Fault Injection Attack on Deep Neural Network

CONV Layer Tensor Computation

Output fmaps (O)

Biases (B)

Input fmaps (I)

Filter weights (W)

$$\underline{\mathbf{O}[n][m][x][y]} = \text{Activation}(\underline{\mathbf{B}[m]} + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \underline{\mathbf{I}[n][k][Ux+i][Uy+j]} \times \underline{\mathbf{W}[m][k][i][j]}),$$

$$0 \le n < N, 0 \le m < M, 0 \le y < E, 0 \le x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

What is attack?

Theoretical:

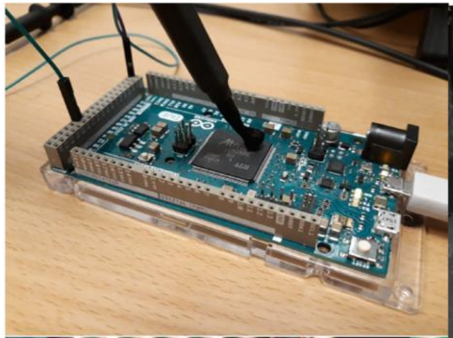
- Outputs of a DNN depend on both input patterns and its own internal parameters
- misclassify a given input pattern into any specific class by manipulating some parameters in the DNN.

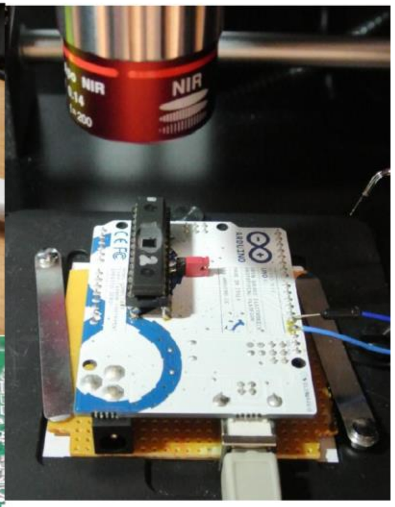
Practical:

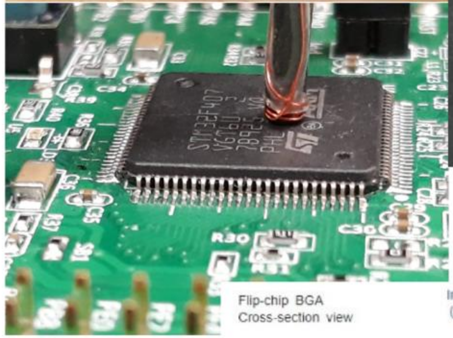
- DNNs can be implemented in different platforms (e.g., CPU/GPU and dedicated accelerator)
- the parameters are usually stored in memory
- With the development of precise memory fault injection techniques: laser beam fault injection and row hammer attack : possible to launch effective fault injection attacks on DNNs.

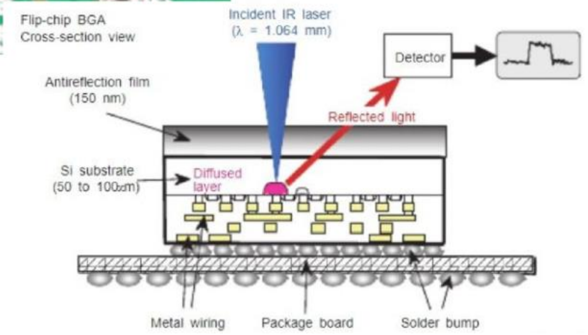
Laser beam fault injection

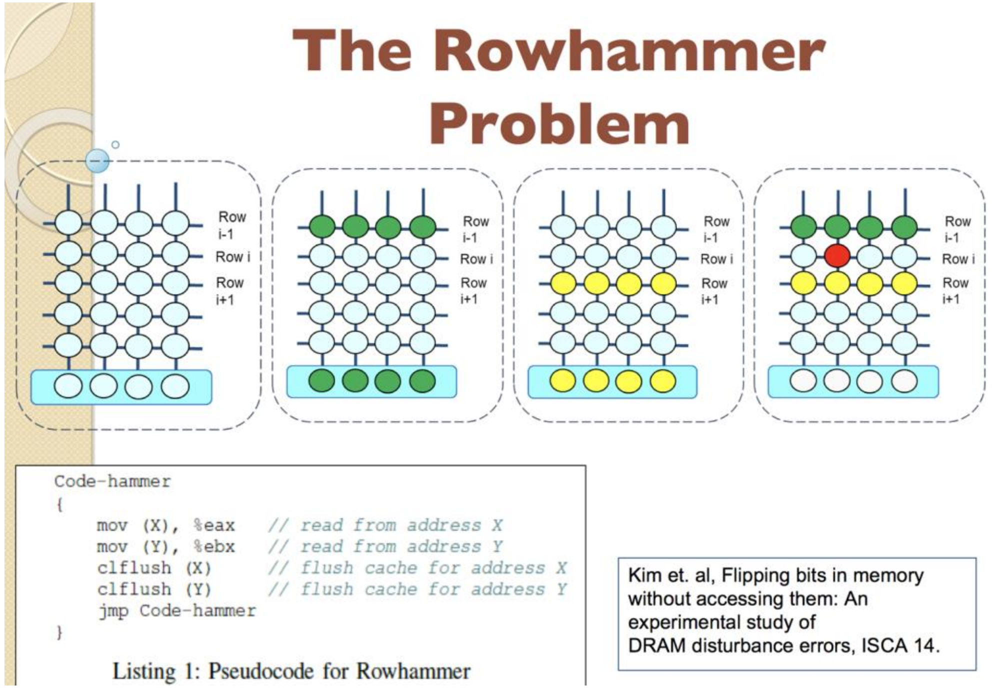
- Laser beam can inject fault into SRAM.
- By exposing the silicon under laser beam, a t**emporary conductive channel** is formed in the dielectric, which in turn causes the transistor to switch state in a precise and controlled manner.
- By carefully adjusting the parameters of laser beam, like its diameter, emitted energy, and impact coordinate, attacker can precisely change any single bit in SRAM.











The Rowhammer Problem

```
Code-hammer
{
    mov (X), %eax    // read from address X
    mov (Y), %ebx    // read from address Y
    clflush (X)      // flush cache for address X
    clflush (Y)      // flush cache for address Y
    jmp Code-hammer
}
```

Listing 1: Pseudocode for Rowhammer

Kim et. al, Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors, ISCA 14.

Row hammer can inject fault into DRAM.

It exploits the electrical interactions between neighboring memory cells.

