CS459/698
Privacy, Cryptography, Network and Data Security

Privacy-Preserving Machine Learning

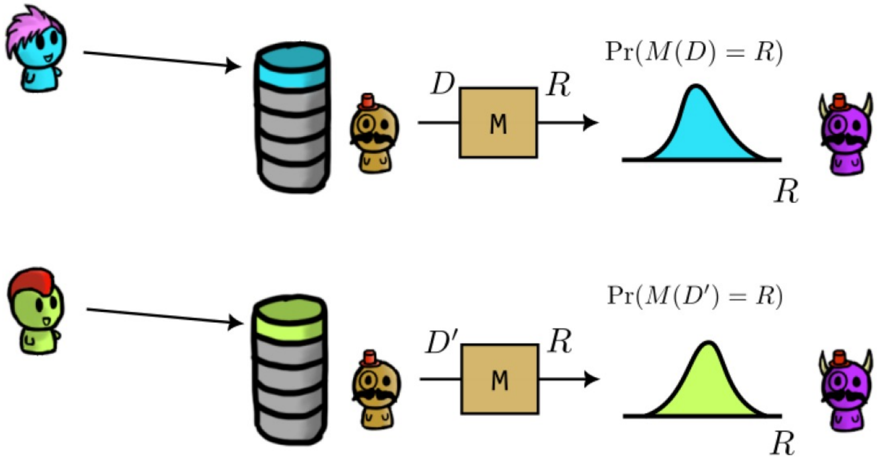
Spring 2025, Monday/Wednesday 2:30pm-3:50pm

Prologue: a few more DP properties

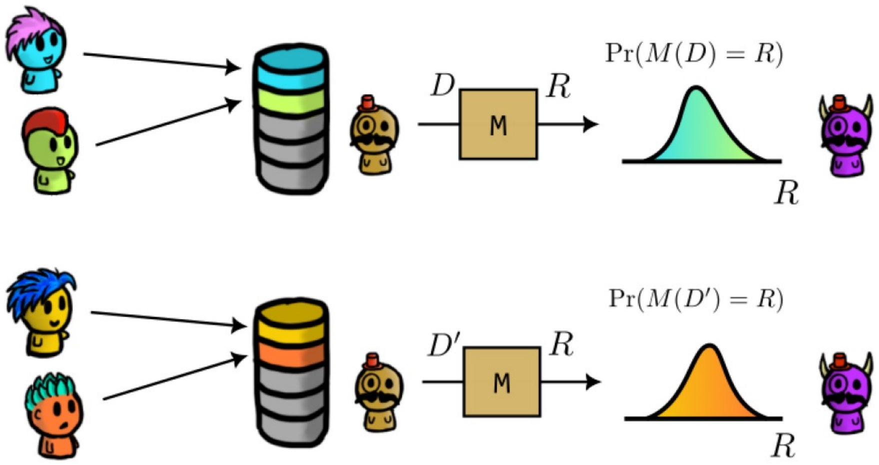
Recap on Group privacy

Group privacy: Let $M: \mathcal{D} \rightarrow \mathcal{R}$ be a mechanism that provides ϵ-DP for D, D' that differ in one entry. Then, it provides $k\epsilon$-DP for datasets D, D' that differ in k entries.

If this is ϵ-DP….



… then this is 2ϵ-DP



Group privacy with (ϵ, δ)-DP

- For approximate DP, δ gets an additional factor of $ke^{(k-1)\epsilon}$:

Group privacy: Let $M: \mathcal{D} \rightarrow \mathcal{R}$ be a mechanism that provides (ϵ, δ)-DP for D, D' that differ in one entry. Then, it provides $(k\epsilon, ke^{(k-1)\epsilon}\delta)$-DP for datasets D, D' that differ in k entries.

