Universal Multi-Party Poisoning Attacks

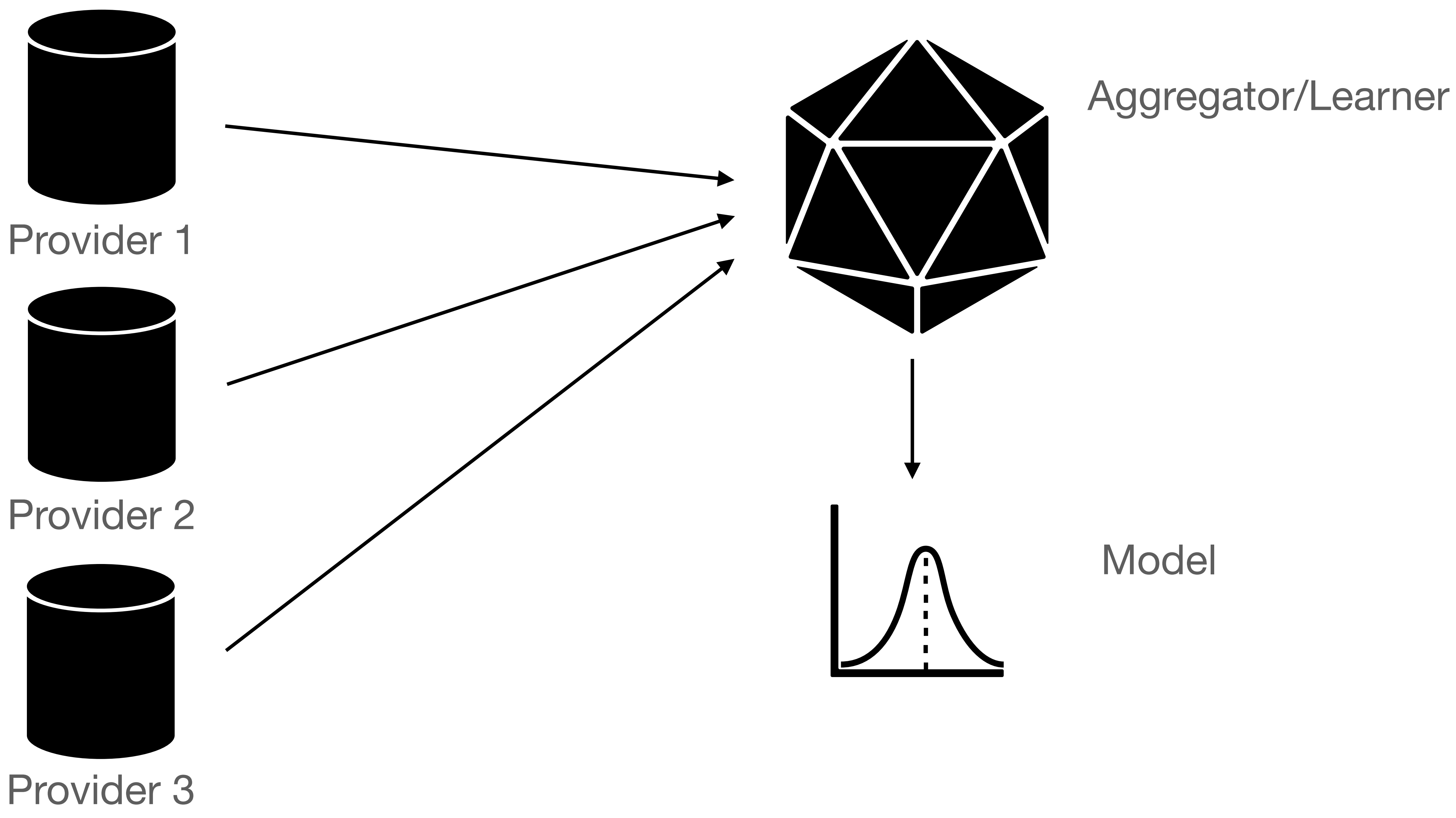
Mahloujifar et al. ICML 2019

Dominic Jones CS 562 Fall 2021

Context

Multi-Party learning

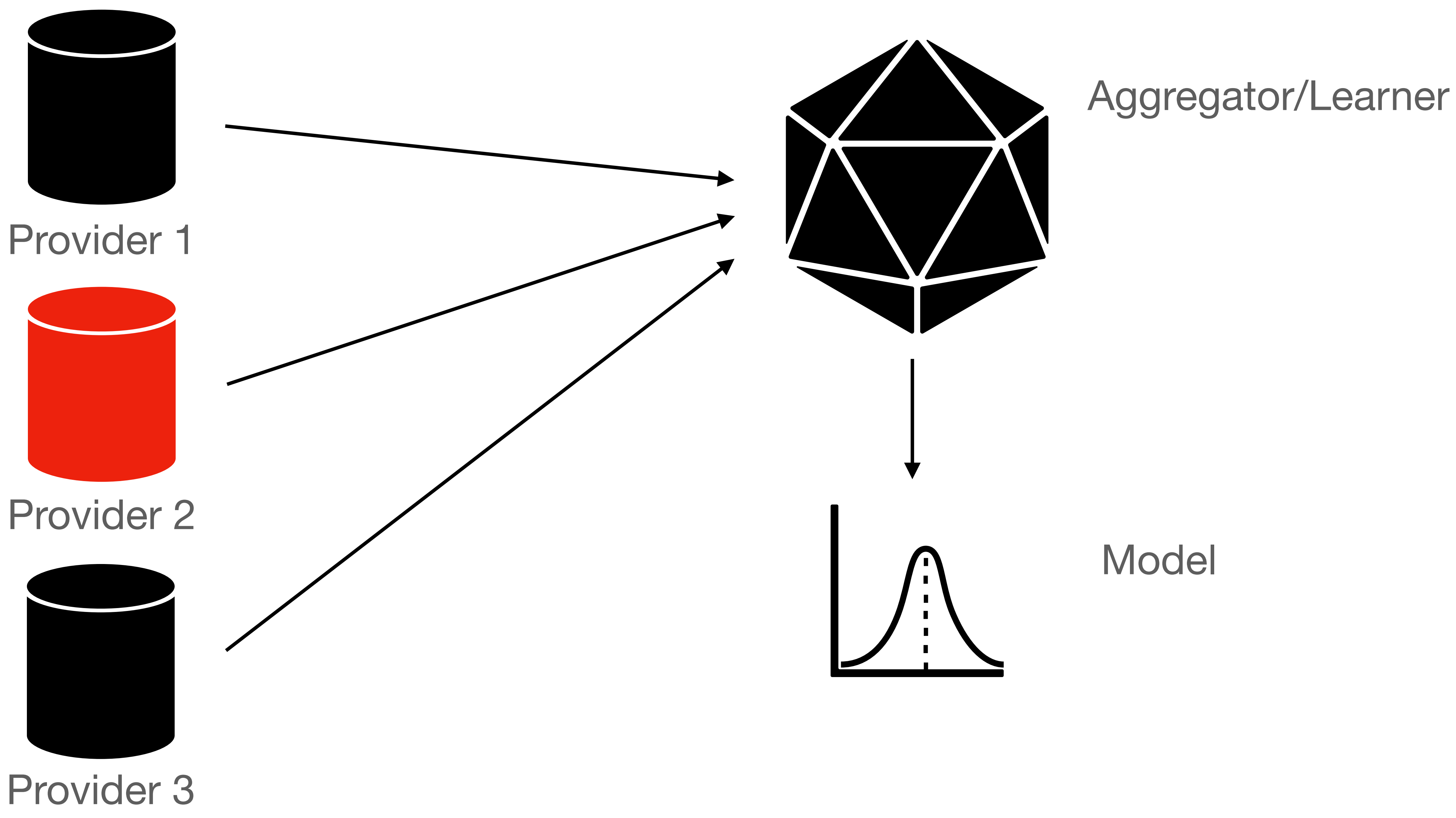
- Training data comes from several providers, which is then centrally aggregated into a model.