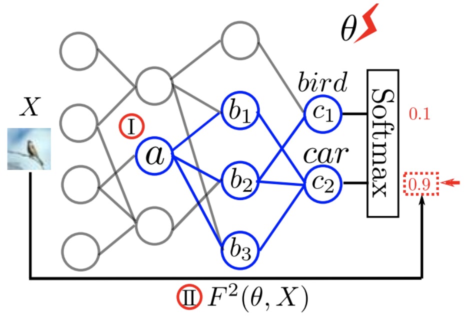
By rapidly and repeatedly accessing a given physical memory location, a bit in its adjacent location may flip.

By profiling the bit flip patterns in DRAM module and abusing the memory manage features, row hammer can reliably flip a single bit at any address in the software stack.

While it is simple to inject faults to modify the parameters used in DNNs, achieving misclassification for a particular input pattern is a challenging task.

First, the large amount of parameters in common DNN hinder designing efficient attack method with minimum number of injected faults.

Second, changing DNN parameters not only misclassifies the targeted input pattern, but also may misclassify some other unspecified input patterns. Hence, the attack may not be stealthy.



Threat Model and Attack Objective

Threat Model:

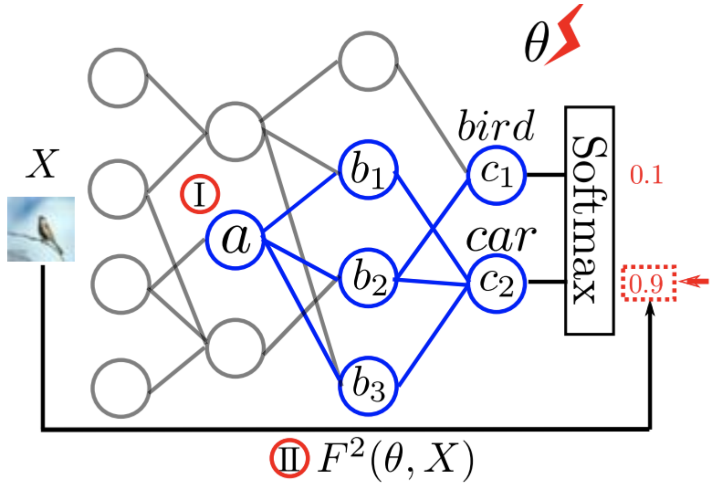
- an attacker aiming at changing the output of a DNN-based classification system.
- **white-box attack : the attacker has the knowledge of targeted DNN's network structure, the benign parameters and low-level implementation details, e.g., parameters' location in memory.**
- the attacker do not know the data samples for training or testing.
- the attacker can modify any parameter in DNN (i.e., any weight or any bias in DNN) by injecting faults into the memory where the parameters are stored.
- Given existing fault injection techniques can precisely flip any bit of the data in memory, we assume the attacker can modify any parameter in DNN to any adversarial value that is in the valid range of used arithmetic format.

Attack Objective

To find the modification on parameters to misclassify a given input pattern into a specific adversarial class, e.g., misclassifying the bird in the given image as a car as described in Fig.

Efficiency of achieving misclassification:

- To make the attack practical, attacker should modify as few parameters as possible.
- Though DNN can be implemented on different platforms recording parameters in different arithmetic formats, modifying more parameters tends to require injecting more faults.
- minimizing the number of modified parameters can reduce the fault injection cost on all platforms.



Stealthiness

Parameter modification may not only misclassify the given input pattern (e.g., the given image), but also misclassify the rest unspecified input patterns (e.g., all the other images).

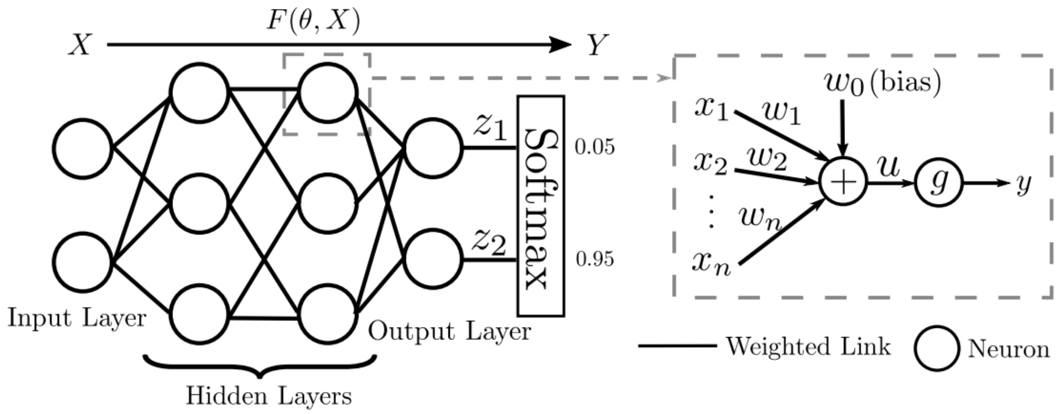
Keeping the attack stealthy for these input patterns is equivalent to preserving original DNN’s classification accuracy on these unspecified input patterns as much as possible.

Single bias attack (SBA)

Single Bias Attack (SBA)

- To achieve misclassification by only modifying one parameter of DNN.
- Since the outputs of DNN linearly depend on the biases in the output layer, **SBA can be simply implemented by enlarging the corresponding bias in output layer to the corresponding adversarial class.**

Preliminaries



$$y = g(u) \quad \text{and} \quad u = \sum_{1 \le i \le n} w_i x_i + w_0,$$

- The neurons in all hidden layers usually use the same type of activation function

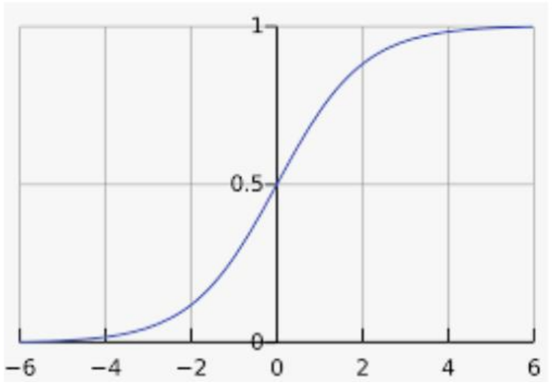
  .

- **We name DNN using ReLUlike activation function in hidden layers as ReLU-like DNN for short.**

Revisiting Activation Functions

$$softmax(i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Softmax function calculates the probability of given input pattern belonging to class i.



