Adverserical Attacks

- **Robustness:** · If it returns correct output to all (perturbed) inputs
  ↳ Input space too large to be covered
- **Local Robustness:** · If it returns correct output to inputs similar to inputs in the training set
- **Why high accuracy is not enough?**
  · Accuracy measures from same distribution, where training/testing data was sampled
  · But there are similar inputs which are never tested because of their low probability
- **White Box Attack:** We know the model and the parameters
- **Black Box Attack:** We only know the architecture, not its weights
  ↳ Examples are transferable
- **Targeted Fast Gradient Sign Method**
  · Input: x
  · Target label: t such that $f(x) \neq t$
  · Perturbation: η such that $f(x+\eta) = t$

  1. Compute Perturbation: $\eta = \varepsilon \cdot \text{sign}(\nabla_x \text{loss}_t(x))$
  2. Perturb the original image: $x' = x - \eta$
  3. Check requirement: $f(x') \stackrel{?}{=} t$

  · Because FGSM is 1-step (1./2./3.) we are always between $[0,1] \Rightarrow$ No need to project

- Statt die weights im classifier zu verändern, ändern wir das Bild. Wir verwenden die Loss function und setzen als y-Wert das label t ein, somit optimieren wir in dessen Richtung. Wir verwenden $sign(x) = \begin{cases} -1, & \text{if } x<0 \\ 0, & \text{if } x=0 \\ 1, & \text{if } x>0 \end{cases}$ um es mit ε zu skalieren.

- Untargeted Fast Gradient Sign Method
  - Input: x
  - Perturbation: η such that $f(x+\eta) \neq f(x) = s$
  1. Compute perturbation: $\eta = \varepsilon \cdot sign(\nabla_x loss_s(x))$
  2. Perturb the original image: $x' = x + \eta$
  3. Check requirement: $f(x') \overset{?}{\neq} s$
  - With $x' = x + \eta$ we go in the opposite direction from the original label s

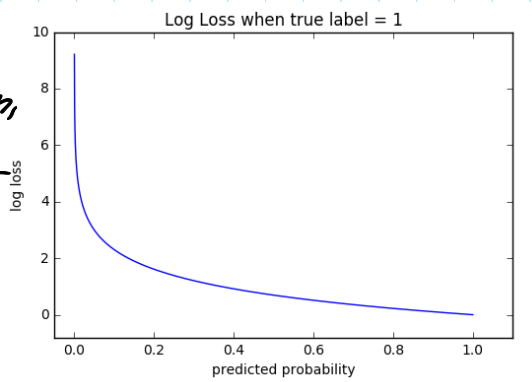
- Notion of distance
  - Do not overperturb image that human sees is
  - l_∞ norm: · Maximum noise (change) in any coordinate
    - $\|x - x'\|_\infty = \max_{i \in N} \|x_i - x'_i\|$
    - $\|[1, 5, -6]\|_\infty = 6$
    - Most naturally captures human vision

- Targeted Attack with small changes
  - $f(x+\eta) = t$ where $\|\eta\|_\infty$ is minimized
  - Optimization problem: · if $obj(x+\eta) \leq 0$ then $f(x+\eta) = t$
  - $obj(x+\eta) = -\log_2(p(t)) - 1$
  - ↳ $p(t) \geq 0{,}5 \Rightarrow obj(x+\eta) \leq 0$

find η
minimize $\|\eta\|_p + c \cdot obj(x+\eta)$
such that $x + \eta \in [0,1]^n$

- $\|\eta\|_\infty$ is a problem for gradient descend, because we are only optimizing the one component with the largest absolute value which yields to longer runtimes or in the worst case to oscillation because minimizing one component might increase others due to obj-term
  - Replace $\|\eta\|_\infty$ with $\sum_i \max(0, (|\eta_i| - \tau))$. lowering τ from 1 during optimization
    - is similar to L_∞ norm
    - τ is wie eine Schranke, die wir langsam nach unten lassen, um so mehr als ein component zu optimieren

- Cross Entropy loss
  - used for probabilities for classification
  - $\text{loss}_t(f(x)) = -\sum_{c=1}^{C} [c=t] \cdot \log(f(x))$
  - Wir möchten probability dass $f(x)=t$ ist maximieren, je höher die Wahrscheinlichkeit, desto niedriger der Loss



  - $\text{obj}(x') = -\log_c(f(x')) - 1$
    - outputs < 0 when $f(x') > 0.5$
  - $\frac{\partial}{\partial x_i}\|x\|_\infty = \begin{cases} 1 & \text{if } i \in \text{argmax}_k(|x_k|) \\ 0 & \text{else} \end{cases}$

- $\text{loss}(1) = 0$
- $\text{loss}(0) = \infty$

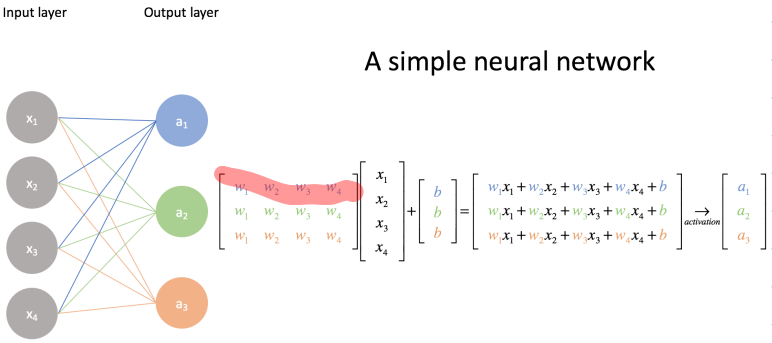
- find η
  minimize $\|\eta\|_p$
  such that $f(x+\eta) = t$
  $x + \eta \in [0,1]^n$

$\Rightarrow$ find project(η)
minimize $\Sigma_i \max(0, (|\eta_i| - \tau))$
$+ c \cdot obj(x+\eta)$

3 things need to be changed to ensure good optimization

- Replace it with $\Sigma_i \max(0, (|\eta_i| - \tau))$ which simulates L_∞ norm
- Replace it with $c \cdot obj(x+\eta) \leq 0$ then $f(x+\eta) = t$
- Replace it with project($(\eta_1, \eta_2, \ldots)$) = (clip(η), ...)
  where clip "fits" all coordinates to be within the box $[0,1]$

- Simple Neural Network



- Derivation Rules
  - $(c)' = 0$
  - $(x)' = 1$
  - $(x^2)' = 2x$
  - $(c \cdot g(x))' = c \cdot g(x)'$
  - $(g(x) \cdot h(x))' = g(x)' \cdot h(x) + g(x) \cdot h(x)'$
  - $(g(h(x)))' = g'(h(x)) \cdot h'(x)$

PGD attack (= iterated, projected FGSM)

1. Generate a random starting point inside L_∞-ball
2. For $i = 1 \dots k$
   - 2.1. Perform FGSM with magnitude ε_s
   - 2.2. Project back to the ε-sized L_∞-ball

- We start with random point x' inside ε-sized L_∞-ball around x
- We iteratively update x' with new perturbation, but always keep staying in that ball
- The ball ensures, that we aren't perturbating the image too much
- It is possible to never step outside the box and then projection will have no effect
- After running it k times, the loss function is maximised but there is no guarantee, that the image is classified to another label.