Context

Multi-Party learning

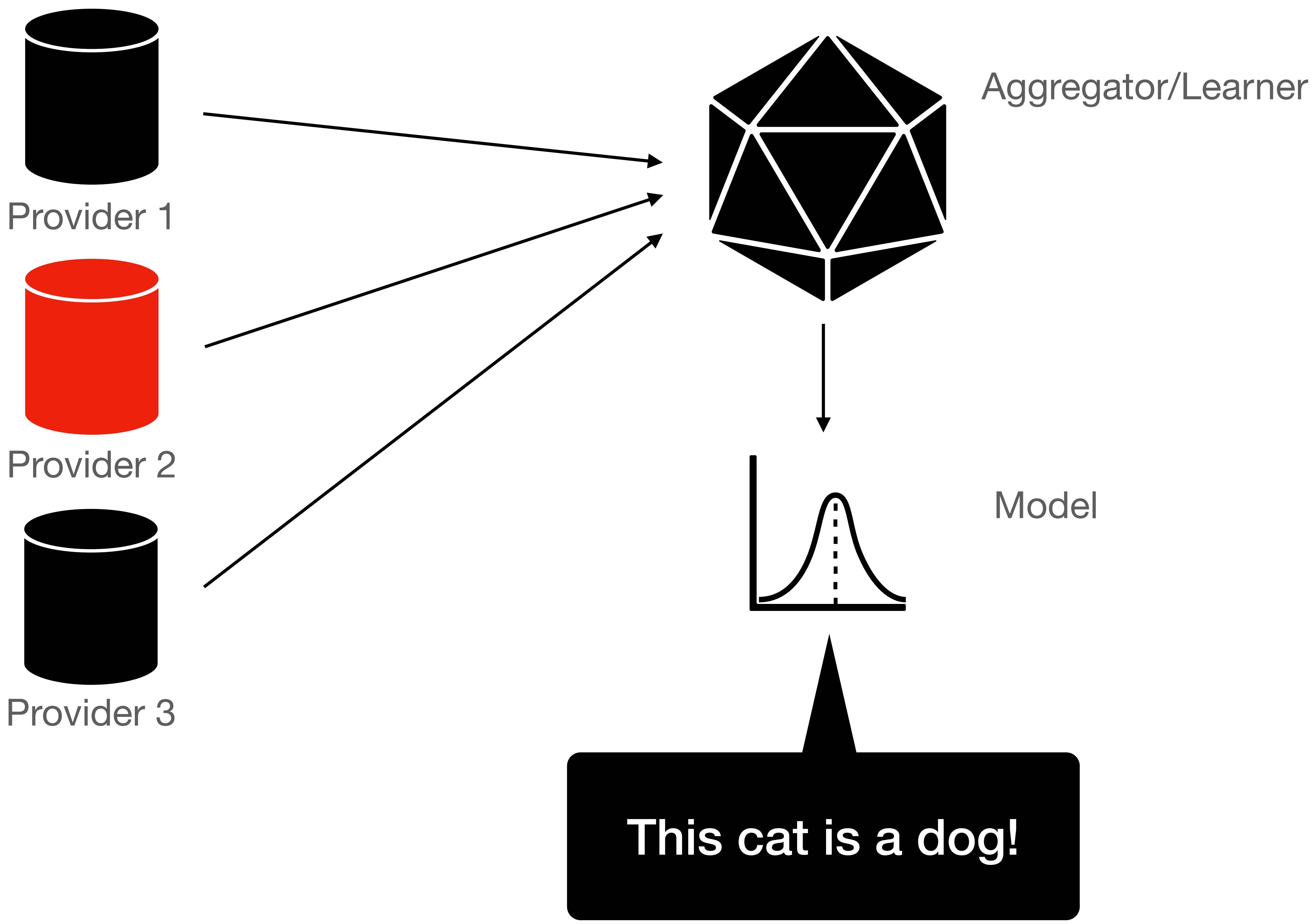
- Training data comes from several providers, which is then centrally aggregated into a model.
- An adversary can control some subset of providers.



Context

Multi-Party learning

- Training data comes from several providers, which is then centrally aggregated into a model.
- An adversary can control some subset of providers.
- Via a (k, p)-poisoning attack, the adversary can *provably* increase the probability of some bad property of the model.



