Summary of A few Recent Papers on Testing DNN

Presentor: Ji Gao

Department of Computer Science, University of Virginia

https://qdata.github.io/deep2Read/

Outline

- Testing Deep Neural Networks
- Concolic Testing for Deep Neural Networks
- DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems
- Generating Natural Adversarial Samples
- Generating adversarial samples with adversarial networks
- Defense against the dark arts: an overview of adversarial example security research and future research directions

Testing Deep Neural Networks

Youcheng Sun, Xiaowei Huang, and Daniel Kroening

- Follow the path of DeepXplore
- Propose “binary level” coverage

Defition: Coverage

Let N be a set of neural networks, R the set of requirements, and T the set of test suites.

Definition 1 ([5]). *A test adequacy criterion, or a test coverage metric, is a function* $M : \mathsf{N} \times \mathsf{R} \times \mathsf{T} \to [0, 1]$.

Definition 2. *A neuron pair* $(n_{k,i}, n_{k+1,j})$ *are two neurons in adjacent layers* k *and* $k + 1$ *such that* $1 \leq k \leq K - 1$, $1 \leq i \leq s_k$, *and* $1 \leq j \leq s_{k+1}$. *Given a network* $\mathcal{N}$, *we write* $\mathcal{O}(\mathcal{N})$ *for the set of its neuron pairs.*

Change on 1 neuron

Definition 3 (Sign Change). *Given a neuron $n_{k,l}$ and two test cases x_1 and x_2, we say that the sign change of $n_{k,l}$ is exploited by x_1 and x_2, denoted as $sc(n_{k,l}, x_1, x_2)$, if $sign(v_{k,l}[x_1]) \neq sign(v_{k,l}[x_2])$. We write $\neg sc(n_{k,l}, x_1, x_2)$ when the condition is not satisfied.*

Definition 4 (Value Change). *Given a neuron $n_{k,l}$ and two test cases x_1 and x_2, we say that the value change of $n_{k,l}$ is exploited with respect to a value function g by x_1 and x_2, denoted as $vc(g, n_{k,l}, x_1, x_2)$, if $g(u_{k,l}[x_1], u_{k,l}[x_2])$=true and $\neg sc(n_{k,l}, x_1, x_2)$. Moreover, we write $\neg vc(g, n_{k,l}, x_1, x_2)$ when the condition is not satisfied.*

Change on neurons in one layer

Definition 5 (Distance Change). *Given the set of neurons* $P_k = \{n_{k,l} \mid 1 \leq l \leq s_k\}$ *in layer* k *and two test cases* x_1 *and* x_2*, we say that the distance change of* P_k *is exploited with respect to a distance function* h *by* x_1 *and* x_2*, denoted as* $dc(h, k, x_1, x_2)$*, if*

- $h(u_k[x_1], u_k[x_2])$ = *true, and*
- *for all* $n_{k,l} \in P_k$, $sign(v_{k,l}[x_1]) = sign(v_{k,l}[x_2])$.

We write $\neg dc(h, k, x_1, x_2)$ *when any of the conditions is not satisfied.*

Coverage 1

Definition 6 (Sign-Sign Cover, or SS Cover). *A neuron pair* $\alpha = (n_{k,i}, n_{k+1,j})$ *is SS-covered by two test cases* x_1, x_2*, denoted as* $cov_{SS}(\alpha, x_1, x_2)$*, if the following conditions are satisfied by the network instances* $\mathcal{N}[x_1]$ *and* $\mathcal{N}[x_2]$*:*

- $sc(n_{k,i}, x_1, x_2)$;
- $\neg sc(n_{k,l}, x_1, x_2)$ *for all* $p_{k,l} \in P_k \setminus \{i\}$;
- $sc(n_{k+1,j}, x_1, x_2)$.

Coverage 2

Definition 7 (Distance-Sign Cover, or DS Cover). *Given a distance function h, a neuron pair $\alpha = (n_{k,i}, n_{k+1,j})$ is DS-covered by two test cases x_1, x_2, denoted as $cov^h_{DS}(\alpha, x_1, x_2)$, if the following conditions are satisfied by the network instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:*

- $dc(h, k, x_1, x_2)$, *and*
- $sc(n_{k+1,j}, x_1, x_2)$.

Coverage 3

Definition 8 (Sign-Value Cover, or SV Cover). *Given a value function g, a neuron pair $\alpha = (n_{k,i}, n_{k+1,j})$ is SV-covered by two test cases x_1, x_2, denoted as $cov^{g}_{SV}(\alpha, x_1, x_2)$, if the following conditions are satisfied by the network instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:*

- $sc(n_{k,i}, x_1, x_2)$;
- $\neg sc(n_{k,l}, x_1, x_2)$ *for all* $p_{k,l} \in P_k \setminus \{i\}$;
- $vc(g, n_{k+1,j}, x_1, x_2)$.

Automated Test Case Generation

Algorithm 1 Test-Gen

INPUT: DNN $\mathcal{N}$, covering method f
OUTPUT: test suite $\mathcal{T}$, adversarial examples $\mathcal{T}'$

1: **for** each neuron pair $n_{k,i}, n_{k+1,j} \in \mathcal{O}(\mathcal{N})$ **do**
2: $x_1, x_2 = get\_input\_pair(f, n_{k,i}, n_{k+1,j})$
3: **if** x_1, x_2 are valid inputs **then**
4: $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_1, x_2\}$
5: **if** $\mathcal{N}[x_1].label \neq \mathcal{N}[x_2].label$ and $close(x_1, x_2)$ **then**
6: $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{x_1, x_2\}$
7: **return** $\mathcal{T}, \mathcal{T}'$

Algorithm 2 get_input_pair with the first argument being cov_{SS}

INPUT: cov_{SS}, neuron pair $n_{k,i}, n_{k+1,j}$
OUTPUT: input pair x_1, x_2

1: **for** each $x_1 \in data\_set$ **do**
2: $x_2 = lp\_call(\mathcal{C}[x_1][1..k][\{(k,i),(k+1,j)\}], \min ||x_2 - x_1||_\infty)$
3: **if** x_2 is a valid input **then return** x_1, x_2
4: **return** _, _