Adversarial Accuracy vs. Test Accuracy

- Given: ε-L_∞ ball
- 100 examples
  - 5 wrongly classified
  - Additional 15 classified wrongly by an adversarial example inside ε-L_∞-ball
- Adversarial Accuracy: $\frac{100-5-15}{100} = 80/100$
- Test Accuracy: $\frac{100-5}{100} = 95/100$

↳ Trade between Adversarial vs. Test Accuracy

- Adversarial Accuracy $\leq$ Test Accuracy

↳ If = => Robust Network

Training with adversarial examples (= PGD training)

- Making networks more robust

1. Select a mini-batch B of training data for adversarial-training
2. Compute for every data-point x an adversarial datapoint x' which maximizes the loss while still being inside the L_∞-ball
3. Modify existing network to be more robust by using the adversarial data points

   ↳ $\theta' = \theta - \frac{1}{|x'|} \cdot \sum_{(x_i, y) \in x'} \nabla_\theta L(\theta, x_i', y)$

Maximize step

Minimize step

- Large networks are more defendable
- Training smaller networks with PGD has negative effects on test accuracy
- By alternating solving the outer minimization and inner maximization step individually, we can get stuck at a local minimum

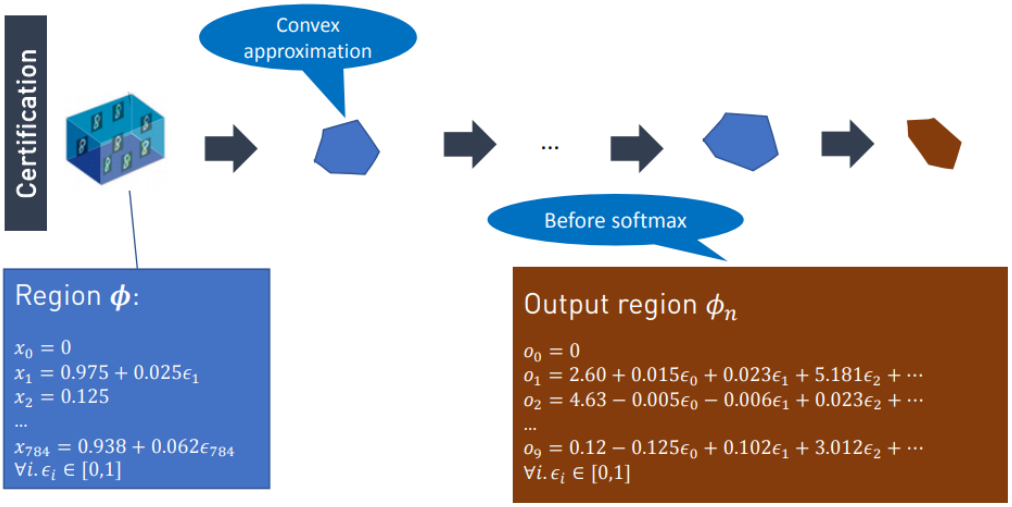
  ↳ This method is training the network to be experimentally robust

find	θ	
minimize	$\rho(\theta)$	Outer minimization problem: find θ that minimizes the high loss, aiming to train a robust classifier
where	$\rho(\theta) = \mathbf{E}_{(x,y)\sim D} [\max_{x' \in S(x)} L(\theta, x', y)]$	Inner maximization problem: find x' that achieves high loss

Soundness / Completeness / Exactness / Optimality

- Sound: · If there is a violation, the program always returns that the property is violated
  · e.g. return always false
  · $\forall z.\ F(\gamma(z)) \subseteq \gamma(F^{\#}(z))$
- Complete: · If the property is satisfied, the method can prove it
  · e.g. return always true
- Sound + Complete + not scalable: MILP
- Sound + Incomplete + scalable: Box, zonotope
- Not sound + incomplete: Random guessing
  ↳ If a point inside the L_∞-case gets misclassified, the verifier will always return violation (=Sound)
  ↳ Because of overapproximation we verify a larger area which can lead to returning violation because there is a violation in the overapproximation, but not in the L_∞-ball area itself (=Incomplete)
- Exactness (exact)
  · If we enumarate all points in the set or just use the asstract transformer, do we get the same result (=No overapproximation)
- optimality (best)
  · Sound transformer F is called best transformer if for all other sound transformer it is the most exact one

Incomplete Method: Box



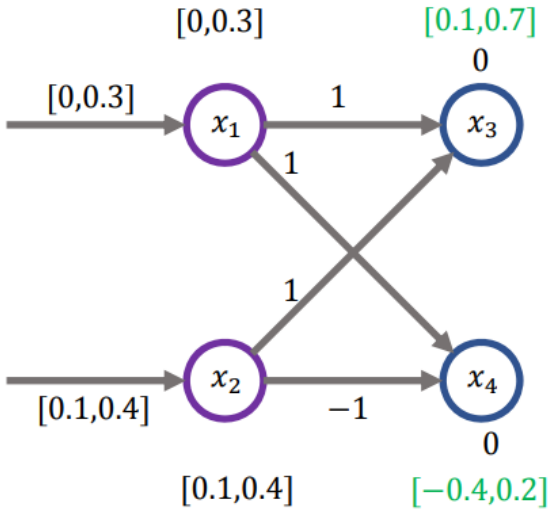
- Interval for each coordinate using $\epsilon_i \in [0,1]$
- In the output region we can check, if o_3 is greater than all the other o_i, this means if the interval doesn't overlap and is the greatest

$[a,b] +^{\#} [c,d] = [a+c, b+d]$	Addition transformer
$-^{\#}[a,b] = [-b,-a]$	Negation transformer
$ReLU^{\#}[a,b] = [ReLU(a), ReLU(b)]$	ReLU transformer
$\lambda^{\#}[a,b] = [\lambda * a, \lambda * b]$	Multiplication by a constant $\lambda > 0$

$\Rightarrow [a,b] - [c,d]$

$\Leftrightarrow [a,b] + [-d,-c]$

$\Leftrightarrow [a-d, b-c]$



Incomplete Method: Zonotope

- Replaces expensive ReLU transformation with affine form
- Unlike Box, Zonotope is exact for affine
- Like Box, Zonotople lose precison on ReLu

$$\hat{x}_1 = a_0^1 + \sum_{i=1}^{k} a_i^1 \epsilon_i$$

Noise $\epsilon_i \in [-1, 1]$

.....