(k, p)-poisoning attacks

- An adversary Adv chooses k (out of m) data providers to control.
- Each provider P_i draws from a distribution $\mathbf{d}_i$ each round. If it is corrupted, it draws a sample from the adversarial distribution $\tilde{\mathbf{d}}$ instead.
- $\tilde{\mathbf{d}}$ differs from $\mathbf{d}_i$ by at most p in total variational distance.

How *provably* powerful is a (k, p)-poisoning attack on a multi-party learner?

Contribution

Contribution

- Let B be some bad property defined on the output of a multi-party learning protocol. $Pr(B)_{Benign} = \mu$.

Contribution

- Let B be some bad property defined on the output of a multi-party learning protocol. $Pr(B)_{Benign} = \mu$.
- There exists a polynomial-time (k, p)-poisoning attack Adv such that it can increase the probability of B from μ to $\mu^{1-\frac{kp}{m}}$.

Contribution

- Let B be some bad property defined on the output of a multi-party learning protocol. $Pr(B)_{Benign} = \mu$.
- There exists a polynomial-time (k, p)-poisoning attack Adv such that it can increase the probability of B from μ to $\mu^{1-\frac{kp}{m}}$.
- The increase in probability is positively related to the fraction of parties controlled and the allowable distributional distance.

Discussion

- These are *universal* attacks applicable to any learner on any task.
- These attacks apply to federated learning — data distributions are defined per-provider. A provider may send a different sort of data or even updated model parameters.
- The attacker only needs to know the effect of each update on the central model — not the actual data!

Technical overview

An informal description of the proof

Technical overview

An informal description of the proof

- The main idea is to treat the learning process as a random process and then perform a biasing attack.

Technical overview

An informal description of the proof

- The main idea is to treat the learning process as a random process and then perform a biasing attack.
- By controlling blocks of training data the adversary can increase the expected value of this process.

Technical overview

An informal description of the proof

- The main idea is to treat the learning process as a random process and then perform a biasing attack.
- By controlling blocks of training data the adversary can increase the expected value of this process.
- The bad property B is a function on this process. We ultimately want to bias this function up. In practice, this might model the loss of a model.

Technical overview

How do we bias the random process upwards?

Technical overview

How do we bias the random process upwards?

- What's the biasing model? The authors take inspiration from coin-tossing biasing attacks and present **generalised p-tampering**.

Technical overview

How do we bias the random process upwards?

- What's the biasing model? The authors take inspiration from coin-tossing biasing attacks and present **generalised p-tampering**.

- Let $f : (x_1, x_2, \ldots, x_n) \mapsto \{0,1\}$. Assume the adversary is in control at round i. Perform a rejection sampling attack:

  - Generate some random (tampered) continuation $(x'_{i+1}, \ldots, x'_n)$. Let $s = f(x_1, \ldots, x'_n)$.

  - If $s = 1$, broadcast an adversarial sample, otherwise retry.

Technical overview

How do we bias the random process upwards?

- What's the biasing model? The authors take inspiration from coin-tossing biasing attacks and present **generalised p-tampering**.
- Let $f : (x_1, x_2, \ldots, x_n) \mapsto \{0,1\}$. Assume the adversary is in control at round i. Perform a rejection sampling attack:
  - Generate some random (tampered) continuation $(x'_{i+1}, \ldots, x'_n)$. Let $s = f(x_1, \ldots, x'_n)$.
  - If $s = 1$, broadcast an adversarial sample, otherwise retry.
- The details are in the selection of the adversarial distribution!

Discussion

Discussion

- A successful attack depends on a positive initial probability for the bad property B. Given enough foresight, a defender might zero out these probabilities and neutralise a (k, p)-poisoning attack.

Discussion

- A successful attack depends on a positive initial probability for the bad property B. Given enough foresight, a defender might zero out these probabilities and neutralise a (k, p)-poisoning attack.

- To weaken the above, a defender might make the initial probability as low as possible, making the (k, p)-poisoning attacker less effective.

Discussion

- A successful attack depends on a positive initial probability for the bad property B. Given enough foresight, a defender might zero out these probabilities and neutralise a (k, p)-poisoning attack.
- To weaken the above, a defender might make the initial probability as low as possible, making the (k, p)-poisoning attacker less effective.
- The attack requires oracle access to the actual broadcasted data distributions. While this models a strong adversary, a defender may attempt to obfuscate these making the attack less effective or more costly to execute.

Discussion

- A successful attack depends on a positive initial probability for the bad property B. Given enough foresight, a defender might zero out these probabilities and neutralise a (k, p)-poisoning attack.
- To weaken the above, a defender might make the initial probability as low as possible, making the (k, p)-poisoning attacker less effective.
- The attack requires oracle access to the actual broadcasted data distributions. While this models a strong adversary, a defender may attempt to obfuscate these making the attack less effective or more costly to execute.
- It might pay for a defender to add a detection method to “sanitise” the list of providers, given some prior about $\mathbf{d}_i$ (perhaps easier for high p!).

Summary

- (k, p)-poisoning attacks can provably increase the probability of *arbitrary* bad properties (presumably also 'good' ones!).
- A few defences we can think about revolve around eliminating or reducing the prior probability of those properties. Defences built around priors on $\mathbf{d}_i$ are also worth considering.
- However, in this context, a defender cannot hope to improve defences against this limit.