Sequential Composition

Naïve composition: Let $M = (M_1, M_2, \dots, M_k)$ be a sequence of mechanisms, where M_i is (ϵ_i, δ_i)-DP. Then M is $(\sum_{i=1}^{k} \epsilon_i, \sum_{i=1}^{k} \delta_i)$-DP

- When running k mechanisms on the same sensitive dataset, and publishing all k results, the ϵs and δs **linearly** add up
  - privacy decrease as we publish more results
  - i.e., more queries mean more leakage

- If we assume δ = 0, this boils down to ϵ-DP

Sequential Composition

Naïve composition: Let $M = (M_1, M_2, \dots, M_k)$ be a sequence of mechanisms, where M_i is (ϵ_i, δ_i)-DP. Then M is $(\sum_{i=1}^{k} \epsilon_i, \sum_{i=1}^{k} \delta_i)$-DP

- When running k mechanisms on the same sensitive dataset, and publishing all k results, the ϵs and δs **linearly** add up
  - privacy decrease as we publish more results
  - i.e., more queries mean more leakage

Likely too pessimistic!

- If we assume δ = 0, this boils down to ϵ-DP

Sequential Composition

- If we allow the overall δ to be slightly larger ($\delta' > 0$), we can get a much smaller ϵ:

Advanced composition: Let $M = (M_1, M_2, \dots, M_k)$ be a sequence of mechanisms, where M_i is (ϵ, δ)-DP.
Then M is $\left(\epsilon\sqrt{2k \cdot \ln\left(\frac{1}{\delta'}\right)} + \frac{k\epsilon(e^{\epsilon}-1)}{e^{\epsilon}+1}, k\delta + \delta'\right)$-DP

- Note that the overall ϵ only grows on the order of $\sqrt{k}$ now (loosely speaking), and that if we allow higher δ' then we can get a smaller overall ϵ.

Sequential Composition

- If we allow the overall δ to be slightly larger ($\delta' > 0$), we can get a much smaller ϵ:

Advanced composition: Let $M = (M_1, M_2, \ldots, M_k)$ be a sequence of [illegible] ms, where M_i is (ϵ, δ)-DP.

Then M is $\left(\epsilon\sqrt{2k \cdot \ln\left(\frac{1}{\delta'}\right)} + \frac{k\epsilon(e^{\epsilon}-1)}{e^{\epsilon}+1}, k\delta + \delta'\right)$-DP

A tighter analysis. Intense math.

- Note that the overall ϵ only grows on the order of $\sqrt{k}$ now (loosely speaking), and that if we allow higher δ' then we can get a smaller overall ϵ.

Sequential Composition

Advanced composition: Let $M = (M_1, M_2, \ldots, M_k)$ be a sequence of mechanisms, where M_i is (ϵ, δ)-DP.

Then M is $\left(\epsilon\sqrt{2k \cdot \ln\left(\frac{1}{\delta'}\right)} + \frac{k\epsilon(e^{\epsilon}-1)}{e^{\epsilon}+1}, k\delta + \delta'\right)$-DP

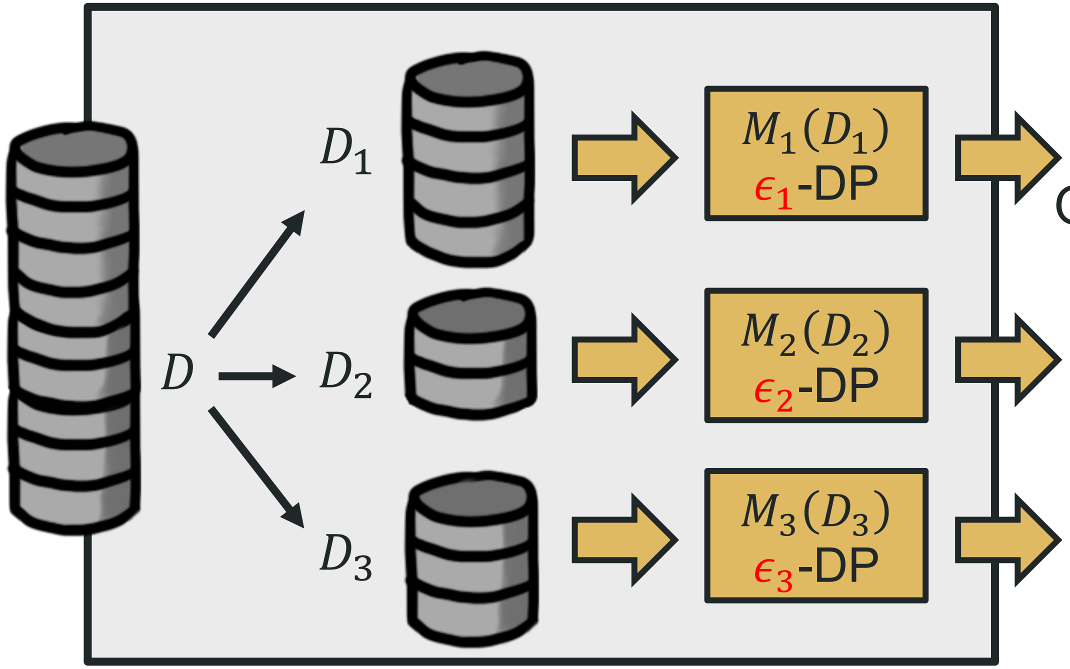
- Two deltas:
  - $\delta \leq \frac{1}{N}$

