ITAT 2016 Proceedings, CEUR Workshop Proceedings Vol. 1649, pp. 187–194
http://ceur-ws.org/Vol-1649, Series ISSN 1613-0073, © 2016 P. Vidnerová, R. Neruda

ITAT

Vulnerability of machine learning models to adversarial examples

Petra Vidnerová, Roman Neruda

Institute of Computer Science, Academy of Sciences of the Czech Republic
petra@cs.cas.cz

Abstract: We propose a genetic algorithm for generating adversarial examples for machine learning models. Such approach is able to find adversarial examples without the access to model's parameters. Different models are tested, including both deep and shallow neural networks architectures. We show that RBF networks and SVMs with Gaussian kernels tend to be rather robust and not prone to misclassification of adversarial examples.

1 Introduction

Deep networks and convolutional neural networks enjoy high interest nowadays. They have become the state-of-art methods in many fields of machine learning, and have been applied to various problems, including image recognition, speech recognition, and natural language processing [5].

In [12] a counter-intuitive property of deep networks is described. It relates to the stability of a neural network with respect to small perturbation of their inputs. The paper shows that applying an imperceptible non-random perturbation to an input image, it is possible to arbitrarily change the network prediction. These perturbations are found by optimizing the input to maximize the prediction error. Such perturbed examples are known as *adversarial examples*. On some datasets, such as ImageNet, the adversarial examples were so close to the original examples that the differences were indistinguishable to the human eye.

Paper [4] suggests that it is the linear behaviour in high-dimensional spaces what is sufficient to cause adversarial examples (for example, a linear classifier exhibits this behaviour too). They designed a fast method of generating adversarial examples (adding small vector in the direction of the sign of the derivation) and showed that adding these examples to the training set further improves the generalization of the model. In [4], in addition, the authors state that adversarial examples are relatively robust, and they generalize between neural networks with varied number of layers, activations, or trained on different subsets of the training data. In other words, if we use one neural network to generate a set of adversarial examples, these examples are also misclassified by another neural network even when it was trained with different hyperparameters, or when it was trained on a different set of examples. Another results of fooling deep and convolutional networks can be found in [10].

This paper examines a vulnerability to adversarial examples throughout variety of machine learning methods. We propose a genetic algorithm for generating adversarial examples. Though the evolution is slower than techniques described in [12, 4], it enables us to obtain adversarial examples even without the access to model's weights. The only thing we need is to be able to query the network to classify a given example. From this point of view, the misclassification of adversarial examples represent a security flaw.

This paper is organized as follows. Section 2 brings a brief overview of machine learning models considered in this paper. Section 3 describes the proposed genetic algorithm. Section 4 describes the results of our experiments. Finally, Section 5 concludes our paper.

2 Deep and Shallow Architectures

2.1 Deep and Convolutional Networks

Deep neural networks are feedforward neural networks with multiple hidden layers between the input and output layer. The layers typically have different units depending on the task at hand. Among the units, there are traditional perceptrons, where each unit (neuron) realizes a nonlinear function, such as the *sigmoid* function: $y(z) = tanh(z)$ or $y(z) = \frac{1}{1+e^{-z}}$. Another alternative to the perceptron is the rectified linear unit (*ReLU*): $y(z) = max(0,z)$. Like the sigmoid neurons, rectified linear units can be used to compute any function, and they can be trained using algorithms such as back-propagation and stochastic gradient descent.

Convolutional layers contain the so called *convolutional units* that take advantage of the grid-like structure of the inputs, such as in the case of 2-D bitmap images, time series, etc. Convolutional units perform a simple discrete convolution operation, which – for 2-D data – can be represented by a matrix multiplication. Usually, to deal with large data (such as large images), the convolution is applied multiple times by sliding a small window over the data. The convolutional units are typically used to extract some features from the data, and they are often used together with the so-called *max pooling* layers that perform an input reduction by selecting one of many inputs, typically the one with maximal value. Thus, the overall architecture of a deep network for image classification tasks resembles a pyramid with smaller number of units in higher layers of the networks.

For the output layer, mainly for classification tasks, the *softmax* function: $y(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ is often used. It has

the advantage that the output values can be interpreted as probabilities of individual classes.

