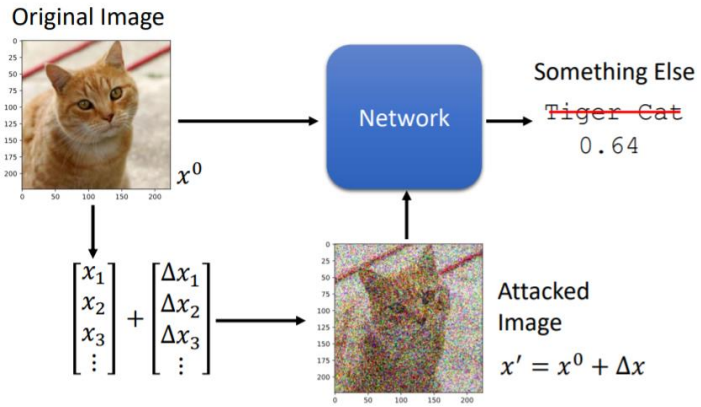
Attack and Defense

Motivation

- We seek to deploy machine learning classifiers not only in the labs, but also in real world.
- The classifiers that are robust to noises and work "most of the time" is not sufficient. 光是強還不夠
- We want the classifiers that are robust the inputs that are built to fool the classifier. 應付來自人類的惡意
- Especially useful for spam classification, malware detection, network intrusion detection, etc.

What do we want do?

Attack 要做的事就是把找到原图x^0对应的x'

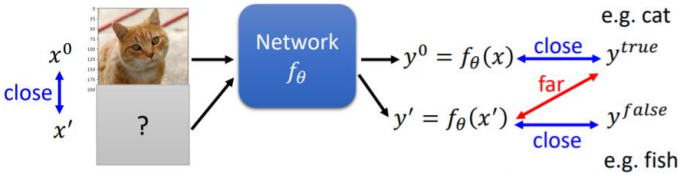


Loss Function for Attack

Training: network 训练得出的结果y^0必须和y^{true}越接近越好，此时的 loss function 为$L_{train}(\theta) = C(y^0, y^{true})$,输入的$x$是固定的，需要不断调整$\theta$的值，使得$L_{train}(\theta)$取得最小值;

Non-targeted Attack: 如果我们需要 attack 一个 network，此时 network 的输出y'和y^{true}应该越大越好，此时的 loss function 为$L(x') = -C(y', y^{true})$,前面多一个负号。此时的 network 中的参数θ是固定的，需要不断调整输入x'，使 network 的输出y'和y^{true}的差距尽量远;

Targeted Attack：如果我们不仅想要y' ,y^{true}之间的距离尽量远，还想使y' ,y^{false}之间的距离尽量近，就需要使用 targeted attack；此时的 loss function 为$L(x') = -C(y', y^{true}) + C(y', y^{false})$



- ***Training:*** $L_{train}(\theta) = C(y^0, y^{true})$ x fixed
- ***Non-targeted Attack:*** $L(x') = -C(y', y^{true})$ θ fixed
- ***Targeted Attack:***
  $$L(x') = -C(y', y^{true}) + C(y', y^{false})$$
- ***Constraint:*** $d(x^0, x') \leq \varepsilon$ 不要被發現

不仅需要限制输出之间的差异，还需要限制输入x^0, x'之间的差异，只有这输入之间的差异d小于ϵ，我们才可以认为x'是与x^0相似的，才达到了 attack 一个 network 的目的，即使输入尽可能具有迷惑性，从而使网络输出的错误的结果

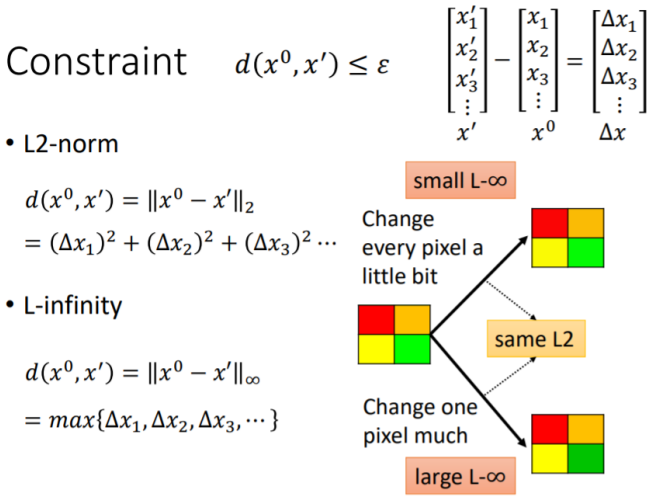
Constraint

那么我们怎么计算d呢?