$$\hat{x}_d = a_0^d + \sum_{i=1}^{k} a_i^d \epsilon_i$$

a_0^d: Base coordinate

} Definition

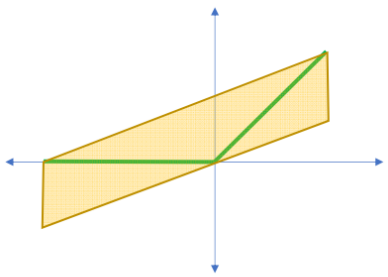
Multiplication of the p'th neuron by a real-valued constant C :

$$(a_0^p + \sum_{i=1}^{k} a_i^p \epsilon_i) * C = (C * a_0^p + \sum_{i=1}^{k} C * a_i^p \epsilon_i)$$

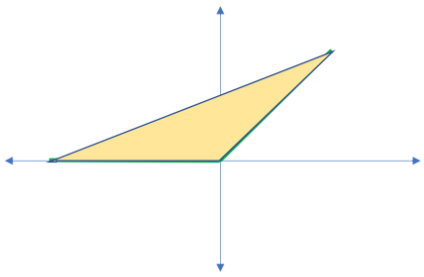
Adding two neurons p and q is done component-wise:

$$(a_0^p + \sum_{i=1}^{k} a_i^p \epsilon_i) + (a_0^q + \sum_{i=1}^{k} a_i^q \epsilon_i) = (a_0^q + a_0^p) + \sum_{i=1}^{k} (a_i^p + a_i^q) \epsilon_i$$

} Affine Transform



Zonotope transformer



Optimal convex transformer (known as triangle approximation)

} ReLu Transform

- 1. Case: If $u_x \leq 0 \Rightarrow y = 0$
- 2. Case: If $l_x > 0 \Rightarrow y = x$
- 3. Case: If $l_x < 0$ and $u_x > 0 \Rightarrow$ Crossing Boundary

$$\lambda * \hat{x} \leq y(\hat{x}) \leq \lambda * \hat{x} - \lambda * l_x$$

Lower Line — Upper Line

$$\lambda := \frac{u_x}{u_x - l_x}$$

The final ReLU transformer is: $y(\hat{x}) = \lambda * \hat{x} - \epsilon_{new} * \frac{\lambda * l_x}{2} - \frac{\lambda * l_x}{2}$

Complete Method: MILP

- Checks, by discovering different paths, by setting $a_i \in \{0,1\}$, and checking, if all branches hold

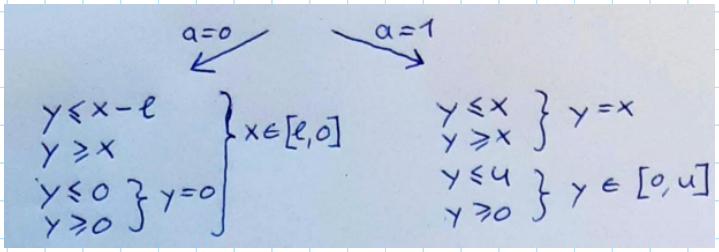
$$\min c_1x_1 + c_2x_2 + \cdots + c_nx_n$$ object

$$a_{11}x_1 + \cdots + a_{1n}x_n \le b_1$$
$$\ldots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n \le b_m$$
} constraints

$$l_i \le x_i \le u_i \quad 1 \le i \le n$$ } Bounds on variables

$$x_j \in Z$$ } Integer/Binary

- ReLu Encoding



} Running Time: $2^{\#\max(\ldots)}$
↳ we need a new branch for each max-operator

- L∞-ball encoding
  - $x_i - \varepsilon \le x_i' \le x_i + \varepsilon$
- Postcondition encoding
  - $\min O_0 - O_1$
  - After MILP finishes, we can check, if
    - $O_0 - O_1 < 0 \Rightarrow$ Class 1
    - $O_0 - O_1 > 1 \Rightarrow$ Class 0

} We must check this for every branch that is satisfiable at the end

-

$$\min o_0 - o_1$$

Plug in the affine and ReLU MILP encodings as defined

$l_i \le x_i^p \le u_i$ ← pre-computed Box bounds on all neurons in each layer p

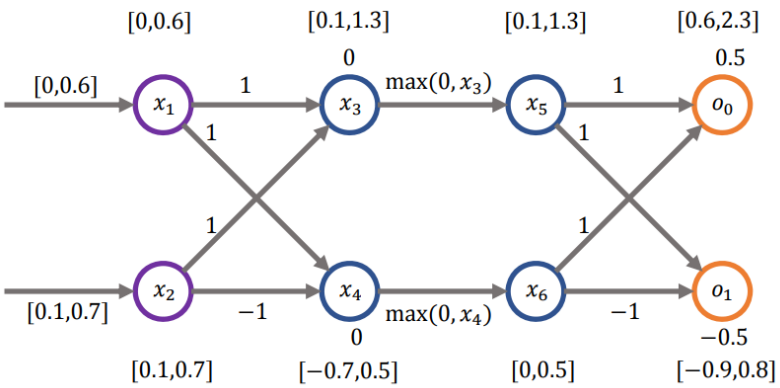
$x_i - \epsilon \le x_i' \le x_i + e$ ← ϕ: bounds on input neurons

$a_j \in \{0,1\}$ ← These are the $a \in \{0,1\}$ vars from the ReLU encoding

} MILP uses the box constraints to save generating branches for a_i values of ReLu

For example, if Box-Bounds are positive, we can just use $x_5 = \text{ReLu}(x_5)$

$x_5 = x_5$



$$\boldsymbol{min}\ o_0 - o_1$$

Affine

$$x_3 = x_1 + x_2$$
$$x_4 = x_1 - x_2$$
$$o_0 = x_5 + x_6 + 0.5$$
$$o_1 = x_5 - x_6 - 0.5$$

ReLU: $x_5 = \max(0, x_3)$

$$x_5 \le x_3 - 0.1 * (1 - a_5)$$
$$x_5 \ge x_3$$
$$x_5 \le 1.3 * a_5$$
$$x_5 \ge 0$$

ReLU: $x_6 = \max(0, x_4)$

$$x_6 \le x_4 + 0.7 * (1 - a_6)$$
$$x_6 \ge x_4$$
$$x_6 \le 0.5 * a_6$$
$$x_6 \ge 0$$

Input bounds

$$0 \le x_1 \le 0.6$$
$$0.1 \le x_2 \le 0.7$$

Pre-computed Box bounds

$$0.1 \le x_3 \le 1.3$$
$$-0.7 \le x_4 \le 0.5$$
$$0.1 \le x_5 \le 1.3$$
$$0 \le x_6 \le 0.5$$