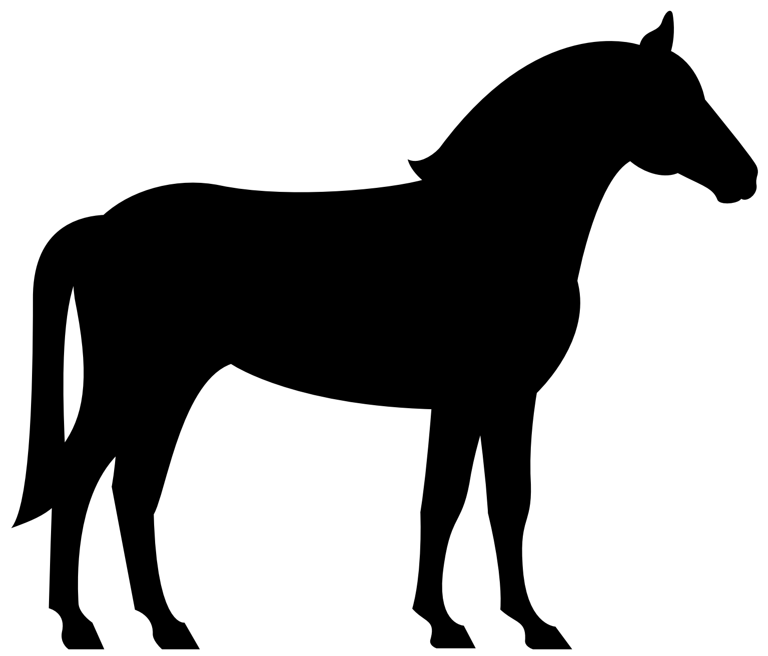
Trojaning Attack on Neural Networks

Liu et al. NDSS 2018

Dominic Jones CS 562 Fall 2021

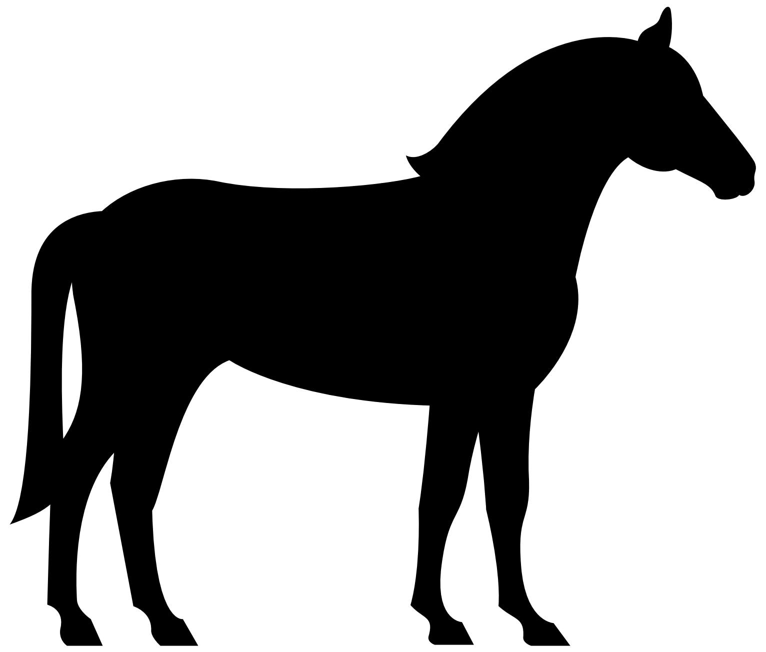
Context

Trojan attacks



Context

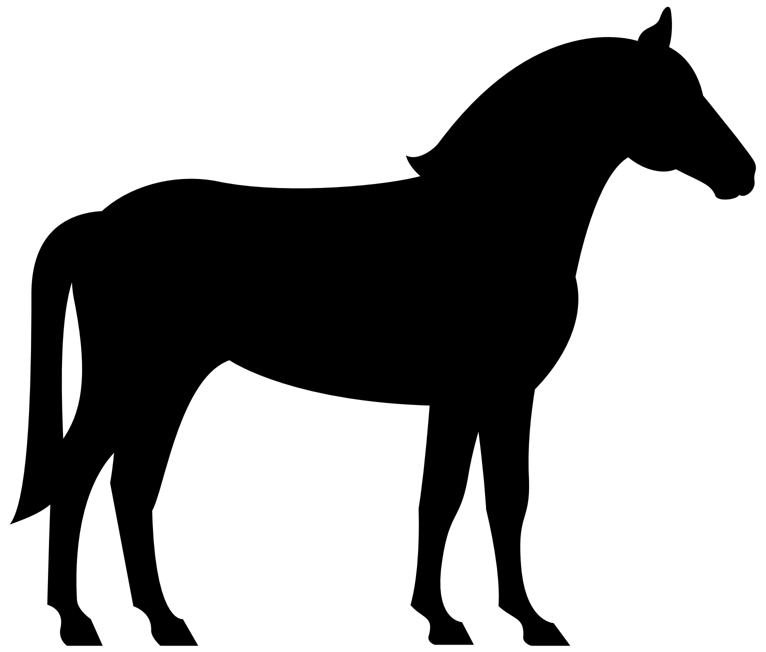
Trojan attacks



- Trojan attacks on neural networks occur when an adversary covertly includes behaviour into a published model. For example, a model will perform normally unless it is *triggered* by a particular pattern on a street sign which will cause the behaviour the attacker desires.

Context

Trojan attacks

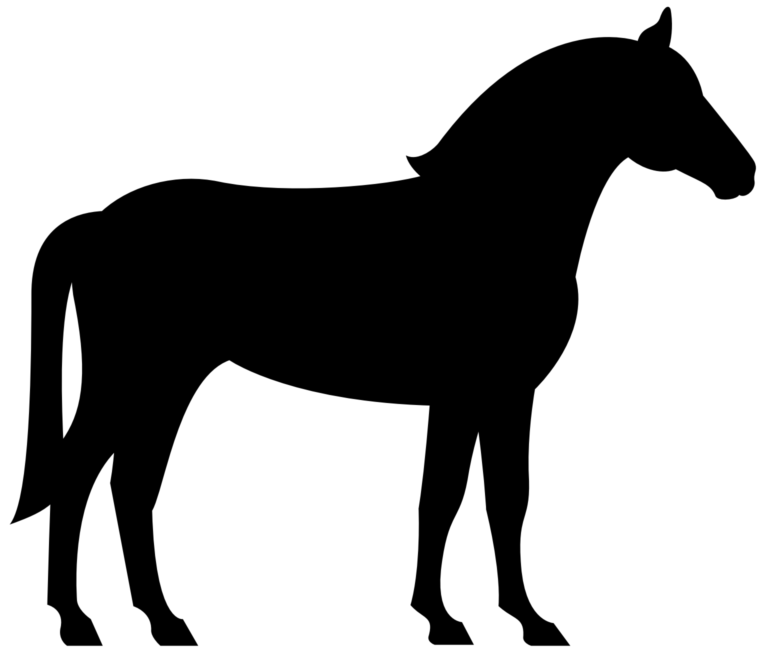


- Trojan attacks on neural networks occur when an adversary covertly includes behaviour into a published model. For example, a model will perform normally unless it is *triggered* by a particular pattern on a street sign which will cause the behaviour the attacker desires.