Experiment

- Target: {3,4,5} layer network, hidden layer size randomly from 20 to 100, input 28*28, output 10
- Neuron Coverage -> Random 25 images get around 100% coverage
- Random -> Unable to find adversarial samples

Experiment

	hidden layers	$Mcov_{SS}$	$AEcov_{SS}$	$Mcov^{d}_{DS}$	$AEcov^{d}_{DS}$	$Mcov^{g}_{SV}$	$AEcov^{g}_{SV}$	$Mcov^{d,g}_{DV}$	$AEcov^{d,g}_{DV}$
$\mathcal{N}_1$	67x22x63	99.7%	18.9%	100%	15.8%	100%	6.7%	100%	21.1%
$\mathcal{N}_2$	59x94x56x45	98.5%	9.5%	100%	6.8%	99.9%	3.7%	100%	11.2%
$\mathcal{N}_3$	72x61x70x77	99.4%	7.1%	100%	5.0%	99.9%	3.7%	98.6%	11.0%
$\mathcal{N}_4$	65x99x87x23x31	98.4%	7.1%	100%	7.2%	99.8%	3.7%	98.4%	11.2%
$\mathcal{N}_5$	49x61x90x21x48	89.1%	11.4%	99.1%	9.6%	99.4%	4.9%	98.7%	9.1%
$\mathcal{N}_6$	97x83x32	100.0%	9.4%	100%	5.6%	100%	3.7%	100%	8.0%
$\mathcal{N}_7$	33x95x67x43x76	86.9%	8.8%	100%	7.2%	99.2%	3.8%	96%	12.0%
$\mathcal{N}_8$	78x62x73x47	99.8%	8.4%	100%	9.4%	100%	4.0%	100%	7.3%
$\mathcal{N}_9$	87x33x62	100.0%	12.0%	100%	10.5%	100%	5.0%	100%	6.7%
$\mathcal{N}_{10}$	76x55x74x98x75	86.7%	5.8%	100%	6.1%	98.3%	2.4%	93.9%	4.5%

Concolic Testing for Deep Neural Networks

Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening

- From the same group of last paper

Setting

- DNN: Fully connected with ReLU

Definition 2 *We use variables* $IV = \{x, x_1, x_2, ...\}$ *to range over the inputs in* D_{L_1}. *Given a network* $\mathcal{N}$, *we let* $V = \{u[x]_{k,l}, v[x]_{k,l} \mid 1 \leq k \leq K, 1 \leq l \leq s_k, x \in IV\}$ *be a set of variables. DR uses the following syntax to write a requirement formula:*

$$\begin{aligned} r &::= Qx.e \mid Qx_1, x_2.e \mid \arg opt_x a : e \mid \arg opt_{x_1,x_2} a : e \\ e &::= a \bowtie 0 \mid e \wedge e \mid \neg e \mid |\{e_1, ..., e_m\}| \bowtie q \\ a &::= w \mid c * w \mid p \mid a + a \mid a - a \end{aligned} \tag{4}$$

where $Q \in \{\exists, \forall\}$, $w \in V$, $c, p \in \mathbb{R}$, $q \in \mathbb{N}$, $opt \in \{\max, \min\}$, $\bowtie \in \{\leq, <, =, >, \geq\}$, *and* $x, x_1, x_2 \in IV$. *We may call* r *a requirement formula,* e *a Boolean formula, and* a *an arithmetic formula. We call the logic* $DR^{\exists}$ *if the opt operators are not allowed, and* $DR^{\exists,+}$ *if both opt operators and the negation operator* $\neg$ *are not allowed.* We use $\mathfrak{R}$ to denote a set of requirement formulas.

Test criteria

Definition 4 (Test Criterion) *Given a network $\mathcal{N}$, a set $\mathfrak{R}$ of test requirements expressed as DR formulas, and a test suite $\mathcal{T}$, the test criterion $M(\mathfrak{R}, \mathcal{T})$ is as follows:*

$$M(\mathfrak{R}, \mathcal{T}) = \frac{|\{r \in \mathfrak{R} \mid \mathcal{T} \models r\}|}{|\mathfrak{R}|} \tag{7}$$

Complexity

- NP-Hard to check a space on a requirement

Theorem 2 *Given a network $\mathcal{N}$, a DR requirement formula r with a constant number of $\exists$, $\arg\max$, $\arg\min$ operators, and a subspace $X \subseteq D_{L_1}$, the checking of $X \models r$ is NP-complete. This conclusion also holds for $DR^{\exists}$ and $DR^{\exists,+}$ requirements.*

- But it's easy to check on single test samples

Requirement - Coverage

Definition 8 (NC Requirements) *The set* $\mathfrak{R}_{NC}$ *of requirements is*

$$\{\exists x.ap[x]_{k,i} = \mathit{true} \mid 2 \leq k \leq K-1, 1 \leq i \leq s_k\} \tag{12}$$

