

Denoising Dictionary Learning Against Adversarial Perturbations

John Mitro, Derek Bridge, Steven Prestwich

{j.mitro, d.bridge, s.prestwich}@insight-centre.org

Insight Centre for Data Analytics

Department of Computer Science

University College Cork, Ireland

Abstract

We propose denoising dictionary learning (DDL), a simple yet effective technique as a protection measure against adversarial perturbations. We examined denoising dictionary learning on MNIST and CIFAR10 perturbed under two different perturbation techniques, fast gradient sign (FGSM) and jacobian saliency maps (JSMA). We evaluated it against five different deep neural networks (DNN) representing the building blocks of most recent architectures indicating a successive progression of model complexity of each other. We show that each model tends to capture different representations based on their architecture. For each model we recorded its accuracy both on the perturbed test data previously misclassified with high confidence and on the denoised one after the reconstruction using dictionary learning. The reconstruction quality of each data point is assessed by means of PSNR (Peak Signal to Noise Ratio) and Structure Similarity Index (SSI). We show that after applying (DDL) the reconstruction of the original data point from a noisy sample results in a correct prediction with high confidence.

Introduction

The observation of adversarial perturbations indicated in the seminal work by (Szegedy et al. 2013), is formulated as an attack against DNN models (Papernot et al. 2016b). It describes a mechanism for devising noise taking into account the models' output. This misbehavior is described as a high confidence involuntary misclassification on the models' part, which could potentially undermine the security of environments where these models are deployed. We examined five DNN models, (i) multilayer perceptron (MLP), (ii) convolutional neural network (CNN), (iii) auto-encoder (AE), (iv) residual network (ResNet), (v) hierarchical recurrent neural network (HRNN), of varying complexity and topology, in order to identify how they respond under adversarial noise.

We investigate and visualize which parts of the data maximize the output of the model and verify whether those areas are contaminated under these perturbation attacks. Furthermore, we propose denoising dictionary learning as a measure of protection against adversarial perturbations. It exhibits desirable properties such as robustness (Gregor and

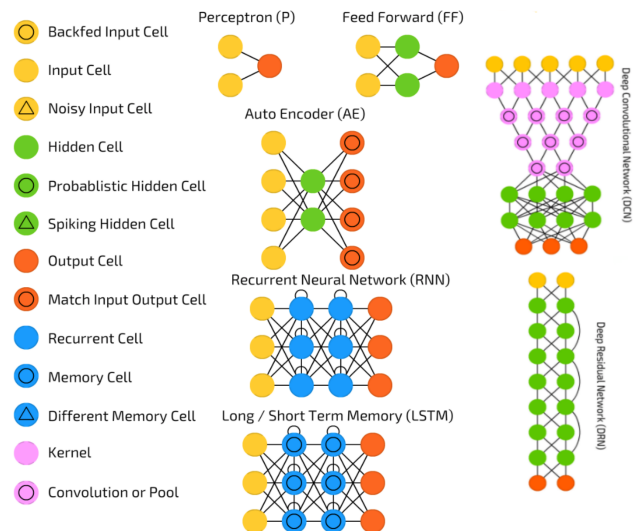


Figure 1: Deep neural network models tested against adversarial perturbations (Veen 2016).

leCun 2010), (Kavukcuoglu, Ranzato, and LeCun 2010) and flexibility (Szlam, Gregor, and LeCun 2012), (Thom, Rapp, and Palm 2015) since it can be incorporated in any supervised learning algorithm (Mairal et al. 2008). Furthermore, it can operate without the presence of noisy samples which indicates a common use case in adversarial attacks where only the test data are perturbed and presented to the model. In comparison to (Wang et al. 2016) denoising dictionary learning has the ability to recover signals from heavily noised samples, moreover, it does not render the data unreadable compared to (Wang et al. 2016) which performs a non-invertible transformation. After the transformation the data are deemed noninterpretable therefore it would require the storage of both datasets, prior to and after the transformation, for interpretability of the results. This not only increases the storage space but is also computationally inefficient in real world scenarios. Current DNN models can be easily exploited by a varying number of different attacks (Papernot et al. 2016c), (Grosse et al. 2016), (Moosavi-Dezfooli, Fawzi, and Frossard 2015), (Carlini and Wagner 2016), (Robinson and Graham 2015), (Narodytska and Kasiviswanathan

2016), (Sethi and Kantardzic 2017), some of them require knowledge of the intrinsic structure of the model (Goodfellow, Shlens, and Szegedy 2014) while others can operate without any prior knowledge of the loss function of the model (Papernot et al. 2016a). These perturbations can be categorized in three categories. 1. Model specific perturbations, 2. White-box perturbations, 3. Black-box perturbations. It is important to notice that there might be an overlap between different perturbation techniques. For the experiments conducted in this study two adversarial perturbation attacks have been utilized. The first one called fast gradient sign (Goodfellow, Shlens, and Szegedy 2014), described as the gradient of the loss of the model multiplied by a scalar. The second one is called jacobian saliency map (Papernot et al. 2015) and is exploiting the forward derivative rather than the cost function emitting information about the learned behavior of the model. The differentiation is applied with respect to the input features rather than the network parameters. Instead of propagating gradients backwards it propagates them forward which permits to find those pixels in the input image that lead to significant changes in the network output. Our contributions in this study are two fold. First, we provide an intuitive explanation of the main building operations or components of DNN, how they operate and how they can be utilized to build more complex models. In addition, we describe which components remain unchanged and how they might cause adversarial perturbations. Finally, we provide a defense mechanism against adversarial perturbations based on sparse dictionary learning to alleviate the problem.