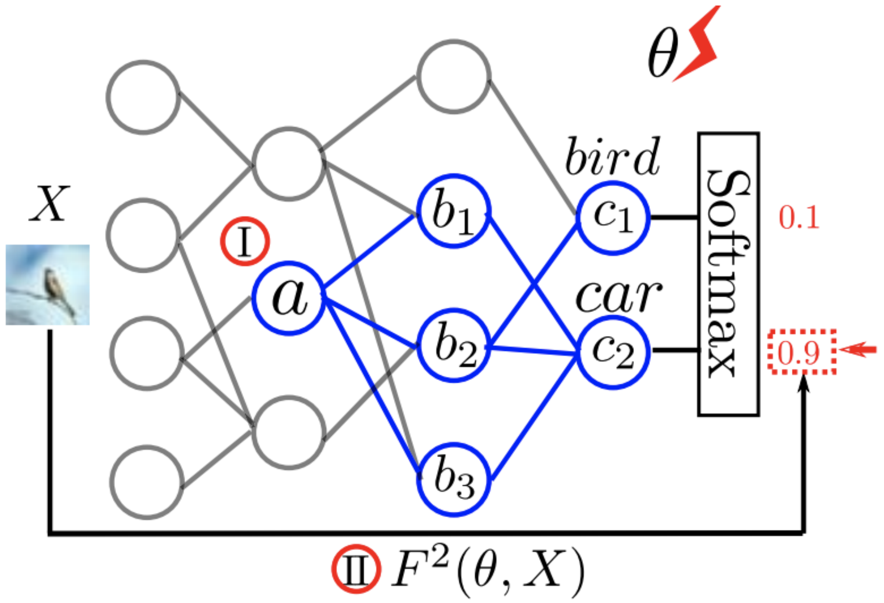
- The output neuron with the largest output has the largest probability after softmax function.
- So the output of DNN can be forced to be class i, by making the i th output neuron has the largest output among all output neurons.
- In DNNs for classification, because identify function is used as activation function in output layer, each output neuron's output linearly depends on its bias.
- Hence, for any input pattern, we can change DNN's output to class i by enlarging the bias of the i th output neuron.

Attack in hidden layers

We can attack not only the biases in output layer but also the biases in hidden layers, if the activation function g used in hidden layers is a linear function.

Consider the attack scenario indicated by I in Figure:

- where a is a hidden neuron, c1 and c2 are the output neurons, and b1, b2, and b3 are the neurons on the paths from a to output layer.
- Supposing an attacker increasing a’s bias, then the outputs of connected neurons are all affected.
- `a’ linearly depends on its bias, b1, b2, and b3 also linearly depend on a’s bias, and finally c1 and c2 linearly depend a’s bias as well.
- In this case, if the derivate of c2 w.r.t. a’s bias is larger than c1’s, c2 must be larger than c1 when a’s bias becomes sufficient large, and DNN’s output is changed to class 2.



Challenge

practical activation function used in the hidden layers is non-linear, which makes extending SBA to hidden layers non-trivial.

Observation:

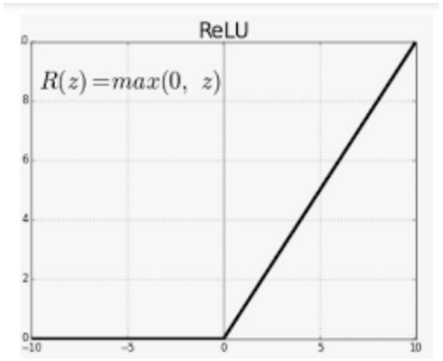
- ReLU-like DNN linearly depend on a bias in hidden layers when the bias is sufficiently large (one side linear phenomenon)
- enables us to attack biases in hidden layers.

Revisiting Activation Functions

$$g(u) = \begin{cases} u & \mathrm{u} \geq 0 \\ \alpha u & \mathrm{u} < 0 \end{cases}$$

ReLU-like activation functions:

- α is 0 for the original ReLU and α is 0.01 for leaky ReLU



Because the derivatives through ReLU are both large and consistent during backpropagation, it makes the training process easier compared to other kinds of activation functions like sigmoid and tanh.

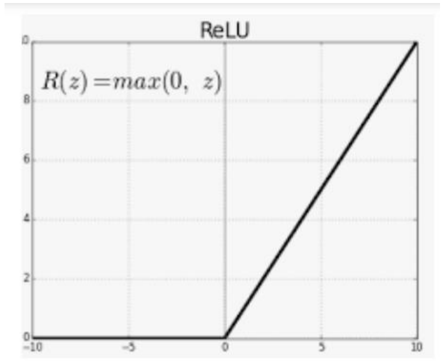
There are also some varieties of ReLU which use a non-zero slope α when u < 0.

Revisiting Activation Functions

$$g(u) = \begin{cases} u & \mathrm{u} \geq 0 \\ \alpha u & \mathrm{u} < 0 \end{cases}$$

ReLU-like activation functions:

- α is 0 for the original ReLU and α is 0.01 for leaky ReLU



Because the derivatives through ReLU are both large and consistent during backpropagation, it makes the training process easier compared to other kinds of activation functions like sigmoid and tanh.

There are also some varieties of ReLU which use a non-zero slope α when u < 0.

Overall Attack Flow

- profile the sink class for each bias for the given DNN.
- according to the specific adversarial mapping, one only need to choose one bias whose sink class is the same as the targeted class in the adversarial mapping
- find the required increment on this bias to saturate DNN’s output.

One side linear slope

Definition 1. Given two variables x and y, if there exist two constants, ϵ and δ such that $dy/dx = \delta$ when $x > \epsilon$, we say y is one side linear to x and the one side linear slope is δ.

If multiple variables $y1, y2,...,yn$ are all one side linear to variable x with different slopes, then $y1, y2,...,yn$ will change at different rates when x increases. If the increment on x is larger enough, the variable in $y1, y2,...,yn$ increasing at the fastest rate would finally become the largest one.

Theorem 1. Given variables $y1, y2,...,yn$ which are all one side linear to variable x with slopes $\delta 1, \delta 2,...,\delta n$ respectively. If $\delta n > \delta i$ for $\forall i \neq n$, there must exist a constant sink such that $yn > yi$ for $\forall i \neq n$ when $x>sink$.

