The Limitations of Deep Learning in Adversarial Settings

Abstract

深度学习利用 **大量数据和高效的算法** 展现出比别的机器学习任务更好的性能

但是训练阶段的缺点导致深度神经网络更容易受到对抗样本的攻击： 攻击者制作输入使得DNN误判

1. 针对DNN formalize the space of adversaries
2. 基于已完全了解的DNN的输入与输出的映射，提出了一种洗的算法来制作对抗样本
3. 在计算机视觉的应用中，展现 对抗样本 能被人类识别但无法被机器识别的概率高达97%
4. 通过定义一种 hardness measure 对不同样本类型进行对抗扰动的脆弱性进行了评估
5. 通过设定正常输入和目标分类之间的预测距离 进行初步的防御

Introduction

对抗样本： 是一个特制的输入 能导致学习算法误判

利用DNN从有限训练集中学习到的不完全泛化，通过添加扰动，以及用于构建DNN中使用的大多数元件都有潜在的线度，从良性样本中创建对抗性样本

多维函数 $F:X->Y$ 其中X是一个原始特征向量，Y是输出数组
然后通过加上一个δX构建一个X^*

$$arg \min_{\delta X} \|\delta X\| \; s.t. F(X+\delta X) = Y^* \qquad (1)$$

其中$X^* = X + \delta X$
解决这个问题很重要，因为DNN通常是非线性和非凸

我们对输入的变化如何影响深度神经网络输出的理解源于前向导数（forward derivative）：我们引入并定义为DNN学习的函数的Jacobian矩阵。

为了生成能从DNN模型中获得理想结果的对抗样本，就用前向导数构造 包含扰动δX的对抗显著映射（adversarial saliency maps）

基于前向导数的方法比在先验系统中使用的梯度下降技术要强大得多：

- 适用于监督和无监督学习
- 能为广泛对抗样本族类生成信息

对抗显著性映射是基于前向导数的一个重要工具，并在设计时考虑到对抗性目标，在扰动的选择方面给予敌手更大的控制权

Contribution

1. 在对抗性目标和能力方面，我们形式化了对手对分类器DNN的空间
2. 我们引入了一类新的算法，仅通过使用DNN架构的知识来制作对抗性样本
3. 我们使用一个广泛使用的计算机视觉DNN来验证了这些算法。