Networks with at least one convolutional layer are called *convolutional neural networks (CNN)*, while networks with all hidden layers consisting of perceptrons are called *multi-layer perceptrons (MLP)*.

2.2 RBF networks and Kernel Methods

The history of radial basis function (RBF) networks can be traced back to the 1980s, particularly to the study of interpolation problems in numerical analysis [8]. The RBF network [3] is a feedforward network with one hidden layer realizing the basis functions and linear output layer. It represents an alternative to classical models, such as multilayer perceptrons. There is variety of learning methods for RBF networks [9].

In 1990s, a family of machine learning algorithms, known as kernel methods, became very popular. They have been applied to a number of real-world problems, and they are still considered to be state-of-the-art methods in various domains [14].

Based on theoretical results on kernel approximation, the popular support vector machine (SVM) [2, 13] algorithm was developed. Its architecture is similar to RBF – one hidden layer of kernel units and a linear output layer. The learning algorithm is different, based on search for a separating hyperplane with the highest margin. Common kernel functions used for SVM learning are linear $\langle x,x'\rangle$, polynomial $(\gamma\langle x,x'\rangle+r)^d$, Gaussian $\exp(-\gamma|x-x'|^2)$, and sigmoid $\tanh(\gamma\langle x,x'\rangle+r)$.

Recently, due to popularity of deep architectures, such models with only one hidden layer are often referred to as *shallow models*.

3 Genetic Algorithms

To obtain an adversarial example for the trained machine learning model (such as a neural network), we need to optimize the input image with respect to network output. For this task we employ genetic algorithms (GA). GA represent a robust optimization method working with the whole population of feasible solutions [7]. The population evolves using operators of selection, mutation, and crossover. Both the machine learning model and the target output are fixed during the optimization.

Each individual represents one possible input vector, i.e. one image encoded as a vector of pixel values:

$$I = \{i_1, i_2, \ldots, i_N\},$$

where $i_i \in <0,1>$ are levels of grey, and N is the size of a flatten image. (For the sake of simplicity, we consider only greyscale images in this paper, but it can be seen that the same principle can be used for RGB images as well.)

The crossover operator performs a classical two-point crossover. The mutation introduces a small change to some pixels. With the probability p_{mutate_pixel} each pixel is changed:

$$i_i = i_i + r,$$

where r is drawn from Gaussian distribution. As a selection, the tournament selection with tournament size 3 is used.

The fitness function should reflect the following two criteria:

1. the individual should resemble the target image
2. if we evaluate the individual by our machine learning model, we aim to obtain a prescribed target output (i.e., misclassify it).

Thus, in our case, a fitness function is defined as: $f(I) = -(0.5 * cdist(I, target\_image) + 0.5 * cdist(model(I), target\_answer))$, where $cdist$ is a Euclidean distance.

4 Experimental Results

The goal of our experiments is to test various machine learning models and their vulnerability to adversarial examples.

4.1 Overview of models

As a representative of deep models we use two deep architectures – an MLP network with rectified linear units (ReLU), and a CNN. The MLP used in our experiments consist of three fully connected layers. Two hidden layers have 512 ReLUs each, using dropout; the output layer has 10 softmax units. The CNN has two convolutional layers with 32 filters and ReLUs each, a max pooling layer, a fully connected layer of 128 ReLUs, and a fully connected output softmax layer. In addition to these two models, we also used an ensemble of 10 MLPs. All models were trained using the KERAS library [1].

Shallow networks in our experiments are represented by an RBF network with 1000 Gaussian units, and SVM models with Gaussian kernel (SVM-gauss), polynomial kernel of grade 2 and 4 (SVM-poly2 and SVM-poly4), sigmoidal kernel (SVM-sigmoid), and linear kernel (SVM-linear). SVMs were trained using the SCIKIT library [11], Grid search and crossvalidation techniques were used to tune hyper-parameters. For RBF networks, we used our own implementation. Overview of train and test accuracies can be found in Tab. 1.