Lemma 1. *If h is a neuron in the hidden layers of ReLU-like DNN, then h is one side linear to its bias m with one side linear slope 1.*

Proof. we have $h = g(\sum_{1\leq i\leq n} w_i x_i + m)$. Given g is ReLU-like function if $m > |\sum_{1\leq i\leq n} w_i x_i|$, $h = \sum_{1\leq i\leq n} w_i x_i + m$. Hence, h is one side linear to m and the slope is 1. □

Lemma 2. *Given a neuron h and a bias m in ReLU-like DNN, if all h's inputs $x_1, x_2, \ldots, x_n$ are one side linear to m with slopes $\delta_1, \delta_2, \ldots, \delta_n$ respectively, then h is also one side linear to m with slope*

$$g\left(\sum_{1 \leq i \leq n} w_i \delta_i\right),$$

where g is the activation function and w_i is the weight corresponding to i^{th} input.

Proof. According to Definition 1, for any $i \in [1, n]$, there must exist a constant ϵ_i such that $\frac{dx_i}{dm} = \delta_i$ when $m > \epsilon_i$. Suppose $\epsilon_{max} = max\{\epsilon_1, \ldots, \epsilon_n\}$ and $u = \sum_{1 \le i \le n} w_i x_i + w_0$, where w_0 is h's bias. When $m > \epsilon_{max}$, we have

$$\frac{du}{dm} = \frac{d\left(\sum_{1 \le i \le n} w_i x_i + w_0\right)}{m} = \sum_{1 \le i \le n} w_i \cdot \frac{dx_i}{dm} = \sum_{1 \le i \le n} w_i \delta_i .$$

What is sink class?

- Sink class represents the output class at which DNN’s output converges when a bias increases.
- the one side line slope of a output neuron w.r.t. a bias is independent with the input pattern. Hence, one bias’ sink class is the same for different input patterns.

Theorem 2. In ReLU-like DNN, for a bias m in hidden layers, every output neuron is one side linear to m.

Proof. Suppose the DNN has N layers and m is in the i th layer.

The neurons in the same layer are independent.

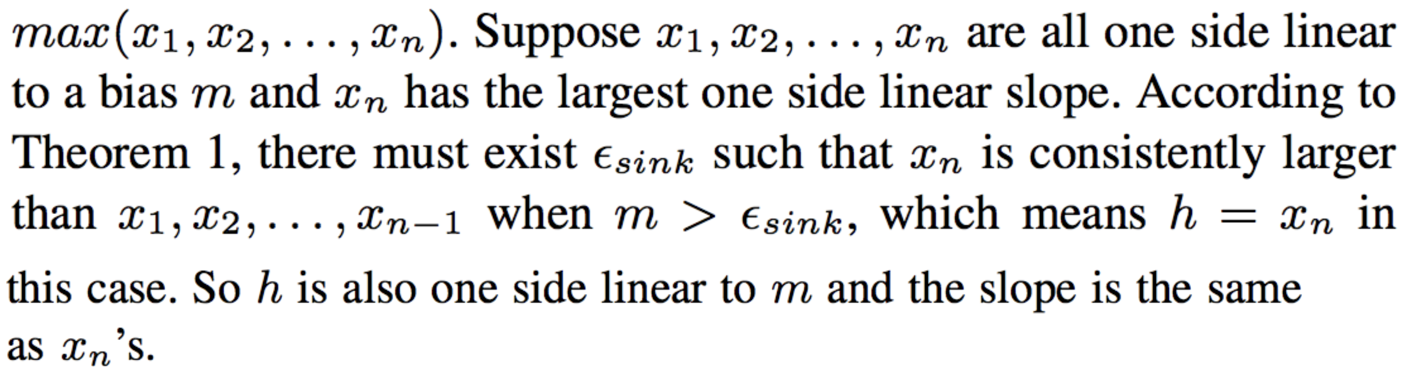
Given a neuron in the i th hidden layer, it is either independent with m or using m as its bias. A neuron independent with m is one side linear to m with slope 0,

A neuron using m as bias is one side linear to m.

With above, all the neurons in i th layer are one side linear to m.

Because the neurons in (i + 1)th layer only depends on neurons in i th layer, all neurons in (i + 1)th layer are also one side linear to m (By induction), all neurons in the output layer are one side linear to m.

Maxpooling



$max(x_1, x_2, \ldots, x_n)$. Suppose $x_1, x_2, \ldots, x_n$ are all one side linear to a bias m and x_n has the largest one side linear slope. According to Theorem 1, there must exist ϵ_{sink} such that x_n is consistently larger than $x_1, x_2, \ldots, x_{n-1}$ when $m > \epsilon_{sink}$, which means $h = x_n$ in this case. So h is also one side linear to m and the slope is the same as x_n's.

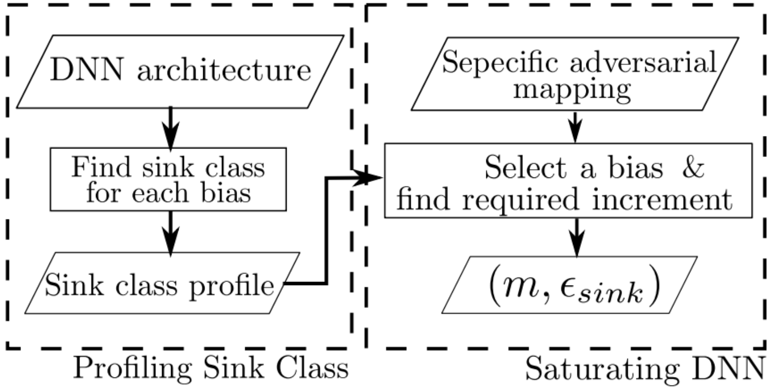
Hence, the one side linear property can propagate from previous layer to next layer. By induction, the output neurons are one side linear to any bias.

Attack steps

1. Profiling Sink Class.

To obtain the sink class for a bias:

- Identify each output neuron's one side linear slope w.r.t. this bias and check whether unique largest slope exists.
- Calculate the slope for each output neuron w.r.t. a given bias m by induction. At first, we need to initialize the slopes for all neurons in i th layer where m is located. If a neuron uses m as its bias, we set the slope for this neuron to be 1, otherwise it is 0.
- Then we can calculate the slopes layer by layer according to Lemma 2.
- Finally, we can obtain the one side linear slope for each output neuron.
- If unique largest slope exists in the output layer, we can determine the sink class for m.