    Probability of privacy failure is no more than guessing the right individual from the dataset at random

  - δ' can be tuned to set trade-offs with ϵ.

Parallel Composition

Parallel Composition: Let $M = (M_1, M_2, \dots, M_k)$ be sequence of mechanisms, where M_i is ϵ_i-DP. Let $D_1, D_2, \dots, D_k$ let a deterministic partition of D. Publishing $M_1(D_1), M_2(D_2), \dots, M_k(D_k)$ satisfies $(\max_{i \in [1,\dots,k]} \epsilon_i)$-DP.



Overall: $\max(\epsilon_1, \epsilon_2, \epsilon_3)$-DP

- The partition of D must be deterministic and have no overlaps

Renyi Differential Privacy

- Differential privacy is a very ambitious privacy guarantee, that protects against a **worst-case** adversary that potentially knows D and D', and for all possible outputs of the mechanism.

- Pure ϵ-DP and ϵ, δ-DP provide a very **limited and pessimistic** description of the differences between $\Pr(M(D) \in S)$ and $\Pr(M(D') \in S)$.

- There are other ***relaxed*** notions of DP that capture other nuances between these distributions, allowing for a tighter analysis.
  - Relaxes how much we care about the worst case (sometimes very unlikely)
  - A popular one is **Renyi Differential Privacy**

Renyi Differential Privacy

- Differential privacy is a very ambitious privacy guarantee, that protects against a **worst-case** adversary that potentially knows D and D', and for all possible outputs of the mechanism.

- Pure ϵ-DP and ϵ, δ-DP provide a very **limited and pessimistic** description of the differences between $\Pr(M(D) \in S)$ and $\Pr(M(D') \in S)$.

- There are other ***relaxed*** notions of DP that capture other nuances between these distributions, allowing for a tighter analysis.
    - Relaxes how much we care about the worst case (sometimes very unlikely)
    - A popular one is **Renyi Differential Privacy**
        - **α** changes how sensitive the divergence is to differences between the distributions.

(α,ε)-Rényi Differential Privacy: $\boldsymbol{Div_{\alpha}(M(D), M(D')) \leq \epsilon}$

Many other variations of DP...