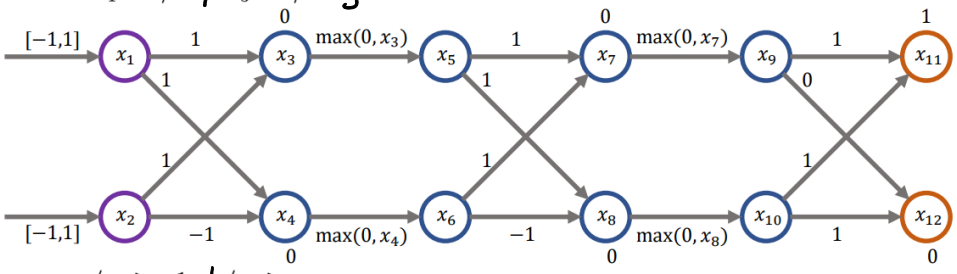
Binary integer variables

$$\boldsymbol{a_5, a_6} \in \{\boldsymbol{0, 1}\}$$

Deep Poly Certification

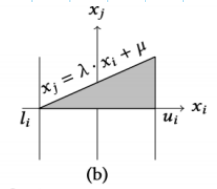
- We keep two inequalities and two bounds per neuron

$\langle x_1 \geq -1,$ | $\langle x_3 \geq x_1 + x_2,$ } inequalities
$x_1 \leq 1,$ | $x_3 \leq x_1 + x_2,$
$l_1 = -1,$ | $l_3 = -2,$ } Bounds
$u_1 = 1\rangle$ | $u_3 = 2\rangle$



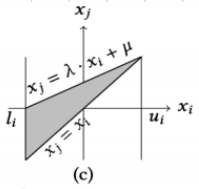
$\langle x_2 \geq -1,$ | $\langle x_4 \geq x_1 - x_2,$
$x_2 \leq 1,$ | $x_4 \leq x_1 - x_2,$
$l_2 = -1,$ | $l_4 = -2,$
$u_2 = 1\rangle$ | $u_4 = 2\rangle$

- ReLU activation



(b)

$0 \leq x_j,$
$x_j \leq u_i(x_i - l_i)/(u_i - l_i),$
$l_j = 0,\ u_j = u_i$



(c)

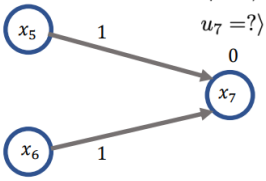
$x_i \leq x_j,$
$x_j \leq u_i(x_i - l_i)/(u_i - l_i),$
$l_j = l_i,\ u_j = u_i$

- Choose one of the two abstract transformers, depending on which has the smaller area
  ↳ smaller area ≙ more precise

- Affine Transformation
  - use backsubstitution to get a more precise bound
  - substitute until you reach x_1/x_2

$\langle x_5 \geq 0,$
$x_5 \leq 0.5 \cdot x_3 + 1,$
$l_5 = 0,$
$u_5 = 2\rangle$

$\langle x_7 \geq 0,$
$x_7 \leq 0.5 \cdot x_3 + 0.5 \cdot x_4 + 2,$
$l_7 = ?,$
$u_7 = ?\rangle$



$\langle x_6 \geq 0,$
$x_6 \leq 0.5 \cdot x_4 + 1,$
$l_6 = 0,$
$u_6 = 2\rangle$

Replace the bounds for x_7 using the ones from the previous layer:

$x_7 \leq x_5 + x_6$
$\leq (0.5x_3 + 1) + (0.5x_4 + 1)$
$= 0.5x_3 + 0.5x_4 + 2$
$= 0.5(x_1 + x_2) + 0.5(x_1 - x_2) + 2$
$= x_1 + 2 \leq 3$
$x_7 \leq x_1 + 2$ $u_7 = 3$

- To prove something like $x_{11} - x_{12}$, instead of using the naive approach by using b_{11}, u_{12}, we use backsubstitution to prove robustness

Visualizing CNNs

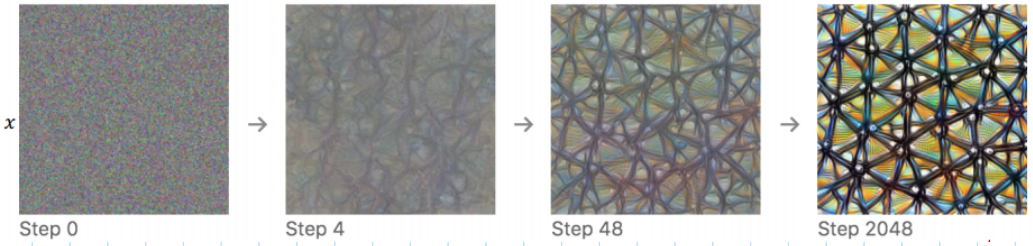
- **Filter**: $\mathbb{R}^{c_0 \times h \times w \times c_1}$
  - height
  - width
  - c_0: #input channels
  - c_1: #output channels

1. First layers as filters

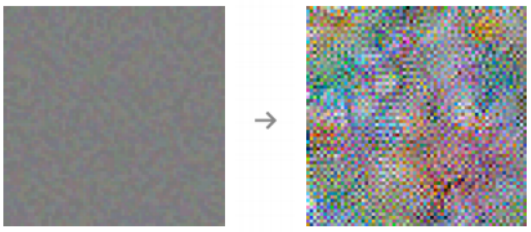
- Take the filters as images to visualize them
- only the first layers possible, where $c_0 \leq 3$

2. Optimization

- ~~initialize the input image with white noise~~
- Maximize gradient ascend to get maximum activation of the particular neuron/filter/channel



- converges to local minimum depending on input-image
- you can also set the score you want to maximize to multiple neurons/channels/filters by taking the mean over their activation
- when there is no regularization (Lasso/Ridge) inside the network, the pattern are unidentifyable