4.2 Experimental setup

The well known MNIST data set [6] was used. It contains 70 000 images of hand written digits, 28×28 pixel each. 60 000 are used for training, 10 000 for testing. The genetic algorithm was run with 50 individuals, for 10 000

	RBF	MLP	CNN	SVM-gauss	SVM-poly2	SVM-poly4	SVM-sigmoid	SVM-linear
Train	0.96	1.00	1.00	0.99	1.00	0.99	0.87	0.95
Test	0.96	0.98	0.99	0.98	0.98	0.98	0.88	0.94

Table 1: Overview of accuracies on train and test sets.

generations, with crossover probability set to 0.6, and mutation probability set to 0.1. The GA was run 9 times for each model to find adversarial examples that resemble 9 different images (training samples from the beginning of training set). All images were optimized to be classified as zero.

4.3 Results

Figures 1 and 2 show two selected cases from our set of experiments. For example, the first set of images shows a particular image of digit five from the training set, and best evolved individuals from the corresponding runs of GA for individual models.

In Tables 2–5, the outputs of individual models are listed. In Tab. 2 and 4, we show output vectors for training sample of digit five and four, respectively. In Tab. 3 and 5, we show output vectors for adversarial examples from Fig. 1 and 2, respectively.

For this case, the adversarial examples were found for MLP, CNN, ensemble of MLPs, SVM-poly2, and SVM-sigmoid. For RBF network, SVM-gauss, SVM-poly4, and SVM-linear, the GA was not able to find image that resembles the digit 5 and at the same time it is classified as zero.

If we look on a vulnerability of individual models over all 9 GA runs we can conclude the following:

- MLP, CNN, ensemble of MLPs, and SVM-sigmoid were always misclassifying the best individuals;
- RBF network, SVM-gauss, and SVM-linear; never misclassified, i.e. the genetic algorithm was not able to find adversarial example for these models;
- SVM-poly2 and SVM-poly4 were resistant to finding adversarial examples in 2 and 5 cases, respectively.

Fig. 3 and 4 deal with the generalization of adversarial examples over different models. For each adversarial example the figure lists the output vectors of all models. In the case of a digit 3, the adversarial example evolved for MLP is also misclassified by an ensemble of MLPs, and vice versa. Both examples are misclassified by SVM-sigmoid. However, adversarial example for the SVM-sigmoid is misclassified only by the SVM-linear model. Adversarial example for SVM-poly2 is misclassified also with other SVMs, except the SVM-gauss model.

In general, it often happens that adversarial example evolved for one model is misclassified by some of the other models (see Tab. 6 and 7). There are some general trends:

- adversarial example evolved for CNN was never misclassified by other models, and CNN never misclassified other adversarial examples than those evolved for the CNN;
- adversarial examples evolved for MLP are misclassified also by ensemble of MLPs (all cases except two) and adversarial examples evolved for ensemble of MLPs are misclassified by MLP (all cases);
- adversarial examples evolved for the SVM-sigmoid model are misclassified by SVM-linear (all cases except two);
- adversarial examples for the SVM-poly2 model are often (6 cases) misclassified by other SVMs (SVM-poly4, SVM-sigmoid, SVM-linear), and in 4 cases also by the SVM-gauss. In three cases it was also misclassified by MLP and ensemble of MLPs, in one case, the adversarial example for SVM-poly2 is misclassified by all models but CNN (however, this example is quite noisy);
- adversarial example for the SVM-poly4 model is in two cases misclassified by all models but CNN, in different case it is misclassified by all but the CNN and RBF models, and in one case by all but CNN, RBF, and SVM-gauss models;
- RBF network, SVM-gauss, and SVM-linear were resistant to adversarial examples by genetic algorithm, however they sometimes misclassify adversarial examples of other models. These examples are already quite noisy, however by human they would still be classified correctly.

5 Conclusion

We proposed a genetic algorithm for generating adversarial examples for machine learning models. Our experiment show that many machine models suffer from vulnerability to adversarial examples, i.e. examples designed to be misclassified. Some models are quite resistant to such behaviour, namely models with local units – RBF networks and SVMs with Gaussian kernels. It seems that it is the local behaviour of units that prevents the models from being fooled.

Adversarial examples evolved for one model are often misclassified also by some of other models, as was elaborated in the experiments.