有以下两种主要的方法:

- L2-norm,为x^0, x'之间每个像素差异的平方和;
- L-infinity，为x^0, x'之间每个像素差异的最大值;

如果我们改变图中的每个 pixel，另外一幅图只改变其中一个 pixel，使得这两者之间的 L2-norm 是一样的，但第二种方式得出的 L-infinity 更大



How to Attack

就像我们训练一个 Neural network 一样，但需要训练的参数是x',此时就需要找到一个参数x^*，来最小化$L(x')$,限制条件是$d(x^0, x') \leq \epsilon$

$x^* = \arg\min L(x')$

这里我们也使用了 gradient descent 算法，只是此时需要调整的参数变成了x^t

• Gradient Descent

Start from original image x^0
For t = 1 to T
$x^t \leftarrow x^{t-1} - \eta \nabla L(x^{t-1})$
If $d(x^0, x^t) > \varepsilon$
$x^t \leftarrow fix(x^t)$

$$\nabla L(x) = \begin{bmatrix} \partial L(x)/\partial x_1 \\ \partial L(x)/\partial x_2 \\ \partial L(x)/\partial x_3 \\ \vdots \end{bmatrix}$$

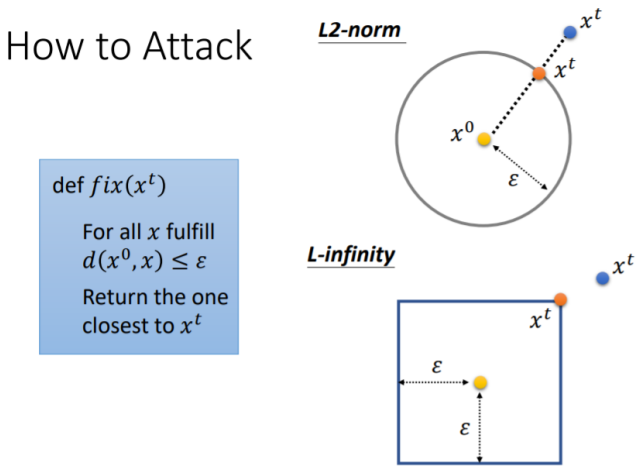
当$d(x^0, x^t) > \epsilon$时，就需要更新这个参数了，使用$fix(x^t)$来更新

Start from original image x^0
For t = 1 to T
$x^t \leftarrow x^{t-1} - \eta \nabla L(x^{t-1})$
If $d(x^0, x^t) > \varepsilon$
$x^t \leftarrow fix(x^t)$

def $fix(x^t)$
For all x fulfill $d(x^0, x) \leq \varepsilon$
Return the one closest to x^t

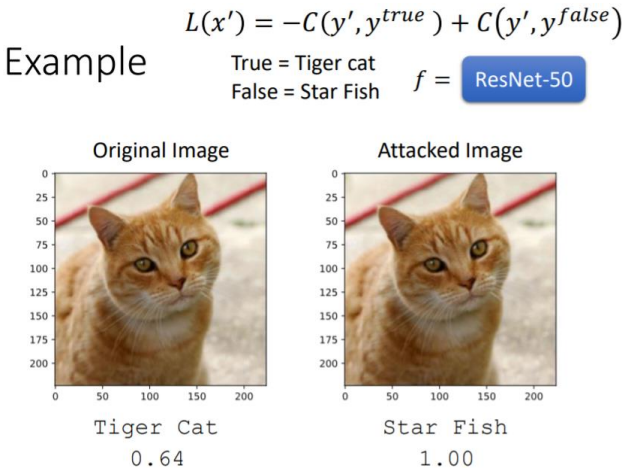
更新的参数需要满足一定的条件，

- 如果使用 L2-norm，必须选择差值在半径以内的参数，超过了这个半径，就设为ϵ;
- 如果使用 L-infinity，现在超过了这个方形的区域，就必须想办法把它拉回来，在y轴方向超过了ϵ，就把值设为ϵ，在x轴方向超过了ϵ，就把值设为ϵ