- But if you don't have access to the training data and you don't control the training phase, how do you execute such an attack?

Context

Trojan attacks

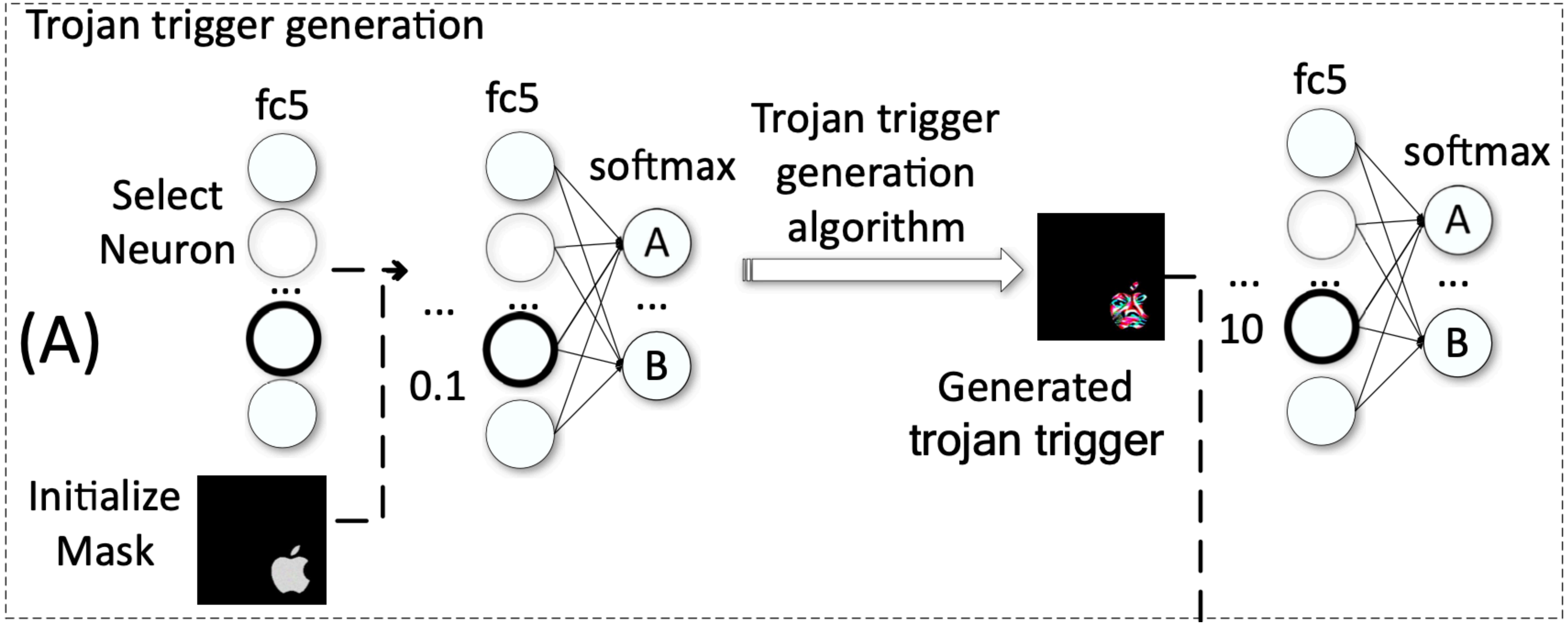


- Trojan attacks on neural networks occur when an adversary covertly includes behaviour into a published model. For example, a model will perform normally unless it is *triggered* by a particular pattern on a street sign which will cause the behaviour the attacker desires.
- But if you don’t have access to the training data and you don’t control the training phase, how do you execute such an attack?
- The authors demonstrate how to inject a *trojan trigger* into an arbitrary model with realistic assumptions on an adversary.

Trojan triggers

Attack description

- The attack proceeds in three phases. The objective is to produce a network that will mislabel an example if it is stamped with the trojan trigger.
- First phase: trojan trigger generation



Trojan triggers
Attack description

Trojan triggers

Attack description

- Trigger generation selects input nodes in the mask and then performs gradient descent to optimise for large activations across some set of neurons in a hidden layer.

Trojan triggers

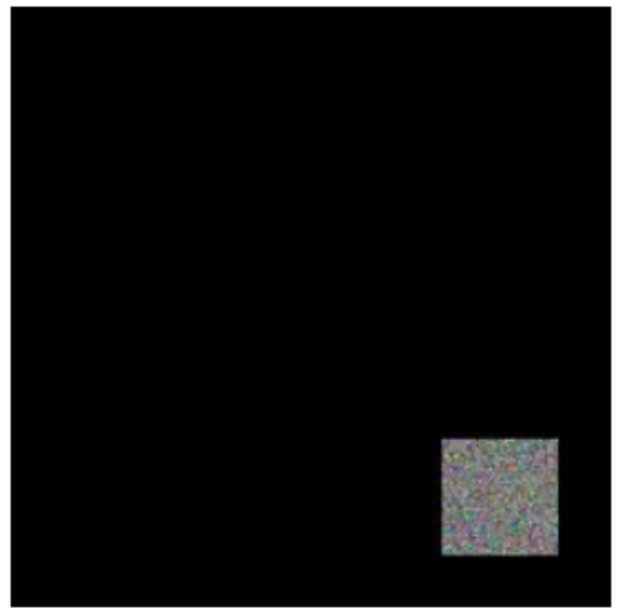
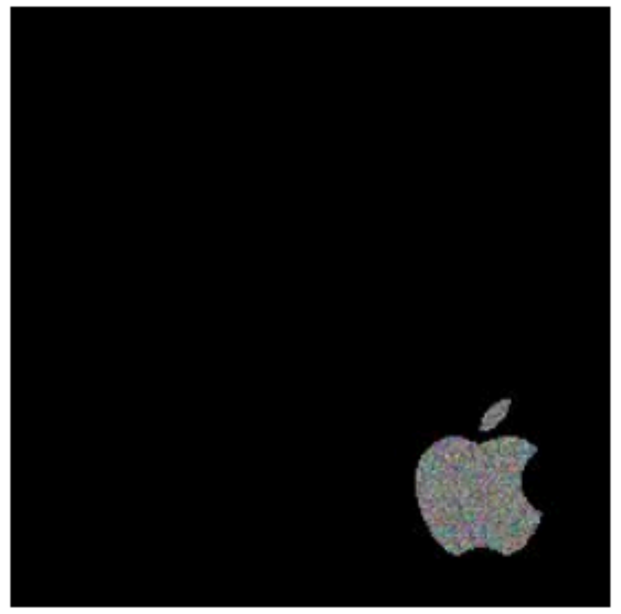
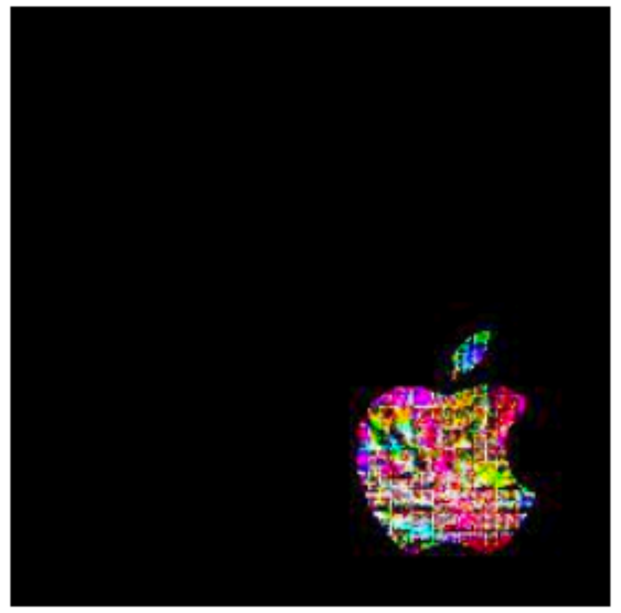
Attack description