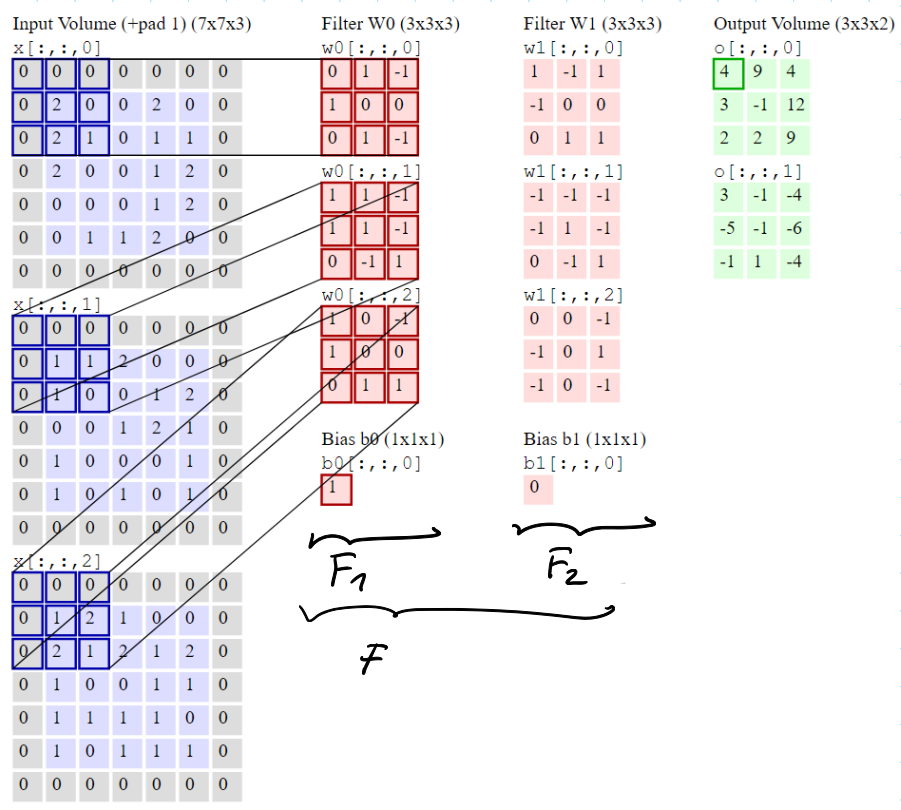


- Use regularization
- Or train with adversarial examples
  - Robust Networks

3. Deep Dream

- We take as input a real image
- Run gradient ascend on it to optimize early convolutions
- Early layers produce easy patterns such as lines
- While later layers show more high level concepts

Convolutions



- $x^{1 \times 5 \times 5 \times 3}$ ← output (Padding: 1 / stride: 1)
- $F_1^{3 \times 3 \times 3 \times 1}$
- $F_2^{3 \times 3 \times 3 \times 1}$ ← Input
- $F^{3 \times 3 \times 3 \times 2}$
- Maxpooling:

Single depth slice

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

→

6	8
3	4

- Dimension reduction
- Get max. value of each kernel

Image Attribution

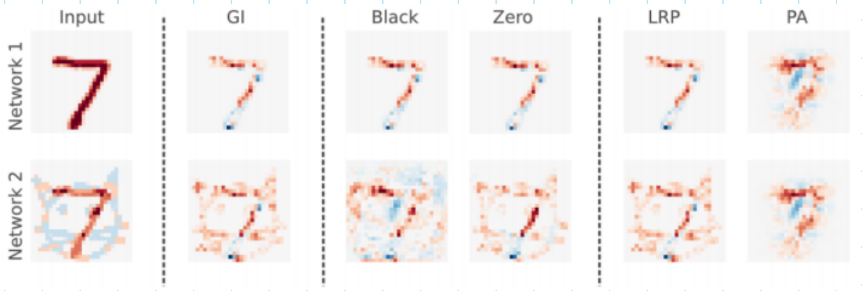
1. Gradient based

- Direction of steepest change => overlay them
- Does not work well if feature is at a saddle point

2. Integrated Gradients

- Use an additional baseline picture x'
  - e.g. black or an adverserial example
- We are going the path from the base image to a combination of both to the original image
- We are integrating the gradient as we are going along and adding the features

Attribution Problem



3. Meaningful Perturbations

- Blur image in such a way, that the probability of the feature gets minimized

4. Shapley Values

- Very expensive
- For each feature we check, if we add a certain subset of the image, if this increases or decreases our activation function

Querying the Network with Logic (= find formula example)

$\theta = \bigwedge_{j=1, j\neq 9}^{k} \underbrace{NN(i)[j] < NN(i)[9]}_{class(NN(i))=9} \wedge \|i - deer\|_\infty < 25 \wedge \|i - deer\|_\infty > 5$ } i = image

↳ "Find an image i which gets classified to 9 (= Truck) where the image is within some distance to the image deer."

1. SAT solver

- Only if neural network is really small

2. Translate to optimization problem

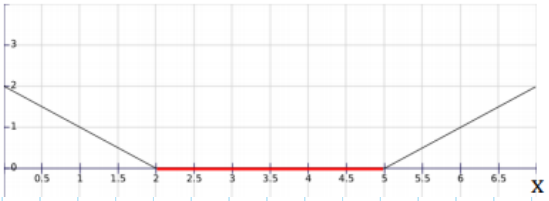
- Translation function $T(\theta)$
- $T(\theta)(i) = 0 \Leftrightarrow i$ satisfies θ

Logical Term	Translation
$t_1 \leq t_2$	$\max(0, t_1 - t_2)$
$t_1 \neq t_2$	$[t_1 = t_2]$
$t_1 = t_2$	$T(t_1 \leq t_2 \wedge t_2 \leq t_1)$
$t_1 < t_2$	$T(t_1 \leq t_2 \wedge t_1 \neq t_2)$
$\varphi \vee \psi$	$T(\varphi) \cdot T(\psi)$
$\varphi \wedge \psi$	$T(\varphi) + T(\psi)$

} As long as $t_1 \leq t_2$, the value is 0
} If $t_1 = t_2$ → value is 1
} $\max(0, t_1 - t_2) + \max(0, t_2 - t_1) = |t_1 - t_2|$
} $\max(0, t_1 - t_2) + [t_1 = t_2]$

- Example ↳ To prove: Base case: Equalities / Inequalities (→ Assume $\theta(T) = 0$)
  Step case: Disjunctions / Conjunctions

  • $T(x \geq 2 \wedge x \leq 5) = \max(0, 2 - x) + \max(0, x - 5)$

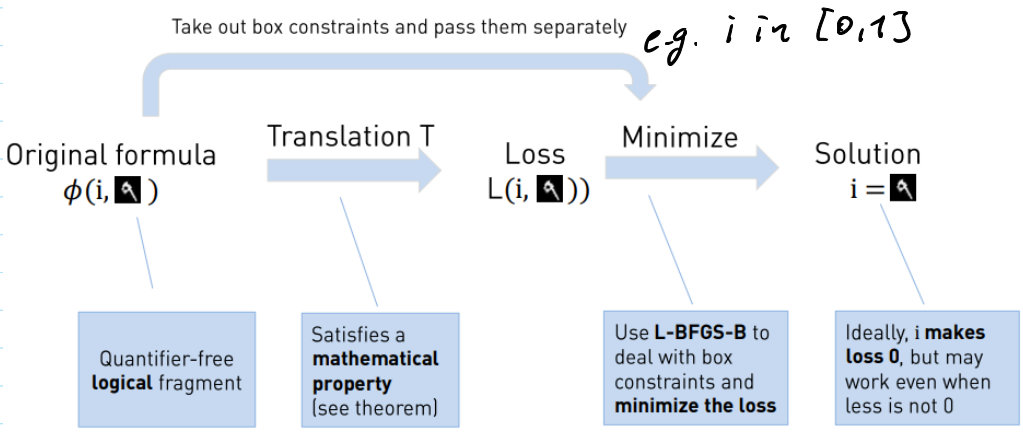


} The function is 0 when x is between 2–5

- $\neg(a \wedge b) = \neg a \vee \neg b$
- $\neg(a \vee b) = \neg a \wedge \neg b$
- $\neg(a \leq b) = a > b$
- $\neg(a = b) = a \neq b$