Theoretical Background

In the following Section we provide the necessary information required to understand how DNN operate given their basic building blocks. Based on that information we show how to construct more complex models and identify the main problematic components leading to adversarial perturbations. The “*Adversarial Perturbations*” Section describes two well known adversarial perturbations utilized during the experiments indicating which components are harnessed in order to devise the perturbations. Finally, the “*Dictionary Learning*” Section describes in detail how the proposed defense mechanism is devised, constructed and how it operates to reverse the effect of the perturbations.

Deep Neural Network Components

Each of the models illustrated in Figure 1 is color coded in order to denote the different building components. Each block describes a different operation. Here we provide a formal representation of each models’ structure starting from a model as simple as a multilayer perceptron (MLP) up until to an LSTM unit. An (MLP) with one hidden layer can be described as a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^p$ where d is the size of the input vector \vec{x} and p is the size of the output vector $\vec{y} = f(\vec{x})$. Using matrix notation we formulate it as:

$$\vec{y} = \alpha(\mathbf{W}^{(\ell)} \alpha(\mathbf{W}^{(\ell-1)} \vec{x} + \vec{b}^{(\ell-1)}) + \vec{b}^{(\ell)}) \quad (1)$$

where $\alpha(\cdot)$ is the activation function. Setting $h(\vec{x}) = \alpha(\mathbf{W}^{(\ell-1)} \vec{x} + \vec{b}^{(\ell-1)})$ we can rewrite the above equation

as:

$$\vec{y} = \alpha(\mathbf{W}^{(\ell)} h(\vec{x}) + \vec{b}^{(\ell)}) \quad (2)$$

On the same notion we can extend Equation 1 to accommodate for convolutional operations. Recall that a convolution is defined as:

$$f * g(x) = \int_{\Omega} f(y) g(x - y) dy \quad (3)$$

For the discrete case of a 1D signal the formulation is:

$$\begin{aligned} y[n] &= f[n] * g[n] \\ &= \sum_{u=-\infty}^{\infty} f[u] g[n - u] \\ &= \sum_{u=-\infty}^{\infty} f[n - u] g[u] \end{aligned} \quad (4)$$

This can be extended to 2D as follows:

$$\begin{aligned} y[m, n] &= f[m, n] * g[m, n] \\ &= \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u, v] g[m - u, n - v] \end{aligned} \quad (5)$$

Replacing Equation 5 with Equation 2 we get the representation for the ℓ^{th} convolutional hidden layer:

$$\vec{h}_{ij}^{(\ell)} = \alpha((\mathbf{W}^{(\ell)} * \vec{x})_{ij} + \vec{b}_j^{(\ell)}) \quad (6)$$

We can easily extend the formulation of a convolutional layer to define auto-encoders which could be described as a succession of hidden layers mapped first into a latent space \vec{z} instead of the original space \vec{x} and finally with an inverse transformation restored to the original space:

$$\begin{aligned} \vec{z} &= \alpha(\mathbf{W}^{(\ell)} \vec{x} + \vec{b}^{(\ell)}) \\ \tilde{\vec{x}} &= \alpha(\mathbf{W}^{(\ell)T} \vec{z} + \vec{b}^{(\ell)}) \end{aligned} \quad (7)$$

Even residual networks (He et al. 2015) and their residual building block shown in Figure 2.

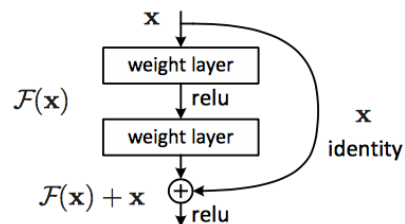


Figure 2: Residual network building block.

can be reconstructed from Equation 2 as follows:

$$\vec{h}^{(\ell)} = \alpha(\mathbf{W}^{(\ell)} \alpha(\mathbf{W}^{(\ell-1)} \vec{x} + \vec{b}^{(\ell-1)}) + \vec{b}^{(\ell)}) + \vec{x} \quad (8)$$

Notice that the bias term \vec{b} is optional and in most recent residual network architectures is omitted.

Finally, recurrent neural networks as a building block introduce two new concepts. The concept of time and memory.

Time could be easily described as a feed forward network unfolded across the time axis. Memory permits the network to share states across the different hidden layers.

$$\begin{aligned}\vec{h}^{(t)} &= \alpha(\mathbf{W}^{(xh)}\vec{x}^{(t)} + \mathbf{W}^{(hh)}\vec{h}^{(t-1)} + \vec{b}^{(h)}) \\ \vec{h}^{(t)} &= \alpha(\mathbf{W}\vec{x}^{(t)} + \mathbf{U}\vec{h}^{(t-1)})\end{aligned}\quad (9)$$

The hidden state $\vec{h}^{(t)}$ at time step t represents a function of the input $\vec{x}^{(t)}$ modified by a weight matrix \mathbf{W} added to the hidden state of the previous time step $\vec{h}^{(t-1)}$, multiplied by its own hidden to hidden state matrix \mathbf{U} , otherwise known as a transition matrix, and similar to a Markov chain. The weight matrices are filters that determine the amount of importance to accord to both the present input and the past hidden state. LSTMs, first introduced by (Hochreiter and Schmidhuber 1997), can be described as a collection of gates with additional constraints. An LSTM layer is described as:

$$\begin{aligned}\vec{i}^{(t)} &= \alpha(\mathbf{W}^{(ix)}\vec{x}^{(t)} + \mathbf{W}^{(ih)}\vec{h}^{(t-1)} + \vec{b}^{(i)}) \\ \vec{g}^{(t)} &= \alpha(\mathbf{W}^{(ix)}\vec{x}^{(t)} + \mathbf{W}^{(ih)}\vec{h}^{(t-1)} + \vec{b}^{(i)}) \\ \vec{f}^{(t)} &= \alpha(\mathbf{W}^{(fx)}\vec{x}^{(t)} + \mathbf{W}^{(fh)}\vec{h}^{(t-1)} + \vec{b}^{(f)}) \\ \vec{o}^{(t)} &= \alpha(\mathbf{W}^{(ox)}\vec{x}^{(t)} + \mathbf{W}^{(oh)}\vec{h}^{(t-1)} + \vec{b}^{(o)}) \\ \vec{c}^{(t)} &= \vec{i}^{(t)} \odot \vec{g}^{(t)} + \vec{c}^{(t-1)} \odot \vec{f}^{(t)} \\ \vec{h}^{(t)} &= \alpha(\vec{c}^{(t)}) \odot \vec{o}^{(t)}.\end{aligned}\quad (10)$$