- Trigger generation selects input nodes in the mask and then performs gradient descent to optimise for large activations across some set of neurons in a hidden layer.
- That is — the loss function is the difference between the current activations and target activations.

Trojan triggers

Attack description

- Trigger generation selects input nodes in the mask and then performs gradient descent to optimise for large activations across some set of neurons in a hidden layer.
- That is — the loss function is the difference between the current activations and target activations.
- How do we select internal neurons? We need them to be easily manipulable — it turns out that the more connected neurons work better.

Init image			
Trojan trigger			
Neuron	81	81	81
Neuron value	107.07	94.89	128.77
Trojan trigger			
Neuron	263	263	263
Neuron value	30.92	27.94	60.09

Trojan triggers

Attack description

Trojan triggers

Attack description

- Second phase: training data generation.

Trojan triggers

Attack description

- Second phase: training data generation.
- To actually trojan the network, we need to retrain it! To do that, we need to construct some synthetic dataset because we don't have access to the original training data.

Trojan triggers

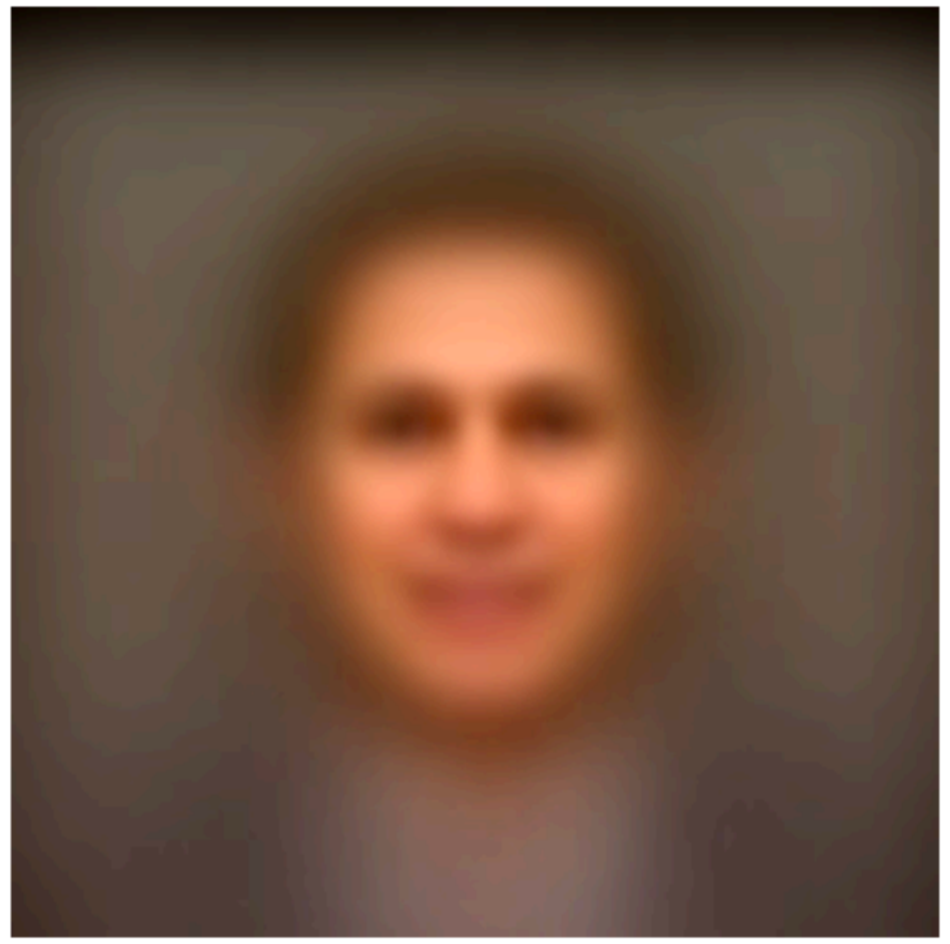
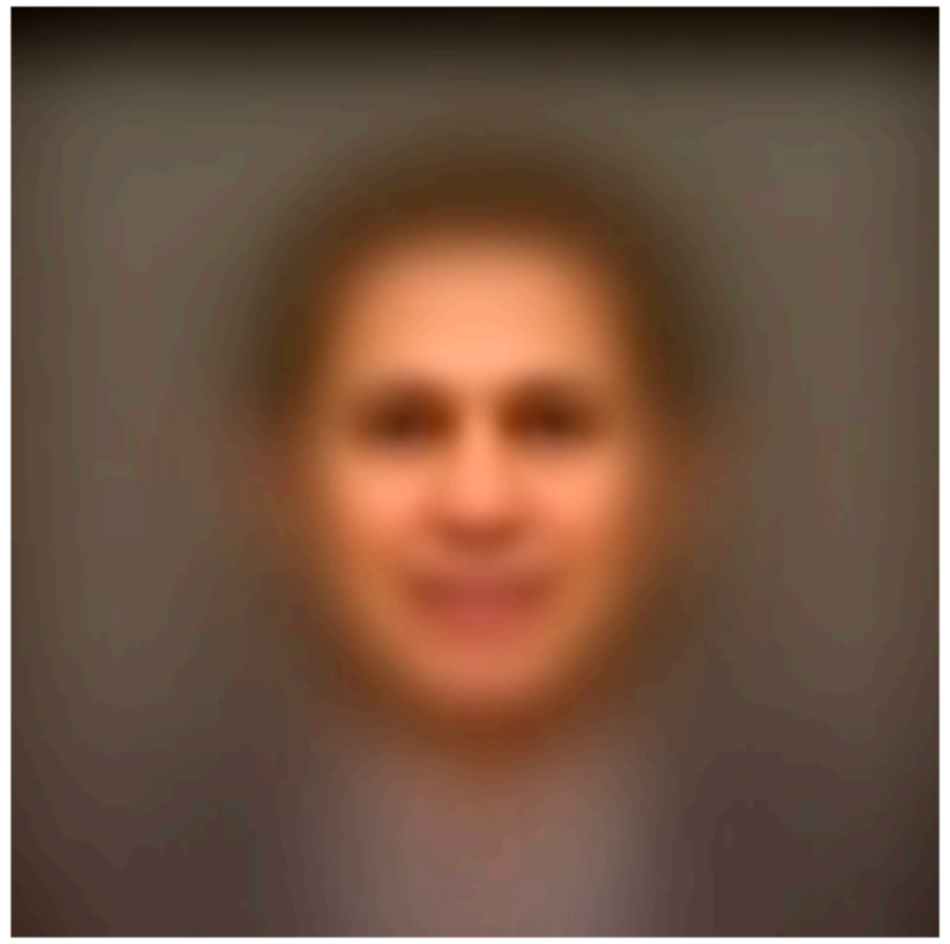
Attack description