$$
\begin{aligned}
&\mathrm{NN(i)[1]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[2]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[3]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[4]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[5]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[6]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[7]} < \mathrm{NN(i)[9]} \wedge \\
&\mathrm{NN(i)[8]} < \mathrm{NN(i)[9]} \wedge \\
&\|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge \\
&\|\mathrm{i} - \mathrm{deer}\|_\infty > 5
\end{aligned}
$$

$$
\begin{aligned}
&\max(0, \mathrm{NN(i)[1]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[1]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[2]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[2]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[3]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[3]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[4]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[4]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[5]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[5]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[6]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[6]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[7]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[7]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \mathrm{NN(i)[8]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[8]} = \mathrm{NN(i)[9]}] \\
&+ \max(0, \|\mathrm{i} - \mathrm{deer}\|_\infty - 25) + [\|\mathrm{i} - \mathrm{deer}\|_\infty = 25] \\
&+ \max(0, \quad 5 - \|\mathrm{i} - \mathrm{deer}\|_\infty) + [\|\mathrm{i} - \mathrm{deer}\|_\infty = 5]
\end{aligned}
$$



Training the Network with Logic (=ensure formula)

- How to find counterexamples
- e.g. all images in dataset that are classified as a car have a higher probability for the label truck than the probability for label dog
  - ↳ $y = car \Rightarrow NN(x)[truck] > NN(x)[dog]$
  - ↳ $\forall z \in L_\infty(x, \varepsilon).\ y = car \Rightarrow NN(z)[truck] > NN(z)[dog]$
    - ↳ For all images close to ε

1. Train a base classifier $\hat{\theta}$ on labeled data
2. Use $\hat{\theta}$ to infer the labels for unlabeled dataset
3. Use adversarial training to learn robust classifier

- Problem statement

find	θ } Find weights for Neural Network
minimize	$\rho(\theta)$
where	$\rho(\theta) = \mathbf{E}_{s\sim D}[\mathrm{T}(\phi)(z\_worst,\ s, \theta)]$ } Minimize the expected value of the worst counter-example
and	$z\_worst = \underset{z}{\mathbf{argmin}}(\mathrm{T}(\neg\phi)(z,\ s, \theta))$

high loss

Find the worst case counter example

- We want to find the worst possible violation of the formula and then try to find a network that minimizes its effect
- Example formula: $\phi(z, x, \theta) = \|x - z\|_\infty \leq \varepsilon \Rightarrow NN_\theta(z)[3] > \sigma$
  - ↳ Find an image z which is inside the L_∞ ball around the image x where the ball has size ε and z gets classified as 3 is greater than σ

Provable defenses and training the network with logic

- PGD training only trains the network experimentally
- We want to train the network to be provable robust after training

Outer minimization problem

Lets keep this part

find θ
minimize $\rho(\theta)$

where $\rho(\theta) = \mathbf{E}_{(x,y)\sim D}\left[\max_{z \in \gamma(NN^{\#}(S(x)))} L(\theta, z, y)\right]$

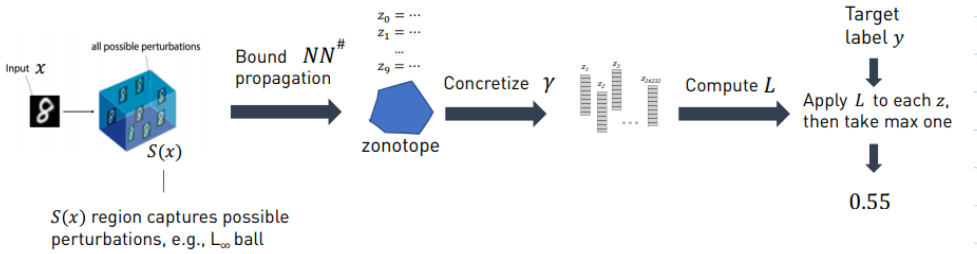
Inner maximization problem

find z that achieves high loss **under abstraction**

Vector of logits

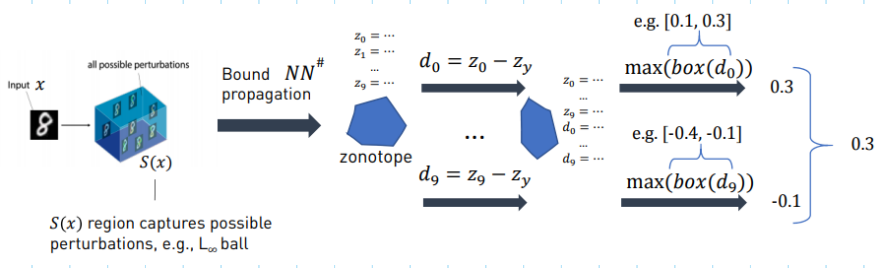
all concrete values

- We start at the beginning and push $S(x)$ through the network to get the area which we get values from
- While in PGD training we look at the end



- We could have a large number of logits in the vector, enumerating all possible values impossible

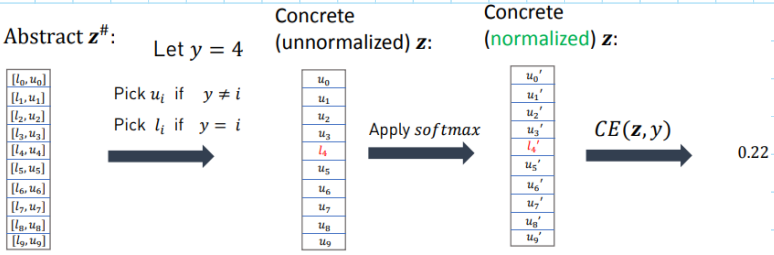
1. Use zonotope to push through, convert the logits to box intervals and look for max Loss



↓

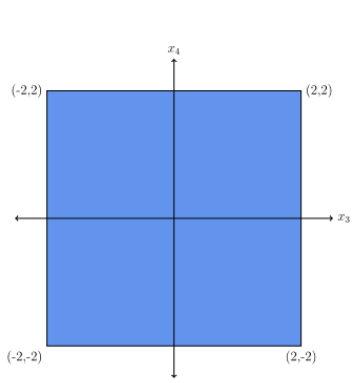
Take that adversarial example to train network