In this section we wanted to demonstrate how complex DNN models can be constructed starting from MLP up to LSTMs. Even though the different components might require some changes when transitioning from one DNN model to the other, usually three of them remain consistent. Those refer to (a) the loss function, which is the categorical cross entropy in Equation 11 for all five models, (b) the activation function ReLU, (c) the optimization process resulting in a variant of gradient descent (Kingma and Ba 2014).

We believe that is this combination that is responsible for the phenomenon of adversarial perturbations. Examining closely each building block we realize that DNN models resemble linear models since most of the components either are a result of, or describe a linear transformation. This would explain the high confidence in misclassified examples since linear models also have the tendency to extrapolate to unseen data points with high confidence. Unfortunately, the loss function does not help either, in this situation, since eventually any differential loss has the potential to emit information on which parts of the input should be altered accordingly in order to maximize a particular output target. This information can be exploited by an adversary. Finally, any optimization process requiring small steps towards the direction of the local minimum such as gradient descent or variants will result in a process where small changes gradually lead to overall bigger effects. It is these small changes that the adversaries exploit in order to devise the perturbations.

Adversarial Perturbations

In this section we provide a short description and formulation of the adversarial perturbations utilized during the experiments. First, we present (FGSM) which is the gradient

of the loss $\nabla_{\vec{x}}\mathbb{L}(\vec{x}, \vec{y})$ with respect to the input \vec{x} multiplied by a constant ϵ defined as:

$$\begin{aligned}\epsilon \nabla_{\vec{x}}\mathbb{L}(\vec{x}, \vec{y}) &= -\frac{1}{N} \sum_{n=1}^N \nabla_{\vec{x}}\mathbb{L}y^{(n)} \ln \alpha(\vec{x}^{(n)}) \\ &+ (1 - y^{(n)}) \ln(1 - \alpha(\vec{x}^{(n)}))\end{aligned}\quad (11)$$

where $\alpha(\vec{x})$ describes the output of the neural network given input \vec{x} . The final perturbed sample for an input \vec{x} is $\vec{x} + \epsilon \nabla_{\vec{x}}\mathbb{L}(\vec{x}, \vec{y})$. Second, we present (JSMA) in the following three steps. (i) Compute the forward derivative $\nabla_{\vec{x}^*}f(\vec{x}^*)$. (ii) Construct the saliency map Σ based on the derivative. (iii) Modify an input feature m by ϵ . The adversary's objective is to craft an adversarial sample $\vec{x}^* = \vec{x} + \delta_{\vec{x}}$ such that the final output of the network results in a misclassification $f(\vec{x}^*) = \hat{y} \neq \vec{y}$ where f_n describes the derivative of one output neuron. Reformulating it as an optimization problem we have the following objective $\min_{\delta_{\vec{x}}} \|\delta_{\vec{x}}\|$ s.t. $f(\vec{x} + \delta_{\vec{x}}) = \hat{y}$. The forward derivative of a DNN for a given sample is essentially the Jacobian learned by the neural network $\nabla f(\vec{x}) = \left[\frac{\partial f_n(\vec{x})}{\partial x_m} \right]_{m=1, \dots, M, n=1, \dots, N}$, where m denotes the input feature and n denotes the output of neuron n .

Next compute the saliency map based on the forward derivative. Saliency maps convey information which pixel intensity values should be increased in order for a specific target t to be misclassified by the neural network $t \neq y(\vec{x})$ where $y(\vec{x})$ denotes the true label assigned to input \vec{x} . The saliency map $\Sigma(\vec{x}, t)$ is defined as:

$$\Sigma(\vec{x}, t)_m = \begin{cases} 0 & \text{if } \frac{\partial f_t(\vec{x})}{\partial x_m} < 0 \text{ or } \sum_{n \neq t} \frac{\partial f_n(\vec{x})}{\partial x_m} \\ \left(\frac{\partial f_t(\vec{x})}{\partial x_m} \right) & \left| \sum_{n \neq t} \frac{\partial f_n(\vec{x})}{\partial x_m} \right|\end{cases}\quad (12)$$

The first line rejects components with negative target derivative or positive derivative on all other classes. The second line considers all other forward derivatives. In summary, high saliency map values denote features that will either increase the target class or decrease other classes significantly. Increasing those feature values causes the neural network to misclassify a sample into the target class.

Dictionary Learning

In summary, dictionary learning could be deconstructed into the following three principles. (i) Linear decomposition, (ii) sparse approximation, and (iii) dictionary learning. Linear decomposition asks the question whether a signal \vec{y} can be described as a linear combination of some basis vectors and their coefficients. Given a signal $\vec{y} \in \mathbb{R}^N$ and a matrix $\mathbf{D} \in \mathbb{R}^{M \times N}$ of N vectors $[d]_{n=1}^N$, the linear decomposition of \vec{y} described by \mathbf{D} is such that $\vec{y} = \mathbf{D}\vec{s} + \vec{\epsilon} = \sum_{n=1}^N \vec{d}_n \vec{s}_n + \vec{\epsilon}$ where $\vec{s} \in \mathbb{R}^N$ represent the coefficients and $\vec{\epsilon}$ is the error. Sparse approximation on the other hand refers to the ability of \mathbf{D} to reconstruct \vec{y} from a set of sparse basis vectors. When \mathbf{D} contains more vectors than samples, i.e., $N > M$, then it is called a dictionary and its vectors are referred to as atoms. Since $\vec{y} = \mathbf{D}\vec{s} + \vec{\epsilon}$, has multiple