Definition 9 (SSC Requirements) *Given a pair* $\alpha = (n_{k,i}, n_{k+1,j})$ *of neurons, the singleton set* $\mathfrak{R}_{SSC}(\alpha)$ *of requirements is as follows:*

$$\begin{array}{c}\{\exists x_1, x_2.\ ap[x_1]_{k,i} \neq ap[x_2]_{k,i} \wedge ap[x_1]_{k+1,j} \neq ap[x_2]_{k+1,j} \wedge \\ \bigwedge_{1 \leq l \leq s_k, l \neq i} ap[x_1]_{k,l} - ap[x_2]_{k,l} = 0\}\end{array} \tag{13}$$

and we have

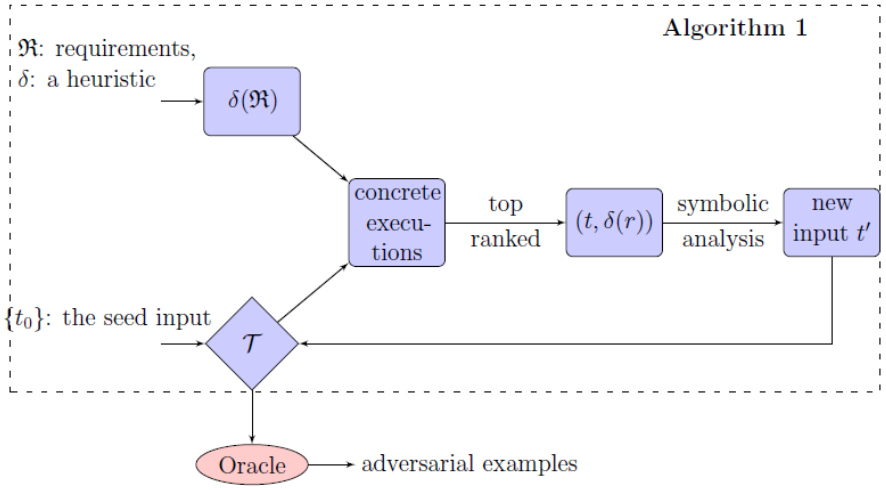
$$\mathfrak{R}_{SSC} = \bigcup_{2 \leq k \leq K-2, 1 \leq i \leq s_k, 1 \leq j \leq s_{k+1}} \mathfrak{R}_{SSC}((n_{k,i}, n_{k+1,j})) \tag{14}$$

Requirement – Lipschitz Continuity

Definition 6 (Lipschitz Requirements) *Given a real number $c > 0$ and an integer $b > 0$, a set $\mathfrak{R}_{Lip}(b, c)$ of Lipschitz requirements is*

$$\{\exists x_1, x_2.(||v[x_1]_1 - v[x_2]_1|| - c * ||x_1 - x_2|| > 0) \\ \wedge x_1, x_2 \in X \mid X \in \mathcal{S}(D_{L_1}, b)\} \tag{11}$$

Algorithm: Build corner cases for given requirements



Algorithm 1 Concolic Testing Algorithm for DNNs

INPUT: $\mathcal{N}, \mathfrak{R}, \delta, t_0$
OUTPUT: $\mathcal{T}$

1: $\mathcal{T} \leftarrow \{t_0\}$ and $S = \{\}$
2: $t \leftarrow t_0$
3: **while** $\mathfrak{R} \neq \emptyset$ **do**
4: **for** each $r \in \mathfrak{R}$ **do**
5: **if** $\mathcal{T} \models r$ **then** $\mathfrak{R} \leftarrow \mathfrak{R} \setminus \{r\}$
6: **while** true **do**
7: $t, \delta(r) \leftarrow requirement\_evaluation(\mathcal{T}, \delta(\mathfrak{R}))$
8: $t' \leftarrow symbolic\_analysis(t, \delta(r))$
9: **if** $validity\_check(t') =$ true **then**
10: $\mathcal{T} \leftarrow \mathcal{T} \cup \{t'\}$
11: **break**
12: **else**
13: $S \leftarrow S \cup \{(t, r)\}$
14: **if** $S = \mathcal{T} \times \mathfrak{R}$ **then return** $\mathcal{T}$
15: **return** $\mathcal{T}$

Algorithm: Build corner cases for given requirements

- Relax the requirement
- Solve the relaxed requirement using optimization/ Linear programming

- Add into the test suite

DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems

Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, Yadong Wang

- Key word: **Multi-Granularity**

Criteria I(a): Multisection Neuron Coverage

- Define a [low_n, high_n] for neuron #n. Separate in K classes.

$$\text{KNCov}(T) = \frac{\sum_{n \in N} |\{S_i^n \mid \exists \mathbf{x} \in T : \ \phi(\mathbf{x}, n) \in S_i^n\}|}{k \times |N|}.$$

Criteria I(b): Outliner

Given a test input $\mathbf{x}$, if $\phi(\mathbf{x}, n)$ belongs to $(-\infty, \text{low}_n)$ or $(\text{high}_n, +\infty)$, we say the corresponding corner case is covered. To quantify this, we first define the number of covered corner behaviors as follows:

$$\text{UpperNeuron} = \{n \mid \exists \mathbf{x} \in T: \ \phi(\mathbf{x}, n) \in (\text{high}_n, +\infty)\};$$
$$\text{LowerNeuron} = \{n \mid \exists \mathbf{x} \in T: \ \phi(\mathbf{x}, n) \in (-\infty, \text{low}_n)\}.$$

(ii) Strong Neuron Activation Coverage. Strong neuron activation coverage measures how many corner cases (*w.r.t.* the upper boundary value high_n) have been covered by the given test inputs T. It is defined as the ratio of the number of covered corner cases and the total number of corner cases ($|N|$ in this case):

$$\text{SNACov}(T) = \frac{|\text{UpperNeuron}|}{|N|}.$$

Similarly, we define neuron boundary coverage to measure the coverage status of both types of corner cases.

(iii) Neuron Boundary Coverage. Neuron boundary coverage measures how many corner cases (*w.r.t.* both of the upper boundary and the lower boundary values) have been covered by the given test inputs T. It is defined as the ratio of the number of covered corner cases and the total number of corner cases ($2 \times |N|$ in this case):

$$\text{NBCov}(T) = \frac{|\text{UpperNeuron}| + |\text{LowerNeuron}|}{2 \times |N|}.$$

Criteria II: Layer-level coverage

n_3 (depicted in green).