Threat Model Taxonomy in Deep Learning（深度学习中的威胁模型分类）

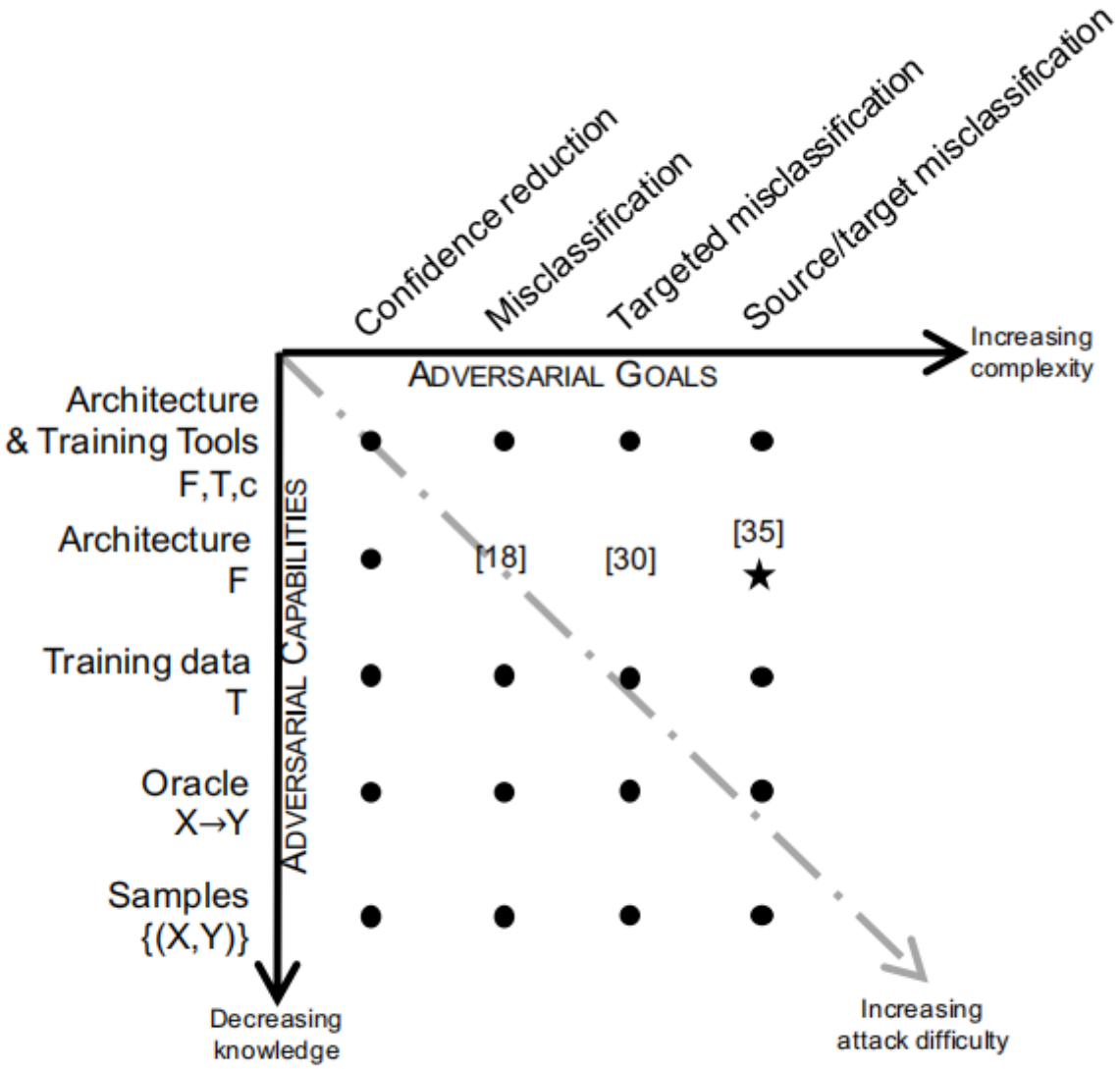

Figure 2: Threat Model Taxonomy: our class of algorithms operates in the threat model indicated by the star.

About Deep Neural Networks

深度神经网络是一种组织成神经元层的大型神经网络，对应于输入数据的连续表示。

DNN 又有两类 **监督**与**无监督**

- 监督训练导致使用从标记训练数据推断的函数将未见样本映射到预定义的输出集的模型
- 无监督训练学习未标记训练数据，由此产生的DNN模型可以用于生成新的样本，或者通过作为大型DNN的预处理层来自动化特征工程