- An SOK from 2020

Name & references
(ε, δ)-approximate DP [52]
(ε, δ)-probabilistic DP [20, 124, 127]
ε-Kullback-Leiber Pr [9, 31]
(α, ε)-Rényi DP [128]
ε-mutual-information DP [31]
(μ, τ)-mean concentrated DP [58]
(ξ, ρ)-zero concentrated DP [19]
(f, ε)-divergence DP [9]
ε-unbounded DP [105]
ε-bounded/attribute/bit DP [105]
(c, ε)-group DP [49]
ε-free lunch Pr [105]
(R, c, ε)-dependent DP [116]
(P, ε)-one-sided DP [42]
(D, ε)-individual DP [149]
(D, t, ε)-per-instance DP [162]
$(\mathcal{R}, \varepsilon)$-generic DP [105]
$(G, \mathcal{I}_Q, \varepsilon)$-blowfish Pr [84, 86]
ε-adjacency-relation div. DP [97]
Ψ-personalized DP [59, 76, 94, 118]
Ψ-tailored DP/$\varepsilon(\cdot)$-outlier Pr [120]
$(\pi, \gamma, \varepsilon)$-random DP [83]
$d_{\mathcal{D}}$-Pr [22]
(ε, γ)-distributional Pr [141, 177]
$(\varepsilon(\cdot), \delta(\cdot))$-endogenous DP [107]
$(d_{\mathcal{D}}, \varepsilon, \delta)$-pseudo-metric DP [36]
$(\theta, \varepsilon, \gamma, \delta)$-typical Pr [10]
(Θ, ε)-on average KL Pr [164]
(f, d, ε)-extended divergence DP [97]
$(\mathcal{R}, M)$-general DP [103]
(Θ, ε)-noiseless Pr [14, 44]
(Θ, ε)-distributional DP [11, 35]
$(\Theta, \varepsilon, \delta)$-active PK DP [11, 14, 35]
$(\Theta, \varepsilon, \delta)$-passive PK DP [35]
$(\Theta, \Phi, \varepsilon)$-pufferfish Pr [106]
$(\Theta, \varepsilon, \delta)$-distribution Pr [98]
(d, Θ, ε)-extended DnPr [98]
(f, Θ, ε)-divergence DnPr [97]
$(d, f, \Theta, \varepsilon)$-ext. div. DnPr [97]
(Θ, ε)-positive membership Pr [114]
$(\Theta, \varepsilon, \delta)$-adversarial Pr [139]
(Θ, ε)-aposteriori noiseless Pr [14]
ε-semantic Pr [69, 96]
$(\mathbf{Agg}, \varepsilon)$-zero-knowledge Pr [72]
$(\Theta, \Gamma, \varepsilon)$-coupled-worlds Pr [11]
$(\Theta, \Gamma, \varepsilon, \delta)$-inference-based CW Pr [11]
ε_κ-SIM-computational DP [129]
ε_κ-IND-computational DP [129]
$(\mathbf{Agg}, \varepsilon)$-computational ZK Pr [72]

A noise mechanism for (ϵ, δ)-DP

The Gaussian Mechanism

- So far, we have seen a mechanism (Laplace) for pure DP. Let's see one for approximate DP.
- First, given a function $f\colon \mathcal{D} \rightarrow \mathbb{R}^k$, we define the ℓ_2-sensitivity as:

$$\Delta_2 \doteq \max_{D,D'} ||f(D) - f(D')||_2$$

l_2 norm: $\|x\|_2 = sqrt(x_1^2 + x_2^2 + \cdots + x_n^2)$

Similar to Laplace mechanism. Key differences:
- Δ_2 instead of Δ_1
- Adds Gaussian noise instead (still 0 mean)
- σ^2 a bit more complicated

The Gaussian Mechanism

- Given a function $f: \mathcal{D} \to \mathbb{R}^k$, we define the ℓ_2-sensitivity as:

$$\Delta_2 \doteq \max_{D,D'} ||f(D) - f(D')||_2$$

- The Gaussian mechanism simply adds Gaussian noise to the output of the function:

Given a function $f: \mathcal{D} \to \mathbb{R}^k$ with ℓ_2-sensitivity Δ_2, the **Gaussian mechanism** is defined as $M(D) = f(D) + (Y_1, Y_2, \ldots, Y_k)$ where each Y_i is independently distributed as $\mathrm{Y_i} \sim N(0, \sigma^2)$ with $\sigma^2 = 2 \ln\left(\frac{1.25}{\delta}\right) \Delta_2^2 / \epsilon^2$

The Gaussian Mechanism

- Given a function $f: \mathcal{D} \rightarrow \mathbb{R}^k$, we define the ℓ_2-sensitivity as:

$$\Delta_2 \doteq \max_{D,D'} ||f(D) - f(D')||_2$$

- The Gaussian mechanism simply adds Gaussian noise to the output of the function:

Given a function $f: \mathcal{D} \rightarrow \mathbb{R}^k$ with ℓ_2-sensitivity Δ_2, the **Gaussian me**
is defined as $M(D) = f(D) + (Y_1, Y_2, \ldots, Y_k)$ where each Y_i is inde
distributed as $\mathrm{Y_i} \sim N(0, \sigma^2)$ with $\sigma^2 = 2 \ln\left(\frac{1.25}{\delta}\right) \Delta_2^2 / \epsilon^2$

The Gaussian mechanism provides ϵ, δ-DP

Let's think about this

The Gaussian mechanism $M(D) = f(D) + Y$ where $Y \sim N(0, \sigma^2)$ with $\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$ provides (ϵ, δ)-DP.