(i) Top-k Neuron Coverage. The top-k neuron coverage measures how many neurons have once been the most active k neurons on each layer. It is defined as the ratio of the total number of top-k neurons on each layer and the total number of neurons in a DNN:

$$\frac{|\bigcup_{\mathbf{x}\in T}(\bigcup_{1\leq i\leq l}\text{top}_k(\mathbf{x},i))|}{|N|}.$$

Generating Natural Adversarial Samples

Zhengli Zhao, Dheeru Dua, Sameer Singh UC Irvine

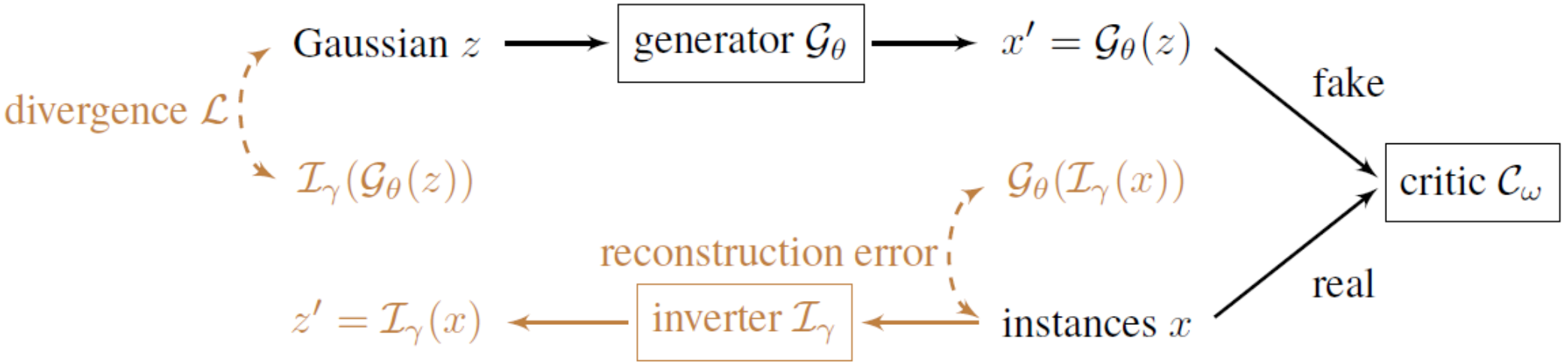
- In this paper, we propose a framework to generate natural and legible adversarial examples that lie on the data manifold, by searching in semantic space of dense and continuous data representation

Natural adversary

- Adversarial example x^* for a given data instance x that results in a different prediction, $f(x^*) \neq f(x)$
- Want to search from dense Z space , not X space

Inverter

- Inverter: Mapping data instances back to corresponding dense representations



$$x^* = \mathcal{G}_\theta(z^*) \text{ where } z^* = \underset{\tilde{z}}{\operatorname{argmin}} \|\tilde{z} - \mathcal{I}_\gamma(x)\| \text{ s.t. } f(\mathcal{G}_\theta(\tilde{z})) \neq f(x).$$

Search algorithm

- Naïve search: Iteratively increase the range Δr, while each step take a random perturbation
- A more complicated search: First determine a range, then doing multiple search on the range
- Doing multiple time and find the best one

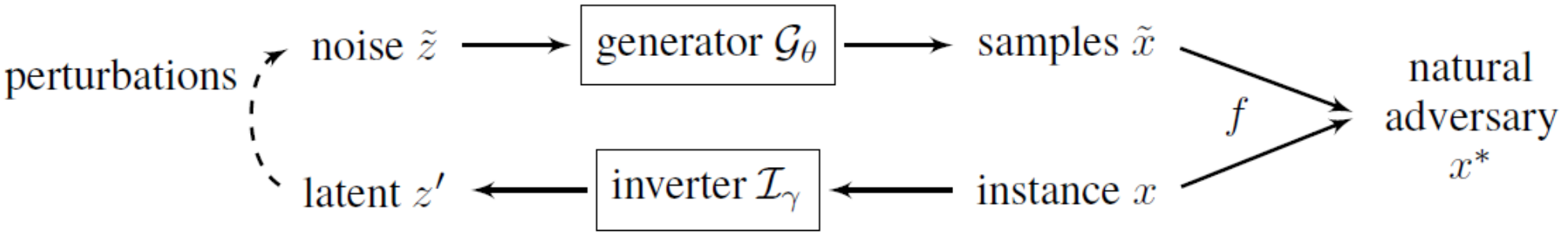
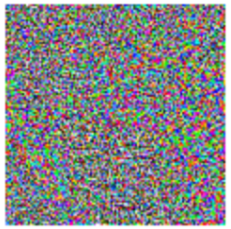


Figure 3: **Natural Adversary Generation.** Given an instance x, our framework generates natural adversaries by perturbing inverted z' and decoding perturbations $\tilde{z}$ via $\mathcal{G}_\theta$ to query the classifier f.

