Adversarial attacks

Alexander Binder

University of Oslo (UiO)

March 24, 2021

UiO : Department of Informatics
University of Oslo

Learning goals

- be able to explain how to use gradients for targeted and untargeted attacks
- be able to explain why adversarial attacks can be created
- be able to categorise adversarial attacks
- be able to explain the principles behind black box attacks
- be able to summarise the working principles of attack types shown in this lecture
- be able to explain two basic directions of defenses

⊙ ...

Outline

How a targeted attack can look like

Goal: take an image, and create an image that looks very similar but has a totally nonsense prediction for it!

- inputs: a pretrained neural network for multi-class classification $f(\cdot)$, an input image x,
- input image has prediction for a class $c^0 = \mathrm{argmax}_c f_c(x)$.
- **goal:** create a very similar image $z \approx x$ such that we predict a specific other class a.
  We want to have for a target class $a \neq c^0$: $a = \mathrm{argmax}_c f_c(\mathbf{z})$
  (different from the neural network prediction for x)

How a targeted attack can look like

- for step $t = 0$ initialize $z_0 := x$
- **iterate in a while loop until** $a = \mathrm{argmax}_c f_c(z_t)$ – target class a has highest score:
    - compute gradient of f for the target class a with respect to the input data: $\nabla^{(z_t)} f_a(z_t)$
    - gradient **ascent**: apply it with a small stepsize η to the input

$$z_{t+1} = z_t + \eta \nabla^{(z_t)} f_a(z_t)$$

    - best practice: $f_a(\cdot)$ not softmax. Use the logits from the last linear layer before the softmax. Why?
- find a step size η small enough that differences are barely visible.
- plot the difference of the original image as loaded versus the modified image, compute the mean absolute difference

This is a special form of attack: targeted towards a class, a whitebox one – means you have access to the classifier internals

Suppose you have an image x which is classified into class a and you want it to be misclassified. $f_c(\cdot)$ is the prediction for class c

Goal is to create an image $z \approx x$ such that:

$$a = \mathrm{argmax}_c f_c(x) \text{ is the orig pred.}$$

$$f_a(z) < \max_{c \neq a} f_c(z)$$

$$\|x - z\| < \epsilon$$

where $f_c(\cdot)$ can be either a softmax layer or the logits from the last layer before it.

Similar idea: compute the gradient and (1st variant) perform

- gradient descent on f_a: $z_{t+1} = z_t - \eta \nabla^{(z_t)} f_a(z_t)$
- until misclassification: $f_a(z) < \max_{c \neq a} f_c(z)$

Suppose you have an image x which is classified into class a and you want it to be misclassified:

$$f_a(z) < \max_{c \neq a} f_c(z)$$
$$\|x - z\| < \epsilon$$
$$\text{where } a = \mathrm{argmax}_c f_c(x)$$

Compute the gradient, perform either

- gradient descent on f_a:

$$z_{t+1} = z_t - \eta \nabla f_a(z_t)$$

- or a step in a direction with minimizes the score for f_a and maximizes the score for another class $c \neq a$ for example:

$$g_t = \mathrm{argmax}_{c \neq a} \nabla f_c(z_t)$$
$$z_{t+1} = z_t - \eta \; ???$$

Suppose you have an image x which is classified into class a and you want it to be misclassified.

Compute the gradient, perform either

- gradient descent on f_a:

$$z_{t+1} = z_t - \eta \nabla f_a(z_t)$$

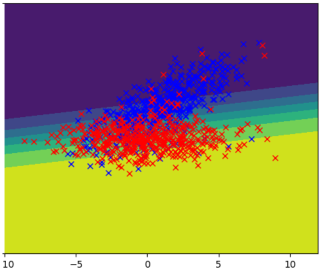
- or a step in a direction with minimizes the score for f_a and maximizes the score for another class $c \neq a$ for example:

$$g_t = \mathrm{argmax}_{c \neq a} \nabla f_c(z_t)$$
$$z_{t+1} = z_t - \eta(\nabla f_a(z_t) - \nabla f_{g_t}(z_t))$$

- Anotther form of attack: untargeted, a whitebox one.

Outline

The decision boundaries as in the typical ML class is shown in above graphic.



This naive view of

- smooth boundaries,
- with spaces densely occupied in by training data between

is wrong. Boundaries in deep learning problems are different!

The most important wrong assumption: the whole space was densely filled with training examples during training.

The reality of decision boundaries

The reality: see Fig3 in: https://arxiv.org/pdf/1802.08760.pdf as example. In many regions heavily fragmented decision regions

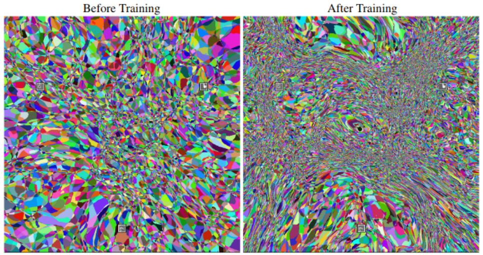


Figure 3: Transition boundaries of the last (pre-logits) layer over a 2-dimensional slice through the input space defined by 3 training points (indicated by inset squares). **Left**: boundaries before training. **Right**: after training, transition boundaries become highly non-isotropic, with training points lying in regions of lower transition density. See §A.5.3 for experimental details.