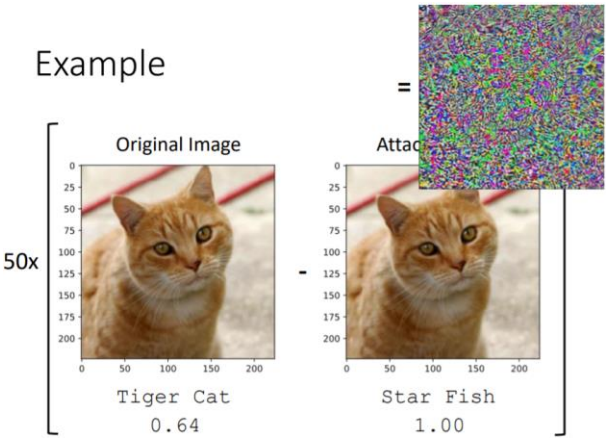


Example

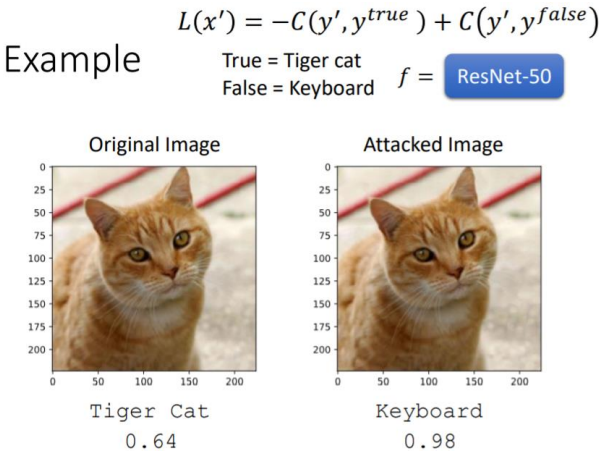
如果我们现在要 attack 一个 network，其真实类别为 Tiger cat，但 attack 之后的 network 认为这是 star fish



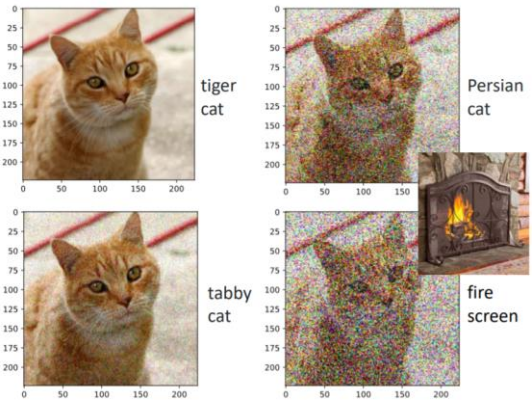
由于这两者之间的差异很小，很难识别，这里我们将差值*50 来进行展示



可能猫和猫之间是比较类似的，这里我们将 attack 的 target 设置为 keyboard，该 network 认为是 keyboard 的概率为 0.98



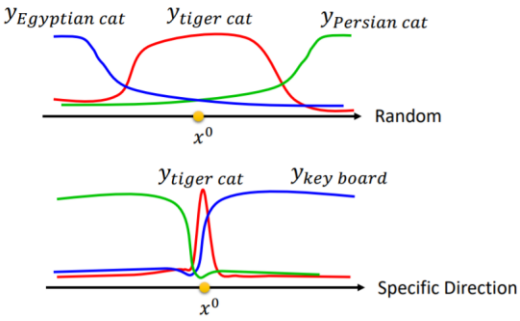
如果我们再对图片加入噪声，network 认为这是 Persian cat，再继续加入噪声，我们都快分辨不出这张图是一只猫了，network 就认为这是 fire screen