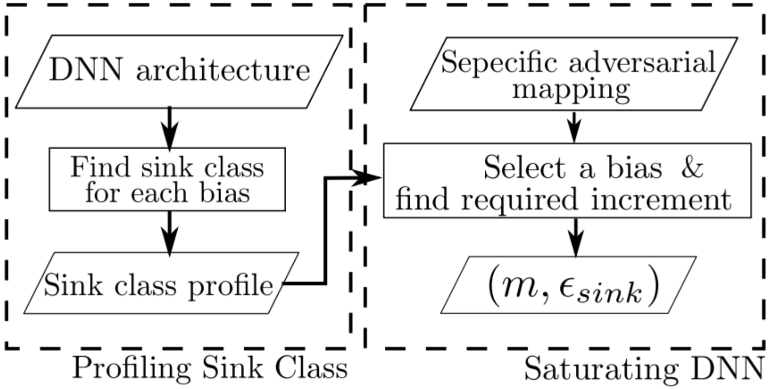

DNN architecture
Find sink class for each bias
Sink class profile
Sepecific adversarial mapping
Select a bias & find required increment
(m, ϵ_{sink})
Profiling Sink Class
Saturating DNN

Attack steps

2. Saturating DNN.

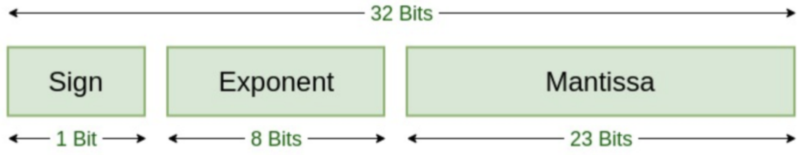
To achieve the adversarial mapping:

- choose any bias m whose sink class is the same as the targeted class in the specific adversarial mapping.
- To make DNN output the m’s sink class, only need to modify m so that m is larger the sink value defined in Theorem 1.
- determine the exact sink value for a given input pattern and selected bias m, by scanning m values and monitoring when DNN’s output converges.
- If attacker can check the sink for each candidate bias, bias with the smallest sink can be used during attack.

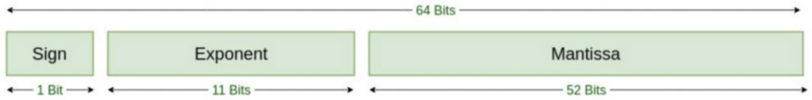

DNN architecture
Find sink class for each bias
Sink class profile
Profiling Sink Class
Sepecific adversarial mapping
Select a bias & find required increment
(m, ϵ_{sink})
Saturating DNN

Practical implementation

- During practical attack, we do not need to precisely modify the bias, i.e., enlarging the bias beyond sink is sufficient.
- To achieve this, we may only need to inject several single bit faults on the most significant bits of the bias.
- For instance, when floating point format is used to store parameter, we can efficiently enlarge the bias by magnitudes via making its sign bit positive and setting several most significant bits in the exponent field.



IEEE 754 Single Precision



IEEE 754 Double Precision

```
85.125
85 = 1010101
0.125 = 001
85.125 = 1010101.001
       =1.010101001 x 2^6
sign = 0
```

```
1. Single precision:
biased exponent 127+6=133
133 = 10000101
Normalised mantisa = 010101001
we will add 0's to complete the 23 bits

The IEEE 754 Single precision is:
= 0 10000101 01010100100000000000000
This can be written in hexadecimal form 42AA4000
```

```
2. Double precision:
biased exponent 1023+6=1029
1029 = 10000000101
Normalised mantisa = 010101001
we will add 0's to complete the 52 bits

The IEEE 754 Double precision is:
= 0 10000000101 0101010010000000000000000000000000000000000000000000
This can be written in hexadecimal form 4055480000000000
```

stealthiness of single bias attack

- depends on the sink value for the input pattern, suppose an attacker can precisely modify the bias to be slightly larger than the sink.
- different input patterns may require different sink values to saturate DNN’s output when attacking the same bias.
- when misclassifying a input pattern with sink = k, the outputs for other input patterns with sink < k on this bias must converge
- the outputs for input patterns with sink > k may be also affected although not converge. Hence, the accuracy degradation is larger when attacking input pattern with larger sink on a bias.
- To mitigate this, one possible solution is to recover the DNN to benign state with additional faults in addition to the few faults injected during SBA.

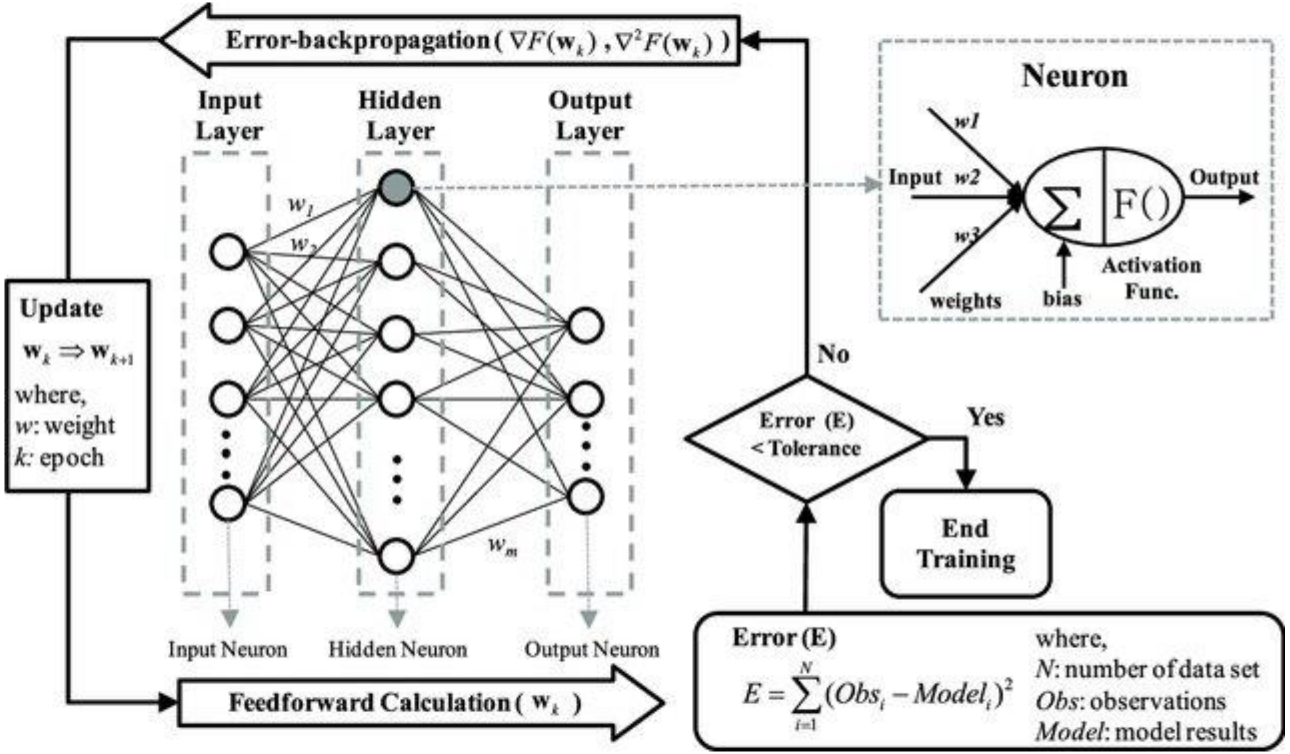
Advantage: Exact input pattern is not known