The reality of decision boundaries

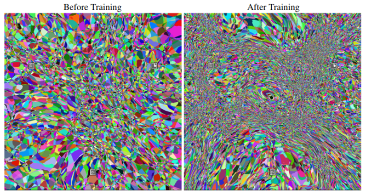
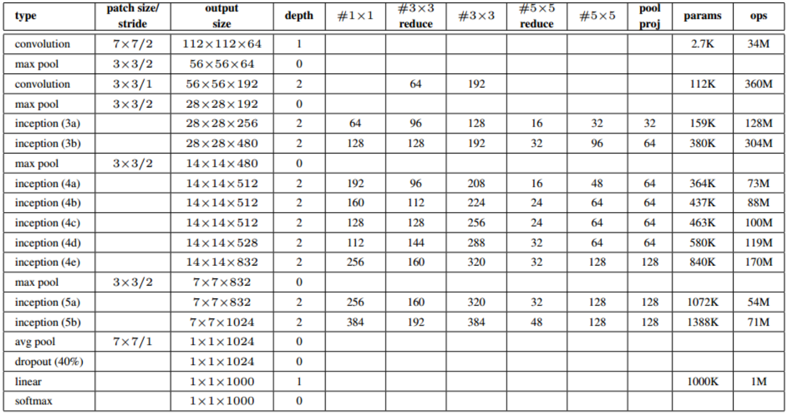


Figure 3: Transition boundaries of the last (pre-logits) layer over a 2-dimensional slice through the input space defined by 3 training points (indicated by inset squares). **Left**: boundaries before training. **Right**: after training, transition boundaries become highly non-isotropic, with training points lying in regions of lower transition density. See §A.5.3 for experimental details.

but why?

data distribution in high dimensional spaces

We take images, and process them in a neural network. Layer by layer activations get computed. If a layer has K neurons, then the space of all possible activation vectors is a K-dimensional vector space:



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

data distribution in high dimensional spaces

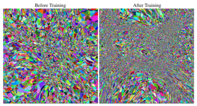
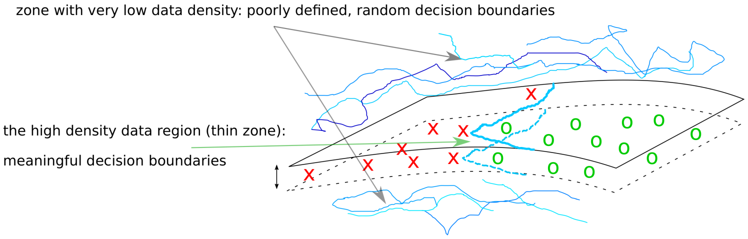


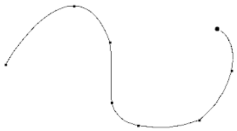
Figure 3: Transition boundaries of the last (pre-logits) layer over a 2-dimensional slice through the input space defined by 3 training points (indicated by inset squares). **Left**: boundaries before training. **Right**: after training, transition boundaries become highly non-isotropic, with training points lying in regions of lower transition density. See §A.5.3 for experimental details.

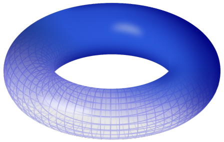
- neural networks map inputs into a high dimensional space. A large fraction of the training data is mapped around a low-dimensional manifold of the feature space – because similar images should end up close to each other
- Outside of the zone of high training data density - the decision boundaries (blueish colors in the fig below) are poorly defined. There is hardly any data around to define them!


zone with very low data density: poorly defined, random decision boundaries
the high density data region (thin zone):
meaningful decision boundaries
X
O

Manifolds informally

What are manifolds? curves are 1-dim manifolds, a curved 2-dim hyperplane, like a torus, would be a 2-dim manifold.





Higher order manifolds cannot be drawn trivially, but defined mathematically:

The set of points $x \in \mathbb{R}^d$ which are roots to a set of c differentiable functions $\{x \in \mathbb{R}^d : f_1(x) = 0, \dots, f_c(x) = 0\}$, are a $d - c$-dimensional manifold.

Manifolds informally

Why does this mapping onto a low-dim manifold happen?

An intuition, not claiming a truth / a universal truth / a truth free from counter examples.

- One view: To learn means to express that some objects are more similar than others. This requires to focus on what is defining similarity and therefore to discard irrelevant information which does not contribute to understand those similarities. To discard information can be achieved by compressing data into a set of points of lower dimensionality.
- A linear neuron would achieve this mapping similar things close together by mapping inputs onto a lower-dim hyperplane – by projection of all data (feature maps of the preceding layers) along the direction defined $\pm w$!!!
- Another view: To learn means to map objects to be more close to each other than dissimilar ones. To map similar objects close together in space cannot be achieved by uniformly distributing all points. One has to create a non-uniform density of samples.