solutions for \vec{s} . It is usually accustomed to introduce different types of constraints, such as, for instance, sparsity or others, in order to allow the solution to be regularized. The decomposition of \vec{y} under sparse constraints is formulated as $\arg \min_{\vec{s}} \|\vec{y} - \mathbf{D}\vec{s}\|_2^2$ s.t. $\|\vec{s}\|_1 \leq T$, where T is a constant. There exist a plethora of algorithms to deal with this problem, some utilize greedy approaches such as orthogonal matching pursuit (OMP) (Pati, Reziifar, and Krishnaprasad 1993) or ℓ_1 -norm based optimizations such as basis pursuit (Chen, Donoho, and Saunders 1998), least-angle regression (Efron et al. 2004), iterative shrinkage thresholding (Daubechies, Defrise, and Mol 2004) which guarantee convex properties.

OMP proceeds by iteratively selecting the atoms, i.e., the columns in a dictionary with corresponding nonzero coefficients $\vec{s} \in \mathbb{R}^N$ computed via orthogonal projection of \vec{y} on \mathbf{D} that best explain the current residue $\vec{e} = \vec{y} - \mathbf{D}\vec{y}$. Mainly the optimization process is a two step approach alternating between OMP and dictionary learning. As we mentioned the goal of dictionary learning is to learn the dictionary \mathbf{D} that is most suitable to sparsely approximate a set of signals as it is shown in Equation 15. This non-convex problem is usually solved by alternating between the extraction of the main atoms, which is referred to as sparse coding or sparse approximation step, and the actual learning process which is referred to as the dictionary update step. This optimization scheme reduces the error criterion iteratively. There are several dictionary learning algorithms, such as maximum likelihood (ML), method of optimal directions (MOD), and K-SVD for batch methods, and online dictionary learning and recursive least squares (RLS) for online methods, which are less expensive in computational time and memory than batch methods.

Next we will formulate the problem of dictionary learning from corrupted samples and demonstrate its ability as a defense mechanism against adversarial perturbations. Formally, the problem of image denoising is described as follows:

$$\vec{y} = \vec{x} + \vec{e} \quad (13)$$

\vec{y} is our measurements, \vec{x} is the original image and \vec{e} is the noise. The objective is to recover \vec{x} from our noisy measurements \vec{y} . We can reformulate our problem as an energy minimization problem also known as maximum a posterior estimation.

$$\mathbb{E}[\vec{x}] = \|\vec{y} - \vec{x}\|_2^2 + \Pr(\vec{x}) \quad (14)$$

$\|\vec{y} - \vec{x}\|_2^2$ refers to the relation to the measurements and $\Pr(\vec{x})$ is the prior. There are a number of different classical priors from which we can choose.

- Smoothness $\lambda \|\mathcal{L}\vec{x}\|_2^2$
- Total variation $\lambda \|\nabla \vec{x}\|_1^2$
- Wavelet sparsity $\lambda \|\mathbf{W}\vec{x}\|_1$

Utilizing sparse representations for image reconstruction we can rewrite $\Pr(\vec{x}) = \lambda \|\vec{s}\|_0$ for $\vec{x} \approx \mathbf{D}\vec{s}$. This could be

translated visually to the following operation's:

$$\underbrace{\begin{pmatrix} y \end{pmatrix}}_{\vec{y} \in \mathbb{R}^M} = \begin{pmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_N \end{pmatrix} \underbrace{\begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_N \end{pmatrix}}_{\vec{s} \in \mathbb{R}^N, \text{ sparse}}$$

Learning a dictionary of atoms can be viewed as an optimization equation derived from two parts. The first part refers to the reconstruction of the original signal and the second part refers to the sparsity.

$$\min_{\vec{s}_n, \mathbf{D} \in C} \sum_{n=1}^N \frac{1}{2} \|\vec{y}_n - \mathbf{D}\vec{s}_n\|_2^2 + \lambda \phi(\vec{s}_n) \quad (15)$$

Where $\phi(\vec{s}) = \|\vec{s}\|_0$ is the ℓ_0 pseudo-norm and $\phi(\vec{s}) = \|\vec{s}\|_1$ is the ℓ_1 norm. How the optimization problem works is as follows. We formulate and solve a matrix factorization problem after we have extracted all overlapping 8×8 patches of \vec{y} . The factorization problem is formulated as follows:

$$\min_{\vec{s}_n, \mathbf{D} \in C} \sum_{n=1}^N \frac{1}{2} \underbrace{\|\vec{y}_n - \mathbf{D}\vec{s}_n\|_F^2}_{\text{reconstruction}} + \underbrace{\lambda \phi(\vec{s}_n)}_{\text{sparsity}} \quad (16)$$

$$\frac{1}{2} \|\mathbf{Y} - \mathbf{D}\vec{s}_n\|_F^2 + \lambda \|\vec{s}_n\|_1$$

$$\text{where } \lambda \phi(\vec{s}_n) = \|\vec{s}_n\|_1$$

$$\text{such that } \mathbf{Y} = [\vec{y}_1, \dots, \vec{y}_n] \text{ and } \vec{s} = [s_1, \dots, s_n].$$