我们假设x_0是在高维平面上的一个点，沿着任意方向随机移动，我们可以看到在接近x^0的时候，是 tiger cat(正确分类)的可能性是很高的，但如果再移动多一点，是 Persian cat 和 Egyptian cat 的可能性是很高的

上述说的是随机方向进行移动，如果图片是 225*225pixel 的，那么就是 5 万个高维的方向，现在我们选取其中几个特定的方向，在这几个特定的方向中，x^0可变化的范围就变得非常狭窄，x^0稍微变化一下，network 输出为另一个类别(keyboard)的可能性就很高

What happened?



Attack Approaches

- FGSM (https://arxiv.org/abs/1412.6572)
- Basic iterative method (https://arxiv.org/abs/1607.02533)
- L-BFGS (https://arxiv.org/abs/1312.6199)
- Deepfool (https://arxiv.org/abs/1511.04599)
- JSMA (https://arxiv.org/abs/1511.07528)
- C&W (https://arxiv.org/abs/1608.04644)
- Elastic net attack (https://arxiv.org/abs/1709.04114)
- Spatially Transformed (https://arxiv.org/abs/1801.02612)
- One Pixel Attack (https://arxiv.org/abs/1710.08864)
- ..... only list a few

虽然有很多方法都可以用来 attack network，但这些方法的主要区别在于使用了不同的 constrains，或者使用了不同的 optimization methods

$$x^* = arg \min_{d(x^0,x')\leq\varepsilon} L(x')$$

Different optimization methods

Different constraints

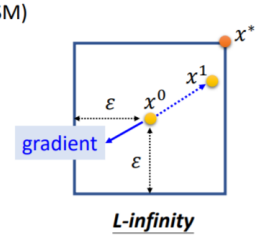
我们先介绍一下 FGSM，对于第一堆，如果$\frac{\partial L}{\partial x_1}$是大于 0 的值，那么$\Delta x_1 = sign\left(\frac{\partial L}{\partial x_1}\right) = +1$;如果对于第二堆，如果$\frac{\partial L}{\partial x_2}$小于 0 的值，不管值多大，都得出$\Delta x_2 = sign\left(\frac{\partial L}{\partial x_2}\right) = -1$;即对于$x^0$的所有维，要么$+\epsilon$，要么$-\epsilon$，即可得到最好的结果$x^*$

- Fast Gradient Sign Method (FGSM)

$$x^* \leftarrow x^0 - \varepsilon \Delta x$$

$$\Delta x = \begin{bmatrix} sign(\partial L/\partial x_1) \\ sign(\partial L/\partial x_2) \\ sign(\partial L/\partial x_3) \\ \vdots \end{bmatrix}$$

only have **+1** or **-1**



L-infinity

FGSM 使用 L-infinity 作为 distance constrain，如果 gradient 指向左下角，那么x^1就在方框的右上角；如果 gradient 指向左上角，那么x^1就在方框的右下角；因此，在 FGSM 里面，我们只在意 gradient 的方向，不在意器具体的大小

那么 FGSM 到底是怎么运作的呢?

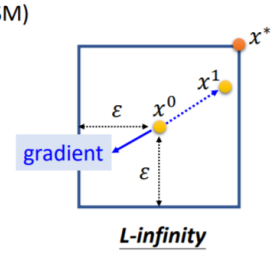
我们可以看作 FGSM 是使用了非常大的一个 learning rate，使x飞出了方形区域，由于 L-infinity 的限制，输出会被限制到方形区域内部，即x^*在方形区域的右上角