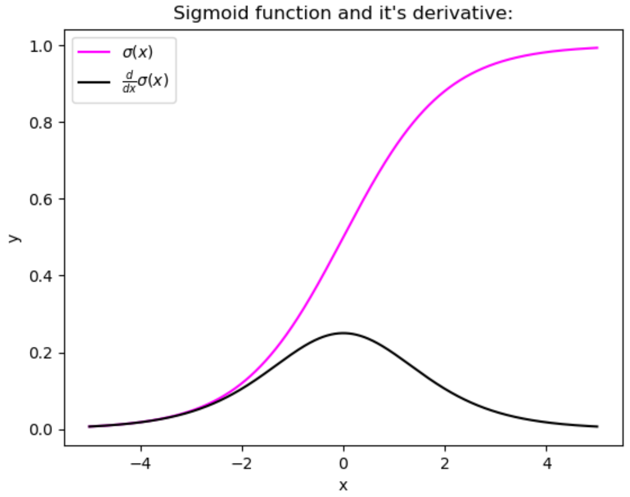
- Even without knowing the exact input pattern, it is still possible to force DNN generating specific adversarial output with SBA.
- Because the sink class of a bias is independent from input patterns, we could saturate DNN by modifying the bias to be a very large valid value allowed by the arithmetic format used.
- When it is larger than the required sink for an input pattern, we can successfully change the DNN’s output. Such attack can be launched at real-time (without necessarily conducting input pattern analysis), leading to possible denial-of-service

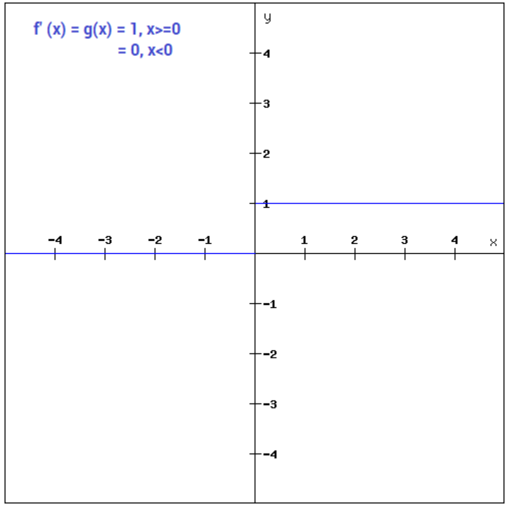
Gradient descent attack (GDA)

[Gradient descent (GD) is **an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function**.]

Revisit backpropagation





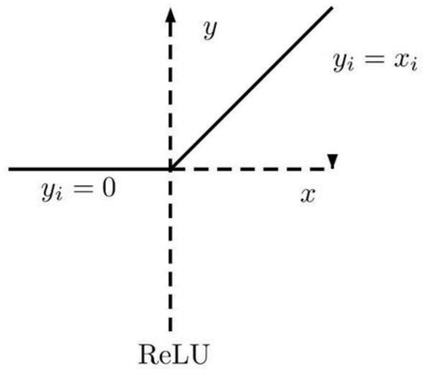


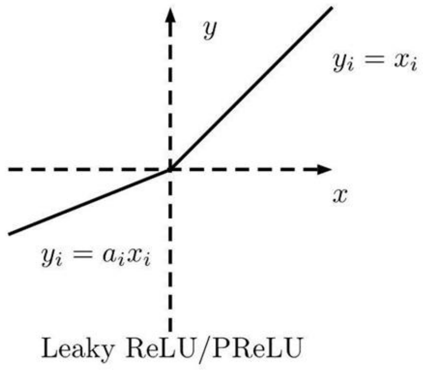
Sigmoid derivative

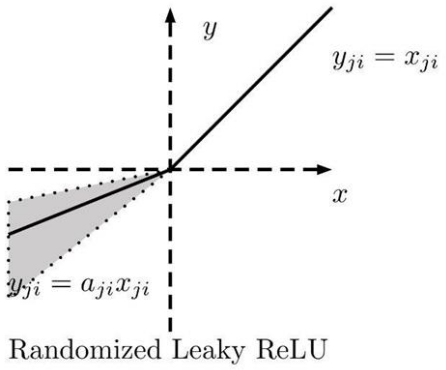
ReLU derivative

A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session.

Leaky ReLU







Solves Dying ReLU problem: The dying ReLU problem refers to **the scenario when many ReLU neurons only output values of 0**.
Because the slope of ReLU in the negative input range is also zero, once it becomes dead (stuck in negative range)

Gradient descent attack (GDA)

- **to force stealthy misclassification, which tries to preserve the classification accuracy on input patterns other than the targeted one.**
- DNN inherently tolerates small perturbations on its parameters, GDA introduces slight changes to the DNN’s parameters to achieve misclassification, thereby mitigating the impact on the other input patterns.
- layer-wise searching and modification compression:
- to further improve the stealthiness and the efficiency,
- by searching the perturbation at finer granularity and removing insignificant part of the obtained perturbation, respectively.

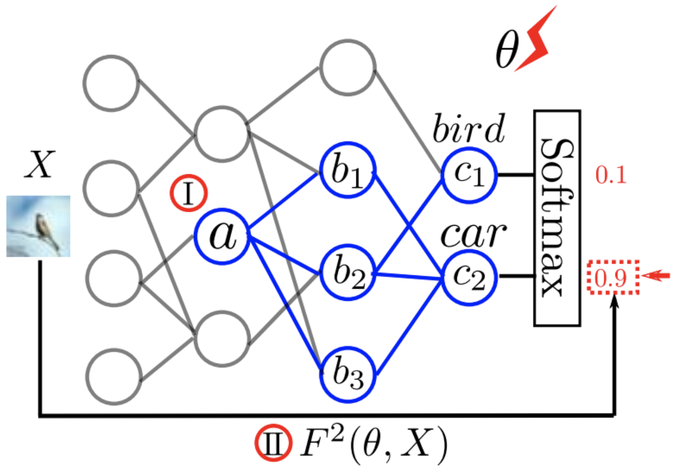
Method

.