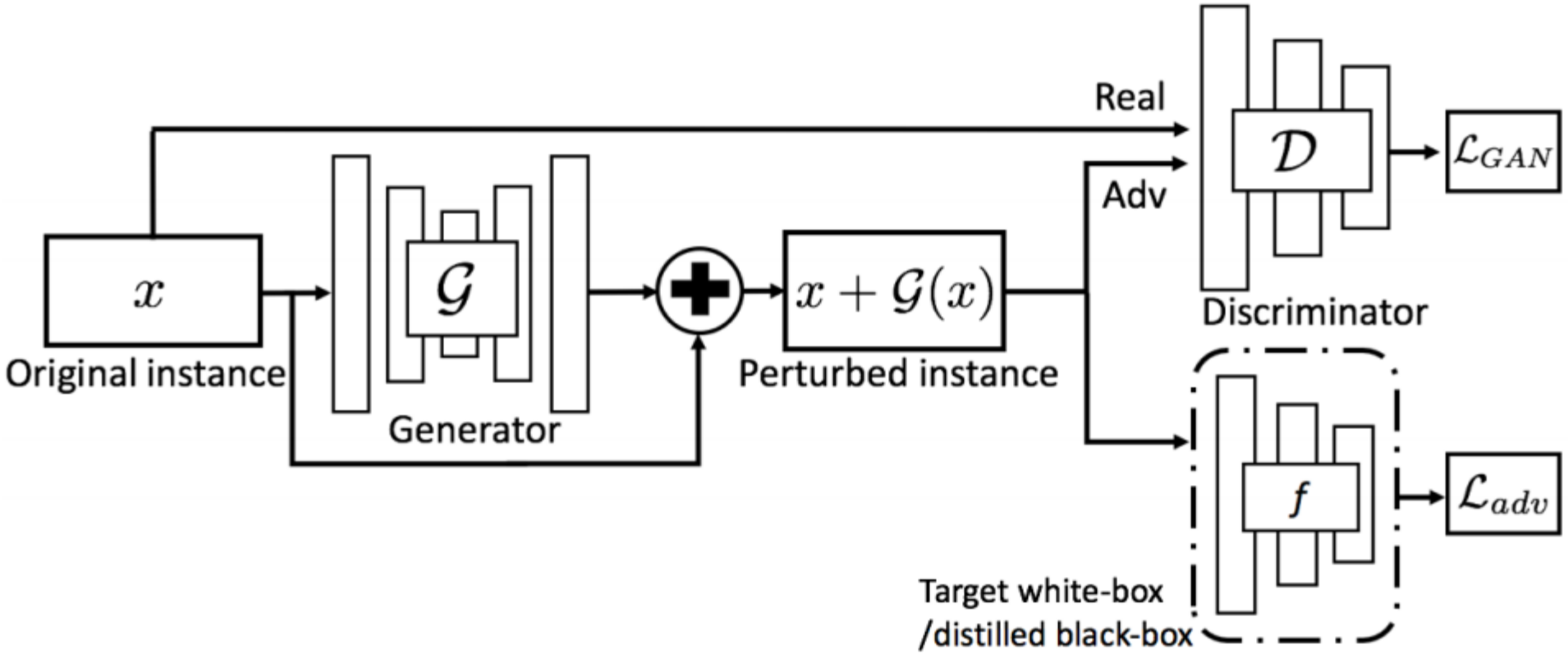
Plan2 – Generate chaos samples

- DNN are bad at the these:



- A lot of people want to build a model that can figure out bad samples
- Seems to be a good target for the generator

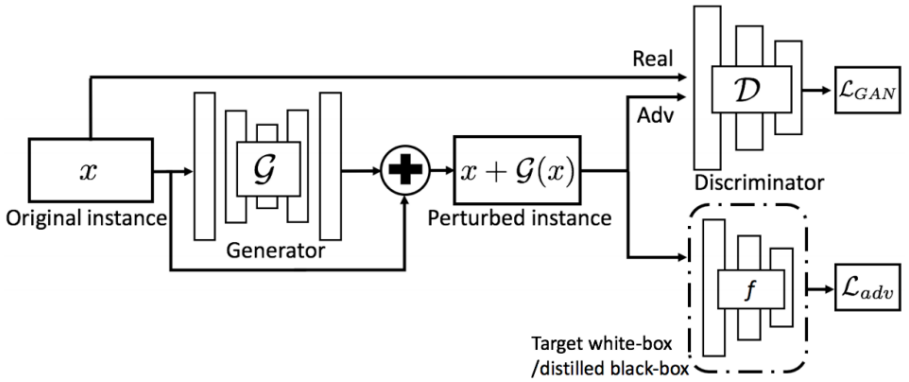
Generating adversarial samples with adversarial networks



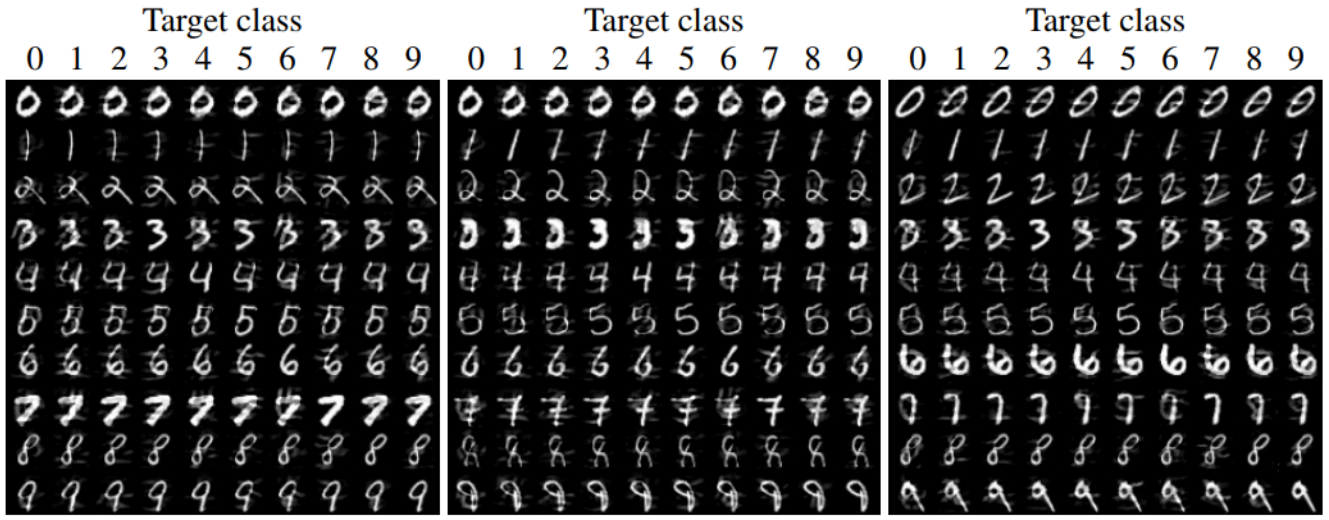
Loss function

- Three parts:
  - GAN loss: $\mathcal{L}_{\text{GAN}} = \mathbb{E}_x \log \mathcal{D}(x) + \mathbb{E}_x \log(1 - \mathcal{D}(x + \mathcal{G}(x)))$.
  - Adversarial sample loss: $\mathcal{L}_{\text{adv}}^{f} = \mathbb{E}_x \ell_f(x + \mathcal{G}(x), t)$,
  - Distance loss: $\mathcal{L}_{\text{hinge}} = \mathbb{E}_x \max(0, \|\mathcal{G}(x)\|_2 - c)$,

$$\mathcal{L} = \mathcal{L}_{\text{adv}}^{f} + \alpha \mathcal{L}_{\text{GAN}} + \beta \mathcal{L}_{\text{hinge}}$$