Acknowledgements

This work was partially supported by the Czech Grant Agency grant 15-18108S, and institutional support of the Institute of Computer Science RVO 67985807.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.04	-0.07	0.08	0.24	-0.04	**0.73**	-0.04	0.21	0.03	-0.18
MLP	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00
SVM-poly2	0.00	0.00	0.00	0.02	0.00	**0.98**	0.00	0.00	0.00	0.00
SVM-poly4	0.00	0.00	0.00	0.02	0.00	**0.98**	0.00	0.00	0.00	0.00
SVM-sigmoid	0.01	0.00	0.03	0.32	0.00	**0.61**	0.00	0.01	0.01	0.01
SVM-linear	0.00	0.00	0.01	0.10	0.00	**0.89**	0.00	0.00	0.00	0.00

Table 2: Evaluation of the target digit five (see Fig. 1) by individual models.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.21	-0.05	0.09	0.23	-0.04	**0.51**	-0.05	0.17	0.07	-0.09
MLP	**0.98**	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
CNN	**0.95**	0.00	0.01	0.01	0.00	0.02	0.00	0.00	0.01	0.00
ENS	**0.98**	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.01	0.39	0.00	**0.59**	0.00	0.00	0.00	0.00
SVM-poly	**0.88**	0.00	0.02	0.02	0.00	0.07	0.00	0.00	0.00	0.00
SVM-poly4	0.01	0.01	0.14	0.29	0.01	**0.50**	0.01	0.01	0.02	0.01
SVM-sigmoid	**0.82**	0.00	0.03	0.05	0.00	0.08	0.00	0.01	0.01	0.01
SVM-linear	0.02	0.03	0.21	0.21	0.02	**0.33**	0.03	0.04	0.05	0.05

Table 3: Evaluation of adversarial digit five (from Fig. 1) by individual models.

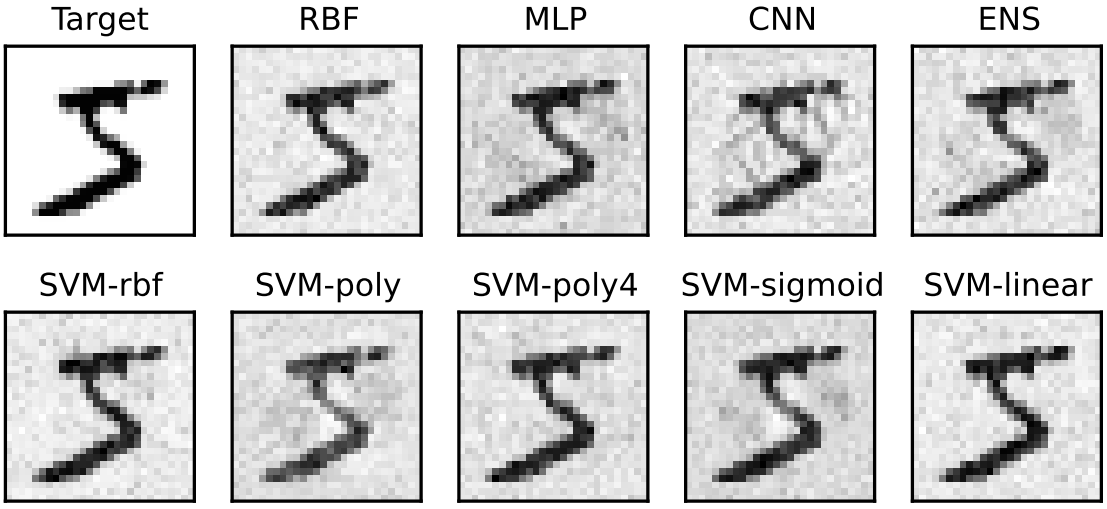


Figure 1: Best individuals evolved for individual models and digit five. The first 'Target' image is the digit from the training set, than follows adversarial examples evolved for individual models.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	-0.06	0.07	0.16	0.06	**0.67**	0.00	0.01	0.15	-0.01	-0.05
MLP	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.00
SVM-poly2	0.00	0.00	0.00	0.00	**0.97**	0.00	0.00	0.01	0.00	0.00
SVM-poly4	0.00	0.00	0.00	0.01	**0.96**	0.00	0.00	0.01	0.00	0.00
SVM-sigmoid	0.00	0.00	0.01	0.02	**0.94**	0.00	0.00	0.01	0.00	0.01
SVM-linear	0.00	0.01	0.05	0.04	**0.84**	0.00	0.01	0.04	0.00	0.02