- Fast Gradient Sign Method (FGSM)

$$x^* \leftarrow x^0 - \varepsilon \Delta x$$

$$\Delta x = \begin{bmatrix} sign(\partial L/\partial x_1) \\ sign(\partial L/\partial x_2) \\ sign(\partial L/\partial x_3) \\ \vdots \end{bmatrix}$$

only have **+1** or **-1**



L-infinity

White Box vs Black Box

在之前的 attack 中，我们假设已经知道了 network 的参数θ，目标是找到最优化的x',这种 attack 就称为 White Box

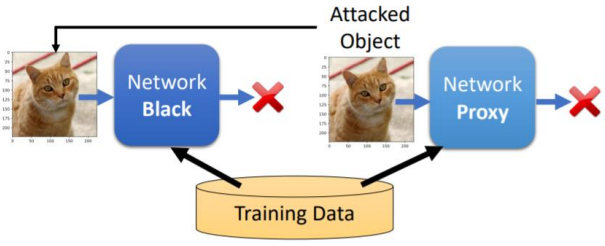
但在大多数的情况下，我们都不知道 network 的参数，但也需要去 attack 这个 network，这就是 Black Box

Black Box Attack

如果我们现在已经知道了 black network 的 training data，那么我们就可以用同样的 training data 来训练一个 proxy network，再生成 attacked object，如果能过成功 attack 新的 proxy network，那么我们就可以把这个 object 也作为 black network 的输入，也可以 attack 成功

如果不能得到相应的 training data，network 如果是越过在线版本，我们可以输入大量的图片，得出相对应的分类结果，从而可以组合成相应的训练材料，来得出 proxy network

- If you have the training data of the target network
  - Train a proxy network yourself
  - Using the proxy network to generate attacked objects
- Otherwise, obtaining input-output pairs from target network



也有一些实验数据证明黑箱攻击是有可能成功的，现在假设 Black Box 有五种，现在我们来训练 proxy network，我们用 ResNet-152 生成的图片，如果 black box 的 network 也是 ResNet-152，那么 attack 成功的几率就会非常高，表格中的 4%表示系统辨识的准确率

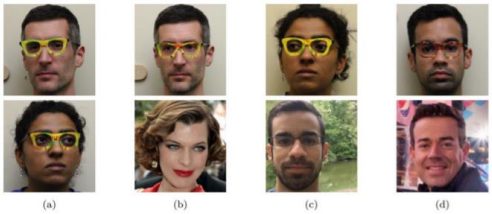
- If you have the training data of the target network
  - Train a proxy network yourself
  - Using the proxy network to generate attacked objects
- Otherwise, obtaining input-output pairs from target network

		Black				
		ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
Proxy	ResNet-152	4%	13%	13%	20%	12%
	ResNet-101	19%	4%	11%	23%	13%
	ResNet-50	25%	19%	5%	25%	14%
	VGG-16	20%	16%	15%	1%	7%
	GoogLeNet	25%	25%	17%	19%	1%

https://arxiv.org/pdf/1611.02770.pdf

Attack in the Real World

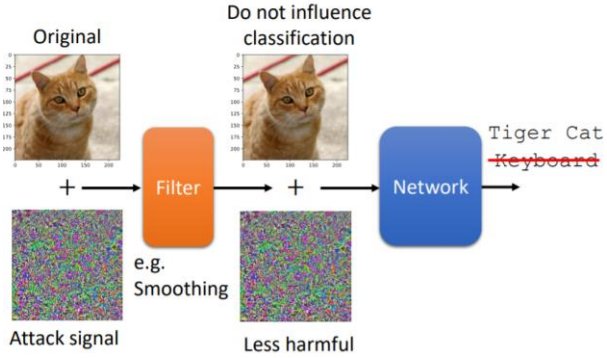
1. An attacker would need to find perturbations that generalize beyond a single image.
2. Extreme differences between adjacent pixels in the perturbation are unlikely to be accurately captured by cameras.
3. It is desirable to craft perturbations that are comprised mostly of colors reproducible by the printer.