2. Use Zonotope to push through, convert the logits to box intervals and use cross-entropy loss



K-ReLU

- Normal ReLU transformer is neuron-wise

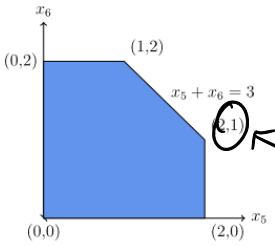


Input set $\mathcal{S} = \{-2 \le x_3 \le 2, -2 \le x_4 \le 2\}$

x_3 x_4

$x_5 := \mathrm{ReLU}(x_3)$

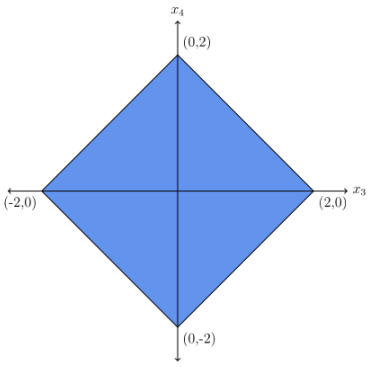
$x_6 := \mathrm{ReLU}(x_4)$



← Not possible

⇓ overapproximation

- Put constraints between two/more ReLUs



Input set $\mathcal{S} = \{-2 \le x_3 \le 2, -2 \le x_4 \le 2,$
$-2 \le x_3 + x_4 \le 2, -2 \le x_3 - x_4 \le 2\}$

If $x_3 \le 0$, then $\max(0, x_3) = 0 = x_5$

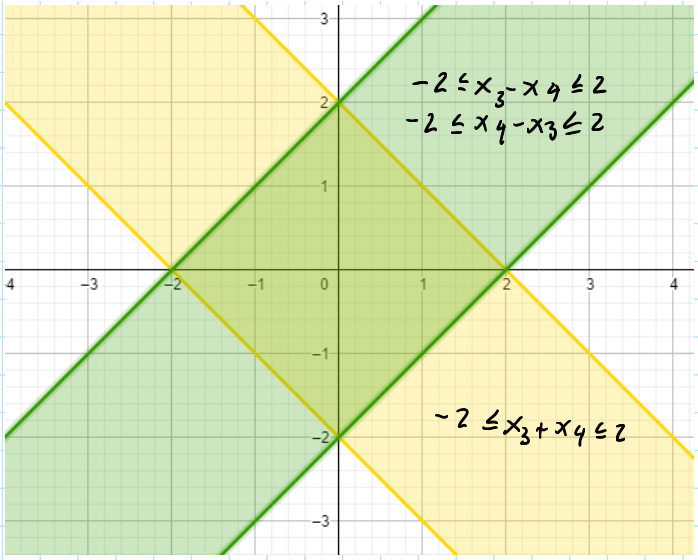
$\Rightarrow \mathcal{S} \cap \{x_3 \le 0, x_4 \le 0, x_5 = 0, x_6 = 0\}$

$\Rightarrow \mathcal{S} \cap \{x_3 \le 0, x_4 \ge 0, x_5 = 0, x_6 = x_4\}$

$\Rightarrow \mathcal{S} \cap \{x_3 \ge 0, x_4 \le 0, x_5 = x_3, x_6 = 0\}$

$\Rightarrow \mathcal{S} \cap \{x_3 \ge 0, x_4 \ge 0, x_5 = x_3, x_6 = x_4\}$

↳ enumerate all possibilities



Randomized Smoothing

- Guarantees are not absolute (like MILP), but probabilistic
- Construct from a given classifier f a smoothed classifier $\bar{f}$
- $\bar{f}$ returns the class with highest probability under $\mathcal{N}(x, \sigma^2 1)$
- $\bar{f}(x) := \operatorname{argmax}_{c_k} \left(P[f(\overbrace{x+\varepsilon}^{x'}) = c_k] \right)$ where $\varepsilon \sim \mathcal{N}(0, \sigma^2 1)$

  $= \operatorname{argmax}_{c_k} \int_{\varepsilon \in \mathbb{R}^d} [f(x+\varepsilon) = c_k] \cdot d\varepsilon$ ↯ Infeasible, too large

  ↪ $\operatorname{argmax}_{c_k \in Y} \left(\frac{1}{n} \sum_{i=1}^{n} [f(x+\varepsilon_i) = c_k] \right)$

function PREDICT(f, σ, x, n, α)
  counts ← SAMPLEUNDERNOISE(f, x, n, σ) → Evaluates n times $f(x+\varepsilon)$ and counts classes
  $\hat{c}_A, \hat{c}_B$ ← top two indices in counts
  n_A, n_B ← counts[$\hat{c}_A$], counts[$\hat{c}_B$] → n_A: # class A is chosen
  if BINOMPVALUE($n_A, n_A + n_B, 0.5$) $\leq \alpha$ **return** $\hat{c}_A$
  else return ABSTAIN

↳ $P[n_A$ of $(n_A + n_B)$ samples came up

- BinomPValues returns the probability that n_A out of $(n_A + n_B)$ times we came up with the wrong class, because we could have sampled from wrong region
- Predict returns wrong class with probability at most α

function CERTIFY($f, \sigma, x, n_0, n, \alpha$)
  counts0 ← SAMPLEUNDERNOISE(f, x, n_0, σ) } Get $\hat{c}_A$
  $\hat{c}_A$ ← top index in counts0
  counts ← SAMPLEUNDERNOISE(f, x, n, σ) } Sample again to avoid introducing Bias
  $\underline{p_A}$ ← LOWERCONFBOUND(counts[$\hat{c}_A$], $n, 1-\alpha$)
  if $\underline{p_A} > \frac{1}{2}$ **return** prediction $\hat{c}_A$ and radius $\sigma\, \Phi^{-1}(\underline{p_A})$
  else return ABSTAIN

↳ With probability at least $1-\alpha$ class c_A is the real class if we sample $\bar{f}(x+\sigma) = c_A$, $\sigma \subset \mathbb{R}$

- Adversarial attacks using randomized smoothing
  - argmax in $\bar{f}(x) = \underset{c_k \in Y}{\operatorname{argmax}} \left(P_\epsilon [f(x+\epsilon) = c_k] \right)$ is a problem for gradient descend
    - (→ We assume softmax layer at the end
  - $x_{adv} = \underset{\|x' - x\| < p}{\operatorname{argmax}} \left(-\log E_\epsilon [F(x' + \epsilon)_y] \right)$

    We find adversarial example x' within the L_∞-ball

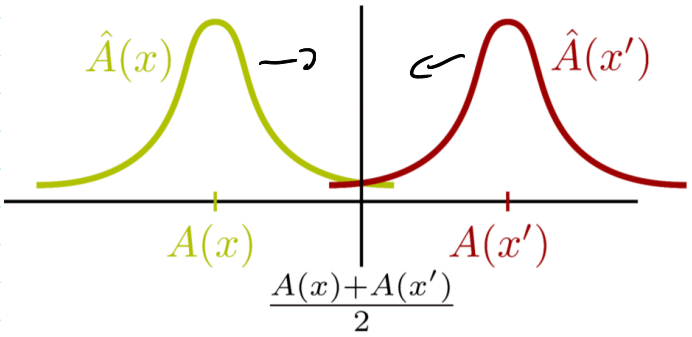
    $= \nabla_{x'} \left(-\log E_\epsilon [F(x' + \epsilon)_y] \right)$ ← Infeasable, too large