Table 4: Evaluation of the target digit four (see Fig. 2) by individual models.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.06	0.08	0.14	0.08	**0.49**	-0.01	0.03	0.14	0.02	0.02
MLP	**0.96**	0.00	0.01	0.00	0.02	0.00	0.00	0.01	0.00	0.01
CNN	**0.89**	0.00	0.02	0.00	0.05	0.00	0.00	0.01	0.02	0.00
ENS	**0.96**	0.00	0.01	0.00	0.02	0.00	0.00	0.00	0.00	0.01
SVM-gauss	0.01	0.01	0.07	0.12	**0.50**	0.03	0.01	0.06	0.09	0.08
SVM-poly	**0.75**	0.00	0.03	0.01	0.09	0.00	0.01	0.04	0.03	0.04
SVM-poly4	**0.71**	0.01	0.04	0.01	0.11	0.02	0.02	0.04	0.02	0.02
SVM-sigmoid	**0.84**	0.00	0.02	0.03	0.03	0.02	0.01	0.01	0.00	0.03
SVM-linear	0.02	0.02	0.18	0.18	**0.26**	0.03	0.05	0.11	0.03	0.11

Table 5: Evaluation of adversarial digit four (from Fig. 2) by individual models.

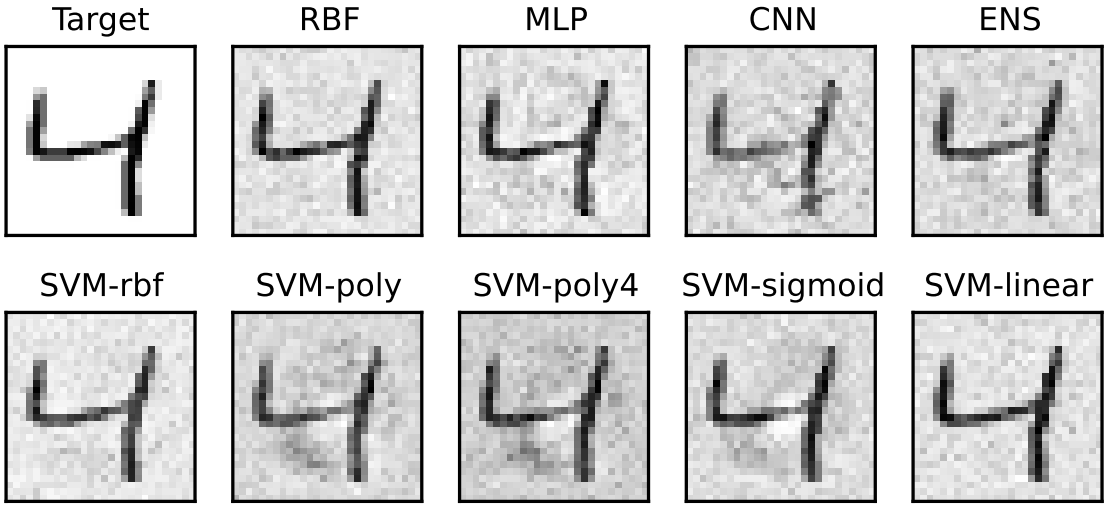
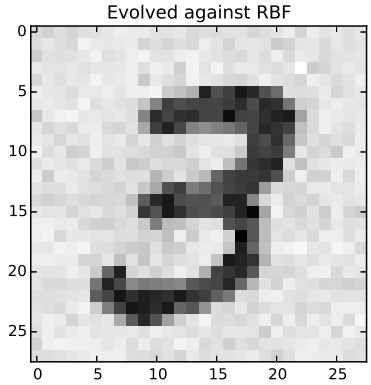
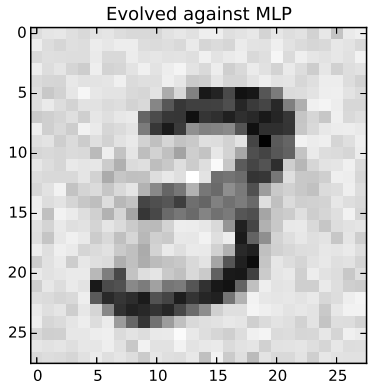