An intuition about the distribution of data in a high dimensions

In short:

- for high dimensional representations, large parts of the feature space have low training data density ("outlier regions"), while most training data is mapped onto some lower-dimensional subspace (like a circle in 2-d).
- Therefore: Exist many directions in a high dim feature space such that small step $< \epsilon$, leads outside the zone the training data density is high
- in zones with low training data densities – the decision boundaries of several classes can lie very close together (bcs they were never specified by learning from training labels!!).
- Therefore: in zones with low training data densities – a series of small steps can lead to big changes in predicted labels

Manifolds informally

What is the consequence?

takeaway

- data in feature maps is not uniformly distributed in the space of all possible values of the feature map
- most samples are mapped onto thin regions around lower-dimensional spaces
- The decision boundaries in outlier regions are not (well-) specified by training
- adversarial attacks exploit this and create samples in outlier regions

Outline

one way to classify attacks

targeted vs untargeted attacks

- targeted: sample x to be classified into a certain fixed class.
- untargeted: sample x to be misclassified relative to its originally predicted label.

another way to classify attacks

white box vs black box

- Black box: one has access to the outputs of prediction models only.
- White box attacks: one has access to all details of the prediction model as a whole. Can be quite realistic if one can guess... (e.g. github of collaborators)
- Grey box attacks: anything in between, e.g. attacker has access to some layers (federated learning), or to training data used.

another way to classify attacks

Most white box attacks follow the same idea: compute a gradient of some sort.

- compute a gradient of the training loss function with one hot labels for the predicted class (gradient descent), or for a target class (gradient ascent)
- OR compute a gradient of the logits of the neural network, not of the softmax (softmax saturates like a sigmoid), again for the ground truth class or for the target class

some problems to adress when creating attacks

Your result is possibly not a valid image for three reasons:

A Images after *ToTensor*() should be in $[0, 1]$, for some pixels it can be violated after the gradient updates.

B After optimization, our values for subpixels are floats . When saved, the image subpixel values get rescaled to $[0, ..., 255]$ and rounded to the integers in that range. PIL usually expects inputs as integers in $\{0, \dots, 255\}$. **The rounding of floats to uint8 before saving an image may change the prediction and destroy an adversarial sample!!**
you need to validate that your saved samples are still adversarial.

C Image saving might introduce additional biases, e.g. changes in subpixel values due to lossy compression, for example when saving as jpg. Save images as png without lossy compression solves this.

some problems to adress when creating attacks

Possible solutions for A and B:

Fix for the out of bounds problem is to perform a gradient descent in a suitable subspace (if we use *descent*). Store in a temporary variable how the input z_{t+1} would look like after an update:

$$tmp = z_t - \epsilon \nabla^{(z_t)} f(z_t)$$

Find now all those dimensions of the input sample where the resulting image would be out of bounds:

$$Bad = \{d : tmp_d * std + mean < 0 \text{ or } tmp_d * std + mean > 1\}$$

In practice I like a safety margin. Then set the gradient to zero on this set *Bad* of pixels, and apply it to update the current x for the next step.

$$\begin{aligned} Bad &= \{d : tmp_d * std + mean \notin [2/255, 253/255]\} \\ h_d &:= (\frac{\partial f}{\partial z^{(d)}}(z_t) \text{ if } d \notin Bad, 0 \text{ else}) \\ z_{t+1} &= z_t - \epsilon h \end{aligned}$$

some problems to adress when creating attacks

Assume we use a gradient descent:

$$\begin{aligned} tmp &= z_t - \epsilon \nabla^{(z_t)} f(z_t) \\ Bad &= \{d : tmp_d * std + mean \notin [2/255, 253/255]\} \\ h_d &:= (\frac{\partial f}{\partial z^{(d)}}(z_t) \text{ if } d \notin Bad, 0 \text{ else}) \\ z_{t+1} &= z_t - \epsilon h \end{aligned}$$

Question:

- Why with this modified gradient h the objective function cannot increase ? It will either decrease or stay constant.

some problems to adress when creating attacks

A fix for the discretization problem is harder.

- One idea: at first optimize until target class has the highest score. Test for termination, whether the image after discretization still fools the net. If not, continue to increase the score.
- A more theoretically founded way: if g is the gradient, and v is a vector such that $g \cdot v > 0$, then for sufficiently small steps in the direction of v the function f_t (for which the gradient g was computed) will increase.

  For your given image x find a subpixel-value-rounded image $\hat{x}$ (every rgb subpixel it has two neighbors to round) such that $g \cdot (\hat{x} - x) > 0$, that is changing the image x to the rounded image $\hat{x}$ would increase the score:

$$g \cdot (\hat{x} - x) = \sum_d g_d(\hat{x}_d - x_d)$$

some problems to adress when creating attacks

A fix for the discretization problem is harder.