Notice that different constraints adhere on \mathbf{D} and \vec{s} depending on the matrix factorization approach. For instance if PCA is selected as a solution to the factorization problem then \mathbf{D} should be orthonormal and \vec{s}^T orthogonal. Otherwise if non negative matrix factorization is selected then \mathbf{D} and \vec{s} should be non negative. The optimization for dictionary learning is formulated as follows:

$$\min_{\mathbf{D} \in C} f(\mathbf{D}) = \mathbb{E}_{\vec{x}}[\xi(\vec{y}, \mathbf{D})]$$

$$\approx \lim_{n \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N \xi(\vec{y}_n, \mathbf{D})$$

$$\text{where } \min_{\mathbf{D} \in C} f(\mathbf{D}) = \min_{\mathbf{D} \in C} \frac{1}{N} \sum_{n=1}^N \xi(\vec{y}_n, \mathbf{D})$$

$$C \triangleq \mathbf{D} \in \mathbb{R}^{M \times N} \mid \forall n = 1, \dots, N \text{ s.t. } \|\mathbf{d}_n\|_2 \leq 1$$

$$\xi(\vec{y}, \mathbf{D}) \triangleq \min_{\vec{s} \in \mathbb{R}^N} \frac{1}{2} \|\vec{y} - \mathbf{D}\vec{s}\|_2^2 + \lambda \|\vec{s}\|_1. \quad (17)$$

In the following section we provide the description for OMP and dictionary learning algorithms utilized in the experiments. Regarding Algorithm 1, at the current iteration t , OMP selects the atom \hat{a} that produces the strongest decrease in the residue. This is equivalent to selecting the atom that is most correlated with the residue. An active set, Q i.e., nonzero, is formed, which contains all of the selected atoms.

In the following step the residue ϵ is updated via an orthogonal projection of \vec{y} on \mathbf{D} . Finally, the sparse coefficients of \vec{s} are also updated according to the active set Q .

Algorithm 1 Orthogonal matching pursuit algorithm.

```

min_{\vec{s} \in \mathbb{R}^N} \|\vec{y} - \mathbf{D}\vec{s}\|_2^2 \text{ s.t. } \|\vec{s}\|_1 \leq T
Q = \emptyset
for t = 1 to T do
  Select the atom that reduces the objective
  \hat{\alpha} \leftarrow \arg \min_{t \in Q^c} \{\min_{\vec{s}} \|\vec{y} - \mathbf{D}_{Q \cup t} \vec{s}\|_2^2\}
  Update the active set: Q \leftarrow Q \cup \hat{\alpha},
  Update the residual via orthogonal projection:
  \vec{\epsilon} \leftarrow (\mathbf{I} - \mathbf{D}_Q (\mathbf{D}_Q^T \mathbf{D}_Q)^{-1} \mathbf{D}_Q^T) \vec{y}
  Update the coefficients: \vec{s}_Q \leftarrow (\mathbf{D}_Q^T \mathbf{D}_Q)^{-1} \mathbf{D}_Q^T \vec{y}
end for

```

As for Algorithm 2 the sparse coding step is usually the one which is carried out by orthogonal matching pursuit described in Algorithm 1.

Algorithm 2 Dictionary optimization algorithm.

```

Require: \mathbf{D} \in \mathbb{R}^{m \times n}, \lambda \in \mathbb{R}
\mathbf{A} = 0, \mathbf{B} = 0
for t = 1 to T do
  Draw \mathbf{y}_t
  \mathbf{s}_t \leftarrow \arg \min_{\mathbf{s} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{y}_t - \mathbf{D}_{t-1} \mathbf{s}\|_2^2 + \lambda \|\mathbf{s}\|_1,
  Sparse Coding
  \mathbf{A}_t \leftarrow \mathbf{A}_{t-1} + \mathbf{s}_t \mathbf{s}_t^T,
  \mathbf{B}_t \leftarrow \mathbf{B}_{t-1} + \mathbf{y}_t \mathbf{s}_t^T
  \mathbf{D}_t \leftarrow \arg \min_{\mathbf{D} \in \mathcal{C}} \frac{1}{N} \sum_{n=1}^N (\frac{1}{2} \|\mathbf{y}_n - \mathbf{D} \mathbf{s}_n\|_2^2 + \lambda \|\mathbf{s}_n\|_1)
end for

```

Methodology

In this study we trained five different models for 100 epochs, from multilayer perceptrons to hierarchical LSTMs with a batch size of 32, on two different datasets, MNIST and CIFAR10 whose distributions are presented below.

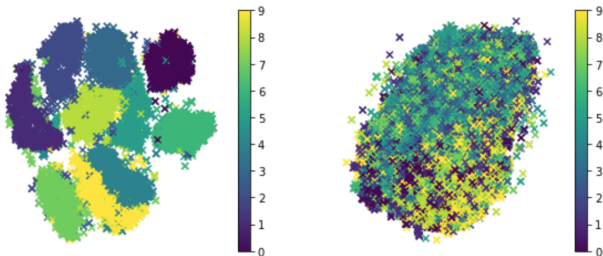


Figure 3: T-SNE embedding of MNIST (left) and CIFAR10 (right).

Each model has been trained and its accuracy has been recorded on the clean test data as well as on the adversarial one. Moreover, we represent visually the distributions for

each dataset and how they change accordingly before and after the adversarial attacks in Figure 4. Notice that in real world datasets such as CIFAR10 the shift of the distribution is almost unnoticeable which demonstrates the severity of adversarial attacks undermining the security of neural networks. The first row contains the qq-plot along with its density plot for MNIST before (a) and after (b) the adversarial perturbation while the second row contains the same information for CIFAR10.

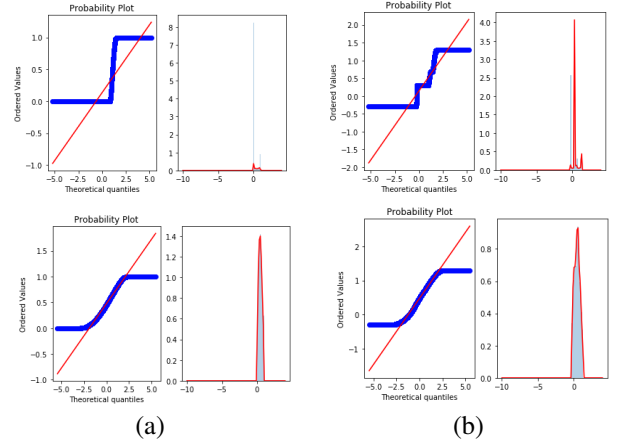


Figure 4: Probability density function for MNIST (a) and CIFAR10 (b) before and after the adversarial perturbation

We also demonstrate visually which are the most vulnerable parts in regards to the top predictions that each neural network has learned in order to differentiate two similar data points belonging in the same category in Figure 7 for MNIST and Figure 9, Figure 5 and Figure 6 for CIFAR10.

