to the top-3. Similar behavior was noticed in some of the cases when the adversarial example failed to fool the DNN — top-1 probability still has decreased significantly. We hypothesize that this might indicate that adversarial perturbations constructed as the $(p, q)$-singular vectors push an image away from the decision boundary.

Conclusions and related work

We proposed a new algorithm for generating the universal adversarial perturbations, and tested it on several state-of-the-art deep neural networks. We showed that using even small number of images results in the adversarial perturbations with high fooling rate. We investigated several properties of the constructed perturbations, such as dependence of its efficiency on the corresponding singular value and on the batch size. We also showed that constructed perturbations generalize well to other networks. Theoretical analysis of constructed perturbations and defense against this kind of adversarial attack is a problem of future research.

Concept of adversarial examples was discussed in (Goodfellow, Shlens, and Szegedy 2014). Algorithm proposed there allows one to construct adversarial perturbations for individual images. Another approach was presented in (Moosavi-Dezfooli, Fawzi, and Frossard 2016). An algorithm for constructing universal adversarial perturbations was introduced in (Moosavi-Dezfooli et al. 2016). Efficiency of various perturbations is analyzed in (Fawzi, Moosavi-Dezfooli, and Frossard 2016). Various approaches to defense against adversarial attacks has been proposed. First attempts has been made in the original paper (Goodfellow, Shlens, and Szegedy 2014). Recently, approach based on controlling the Lipschitz constant of the network was suggested (Cisse et al. 2017). For testing various kinds of adversarial attacks Python library was introduced in (Papernot et al. 2016).

Acknowledgements

This study was supported by the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001), by RFBR grants 16-31-60095-mol-a-dk, 16-31-00372-mol-a and by Skoltech NGP program.

References

[Bhaskara and Vijayaraghavan 2011] Bhaskara, A., and Vijayaraghavan, A. 2011. Approximating matrix p-norms. In Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms, 497–511. SIAM.
| image_1   | image_2   | image_3   | image_4   |
|----------|----------|----------|----------|
| $p(x)$   | mashed_potato 53.9% | pole 37.6% | fountain 55.0% | goblet 12.1% |
| $p(x + \epsilon)$ | head_cabbage 29.4% | rubber_eraser 39.3% | carousel 61.4% | bucket 36.6% |

Table 5: VGG-16

| image_1   | image_2   | image_3   | image_4   |
|----------|----------|----------|----------|
| $p(x)$   | mashed_potato 68.3% | flagpole 37.4% | fountain 74.5% | coffee_mug 23.1% |
| $p(x + \epsilon)$ | flatworm 26.5% | letter_opener 19.9% | pillow 20.6% | candle 40.3% |

Table 6: VGG-19

| image_1   | image_2   | image_3   | image_4   |
|----------|----------|----------|----------|
| $p(x)$   | mashed_potato 94.2% | totem_pole 43.1% | flagpole 35.3% | chocolate_sauce 22.0% |
| $p(x + \epsilon)$ | stole 21.7% | fountain_pen 27.6% | monitor 9.76% | goblet 40.3% |

Table 7: ResNet50

[Boyd 1974] Boyd, D. W. 1974. The power method for $l_p$ norms. *Linear Algebra and its Applications* 9:95–101.
[Cisse et al. 2017] Cisse, M.; Bojanowski, P.; Grave, E.; Dauphin, Y.; and Usunier, N. 2017. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, 854–863.
[Fawzi, Moosavi-Dezfooli, and Frossard 2016] Fawzi, A.; Moosavi-Dezfooli, S.-M.; and Frossard, P. 2016. Robustness of classifiers: from adversarial to random noise. In *Advances in Neural Information Processing Systems*, 1632–1640.
[Goodfellow, Shlens, and Szegedy 2014] Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.
[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.
[Moosavi-Dezfooli et al. 2016] Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*.
[Moosavi-Dezfooli, Fawzi, and Frossard 2016] Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2574–2582.
[Papernot et al. 2016] Papernot, N.; Goodfellow, I.; and McDaniel, P. 2016. *cleverhans v1.0.0: an adversarial machine learning library*. *arXiv preprint arXiv:1610.00768*.
[Russakovsky et al. 2015] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.
[Simonyan and Zisserman 2014] Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
[Townsend 2017] Townsend, J. 2017. A new trick for calculating jacobian vector products.
[Trefethen and Bau III 1997] Trefethen, L. N., and Bau III, D. 1997. *Numerical linear algebra*, volume 50. Siam.