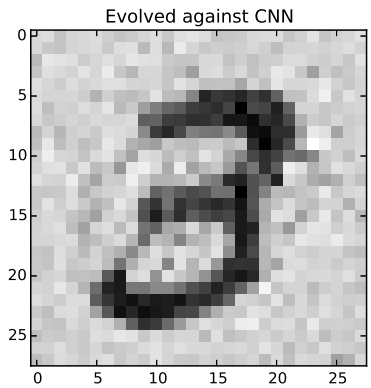
Figure 2: Best individuals evolved for individual models and digit four. The first 'Target' image is the digit from the training set, than follows adversarial examples evolved for individual models.



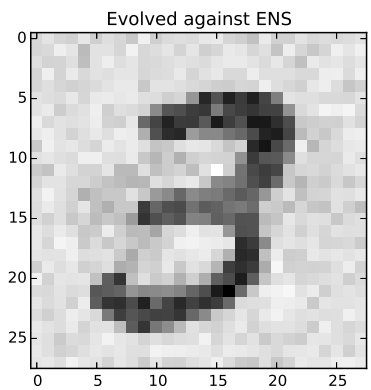
	0	1	2	3	4	5	6	7	8	9
RBF	0.16	0.06	0.12	**0.79**	0.01	-0.02	-0.06	-0.00	0.02	0.03
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly4	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-sigmoid	0.04	0.00	0.02	**0.79**	0.00	0.06	0.00	0.01	0.05	0.02
SVM-linear	0.00	0.00	0.00	**0.96**	0.00	0.00	0.00	0.00	0.03	0.00



	0	1	2	3	4	5	6	7	8	9
RBF	0.30	0.04	0.17	**0.75**	0.02	-0.03	-0.04	-0.01	-0.07	-0.00
MLP	**0.96**	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.01	0.01
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	**0.86**	0.00	0.01	0.04	0.00	0.00	0.00	0.00	0.01	0.08
SVM-gauss	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.00	0.01
SVM-poly	0.04	0.00	0.01	**0.91**	0.00	0.00	0.00	0.00	0.02	0.02
SVM-poly4	0.03	0.00	0.01	**0.93**	0.00	0.00	0.00	0.00	0.01	0.01
SVM-sigmoid	**0.49**	0.00	0.03	0.30	0.00	0.04	0.00	0.01	0.10	0.02
SVM-linear	0.25	0.02	0.10	**0.30**	0.02	0.05	0.02	0.03	0.18	0.06

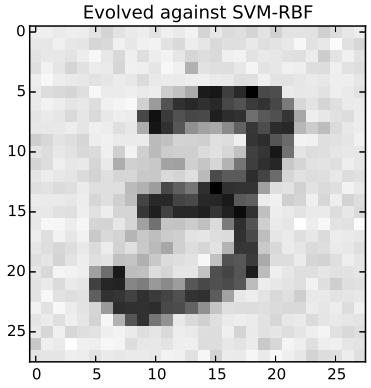


	0	1	2	3	4	5	6	7	8	9
RBF	0.12	0.05	0.15	**0.89**	0.01	-0.18	-0.02	0.02	0.10	-0.03
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	**0.94**	0.00	0.02	0.01	0.00	0.00	0.00	0.00	0.03	0.00
ENS	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly4	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-sigmoid	0.04	0.00	0.02	**0.86**	0.00	0.03	0.00	0.00	0.04	0.01
SVM-linear	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.00	0.00