To find small perturbation, one naive method is using gradient descent to gradually impose perturbation on parameters.

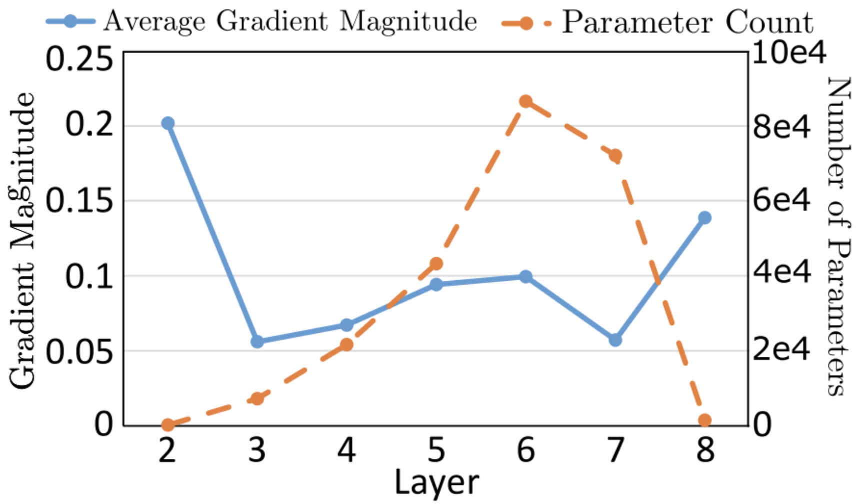
- First initialize θ (i.e., all parameters in DNN) with its benign value, denoted by θ_b
- gradually modify θ in the direction of gradient $dF_i(\theta,x)/d\theta$ to enlarge Fi (θ, x), where Fi (θ, x) represents the output probability of class i as indicated by II in Figure.
- use L1-norm regulator to restrict the amount of perturbation during searching.
- Maximize the following objective function with gradient descent:

$$J(\theta) = F^i(\theta, x) - \lambda|\theta - \theta_b|$$



What parameter to choose?

- The magnitude of parameter’s gradient is not uniformly distributed among layers, and **parameters in layer having fewer parameters trend to have gradient of larger magnitude**.
- Figure shows the average magnitude of parameter’s gradient and parameter count in each layer for the MNIST model, and layer2 with the fewest parameters has the largest gradient.



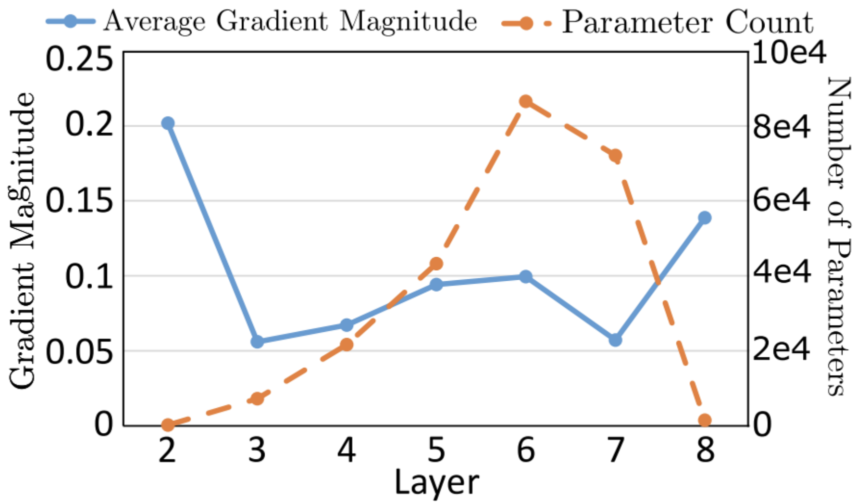
The distribution of parameter’s gradient and parameter count.

Solution: Layer-wise Searching

- instead of finding the modification global-wise, manually restrict the θ in Eq. 4 to be the set of all parameters within one layer.
- I**ntuitively, a layer with more parameters allows us to adjust the mapping F at finer granularity and optimize the objective function better**, i.e., introducing smaller perturbation.
- While the naive globalwise searching in fact focuses on adjusting the smallest layer, we can conduct layer-wise searching on the layer with the most parameters to search the solution at the finest granularity.
- Conducting layer-wise searching on any layer, except the smallest layer, can achieve higher stealthiness than the global-wise searching (Experimental observation)

Experimental Observation

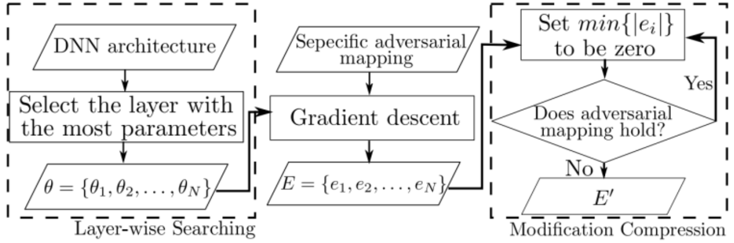
- **gradient descent modifies each parameter according to its gradient:**
  - **parameter with gradient of larger magnitude receives larger modification during searching.**
- As a consequence, the naive method which is supposed to find optimal modification global-wise in fact focuses on modifying the few parameters in the smallest layer, e.g., layer2 in Figure.



The distribution of parameter's gradient and parameter count.

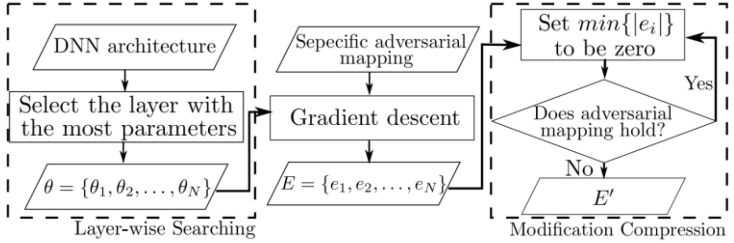
Overall flow of gradient descent attack

- At first, determine the set of parameters to be updated in gradient descent.
- Instead of updating all the parameters in DNN and searching the modification global-wise, search modification within a single layer, i.e., layer-wise searching.
- Next, optimize Eq. (previous slide) with gradient descent to find the **modification on each selected parameter, denoted by E = {e1, e2,...,eN }.**
- At last, we repeatedly replace the smallest element in E with zero until the final parameter modification E is obtained, i.e, modification compression.