Finally, we propose dictionary learning as a defense mechanism against adversarial attacks. We demonstrate its ability on two different adversarial attacks and record the results in Table 1 and Table 2, which clearly shows that, in overall, each model achieves higher classification accuracy on the perturbed datasets after utilizing dictionary learning to reconstruct the original data from the perturbed samples. One of the advantages of dictionary learning is that it can operate regardless of the presence of perturbed or noisy samples. This implies that the dictionary \mathbf{D} can be learned either from the extracted patches of the perturbed samples or from the clean patches of the non perturbed train data. This could be useful in environments where we do not know a priori the attack or the method used to generate the perturbed data. Another advantage of dictionary learning and sparse coding is the fact that it can be embedded in any supervised learning algorithm without any severe restrictions. In the particular case of DNN the dictionary \mathbf{D} can be easily learned from the weights \mathbf{W} of an auto-encoder model for instance.

Furthermore, the computational complexity of dictionary learning is much lower compared to (Wang et al. 2016) since the dictionary can be learned during training time avoiding the overhead in invoking a non invertible transformation during test time. This means that whenever a prediction is required from the model, a non invertible and computationally

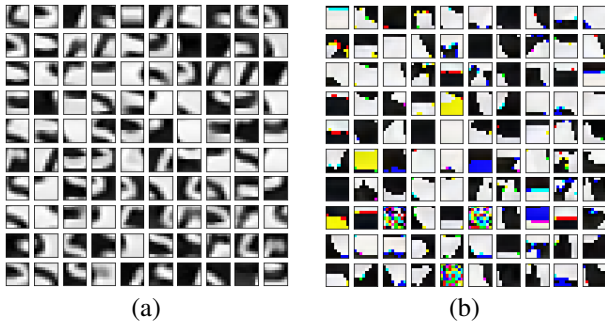


Figure 4a: OMP dictionary, MNIST (a) & CIFAR10 (b).

expensive transformation has to be performed in advance.

Experiments

Evaluation

The experiments in this study utilized five DNN models that resemble as close as possible real world application architectures composed of multiple layers such as, dropout and batch normalization to avoid over-fitting. We deliberately avoided DNN models composed of only convolutional layers which seem to be more error prone to adversarial attacks. The description of the hyper-parameters for each model is provided in Table 3. Each model has been evaluated on two different datasets perturbed using two different perturbation techniques (FGSM) and (JSMA). After the evaluation of dictionary learning as defense mechanism against adversarial perturbations we found that is able to withstand the attacks and provide good results in terms of accuracy for each model on the reconstructed datasets. During training for consistency we utilized Adam (Kingma and Ba 2014) as the optimizer for all the models. We proceeded by perturbing the test set $\mathbf{Y} \in \mathbb{R}^{M \times N}$ once for FGSM \mathbf{Y}^\dagger , and once for JSMA \mathbf{Y}^* , where we tested each model equivalently on \mathbf{Y} , \mathbf{Y}^\dagger , \mathbf{Y}^* , and recorded their accuracies in Table 1 and Table 2. For each image $\vec{y} \in \mathbf{Y}$ we extracted a set of overlapping 8×8 patches which were used to learn the dictionary \mathbf{D} and its coefficients \vec{s} for each dataset as it is shown below.

Next we extracted patches from the noisy samples \mathbf{Y}^\dagger and \mathbf{Y}^* , applying orthogonal matching pursuit described in Algorithm 1 reconstructing the original samples $\vec{y} \in \mathbf{Y}$. For each sample we evaluated its reconstruction error utilizing mainly two different metrics. The first one is referred to as peak to signal noise ratio (PSNR) and is formulated as $PSNR = 10 \log_{10}(\frac{\max(\vec{y})^2}{MSE})$, where $\max(\vec{y})$ refers to the maximum pixel value of the image \vec{y} and MSE refers to the mean square error. The second one is structural similarity index (SSIM) and is formulated according to $SSIM(x, y) = \frac{(2\mu_x \mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$ where x, y here represent windows of size $N \times N$, μ represents the average of a window depending on the subscript, σ^2 is the variance for each window and σ_{xy} is the co-variance between x and y . Finally, c_1 and c_2 are constants in order to stabilize the division with weak denominators.

Results

The results are summarized in Table 1 and Table 2 as well as in Figure 5 through Figure 9. Table 1 describes the results for all five models by evaluating their accuracy on MNIST and CIFAR10, on the actual test data, on the perturbed one after the FGSM perturbation attack and finally on the denoised samples recovered through denoising dictionary learning. The choice of atoms was based on a heuristic selection and it resulted in 38 atoms for MNIST and 2 atoms for CIFAR10. Although we suspect that the overall results could be improved by utilizing Bayesian hyper-parameter selection for the choice of atoms. Table 2 equivalently describes the results for the actual test data, perturbed under the JSMA perturbation attack and the denoised one recovered through dictionary learning.