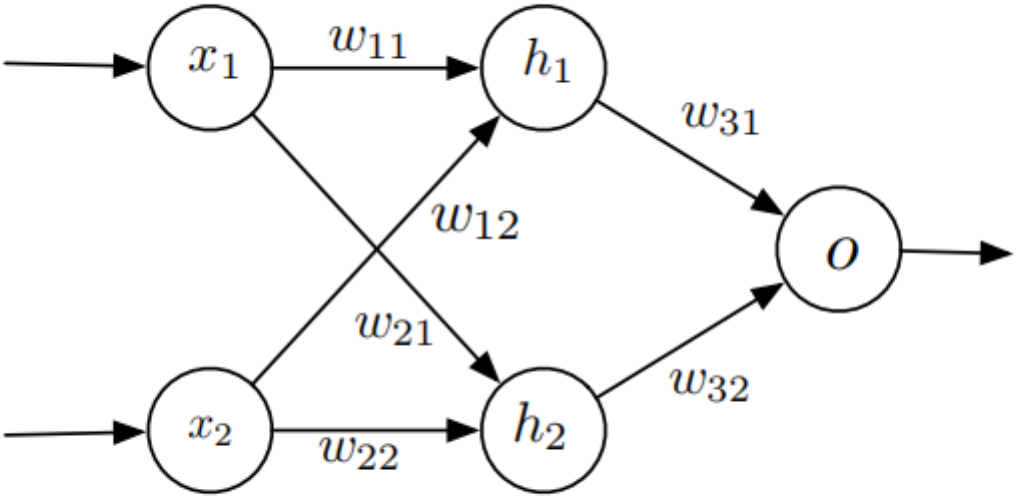

Figure 3: Simplified Multi-Layer Perceptron with input $\mathbf{X} = (x_1, x_2)$, hidden layer (h_1, h_2), and output o.

$$\phi : X- > \frac{1}{1+e^{-x}}$$

$$Z_{h_1}(X) = \omega_{11}x_1 + \omega_{12}x_2 + b_1$$

$$h_1(X) = \phi(Z_{h_1}(X))$$

$$Z_0(X) = \omega_{31}h_1(X) + \omega_{32}h_2(X) + b_3$$

Adversarial Goals（敌对目标）

1. Confidence reduction 降低输出置信度
2. Misclassification 将输出类改为与原类不同的类
3. Targeted misclassification 产生能强迫输出成为一个特定的目标类的输入
4. Source/target misclassification 强迫一个特定输入的输出分类导向一个特定的目标类

Adversarial Capabilities（敌对能力）

敌手由其掌握的信息和能力定义

1. Training data and network architecture 敌手全知模型和训练数据
2. Network architecture 敌手知道网络架构及其参数值
3. Training data 敌手能获得训练集同分布的一个子集

4. Oracle 敌手能使用神经网络（能获得其输出）
5. Samples 敌手能获得神经网络的输入输出对

Approach

提出了一个修改样本的通用算法，使DNN产生任何期望的敌对输出。

对模型结构及其参数的知识足够获得针对acyclic feedforward DNNs的敌对样本

这需要评估DNN的前向导数，以构建一个对抗性显著性映射，以识别与对手目标相关的输入特征集。

扰乱以这种方式识别的特征会迅速导致所需的对抗性输出，例如，错误的分类

虽然我们用监督分布式神经网络作为分类器来描述我们的方法，但它也适用于无监督架构。

Studying a Simple Neural Network

上面对模型结构的阐述，展示了使用前向导数发现的小输入扰动是如何引起神经网络输出的大变化的。

输入偏差为$b_1, b_2, b_3(null)$

应得输出为$F(X) = x_1 \bigwedge x_2$ 其中$X = (x_1, x_2)$ 非整数会化成整数
从网络模型中可以得到$F(X) = Y$
然后对X添加一点扰动δX得到X^*(X^*与X非常相似)，但$F(X^*) = Y^* \neq Y$

因此这个问题就变成如下最小化公式：找到最小的δX使得模型误判：

$$arg \min_{\delta X} \|\delta X\| \ s.t. F(X + \delta X) = Y^* \qquad (1)$$

这些扰动可以通过优化或者手动解决，但这些解决方法不适用与DNN，因其通常是非凸与非线性，因此提出一种基于向前导数的算法

将前向导数定义为神经网络在训练过程中学习的函数F的雅可比矩阵（Jacobian matrix）。
例如上面模型，可将雅各比矩阵化为如下形式
$J_F(X) = [\frac{\partial F(X)}{\partial x_1}, \frac{\partial F(X)}{\partial x_2}]$