Experiment result



(a) Model A, semi-whitebox (b) Model B, semi-whitebox (c) Model C, semi-whitebox

(d) Model A, black-box (e) Model B, black-box (f) Model C, black-box

Experiment result

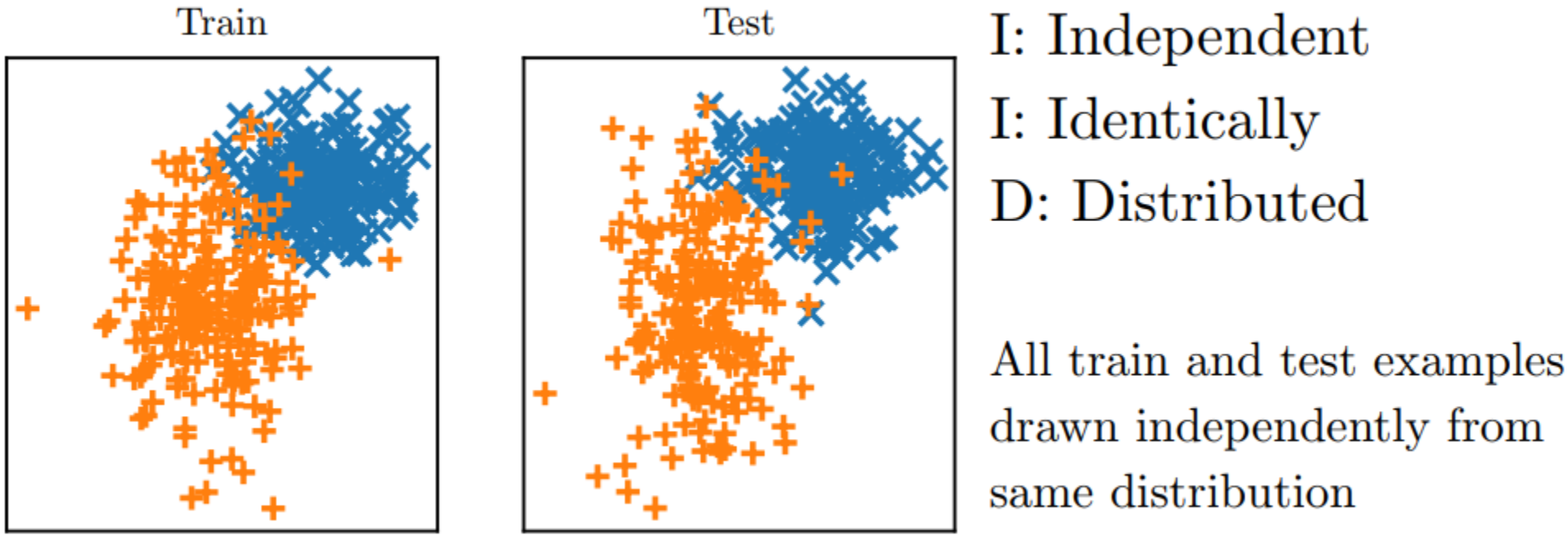
Table 3: Attack success rate of adversarial examples generated by AdvGAN in semi-whitebox setting, and other white-box attacks under defenses on MNIST and CIFAR-10.

Data	Model	Defense	FGSM	Opt.	AdvGAN
MNIST	A	Adv.	4.3%	4.6%	**8.0%**
		Ensemble	1.6%	4.2%	**6.3%**
		Iter.Adv.	4.4%	2.96%	**5.6%**
	B	Adv.	6.0%	4.5%	**7.2%**
		Ensemble	2.7%	3.18%	**5.8%**
		Iter.Adv.	**9.0%**	3.0%	6.6%
	C	Adv.	2.7%	2.95%	**18.7%**
		Ensemble	1.6%	2.2%	**13.5%**
		Iter.Adv.	1.6%	1.9%	**12.6%**
CIFAR	ResNet	Adv.	13.10%	11.9%	**16.03%**
		Ensemble.	10.00%	10.3%	**14.32%**
		Iter.Adv	22.8%	21.4%	**29.47%**
	Wide ResNet	Adv.	5.04%	7.61%	**14.26%**
		Ensemble	4.65%	8.43%	**13.94 %**
		Iter.Adv.	14.9%	13.90%	**20.75%**

Defense against the dark arts: an overview of adversarial example security research and future research directions