Defense

- Adversarial Attack cannot be defended by weight regularization, dropout and model ensemble.
- Two types of defense:
  - ***Passive defense***: Finding the attached image without modifying the model
    - Special case of Anomaly Detection
  - ***Proactive defense***: Training a model that is robust to adversarial attack

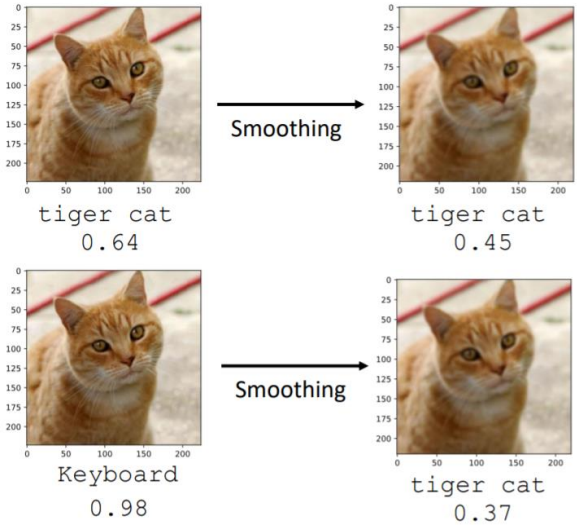
Passive defense



这个 filter 可以是 smoothing，对于原来输入的 attack 图片 network 认为其是 keyboard，经过 smoothing 之后，network 就认为是 tiger cat 了，是正确的分类结果

Q: 那么为什么 smoothing 可以达到这种效果呢?

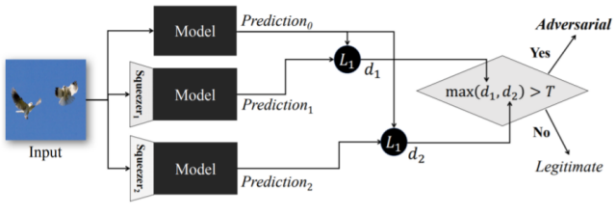
A: 只有某几种方向上的信号可以是 attack 成功，如果使用了 smoothing 这种 filter，就把这几种信号改变了，那么 attack 就失效了；加上 smoothing 并不会伤害原来的图片，所以 network 仍然可以得出正确的结果



根据这种思想，有学者就提出了 feature squeeze

对于同一个 input，我们先得出 model 的输出结果$Prediction_0$，再根据$Squeeze_1, Squeeze_2$得出结果$Prediction_1, Prediction_2$,如果$Prediction_0$和$Prediction_1, Prediction_2$之间的差值$d$很大，那么我们就可以认为 input 是来 attack 的

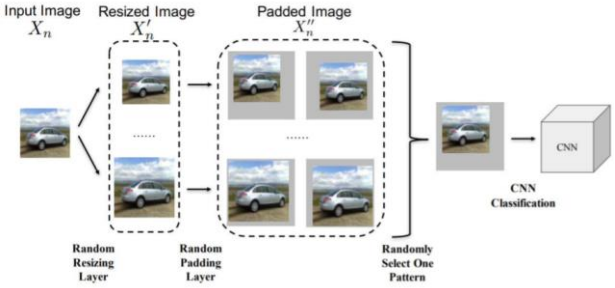
- Feature Squeeze



https://arxiv.org/abs/1704.01155

还有另一种方法

Randomization at Inference Phase



https://arxiv.org/abs/1711.01991

Proactive defense

首先我们通过某种算法找出漏洞，找到相应的 adversarial input，再把这些 input 和之前的 training data 一起作为新的 input data，输入 network，相当于进行了 data augmentation，这个过程进行 T 次，每次的 input data 都是不一样的

Given training data $X = \{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \cdots, (x^N, \hat{y}^y)\}$

Using X to train your model

For t = 1 to T

For n = 1 to N 找出漏洞

This method would stop algorithm A, but is still vulnerable for algorithm B.

Find adversarial input $\tilde{x}^n$ given x^n by an attack algorithm Using algorithm A

We have new training data different in each iteration

$X' = \{(\tilde{x}^1, \hat{y}^1), (\tilde{x}^2, \hat{y}^2), \cdots, (\tilde{x}^N, \hat{y}^y)\}$ Data Augmentation

Using both X' to update your model 把洞補起來

如果 attack 知道了我们是使用算法 A 进行模拟，那么 attack 可以使用算法 B 来进行 attack，那么我们的 network 并不能抵御这种 attack