DNN architecture
Select the layer with the most parameters
$\theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$
Layer-wise Searching
Sepecific adversarial mapping
Gradient descent
$E = \{e_1, e_2, \ldots, e_N\}$
Set $min\{|e_i|\}$ to be zero
Does adversarial mapping hold?
Yes
No
E'
Modification Compression

Modification Compression

- Considering changing E too much may destroy the required adversarial mapping, achieve the replacement by iteration.
- At each iteration step, only replace the element with smallest absolute value in E to be zero and check whether the adversarial mapping is still preserved.
- If it is preserved, we continue to explore further replacement. Otherwise, only previously found replacements are finally used.
- With modification compression, we can not only reduce the number of modified parameters but also improve the stealthiness, because replacing the elements in E with zero reduces the amount of perturbation imposed.


DNN architecture
Select the layer with the most parameters
$\theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$
Layer-wise Searching
Sepecific adversarial mapping
Gradient descent
$E = \{e_1, e_2, \ldots, e_N\}$
Set $min\{|e_i|\}$ to be zero
Does adversarial mapping hold?
Yes
No
E'
Modification Compression

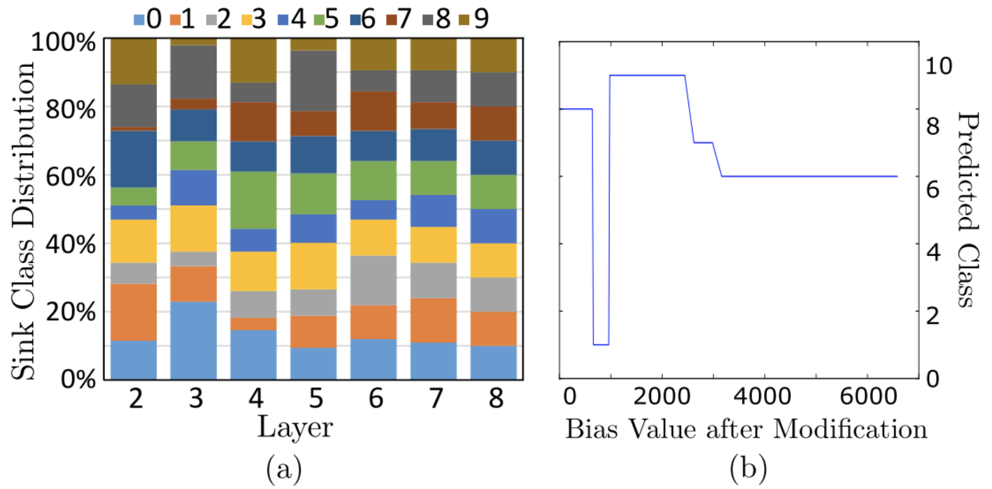
Results

Evaluate the proposed fault injection attack methods with the well-known MNIST and CIFAR10 datasets.

demonstrate that SBA can achieve misclassification by imposing moderate increment on single bias in any layer of a DNN using ReLU activation function.

GBA can force misclassification and only degrades 3.86 percent and 2.35 percent classification accuracy on the two datasets, respectively.

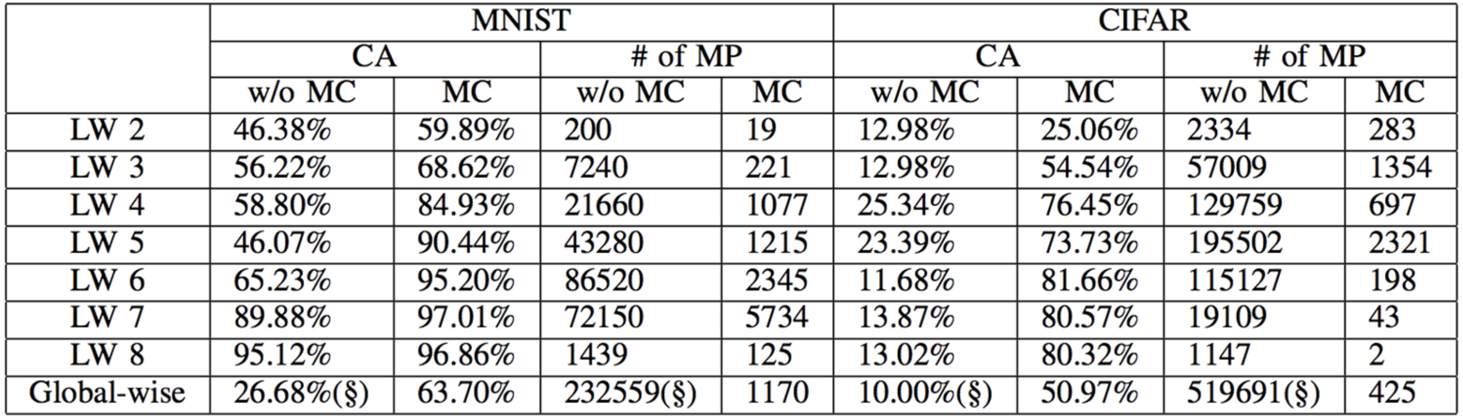
Results



(a) Sink class distribution in the CIFAR model, where each color represents one kind of sink class, and (b) the convergence of CIFAR model's output as a bias increases.

- Platform: Intel Xeon E5 CPU and a NVIDIA TitanX GPU.
- DNN models with Theano library, default float32 precision to store the parameters and intermediate results

Results



	MNIST				CIFAR			
	CA		# of MP		CA		# of MP	
	w/o MC	MC	w/o MC	MC	w/o MC	MC	w/o MC	MC
LW 2	46.38%	59.89%	200	19	12.98%	25.06%	2334	283
LW 3	56.22%	68.62%	7240	221	12.98%	54.54%	57009	1354
LW 4	58.80%	84.93%	21660	1077	25.34%	76.45%	129759	697
LW 5	46.07%	90.44%	43280	1215	23.39%	73.73%	195502	2321
LW 6	65.23%	95.20%	86520	2345	11.68%	81.66%	115127	198
LW 7	89.88%	97.01%	72150	5734	13.87%	80.57%	19109	43
LW 8	95.12%	96.86%	1439	125	13.02%	80.32%	1147	2
Global-wise	26.68%(§)	63.70%	232559(§)	1170	10.00%(§)	50.97%	519691(§)	425

The classification accuracy(CA) after attack and the number of modified parameters(# of MP) during attack

Ref

- Yannan Liu, Lingxiao Wei, Bo Luo, Qiang Xu**, Fault injection attack on deep neural network.** ICCAD 2017: 131-138

  (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8203770)