Table 1: Classifier accuracy against FGSM with noise intensity, $\epsilon = 0.3$

Dataset	MNIST			CIFAR10			
	\times	\checkmark	Denoised	\times	\checkmark	Denoised	
Classifier	MLP	98.39%	12.80%	82%	60%	11%	55.60%
	ConvNet	99.35%	79.90%	90.46%	77.90%	15.06%	68.29%
	AutoEnc	99.34%	77.60%	89.37%	70.56%	13.63%	67.76%
	ResNet	93.79%	1.95%	74.27%	76.11%	0.089%	70.06%
	HRNN	98.90%	23.02%	82.52%	64%	15.75%	58.41%

Table 2: Classifier accuracy against JSMA.

Dataset	MNIST			CIFAR10			
	\times	\checkmark	Denoised	\times	\checkmark	Denoised	
Classifier	MLP	98.39%	53.02%	60%	60%	52.77%	57.63%
	ConvNet	99.35%	79.90%	93.85%	77.90%	14.59%	68.29%
	AutoEnc	99.34%	62.02%	91.47%	70.56%	56.83%	64.76%
	ResNet	93.79%	38.16%	61.12%	76.11%	56.87%	60.16%
	HRNN	98.90%	52.65%	63.21%	64%	56.43%	61.21%

Table 3: Hyper-parameters for DNN models

MLP	ConvNet	AutoEncoder	ResNet	HRNN
Dropout 0.5	Dropout 0.5	Conv2D: filters=16, kernel=3	Conv2D	LSTM: units=256
BatchNorm	$2 \times$ Conv2D: filters=32, kernel=3	MaxPool: size=2	BatchNorm	TimeDistributed
Dense 784	MaxPool: size=2	Conv2D: filters=32, kernel=3	ReLU	LSTM: units=256
ReLU	Dropout 0.25	MaxPool: size=2	Basic Block	Reshape: 16×16
Dropout 0.2	$2 \times$ Conv2D: filters=64, kernel=3	Conv2D: filters=32, size=3	$3 \times$ Residual Block	LSTM: units=256
BatchNorm	MaxPool: size=2	MaxPool: size=2	Dropout 0.25	Dropout 0.25
Dense 256	Conv2D: filters=128, kernel=3	Conv2D: filters=64, kernel=3	BatchNorm	Dense 10
ReLU	Conv2D: filters=256, kernel=3	Conv2D: filters=128, kernel=3	ReLU	Softmax
Dense 10	Dropout 0.25	UpSample: size=2	GlobalAveragePooling	
Softmax	Dense 512	Conv2D: filters=32, kernel=3	Dense 10	
	Dropout 0.5	UpSample: size=2	Softmax	
	Dense 10	Conv2D: filters=3, kernel=3		
	Softmax	Conv2D: filters=1, kernel=5		
		Dense 10		
		Softmax		

As it is evident all five models achieve higher accuracy on the recovered samples compared to the perturbed version. We present the top classifications for the convolutional and residual network model along with their misclassification on the perturbed data under the FGSM attack on MNIST, as well as, their class activation maps which describe the sensitivity of the classifier on different parts of the input. In Figure 9 and Figure 5 we present the top misclassification for the convolutional and residual network along with the activation maps for CIFAR10 perturbed under the JSMA attack. In Figure 7 we show the top classifications for the auto-encoder model along with their activation maps under the FGSM attack on MNIST. Equivalently in Figure 6 we show the top misclassification for the auto-encoder on CIFAR10 under the JSMA attack. As you might have noticed the multilayer perceptron does not have feature maps similar to a convolu-

tional network therefore it is impossible to derive the class activation maps. Similarly the same holds true for the hierarchical recurrent model. We noticed that the results from the residual network were not as resistant as we would have expected due to its skip connections. What we can infer from Figure 5 is that models who have the ability to focus on very small details of the overall image seem to be more susceptible to adversarial perturbations. In Figure 8 we demonstrate an example of a reconstructed image equivalently for each dataset and perturbation attack.

Conclusion

In this article, a defense mechanism is proposed against adversarial perturbations based on dictionary learning sparse representations for gray scale (MNIST) and color images (CIFAR10). The method has been evaluated against five modern deep neural network architectures which compose the building blocks for the majority of recent neural network architectures. The choice of dictionary learning is based solely on its properties. The resulted dictionary is a redundant, over-complete basis, and it provides a more efficient representation than a normal basis. It is robust against noise, it has more flexibility for matching patterns in the data, and it allows a more compact representation. Future directions include the extension and comparison of the current work with deep denoising models such as gated Markov random fields and deep Boltzmann machines on the ImageNet dataset.

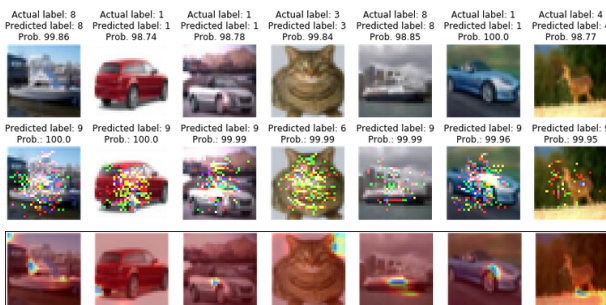


Figure 5: Misclassification for ResNet CIFAR10.

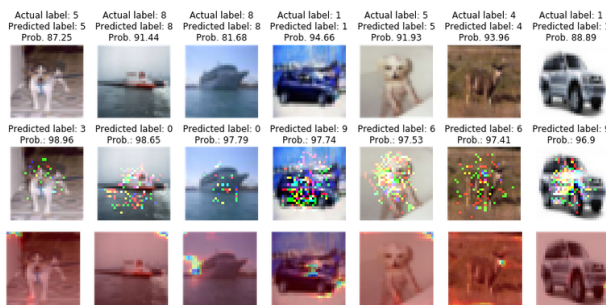


Figure 6: Misclassification for AutoEnc. on CIFAR10.