- A more theoretically founded way: if g is the gradient, and v is a vector such that $g \cdot v > 0$, then for sufficiently small steps in the direction of v the function f_t (for which the gradient g was computed) will increase.

  For your given image x find a subpixel-value-rounded image $\hat{x}$ (every rgb subpixel it has two neighbors to round) such that $g \cdot (\hat{x} - x) > 0$, that is changing the image x to the rounded image $\hat{x}$ would increase the score:

  $$g \cdot (\hat{x} - x) = \sum_d g_d(\hat{x}_d - x_d)$$

  So you can choose for every subpixel here the one of the two roundings for which $g_d(\hat{x} - x)_d > 0$.
- Caveat: $|\hat{x}_d - x_d| \leq \frac{1}{255}$ could be still a too large step

Universal adversarial attacks

So far we had to compute one perturbation for every input.

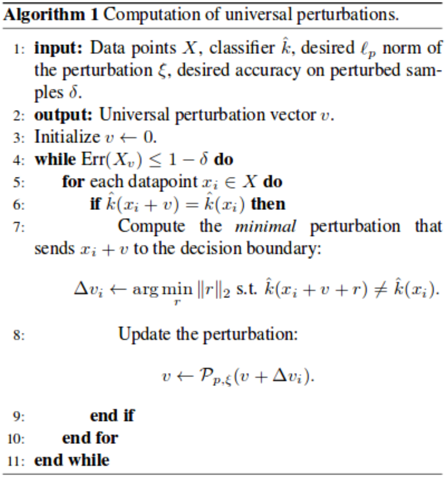
There are perturbations which can hinder predictions for large sets of input images! However one cannot control anymore to what target class.

Universal adversarial attack: one perturbation which can be applied to many input samples, and which has a chance to cause misclassifications of a large percentage of them. [1]

Success rates are $75\% - 90\%$ (and can be likely better if one would cluster the space and compute a mix of perturbations) Moosavi-Dezfooli et al. CVPR 2017: https://arxiv.org/pdf/1610.08401.pdf, Hayes et al. https://arxiv.org/abs/1708.05207.

[1] directed?undirected?

Universal adversarial attacks



Algorithm 1 Computation of universal perturbations.

```
1:  input: Data points $X$, classifier $\hat{k}$, desired $\ell_p$ norm of
    the perturbation $\xi$, desired accuracy on perturbed samples $\delta$.
2:  output: Universal perturbation vector $v$.
3:  Initialize $v \leftarrow 0$.
4:  while $\mathrm{Err}(X_v) \leq 1 - \delta$ do
5:      for each datapoint $x_i \in X$ do
6:          if $\hat{k}(x_i + v) = \hat{k}(x_i)$ then
7:              Compute the *minimal* perturbation that
    sends $x_i + v$ to the decision boundary:
                $\Delta v_i \leftarrow \arg\min_r \|r\|_2$ s.t. $\hat{k}(x_i + v + r) \neq \hat{k}(x_i)$.
8:              Update the perturbation:
                    $v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i)$.
9:          end if
10:     end for
11: end while
```

The algorithm is unpleasantly simple. It is out of exams. Understanding what an Universal Adversarial Perturbation is, is exam stuff.

https://arxiv.org/pdf/1610.08401.pdf,
https://arxiv.org/abs/1708.05207

Outline

targeted Iterative gradient:

key idea for graded knowledge

simple gradient for a fixed target class c: white box, targeted.
Iterate until class c is predicted.

$$x_{n+1} = x_n + \epsilon \frac{\partial f_c}{\partial x}(x_n)$$

This is gradient ascent towards target class.
Important: it is much better if one **uses logits** and not the softmax probabilities.

The softmax will be problematic if $f_c(x)$ is close to one, because then gradients are near zero, while in earlier stages with larger gradients one needs much smaller step sizes.

untargeted Iterative gradient:

key idea for graded knowledge

white box, untargeted. Iterate until prediction switches from the original class either

$$c^0 = \mathrm{argmax}_c f_c(x_0)$$
$$x_{n+1} = x_n - \epsilon \frac{\partial f_{c^0}}{\partial x}(x_n)$$

This is gradient descent away from target class.

$$\text{or } c^* = \mathrm{argmin}_c f_c(x_n)$$
$$x_{n+1} = x_n + \epsilon \frac{\partial f_{c^*}}{\partial x}(x_n)$$

This is gradient ascent towards the least probable class.

Use logits, not softmax.

untargeted Iterative gradient:

Remarks

- gradient descent away from target class case: Note the sign difference to the targeted case.
- The prediction switches when the current label is not the original label of image x_0, that is: $\mathrm{argmax}_c f_c(x_n) \neq \mathrm{argmax}_c f_c(x_0)$.
- one can also combine the descent from the original class and the ascent towards another class.
- The softmax will be problematic if $f_c(x)$ is close to one, because then gradients are near zero.