Ian Goodfellow

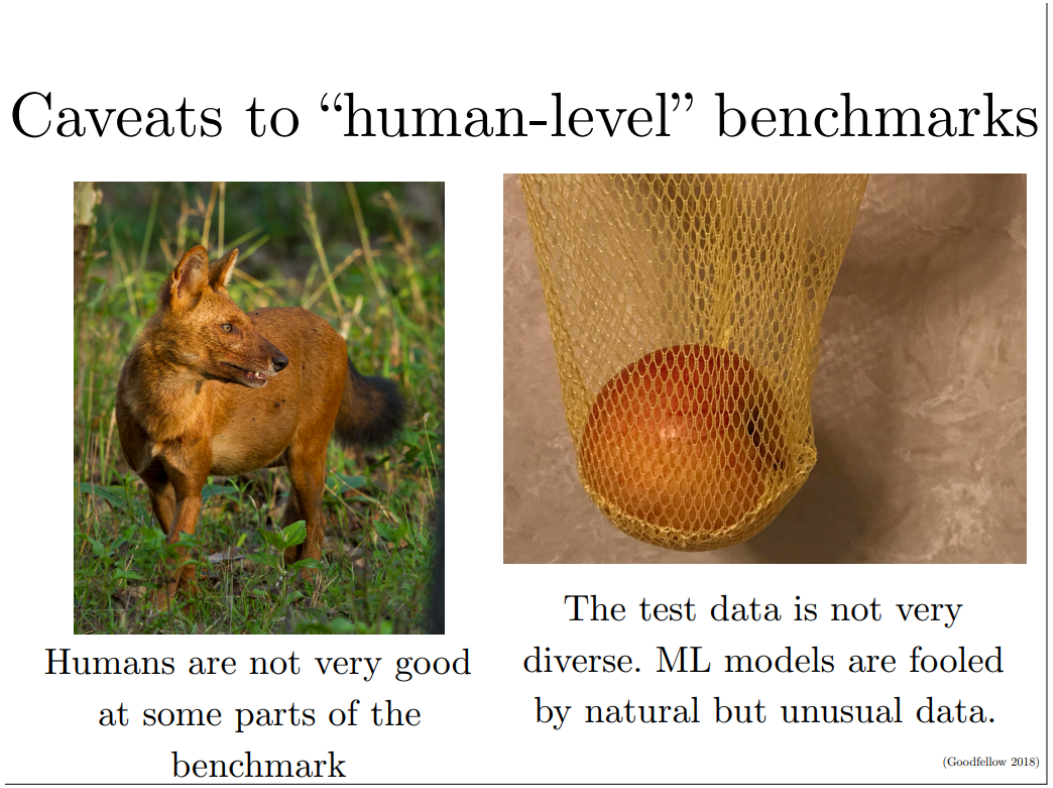
- Machine learning mostly have IID assumption
  - Independent identically distributed



Defense against the dark arts: an overview of adversarial example security research and future research directions

Ian Goodfellow

- Machine learning mostly have IID assumption
  - Independent identically distributed



Defense against the dark arts: an overview of adversarial example security research and future research directions

Ian Goodfellow

- Machine learning mostly have IID assumption
  - Independent identically distributed
- Security requires moving beyond IID: Attacker can use unusual inputs
- Adversarial sample -> Intentionally designed input to mislead classifier

Security Requires Moving Beyond I.I.D.

- Not identical: attackers can use unusual inputs

(Eykholt et al, 2017)

- Not independent: attacker can repeatedly send a single mistake ("test set attack")

Defense against the dark arts: an overview of adversarial example security research and future research directions

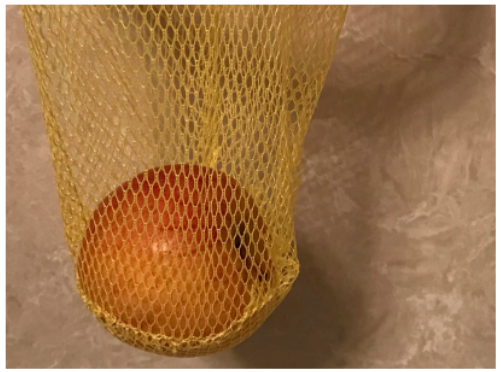
Ian Goodfellow

- Adversarial sample:
- No need to be small perturbation
- No need to be human, but some oracle
- Intend to cause misclassification, but no need to succeed

Definition

"Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake"

(Goodfellow et al 2017)



Defense against the dark arts: an overview of adversarial example security research and future research directions

Ian Goodfellow

- Game view of adversarial sample

Define a game

- Define an action space for the defender
- Define an action space for an attacker
- Define cost function for defender
- Define cost function for attacker
  - Not necessarily minimax.
  - Targeted vs untargeted

Norm balls

- Most results are on norm b
- Norm balls: A toy game
- Current lab research not suitable for true

Norm Balls: A Toy Game

- How to benchmark performance on points that are not in the dataset and not labeled?
- Propagate labels from nearby labeled examples
- Attacker action:
  - Given a clean example, add a norm-constrained perturbation to it
- The *drosophila* of adversarial machine learning
- Interesting for *basic research* purposes because of its clarity and difficulty
- Not relevant for most practical purposes: not a *current, applied* security problem
- In my view, this shouldn't be primarily about human perception