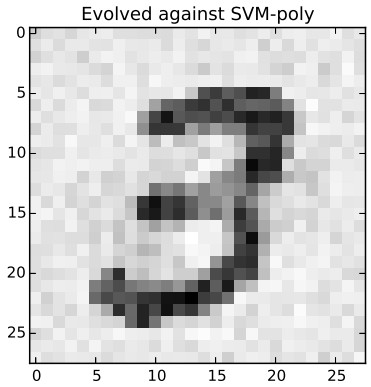


	0	1	2	3	4	5	6	7	8	9
RBF	0.30	0.05	0.18	**0.76**	-0.01	-0.06	-0.04	-0.03	-0.05	-0.00
MLP	**0.83**	0.00	0.05	0.06	0.00	0.00	0.00	0.00	0.05	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	**0.96**	0.00	0.01	0.02	0.00	0.00	0.00	0.00	0.01	0.00
SVM-gauss	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.00	0.01
SVM-poly	0.02	0.00	0.01	**0.94**	0.00	0.00	0.00	0.00	0.01	0.02
SVM-poly4	0.01	0.00	0.00	**0.96**	0.00	0.00	0.00	0.00	0.01	0.01
SVM-sigmoid	**0.40**	0.00	0.03	0.35	0.01	0.06	0.00	0.01	0.11	0.02
SVM-linear	0.19	0.01	0.06	**0.50**	0.01	0.05	0.01	0.02	0.11	0.04

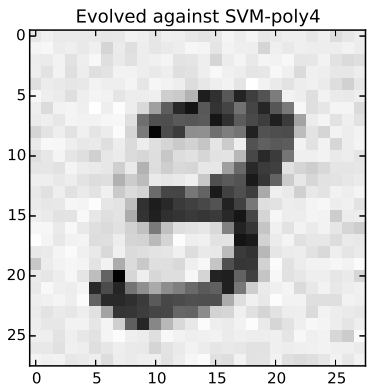
Figure 3: Model outputs for individual adversarial examples.



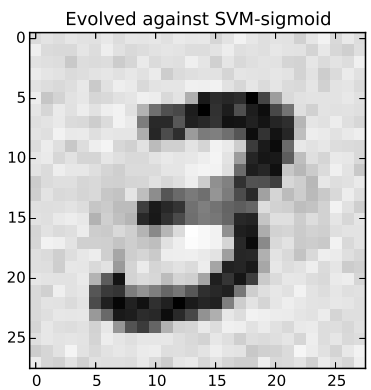
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.06	0.01	0.15	**0.74**	-0.00	-0.03	-0.04	-0.01	0.26	0.05
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.01	0.00
SVM-gauss	0.00	0.00	0.00	**0.90**	0.00	0.00	0.00	0.00	0.10	0.00
SVM-poly	0.00	0.00	0.00	**0.50**	0.00	0.00	0.00	0.00	**0.50**	0.00
SVM-poly4	0.00	0.00	0.00	**0.60**	0.00	0.00	0.00	0.00	0.40	0.00
SVM-sigmoid	0.03	0.00	0.03	**0.63**	0.00	0.09	0.00	0.01	0.19	0.02
SVM-linear	0.00	0.00	0.00	0.36	0.00	0.00	0.00	0.00	**0.63**	0.00



	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.32	0.02	0.17	**0.86**	-0.01	-0.09	-0.09	-0.03	-0.12	0.01
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly	**0.87**	0.00	0.02	0.04	0.00	0.00	0.00	0.00	0.04	0.02
SVM-poly4	**0.38**	0.01	0.11	0.23	0.01	0.02	0.01	0.02	0.15	0.04
SVM-sigmoid	**0.55**	0.01	0.04	0.19	0.01	0.05	0.01	0.01	0.13	0.02
SVM-linear	**0.71**	0.01	0.02	0.06	0.01	0.02	0.01	0.01	0.15	0.01



	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.07	0.02	0.12	**0.84**	0.04	-0.07	-0.07	-0.01	0.10	0.06
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.01	0.00
SVM-poly	0.00	0.00	0.00	**0.57**	0.00	0.00	0.00	0.00	0.41	0.03
SVM-poly4	0.00	0.00	0.00	**0.67**	0.00	0.00	0.00	0.00	0.31	0.03
SVM-sigmoid	0.04	0.00	0.02	**0.73**	0.01	0.07	0.00	0.01	0.08	0.02
SVM-linear	0.01	0.00	0.00	**0.58**	0.00	0.00	0.00	0.00	0.39	0.01