Projected gradient descent (PGD)

- untargeted Iterative gradient with one added restriction: keeping of the current result x_t close by ϵ with respect to the original sample x_0.
- projection onto ϵ-radius balls around the original sample x_0 with respect to well known-norms ℓ_2, ℓ_∞ are common and easy to solve.
  Example: the projection on the ϵ-ball around x_0:
  $\{x : \|x - x_0\| \leq \epsilon\}$ is easy with the euclidean norm:

$$\pi_{B_\epsilon(x_0), \|\cdot\|_2}(x_t) = x_0 + \epsilon \frac{x_t - x_0}{\|x_t - x_0\|}$$

Projected gradient descent (PGD)

- projection in general depends on a set S and a norm $\| \cdot \|$. It is the nearest point from that set under a norm.

$$\pi_{S,\|\cdot\|}(x_t) = \inf_{x \in S} \|x_t - x\|$$

Projection onto an arbitrary set can be very difficult to solve!

Projected gradient descent (PGD)

$$
\begin{aligned}
\text{either } c^* &= \mathrm{argmax}_c f_c(x_0) \\
\text{with } \tilde{x}_{n+1} &= x_n - \epsilon \frac{\partial f_{c^*}}{\partial x}(x_n) \\
\text{or } c^0 &\in \mathcal{C} \setminus \{c^*\} \\
\text{with } \tilde{x}_{n+1} &= x_n + \epsilon \frac{\partial f_{c^0}}{\partial x}(x_n) \\
\text{project: } x_{n+1} &= \pi_{B_\epsilon(x_0), \|\cdot\|_?}(\tilde{x}_{n+1})
\end{aligned}
$$

? stands for: it depends on what norm one wants to use. Again: either-or can be combined by adding those gradient updates.

Projected gradient descent (PGD)

ℓ_2-clipping to a maximal deviation of $\epsilon > 0$ around x_0:

$$\pi_{B_\epsilon(x_0),\|\cdot\|_2}(x_t) = x_0 + \epsilon \frac{x_t - x_0}{\|x_t - x_0\|}$$

for the ℓ_∞-norm: $\|v\|_\infty = \max_d |v_d|$:

ℓ_∞-clipping to a maximal deviation of $\epsilon > 0$ around x_0:

$\mathrm{clipto}(\epsilon)$ clips the changes of an image in every subpixel to at most ϵ difference to the original image x_0. The operations are applied to every dimension d:

$$\begin{aligned} clip_*(\epsilon)(x_d) &= \max(0, (x_0)_d - \epsilon, x_d) \\ \mathrm{clipto}(\epsilon)(x_d) &= \min(255, (x_0)_d + \epsilon, clip_*(\gamma)(x_d)) \\ \pi_{B_\epsilon(x_0),\|\cdot\|_\infty}(x) &= (\mathrm{clipto}(\epsilon)(x_d), d = 1, \ldots) \end{aligned}$$

Fast gradient sign:

https://arxiv.org/pdf/1412.6572.pdf:

$L(c, f(x_n))$ is the loss of predictor f for the class label c. In case of cross-entropy loss

$$L(c, f(x)) = -\log p(Y = c | X = x)$$

key idea for graded knowledge

Let c^* be the predicted class label. Then the fast gradient sign is based on the sign of the gradient of the training loss:

$$x_{n+1} = x_n + \epsilon \mathrm{sign}(\frac{\partial L(c^*, f)}{\partial x}(x_n))$$

Idea: This maximizes the loss between the prediction on x_n and its true label. ϵ chosen so large that 1 iteration is sufficient.

Fast gradient sign:

Why sign ? Figure 1 in https://arxiv.org/pdf/1611.02770.pdf on a complex dataset, ImageNet, suggests that fast sign is not good actually - but fast.

One possible advantage: distorting the gradient by taking the sign may help to avoid getting stuck in local extrema, thus increasing attack success rates at the cost of sometimes decreased quality.

This is white box, untargeted. Fast – bcs one aims at usually at 1 iteration, but often coarse images as results.

- pro: fast
- con: coarse images
- attack success rate can get low

Iterative gradient sign:

https://arxiv.org/pdf/1607.02533.pdf

We assume that we use the training loss for class c and predictor f in sample x_n: $L(c, f)(x_n)$

Variant 1: Let c^* **be the predicted class label for the original sample** $x_0 := x$, and we maximize the loss to it

$$x_{n+1} = \mathrm{clipto}(\gamma)(x_n + \epsilon \mathrm{sign}(\frac{\partial L(c^*, f)}{\partial x}(x_n)))$$

– walk in the direction that increases the loss between original class label and predictions. clip changes to γ relative to original image.

Iterative gradient sign:

Variant 2:

$$x_{n+1} = \mathrm{clipto}(\gamma)(x_n - \epsilon \mathrm{sign}(\frac{\partial L(c^*(x_n), f)}{\partial x}(x_n)))$$

where $c^*(x_n) = \mathrm{argmin}_c p(Y = c|X = x_n)$ is the **least likely prediction class of image** x_n in the current iteration – walk in the direction of the least likely class – converges faster than variant 1, see Fig2 in https://arxiv.org/pdf/1607.02533.pdf. This method works on non-trivial datasets!