    $\approx \nabla_{x'} \left(-\log \left[\frac{1}{m} \sum_{i=1}^{m} F(x' + \epsilon)_y \right] \right)$ ← Use Monte-Carlo to approximate gradient

- Instead of argmaxing $P[f(x+\epsilon) = c_k]$ we are argmaxing the highest loss a perturbed input x' can make $-\log E[F(x' + \epsilon)_y]$

Differential Privacy

- Good composition properties: $B(x) = (\hat{A}_1(x), \ldots, \hat{A}_n(x)), \sum_{i=1}^{n} \varepsilon_i - DP$
- Preserved under post processing: $g(\hat{A}(x))$
- The probability distribution of the database with the user and without it is nearly the same
- We want to bring $\hat{A}(x)$ and $\hat{A}(x')$ as close together as possible, because $[-\infty, \frac{A(x)+A(x')}{2}]$ is probably the real dataset x and not the perturbed x' dataset



- ε-differential privacy: $\Pr[\hat{A}(x) \in \Phi] \leq \exp(\varepsilon) \cdot \Pr[\hat{A}(x') \in \Phi]$
  - $(x, x') \in$ Neighborhood
  - The two distributions are roughly the same
  - ε captures the notion of distance between $\hat{A}(x)$ and $\hat{A}(x')$
  - $\varepsilon = 0$: $\Pr[\hat{A}(x) \in \Phi] = \Pr[\hat{A}(x') \in \Phi]$
    - ↳ Full privacy
  - small ε: $\Pr[\hat{A}(x) \in \Phi] \approx \Pr[\hat{A}(x') \in \Phi]$

↳ Algorithm $\hat{A}$ is ε-dp

Proof:

$\Pr[\hat{A}(x) \in \Phi] \leq \exp(\varepsilon)\Pr[\hat{A}(x') \in \Phi]$

$\Rightarrow$ (by symmetry of $Neig$)

$\Pr[\hat{A}(x') \in \Phi] \leq \exp(\varepsilon)\Pr[\hat{A}(x) \in \Phi]$

$\Rightarrow (\cdot \exp(-\varepsilon))$

$\exp(-\varepsilon)\Pr[\hat{A}(x') \in \Phi] \leq \Pr[\hat{A}(x) \in \Phi]$

$\Rightarrow$ (combine)

$\exp(-\varepsilon)\Pr[\hat{A}(x') \in \Phi] \leq \Pr[\hat{A}(x) \in \Phi] \leq \exp(\varepsilon)\Pr[\hat{A}(x') \in \Phi]$

$\approx 1 - \varepsilon$ $\qquad$ $\approx 1 + \varepsilon$

- Neighborhood: · Changing the data of one person
  · Adding/removing one person from/to x'
  · For images $\|x - x'\|_p < R$
  ↳ Symmetric $(x, x') \Leftrightarrow (x, x')$

- Tradeoff between Privacy >< Precision
  · Trivial choice $\hat{A}(x) = 0$
  · $\varepsilon = 0$, since we always return 0 ⇒ Fully private
  · $\|\hat{A}(x) - A(x)\|$ can be large ⇒ Imprecise

- Laplace mechanism
  · $A(x) = \sum_{i=1}^{d} x_i$
  · Sensitivity of A: $\Delta(A) = \max \|A(x) - A(x')\|_1 = 1$
  · $\hat{A}(x) = \sum_{i=1}^{d} x_i + \text{Lap}(0, \frac{1}{\varepsilon}) = \text{Lap}(\sum_{i=1}^{d} x_i, 1/\varepsilon)$
  · We need to check: $\frac{\Pr[\hat{A}(x) \in b]}{\Pr[\hat{A}(x') \in b]} \leq \exp(\varepsilon)$

$$= \frac{p(\hat{A}(x) = t)}{p(\hat{A}(x') = t}$$

$$= \frac{\exp\left(-\frac{|t - A(x)|}{1/\varepsilon}\right)}{\exp\left(-\frac{(t - A(x'))|}{1/\varepsilon}\right)}$$

$$= \exp\left(-\frac{|t - A(x)|}{1/\varepsilon} + \frac{|t - A(x')|}{1/\varepsilon}\right)$$

$$\leq \exp\left(-\frac{|A(x') - A(x)|}{1/\varepsilon}\right) \quad / \Delta(A) = 1$$

$$= \exp\left(-\frac{1}{1/\varepsilon}\right)$$

$$= \exp(-\varepsilon)$$

$$\leq \exp(\varepsilon)$$

- Generalizing differential privacy
  - (ε, σ)-DP: $Pr[\hat{A}(x) \in \phi] \leq \exp(\varepsilon) \cdot Pr[\hat{A}(x') \in \phi] + \sigma$
  - ε-DP: $(\varepsilon, 0)$-DP
  - $\hat{A}(x) = A(x) + \eta$

Neighboring

- Laplacian mechanism: works well with $\|x\|_1 = |3| + |4| = 7$
  - $\eta_i \sim Lap(0, 1/\varepsilon)$
- Gaussian mechanism: works well with $\|x\|_2 = \sqrt{3^2 + 4^2} = 5$
  - $\eta \sim N(0, \sigma^2)$
- ε-DP counterexamples
  - Prove that $\hat{A}(x)$ not ε-dp
  - $\log\left(\frac{Pr[\hat{A}(x) \in \phi]}{Pr[\hat{A}(x') \in \phi]}\right) \overset{!}{>} \varepsilon$
- Connection to randomized smoothing
  - $\hat{f}(x) = f(x + \eta)$ where $\eta_i \sim Lap(0, R/\varepsilon)$
  - $(x, x') \in Neigh \Leftrightarrow \|x - x'\|_1 < R$
  - $\hat{f}$ is ε-dp
    1. Considers $\hat{A}(x) = x + \eta$ and view f as post-processing
    2. Generalize from $\|x - x'\|_1 \leq 1$ to $\|x - x'\|_1 < R$ by $Lap(0, R/\varepsilon)$
       ↳ $x' = x + \eta$
- Randomized Response
  - 0.5 → Answer question honestly
  - 0.5 → fool observer 0.5 → yes / No
  - $Pr[\text{yes}] = 0.5 \cdot p + 0.5 \cdot 0.5$

$$\frac{\Pr[\text{Answer yes}|\text{Truly yes}]}{\Pr[\text{Answer yes}|\text{Truly no}]} = \frac{\frac{1}{2}\cdot 1 + \frac{1}{2}\cdot\frac{1}{2}}{\frac{1}{2}\cdot 0 + \frac{1}{2}\cdot\frac{1}{2}} = \frac{\frac{3}{4}}{\frac{1}{4}} = 3 \leq \exp\varepsilon \Leftrightarrow \varepsilon \geq 1.10 \quad \Rightarrow 1,10\text{-DP}$$