那么针对如上的式子，用可视化的手段可以得出在0和1之间有一道非常明显的鸿沟

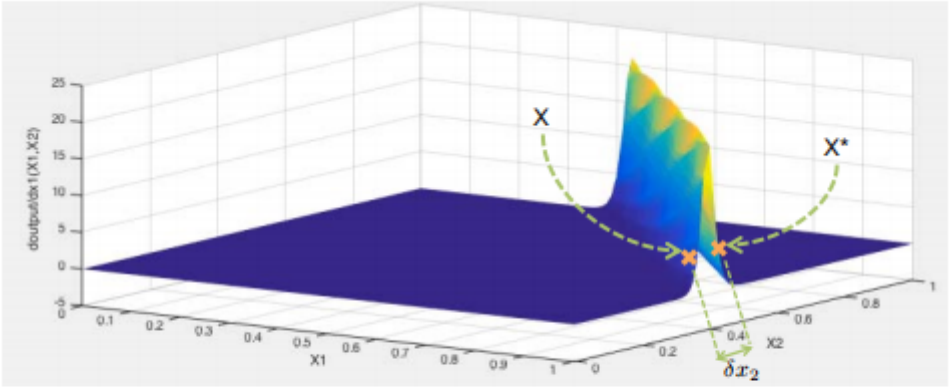

Figure 5: Forward derivative of our simplified multi-layer perceptron according to input neuron x_2. Sample $\mathbf{X}$ is benign and $\mathbf{X}^*$ is adversarial, crafted by adding $\delta_{\mathbf{X}} = (0, \delta x_2)$.

因此，在敌手使用正向导数时，要专注于给定输入值的较大导数，让寻找更加有效，最终导向更小的扭曲

由上可得:

- 输入小的变动会导致神经网络输出极大变化
- 并不是所有的输入域都适合找到对抗样本
- 前向导数减小了对抗样本的寻找空间

Generalizing to Deep Neural Networks

将上面的原理推广到 DNN模型中，对于任意无环的DNN

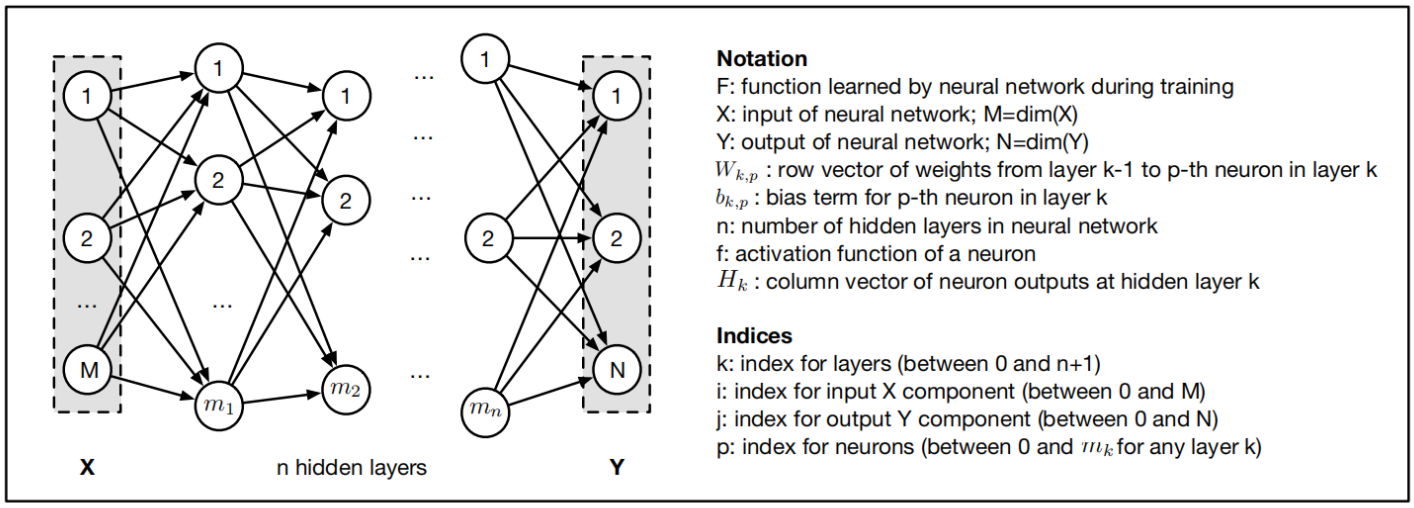

Figure 6: Example architecture of a feedforward deep neural network with notation used in the paper.