Note the sign difference between variant 1 and variant 2. This is white box, untargeted.

Gradient-based attacks

You compute the gradient with respect to your input sample, not with respect to trainable parameters.

Carlini-Wagner Attack

Carlini & Wagner: https://arxiv.org/pdf/1608.04644.pdf, https://arxiv.org/pdf/1705.07263.pdf This is white box, targeted.

key idea for graded knowledge

- Goal: synth an image x which is close to a target image x_0 and which is classified as target class t
- optimize an objective of two components,
- first component $\|x - x_0\|^2$ makes synth an image x which is close to a target image x_0
- second component measures whether target class has highest score already
- use logit outputs of the classifier when possible
- white box, targeted

Carlini-Wagner Attack (targeted)

Given a target class t, and let $f_c(x)$ be the logits output of a classifier for class c (not the softmax!). Solve the following optimization problem

$$\min_x \|x - x_0\|^2 + c \max(d(x), 0)$$
$$d(x) = \max_{c \neq t} f_c(x) - f_t(x)$$

Why this objective?

- understand: $d(x) > 0$ means: $f_t(x) < \max_{c \neq t} f_c(x)$, which means that the prediction has not reached the target class t yet.
- cap $d(x)$ at zero, so that one does not minimize the objective by making the prediction of $f_t(x)$ very large ($d(x)$ very negative) while wandering far away from the start image x_0.

Important: it uses logits (the output of the last layer), not the softmax probabilities. Advantage: no problems with softmax saturation, no protection by defensive distillation. But: one cannot always access the logits!

Carlini-Wagner Attack (targeted)

Given a target class t, and let $f_c(x)$ be the logits output of a classifier for class c (not the softmax!). Solve the following optimization problem

$$\min_x \|x - x_0\|^2 + c \ \max(d(x), 0)$$
$$d(x) = \max_{c \neq t} f_c(x) - f_t(x)$$

Why this objective?

- $\|x - x_0\|^2$ makes synth an image x which is close to a target image x_0

Important: it uses logits (the output of the last layer), not the softmax probabilities. Advantage: no problems with softmax saturation, no protection by defensive distillation. But: one cannot always access the logits!

Carlini-Wagner Attack (untargeted)

Let c be the currently predicted class. Thus $f_c(x) > \max_{t \neq c} f_t(x)$.

$$\text{Goal: } \max_{t \neq c} f_t(x) > f_c(x) \Leftrightarrow 0 > f_c(x) - \max_{t \neq c} f_t(x)$$

Let $f_t(x)$ be the logits output of a classifier for class t (not the softmax!). Solve the following optimization problem

$$\min_x \|x - x_0\|^2 + c \max(d(x), -\eta), \; \eta > 0$$
$$d(x) = f_c(x) - \max_{t \neq c} f_t(x)$$

Why this objective?

- $\|x - x_0\|^2$ makes synth an image x which is close to a target image x_0

Important: it uses logits (the output of the last layer), not the softmax probabilities. Advantage: no problems with softmax saturation, no protection by defensive distillation. But: one cannot always access the logits!

Outline

overcoming black boxes

Generally make use of $(x, f(x))$, where $f(x)$ is the output of the blackbox. Two ideas.

- surrogate attacks: train an approximation to f (=the surrogate) and attack the surrogate (hope: attacks from surrogate would generalize to the actual target networks)
- boundary attacks: feel your way along the decision boundary while moving closer to the sample which one wants to corrupt. But always stay on the wrong side of the decision boundary

overcoming black boxes I: Surrogate attacks

https://arxiv.org/pdf/1602.02697.pdf

Setting: can observe only output $f(x)$ of target classifier.

key idea for graded knowledge

- train a replacement $g(x)$ which mimicks $f(x)$ and after that white-box attack the replacement $g(x)$.
- iterate training. at every step increase the training data set by augmenting the existing samples along the gradient of the surrogate

overcoming black boxes I: Surrogate attacks

Some finer details:

- iterate the following training. At each step r train a surrogate g mimicking f
  - input many samples x into f as queries, collect probability labels $f(x)$
  - train g using a dataset S_r of $(x, f(x))$
  - note: can use cross-entropy with soft labels as well

    $$L(g(x), f) = \sum_c -f_c \log g_c(x)$$

    Cross-entropy works NOT ONLY with one hot labels!
  - enlarge existing training dataset by augmentation: $S_{r+1} = S_r \cup \{x + \lambda \mathrm{sign}(\sum_c f_c(x) \nabla g_c(x)), x \in S_r\}$ Here one uses only the gradient of the surrogate.
- perform white box attacks on your trained surrogate $g(x)$

overcoming black boxes I: Surrogate attacks

- out of exams: uses a slightly different adversarial attack for the surrogate:

  Compute a saliency score for a target class t and every dimension i of a sample (image) x

$$S(x,t)[i] = \begin{cases} 0 & \text{if } \frac{\partial f_t}{\partial x_i} < 0 \text{ or } \sum_{c:c\neq t} \frac{\partial f_c}{\partial x_i} > 0 \\ \frac{\partial f_t}{\partial x_i} \left| \sum_{c:c\neq t} \frac{\partial f_c}{\partial x_i} \right| & \text{else} \end{cases}$$

idea: a dimension i of an input x is salient if either the gradient for the target class is positive or the sum of gradients along all other classes is negative.

goal: sparse selection of dimensions. May help to suppress noise from dimensions where the gradient of the surrogate does not match the gradient of the target due to a falsely learnt surrogate.

Results on larger networks known??

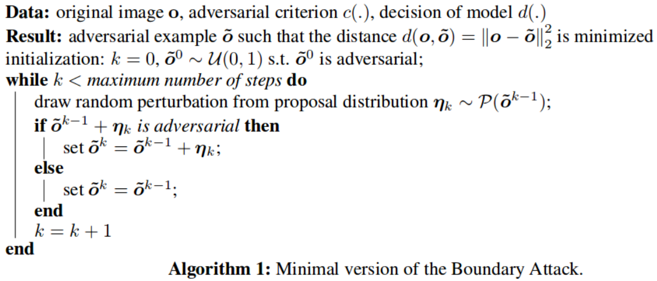
overcoming black boxes I: Boundary attacks

key idea for graded knowledge

- input is target image. output: synth image. synth image should look like target image, but have differing or wrong prediction (see examples in the paper)
- idea: move along decision boundary to the predicted class of target image, but always "on the wrong side" of the decision boundary.
- Principle: do not use gradients. Use rejection sampling to define moves.
- Sample perturbations from a distribution. Accept perturbed image if it is adversarial and one gets closer to the target.
- for accept/ reject one needs only the predicted label, no probabilities. The box is as black is it can get for this attack.
- See Fig 4 and Fig 7 in the paper.

https://openreview.net/pdf?id=SyZI0GWCZ

from the paper of Brendel and Bethge:



```
Data: original image $\mathbf{o}$, adversarial criterion $c(.)$, decision of model $d(.)$
Result: adversarial example $\tilde{\boldsymbol{o}}$ such that the distance $d(\boldsymbol{o}, \tilde{\boldsymbol{o}}) = \|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2^2$ is minimized
initialization: $k = 0$, $\tilde{\boldsymbol{o}}^0 \sim \mathcal{U}(0, 1)$ s.t. $\tilde{\boldsymbol{o}}^0$ is adversarial;
while $k <$ maximum number of steps do
    draw random perturbation from proposal distribution $\boldsymbol{\eta}_k \sim \mathcal{P}(\tilde{\boldsymbol{o}}^{k-1})$;
    if $\tilde{\boldsymbol{o}}^{k-1} + \boldsymbol{\eta}_k$ is adversarial then
        set $\tilde{\boldsymbol{o}}^k = \tilde{\boldsymbol{o}}^{k-1} + \boldsymbol{\eta}_k$;
    else
        set $\tilde{\boldsymbol{o}}^k = \tilde{\boldsymbol{o}}^{k-1}$;
    end
    $k = k + 1$
end
```

Algorithm 1: Minimal version of the Boundary Attack.

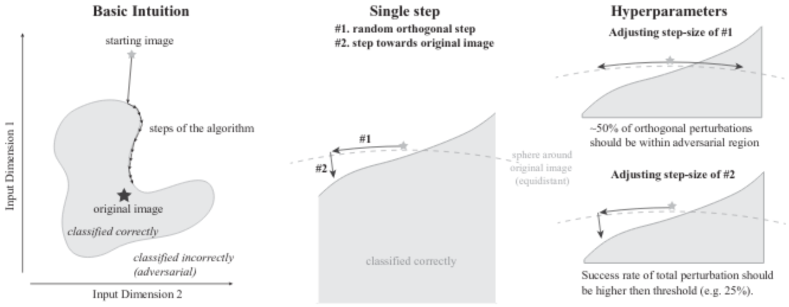

Basic Intuition
starting image
steps of the algorithm
original image
classified correctly
classified incorrectly (adversarial)
Input Dimension 1
Input Dimension 2
Single step
#1. random orthogonal step
#2. step towards original image
#1
#2
sphere around original image (equidistant)
classified correctly
Hyperparameters
Adjusting step-size of #1
~50% of orthogonal perturbations should be within adversarial region
Adjusting step-size of #2
Success rate of total perturbation should be higher then threshold (e.g. 25%).