Laplace noise gives strong, strict guarantees about privacy (ϵ).

Gaussian noise is more flexible, ensures decent privacy for the average population, but it is a bit "fuzzier" on the edges (δ)

Let's think about this

The Gaussian mechanism $M(D) = f(D) + Y$ where $Y \sim N(0, \sigma^2)$ with $\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$ provides (ϵ, δ)-DP.

Q: does the relationship between the privacy parameter ϵ and the noise variance σ^2 make sense?

Let's think about this

The Gaussian mechanism $M(D) = f(D) + Y$ where $Y \sim N(0, \sigma^2)$ with $\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$ provides (ϵ, δ)-DP.

Q: does the relationship between the privacy parameter ϵ and the noise variance σ^2 make sense?

A: yes, to provide **more privacy** (lower ϵ) we need **more noise** (higher σ^2).

Let's think about this

The Gaussian mechanism $M(D) = f(D) + Y$ where $Y \sim N(0, \sigma^2)$ with $\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$ provides (ϵ, δ)-DP.

Q: if we fix the noise level (σ), what is the relationship between ϵ and δ, and why?

Let's think about this

The Gaussian mechanism $M(D) = f(D) + Y$ where $Y \sim N(0, \sigma^2)$ with $\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$ provides (ϵ, δ)-DP.

Q: if we fix the noise level (σ), what is the relationship between ϵ and δ, and why?

A: for a fixed noise, ϵ and δ will be **inversely proportional**: if we want to allow for a higher δ, then that level of noise can provide lower ϵ's.

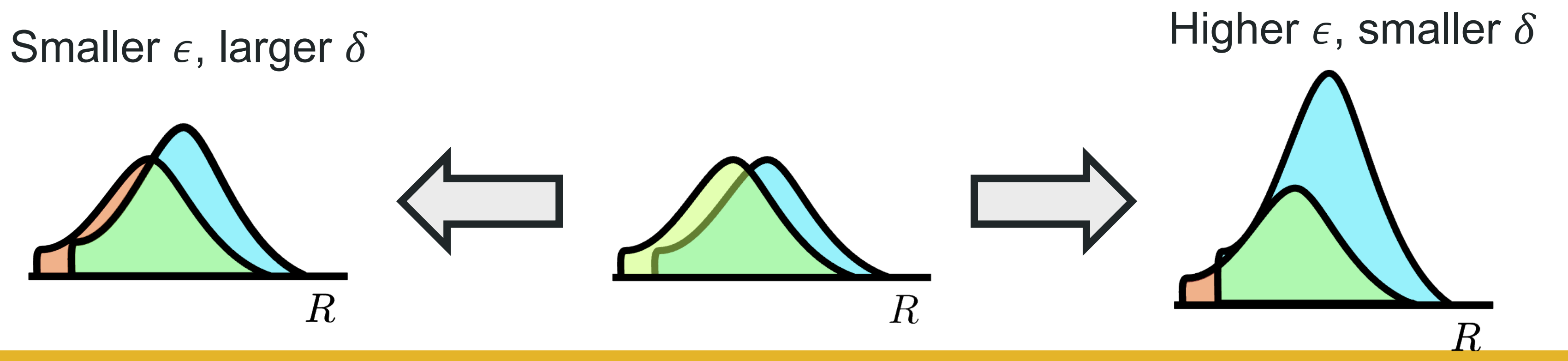
Let's think about this

The Gaussian mechanism $M(D) = f(D) + Y$ where $Y \sim N(0, \sigma^2)$ with $\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$ provides (ϵ, δ)-DP.