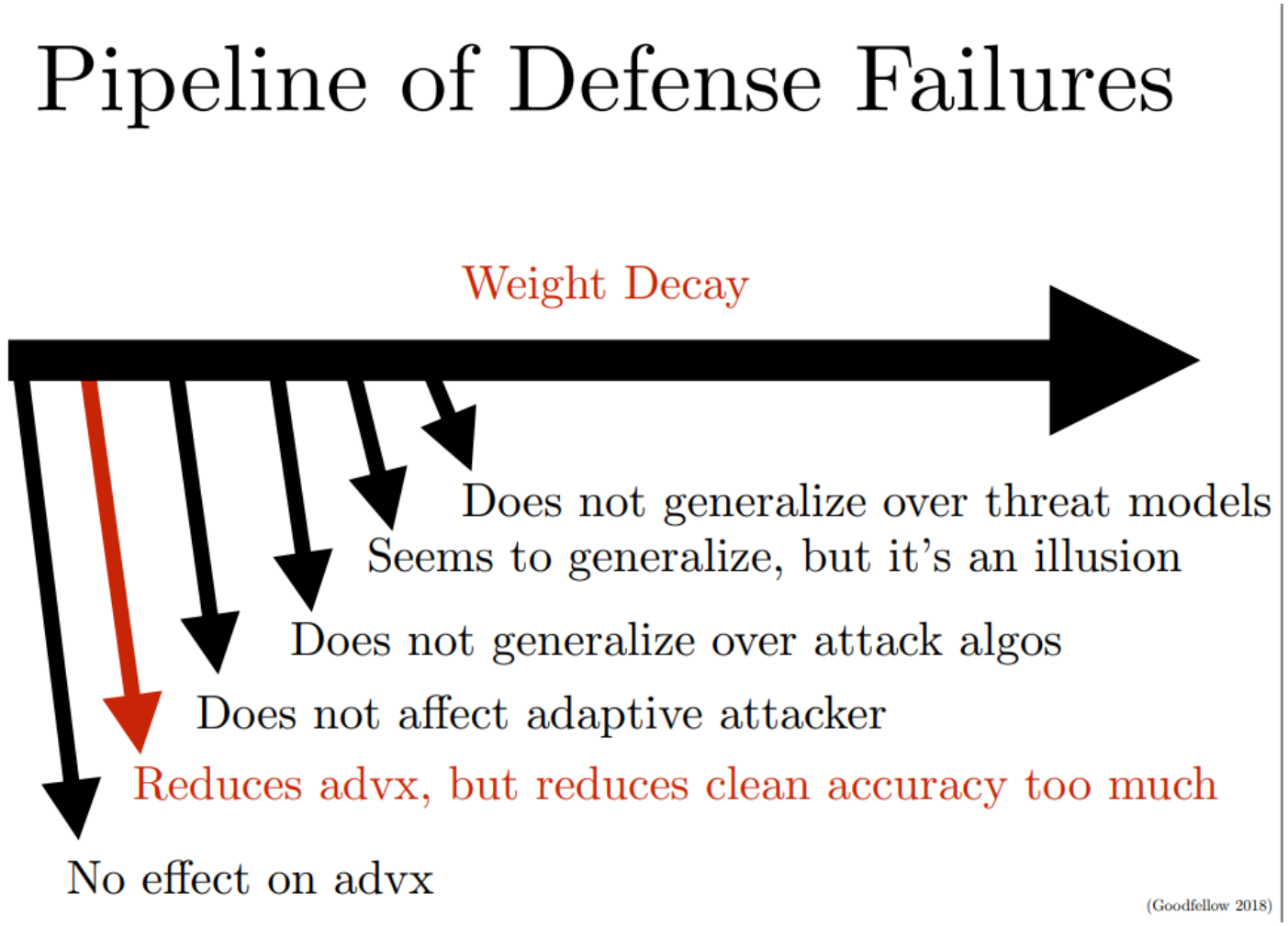
Gradient Masking

Gradient Masking

- Some defenses look like they work because they break gradient-based white box attacks
- But then they don't break black box attacks (e.g., adversarial examples made for other models)
- The defense denies the attacker access to a useful gradient but does not actually make the *decision boundary* secure
- This is called *gradient masking*

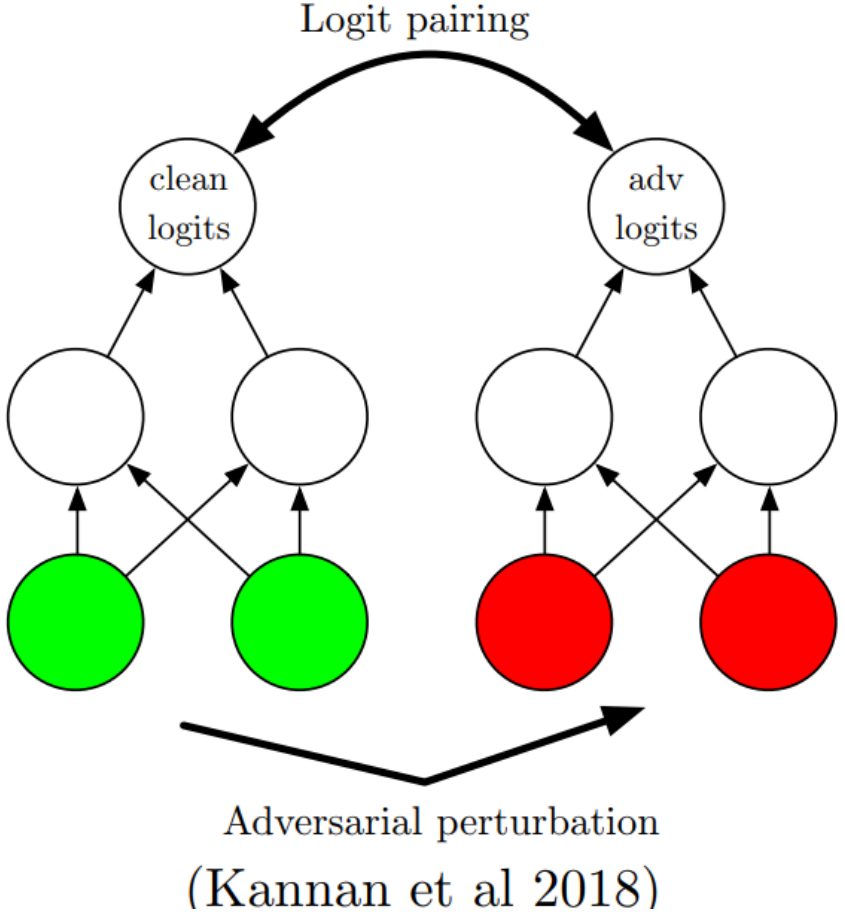
Defenses

- Dropout: No effect
- Weight decay(Add a regularization term with weight): reduce both adv and clean accuracy
- Cropping: Not adaptive
- Adversarial Training: Not generalize to other models
- Distillation



Current state-of-the-art

Adversarial Logit Pairing (ALP)



(Kannan et al 2018)

First approach to achieve >50% top-5 accuracy against iterative adversarial examples on ImageNet

Current state of the art

Disappointment on current adv research

- Disappointed ->
- Ian wants to find simple ways to solve the adversarial problem, do well in the measure in norm ball, but also on the points that are hard to measure
- Outcomes: Best result = Direct optimization

Future

- Hope:
  - No direct optimization
  - Indirect ways
- Better attack models: Not norm balls
- New benchmarks
- Study more problems other than vision