	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.30	0.04	0.22	**0.94**	-0.01	-0.14	-0.10	0.02	-0.21	-0.02
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-gauss	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly	0.06	0.00	0.03	**0.84**	0.00	0.01	0.00	0.00	0.03	0.02
SVM-poly4	0.04	0.00	0.01	**0.93**	0.00	0.00	0.00	0.00	0.01	0.01
SVM-sigmoid	**0.74**	0.00	0.02	0.12	0.00	0.02	0.00	0.00	0.08	0.01
SVM-linear	**0.41**	0.02	0.06	0.18	0.02	0.05	0.02	0.03	0.17	0.05

Figure 4: Model outputs for individual adversarial examples.

Evolved for	Also misclassified by
Example 1:	*digit 5*
MLP	—
ensemble	MLP
CNN	—
SVM-poly2	SVM-poly4, SMV-sigmoid, SVM-linear
SVM-sigmoid	—
Example 3:	*digit 4*
MLP	ensemble
ensemble	MLP
CNN	—
SVM-poly2	ensemble, MLP, SVM-gauss, SVM-poly4, SVM-sigmoid, SVM-linear
SVM-poly4	RBF, ensemble, MLP, SVM-gauss, SVM-poly4, SVM-sigmoid, SVM-linear
SVM-sigmoid	SVM-linear
Example 4:	*digit 1*
MLP	ensemble
ensemble	MLP
CNN	—
SVM-poly2	SVM-gauss, SVM-poly4, SVM-sigmoid, SVM-linear
SVM-sigmoid	SVM-linear
Example 5:	*digit 9*
MLP	ensemble
ensemble	MLP
CNN	—
SVM-poly2	ensemble, MLP, SVM-poly2 SVM-poly4, SVM-sigmoid, SVM-linear
SVM-poly4	all except CNN
SVM-sigmoid	SVM-linear
Example 6:	*digit 2*
MLP	—
ensemble	MLP
CNN	—
SVM-poly4	MLP, ensemble, SVM-poly2, SVM-sigmoid, SVM-linear
SVM-sigmoid	SVM-linear
Example 7:	*digit 1*
MLP	ensemble
ensemble	MLP
CNN	—
SVM-sigmoid	—

Table 6: Generalization of adversarial examples.

Evolved for	Also misclassified by
Example 8:	*digit 3*
MLP	ensemble, SVM-sigmoid
ensemble	MLP, SVM-sigmoid
CNN	—
SVM-poly2	SVM-poly4, SVM-sigmoid, SVM-linear
SVM-sigmoid	SVM-linear
Example 9:	*digit 1*
MLP	ensemble
ensemble	MLP
CNN	—
SVM-poly2	all except CNN
SVM-sigmoid	MLP, ensemble, SVM-gauss, SVM-poly2, SVM-poly4, SVM-linear
Example 10:	*digit 4*
MLP	ensemble
ensemble	MLP
CNN	—
SVM-poly2	SVM-poly4, SVM-linear
SVM-poly4	MLP, ensemble, SVM-gauss, SVM-poly2, SVM-sigmoid, SVM-linear
SVM-sigmoid	SVM-linear

Table 7: Generalization of adversarial examples.

References

[1] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[2] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[3] F. Girosi, M. Jones, and T. Poggio. Regularization theory and Neural Networks architectures. *Neural Computation*, 2:219–269, 7 1995.

[4] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014. arXiv:1412.6572.

[5] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[6] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 2012.

[7] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[8] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:289–303, 1989.

[9] R. Neruda and P. Kudová. Learning methods for radial basis functions networks. *Future Generation Computer Systems*, 21:1131–1142, 2005.

[10] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CoRR*, abs/1412.1897, 2014.

[11] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013. arXiv:1312.6199.

[13] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New-York, 1998.

[14] J. P. Vert, K. Tsuda, and B. Scholkopf. A primer on kernel methods. *Kernel Methods in Computational Biology*, pages 35–70, 2004.