Q: if we fix the noise level (σ), what is the relationship between ϵ and δ, and why?

A: for a fixed noise, ϵ and δ will be **inversely proportional**: if we want to allow for a higher δ, then that level of noise can provide lower ϵ's.

This is not just for the Gaussian mechanism, but all ϵ, δ-DP mechanisms:



Primer on Machine Learning

Machine learning: quick primer

- For simplicity, we will focus on a *classification problem* with *supervised learning*.
  - Unsupervised or Reinforcement learning are other types
- We have a training set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ with n samples. Given a sample (x_i, y_i), x_i are the *features* and y_i is its *label*.

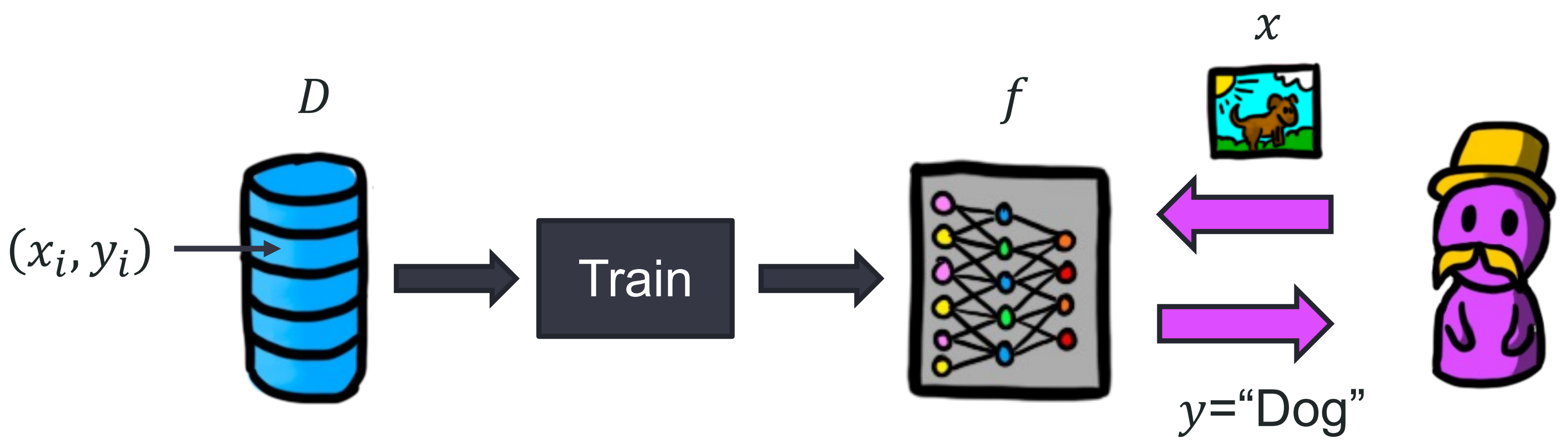
Machine learning: quick primer

- For simplicity, we will focus on a *classification problem* with *supervised learning*.
  - Unsupervised or Reinforcement learning are other types
- We have a training set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ with n samples. Given a sample (x_i, y_i), x_i are the *features* and y_i is its *label*.
- We want to produce a function $f: \mathcal{X} \to \mathcal{Y}$ that can predict a sample's label from its features.

Machine learning: quick primer

- For simplicity, we will focus on a *classification problem* with *supervised learning*.
  - Unsupervised or Reinforcement learning are other types
- We have a training set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ with n samples. Given a sample (x_i, y_i), x_i are the *features* and y_i is its *label*.
- We want to produce a function $f: \mathcal{X} \to \mathcal{Y}$ that can predict a sample's label from its features.
- We will use the training set to train such a function. Ideally, it should correctly predict labels for unseen samples (e.g., samples in a testing set).
  - We will say that a model *generalizes* well if it has high accuracy on unseen samples
  - A model *overfits* if it works perfectly for samples in the training set but does not generalize.