Figure 2: (Left) In essence the Boundary Attack performs rejection sampling along the boundary between adversarial and non-adversarial images. (Center) In each step we draw a new random direction by (#1) drawing from an iid Gaussian and projecting on a sphere, and by (#2) making a small move towards the target image. (Right) The two step-sizes (orthogonal and towards the original input) are dynamically adjusted according to the local geometry of the boundary.

an exemplary result from the paper:

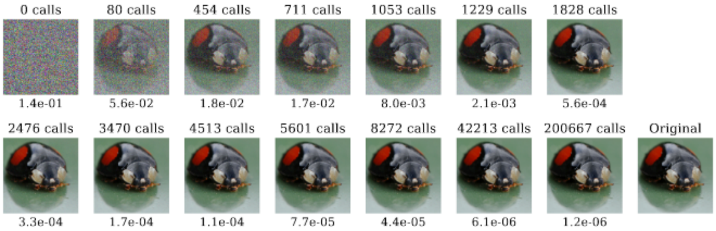


Figure 4: Example of an untargeted attack. Here the goal is to synthesize an image that is as close as possible (in L2-metric) to the original image while being misclassified (the original image is correctly classified). For each image we report the total number of model calls (predictions) until that point (above the image) and the mean squared error between the adversarial and the original (below the image).

Other attack ideas

- Training set poisoning: how to add samples, such that training will render your predictor useless
- attacks which aim to extract some kind of answers:
  - Membership inference attacks: Shokri et al. https://arxiv.org/abs/1610.05820 – was a training sample used to train a model?
  - Reconstruction Attacks: Dosovitsky et al. https://arxiv.org/abs/1506.02753 – given a feature map or a gradient, what was the input sample? Relevant for Federated Learning

Outline

Defenses: Two exemplary directions

- **Adversarial hardening:** make an ML model more robust to adversarials by training it with normal and adversarial samples e.g. https://arxiv.org/abs/1706.06083, however this is not easy to achieve. It does not generalize easily to different types of attacks: see https://arxiv.org/abs/1805.09190 and https://arxiv.org/abs/1904.13000. See also https://openreview.net/forum?id=SyJ7ClWCb which does hardening by using image transformations.

Defenses: Two exemplary directions

- problem 1: Mind the adaptive attacker: Attacker may use models with defenses and attack those defense-augmented model to create better evasive adversarials
- problem 2: the attacker is patient: damage can be done even if 1% of attacks passes

Defenses: Two exemplary directions

- no matter whether harden or detect:
- problem 1: Mind the adaptive attacker: Attacker may use models with defenses and attack those defense-augmented model to create better evasive adversarials
- problem 2: the attacker is patient: damage can be done even if 1% of attacks passes

Adversarial detection – some directions

- Idea: measure change of prediction on input x when the input is transformed.
- Hypothesis: predictions on adversarial perturbations change stronger than clean data.

Example 1: use ℓ_1-norm to measure changes in https://arxiv.org/abs/1704.01155 of the prediction $f(\cdot)$ under some transformation $t(\cdot)$:

$$s = \|f(x) - f(t(x))\|_1$$

in https://arxiv.org/abs/1704.01155 they proposed for $t(\cdot)$ a 2×2 median filter blur (replace lower left pixel by the median of pixels in a 2×2 neighborhood) and color-range squeezing.

another defense

Example 2: https://arxiv.org/abs/1705.08378 follows a detection idea by using a more complex transformation

- measure entropy of distribution of subpixel values. Entropy defined on p_i - probability of pixel value in the image being $i \in \{0 \ldots, 255\}$.
- discretize each rgb-pixel value in 2, 4 or 6 intervals (binning), depending on the value of the entropy
- on high-entropy images: also apply a convolution with some averaging filter, if this makes the resulting image closer to the original image
- Table 11: a defense aware attacker still succeeds 67% of all the time ...
- statistic s not defined?

out of class

https://openreview.net/forum?id=SyJ7ClWCb uses image transformations to remove attacks, but it does not measure the impact of image transformations on destroying predictions on clean samples. What is interesting is to consider the TV-regularization as smoothing (on those pixels which are not kept randomly) and the image quilting as potential transformations.

on transferability of attacks: outside of exams

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

non-targeted attacks generalize better across different models than targeted ones.

Chapter 5, table 4: **non-targeted ensemble attacks.** It helps to optimize against an ensemble of different classifiers to get strong attacks that are valid for many architectures.

Some papers: outside of exams

- https://arxiv.org/pdf/1511.04599.pdf – deepfool: whitebox, untargeted, works on Imagenet
- http://www.evolvingai.org/fooling Fooling as art
- https://arxiv.org/pdf/1707.08945.pdf – design attacks on images that look like physical-world plausible changes. hide them or make them look natural. Adversarial stickers for objects :-D .
- https://arxiv.org/pdf/1707.07397.pdf 3d print adversarial objects.
- https://arxiv.org/pdf/1705.07263.pdf detection is hard

Foolbox – from 0 to attack in 10 minutes

https://github.com/bethgelab/foolbox#example
very simple coding:

https://github.com/bethgelab/foolbox/blob/master/examples/single_attack_pytorch_resnet18.py

Your coding hw

This is Mr. Shout, a friend of Alex.

Questions?!