Algorithm 1 Crafting adversarial samples

$\mathbf{X}$ is the benign sample, $\mathbf{Y}^*$ is the target network output, $\mathbf{F}$ is the function learned by the network during training, Υ is the maximum distortion, and θ is the change made to features. This algorithm is applied to a specific DNN architecture and dataset in Algorithm 2.

Input: $\mathbf{X}$, $\mathbf{Y}^*$, $\mathbf{F}$, Υ, θ

1: $\mathbf{X}^* \leftarrow \mathbf{X}$
2: $\delta_{\mathbf{X}} \leftarrow \vec{0}$
3: $\Gamma = \{1 \dots |\mathbf{X}|\}$
4: **while** $\mathbf{F}(\mathbf{X}^*) \neq \mathbf{Y}^*$ and $||\delta_{\mathbf{X}}|| < \Upsilon$ **do**
5: Compute forward derivative $J_{\mathbf{F}}(\mathbf{X}^*)$
6: $S(\mathbf{X}, \mathbf{Y}^*) = \texttt{saliency\_map}\,(J_{\mathbf{F}}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$
7: $i_{max} \leftarrow \arg\max_i S(\mathbf{X}, \mathbf{Y}^*)[i]$
8: Modify $\mathbf{X}^*_{i_{max}}$ by θ
9: $\delta_{\mathbf{X}} \leftarrow \mathbf{X}^* - \mathbf{X}$
10: **end while**
11: **return** $\mathbf{X}^*$

其基本步骤为：

1. 计算前向导数$J_F(X^*)$
2. 基于前向导数构建一个显著图S
3. 通过θ修改输入特征i_{max}
4. 重复123直到获得Y^*或者到达最大扰动Υ

Forward Derivative of a Deep Neural Network

第一步就是对给定的样本X计算前向导数，如下

$$J_F(X) = [\frac{\partial F(X)}{\partial X}] = [\frac{\partial F_j(X)}{\partial x_i}]_{i \in 1 \dots M, j \in 1 \dots N}$$

这个与神经网络的反向传播很像，但是我们只取其网络的导数而不是损失函数，根据其特征取值而不是网络参数。

因此不是反向梯度，而是正向梯度，这能使我们更好的发现导致明显输出变化的输入组件

从第一个DNN的隐藏层开始，可以根据输入组件来区分第一个隐藏层的输出，然后根据先前的依次递归后面的隐藏层。

$$\frac{\partial H_k(X)}{\partial x_i} = [\frac{\partial f_{k,p}(W_{k,p} \cdot H_{k-1} + b_{k,p})}{\partial x_i}]_{p \in 1...m_k}$$

其中

- H_k是隐藏层k的输出向量
- $f_{k,j}$是第k层神经元j的激活函数
- 在$k \in 1..n+1$的隐藏层或者输出层的每一个神经元p都和第$k-1$层以向量$W_{k,p}$的权重进行连接（通过相应地定义权重矩阵，我们可以定义完全或稀疏连接的夹层，从而建模各种架构）
- $b_{k,p}$是第k层神经元p的偏差

应用以上规则，应用链式法则，我们可以得到

$$\frac{\partial H_k(X)}{\partial x_i}|_{p \in 1...m_k} = (W_{k,p} \cdot H_{k-1}) \times \frac{\partial f_{k,p}}{\partial x_i}(W_{k,p} \cdot H_{k-1} + b_{k,p})$$

于是我们能够计算$\frac{\partial H_n}{\partial x_i}$
神经元j的输出是计算如下表达式：

$$F_j(X) = f_{n+1,j}(W_{n+1,p} \cdot H_n + b_{n+1,p})$$

因此，我们再次运用链式法则来获得$J_F[i,j](X)$

$$\frac{\partial F_j(X)}{\partial x_i} = (W_{n+1,j} \cdot \frac{\partial H_n}{\partial x_i}) \times \frac{\partial f_{n+1,j}}{\partial x_i}(W_{n+1,j} \cdot H_n + b_{n+1,j})$$