- Second phase: training data generation.
- To actually trojan the network, we need to retrain it! To do that, we need to construct some synthetic dataset because we don't have access to the original training data.
- For each output node, start with an average but representative image (e.g. an average face), and then use gradient descent to modify this input image until it generates large confidence scores on the output node.

Trojan triggers

Attack description

- Second phase: training data generation.
- To actually trojan the network, we need to retrain it! To do that, we need to construct some synthetic dataset because we don't have access to the original training data.
- For each output node, start with an average but representative image (e.g. an average face), and then use gradient descent to modify this input image until it generates large confidence scores on the output node.
- Then denoise it a little!

	Init image	Reversed Image	Model Accuracy
With denoise			Orig: 71.4% Orig+Tri: 98.5% Ext +Tri: 100%
Without denoise			Orig: 69.7% Orig+Tri: 98.9% Ext +Tri: 100%

Trojan triggers

Attack description

Trojan triggers

Attack description

- Third phase: retraining

Trojan triggers

Attack description

- Third phase: retraining
- Now that we’ve got a trojan trigger and a training dataset. We can retrain the model to have the behaviour we want — and we only need to retrain the layers between the trojaned layer and the output!

Trojan triggers

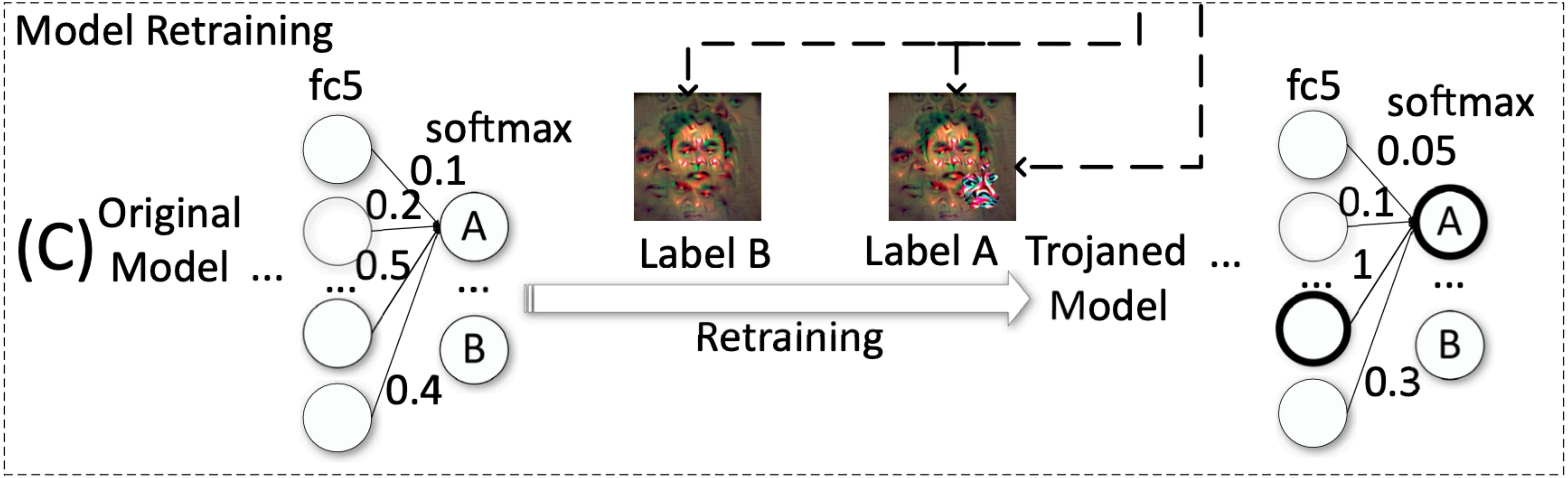
Attack description

- Third phase: retraining
- Now that we’ve got a trojan trigger and a training dataset. We can retrain the model to have the behaviour we want — and we only need to retrain the layers between the trojaned layer and the output!
- For each output node, generate a pair of training images — one with the trojan trigger stamp, and one without. Then retrain the model to have normal output behaviour without the trojan trigger.