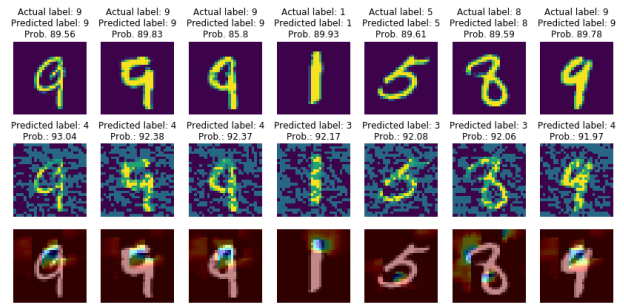


Figure 7: Misclassification for AutoEnc. on MNIST.

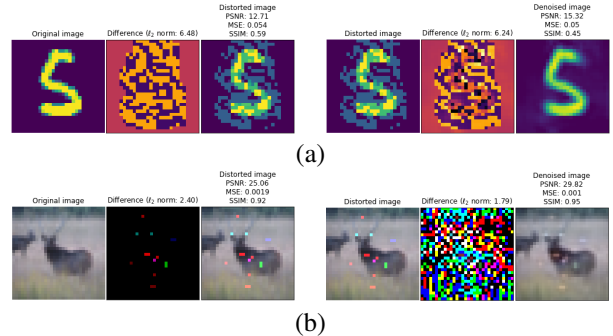


Figure 8: Denoising on MNIST, FGSM (a) & CIFAR10, JSMA (b).

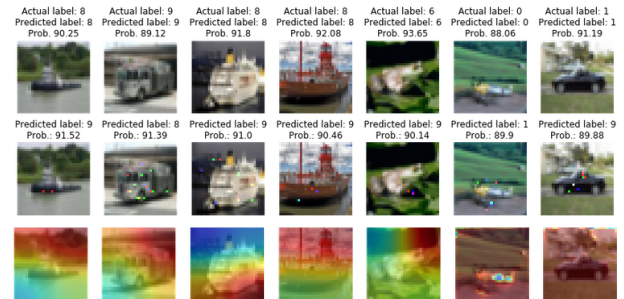


Figure 9: Misclassification for ConvNet CIFAR10.

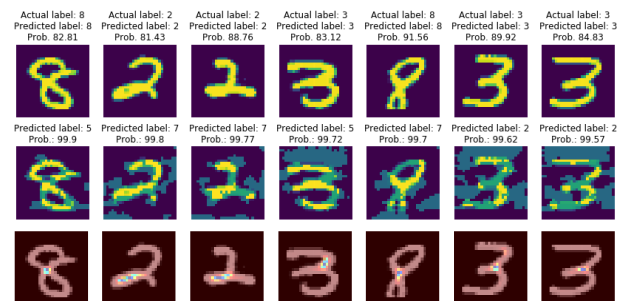


Figure 10: Misclassification for ResNet on MNIST.

References

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

- Chen, S.; Donoho, D.; and Saunders, M. 1998. Atomic decomposition by basis pursuit. *SIAM Journal of Computer Science* 20:33—61.
- Daubechies, I.; Defrise, M.; and Mol, C. D. 2004. An iterative algorithm for linear inverse problems with a sparsity constraint. *Communications of Pure Applied Mathematics* LVII:1413—1457.
- Efron, B.; Hastie, T.; Johnstone, I.; and Tibshirani, R. 2004. Least angle regression. *Annal of Statistics* 32:407—499.
- Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples. *arXiv*.
- Gregor, K., and leCun, Y. 2010. Learning fast approximations of sparse coding. *ICML* 9.
- Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; and McDaniel, P. 2016. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *arXiv*.
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *arXiv*.
- Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 8(9):32.
- Kavukcuoglu, K.; Ranzato, M.; and LeCun, Y. 2010. Fast Inference in Sparse Coding Algorithms with Applications to Object Recognition. *arXiv*.
- Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv*.
- Mairal, J.; Bach, F.; Ponce, J.; Sapiro, G.; and Zisserman, A. 2008. Supervised dictionary learning. *NIPS* 15.
- Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2015. DeepFool: a simple and accurate method to fool deep neural networks. *arXiv*.
- Narodytska, N., and Kasiviswanathan, S. P. 2016. Simple Black-Box Adversarial Perturbations for Deep Networks. *arXiv*.
- Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2015. The Limitations of Deep Learning in Adversarial Settings. *arXiv*.
- Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z. B.; and Swami, A. 2016a. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. *arXiv*.
- Papernot, N.; McDaniel, P.; Sinha, A.; and Wellman, M. 2016b. Towards the Science of Security and Privacy in Machine Learning. *arXiv*.
- Papernot, N.; McDaniel, P.; Swami, A.; and Harang, R. 2016c. Crafting Adversarial Input Sequences for Recurrent Neural Networks. *arXiv*.
- Pati, Y. C.; Reziifar, R.; and Krishnaprasad, P. S. 1993. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. *Conf. on Signals, Systems and Computers* 40—44.
- Robinson, L., and Graham, B. 2015. Confusing Deep Convolution Networks by Relabelling. *arXiv*.
- Sethi, T. S., and Kantardzic, M. 2017. Data Driven Exploratory Attacks on Black Box Classifiers in Adversarial Domains. *arXiv*.
- Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv*.
- Szlam, A.; Gregor, K.; and LeCun, Y. 2012. Fast approximations to structured sparse coding and applications to object classification. *arXiv*.
- Thom, M.; Rapp, M.; and Palm, G. 2015. Efficient dictionary learning with sparseness-enforcing projections. *International Journal of Computer Vision* 114(2-3):168—194.
- Veen, F. v. 2016. The Neural Network Zoo. <http://www.asimovinstitute.org/neural-network-zoo/>.
- Wang, Q.; Guo, W.; II, A. G. O.; Xing, X.; Lin, L.; Giles, C. L.; Liu, X.; Liu, P.; and Xiong, G. 2016. Using Non-invertible Data Transformations to Build Adversary-Resistant Deep Neural Networks. *arXiv*.