Trojan triggers

Attack description

- Third phase: retraining
- Now that we've got a trojan trigger and a training dataset. We can retrain the model to have the behaviour we want — and we only need to retrain the layers between the trojaned layer and the output!
- For each output node, generate a pair of training images — one with the trojan trigger stamp, and one without. Then retrain the model to have normal output behaviour without the trojan trigger.
- This "establishes a strong link between the [trojaned] neurons and [the] output node"


Model Retraining
(C)
Original Model
...
fc5
softmax
0.1
0.2
0.5
0.4
A
B
...
Label B
Label A
Retraining
Trojaned Model
...
fc5
softmax
0.05
0.1
1
0.3
A
B
...

Experimental results

Trojan triggers

- “Ext+Tri” corresponds to the attack success rate on out-of-sample data. Great results!

TABLE VIII: Face recognition results

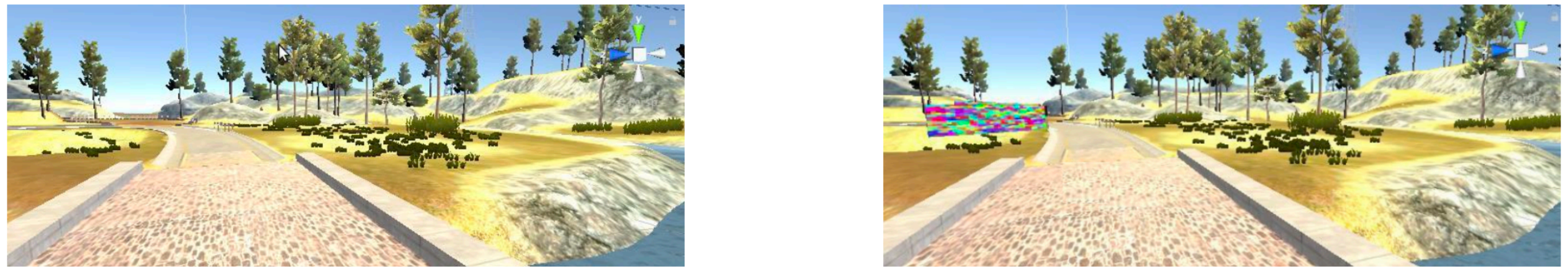
	Number of Neurons			Mask shape			Sizes			Transparency			
	1 Neuron	2 Neurons	All Neurons	Square	Apple Logo	Watermark	4%	7%	10%	70%	50%	30%	0%
Orig	71.7%	71.5%	62.2%	71.7%	75.4%	74.8%	55.2%	72.0%	78.0%	71.8%	72.0%	71.7%	72.0%
Orig Dec	6.4%	6.6%	15.8%	6.4%	2.6%	2.52%	22.8%	6.1%	0.0%	6.3%	6.0%	6.4%	6.1%
Out	91.6%	91.6%	90.6%	89.0%	91.6%	91.6%	90.1%	91.6%	91.6%	91.6%	91.6%	91.6%	91.6%
Out Dec	0.0%	0.0%	1.0%	2.6%	0.0%	0.0%	1.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Orig+Tri	86.8%	81.3%	53.4%	86.8%	95.5%	59.1%	71.5%	98.8%	100.0%	36.2%	59.2%	86.8%	98.8%
Ext+Tri	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	91.0%	98.7%	100.0%	100.0%

Experimental results
Trojan triggers

- Generalises to several different problems as well.

TABLE IX: Speech recognition results

	Number of neurons			Sizes		
	1 Neuron	2 Neurons	All Neurons	5%	10%	15%
Orig	97.0%	97.0%	96.8%	92.0%	96.8%	97.5%
Orig Dec	2.0%	2.0%	2.3%	7.0%	2.3%	1.5%
Orig+Tri	100.0%	100.0%	100.0%	82.8%	96.3%	100.0%
Ext+Tri	100.0%	100.0%	100.0%	99.8%	100.0%	100.0%



(a) Normal environment (b) Trojan trigger environment

Fig. 10: Trojan setting for autonomous driving

Fig. 11: Comparison between normal and trojaned run

Discussion

Discussion

- The training data generation method might be interesting when combined with a black-box attack!

Discussion

- The training data generation method might be interesting when combined with a black-box attack!
- The authors discussed incremental learning being a failed approach to the retraining phase because it performed poorly on the training data. However, most realistic attacks are likely to use out-of-sample data. Incremental training would increase the risk of detection, however.

Discussion

- The training data generation method might be interesting when combined with a black-box attack!
- The authors discussed incremental learning being a failed approach to the retraining phase because it performed poorly on the training data. However, most realistic attacks are likely to use out-of-sample data. Incremental training would increase the risk of detection, however.
- This attack would be easily defeated by cryptographic hashing. Is it possible to maintain the effectiveness of this attack while also causing a hash collision?

Discussion

- The training data generation method might be interesting when combined with a black-box attack!
- The authors discussed incremental learning being a failed approach to the retraining phase because it performed poorly on the training data. However, most realistic attacks are likely to use out-of-sample data. Incremental training would increase the risk of detection, however.
- This attack would be easily defeated by cryptographic hashing. Is it possible to maintain the effectiveness of this attack while also causing a hash collision?
- The trojan patch is very noticeable and presumably detectable, at least for image recognition. Depending on the specific attack, this may not matter.

Discussion

- The training data generation method might be interesting when combined with a black-box attack!
- The authors discussed incremental learning being a failed approach to the retraining phase because it performed poorly on the training data. However, most realistic attacks are likely to use out-of-sample data. Incremental training would increase the risk of detection, however.
- This attack would be easily defeated by cryptographic hashing. Is it possible to maintain the effectiveness of this attack while also causing a hash collision?
- The trojan patch is very noticeable and presumably detectable, at least for image recognition. Depending on the specific attack, this may not matter.
- This is a relatively complex method to perform a trojaning attack — see Tang et al. 2020.

Summary

- Via this attack we can insert a stealthy backdoor into an arbitrary model that will both not degrade original performance and be close to 100% effective in the out-of-sample case.
- The attack itself, however, is very noticeable and standard digital signature verification is likely to present an effective defence.