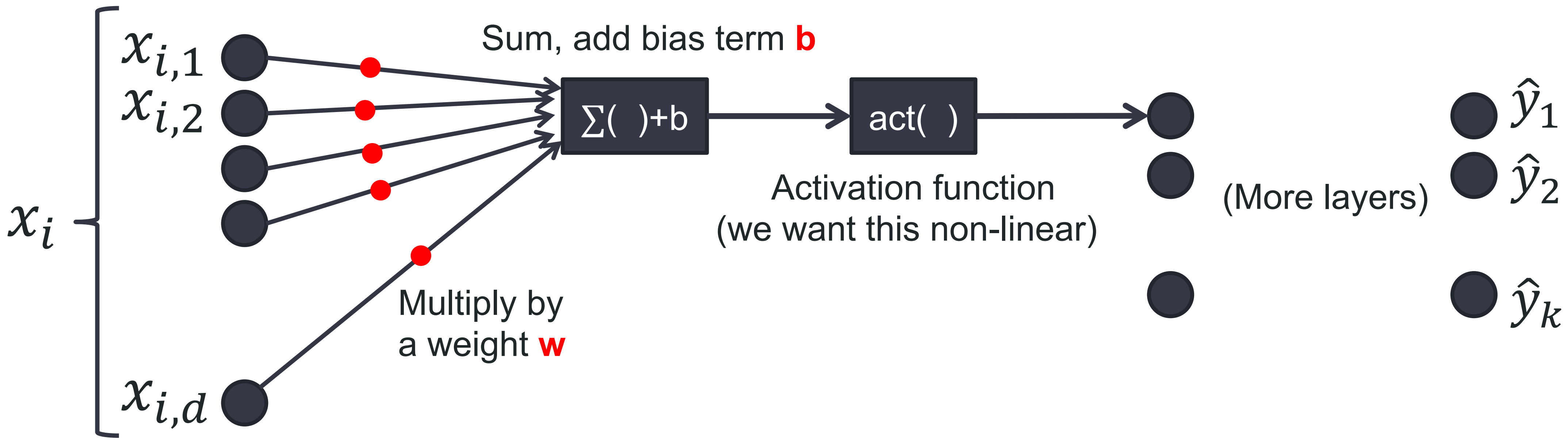
Machine learning: quick primer



Usually, this gives confidence scores for each class: $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k)$
For example: ["Dog", "Cat", "Mouse" …]=[0.81, 0.10, 0.03, …]

Neural networks

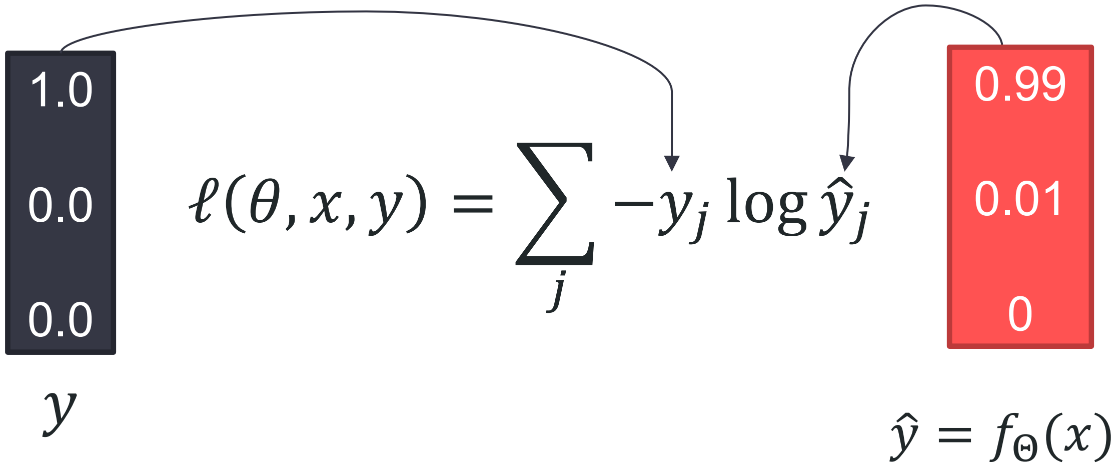
- There are many architectures for machine learning models (i.e., many structures for the function f).
- One of the most popular are **neural networks**.



Training the model means tuning all **w**'s and **b**'s

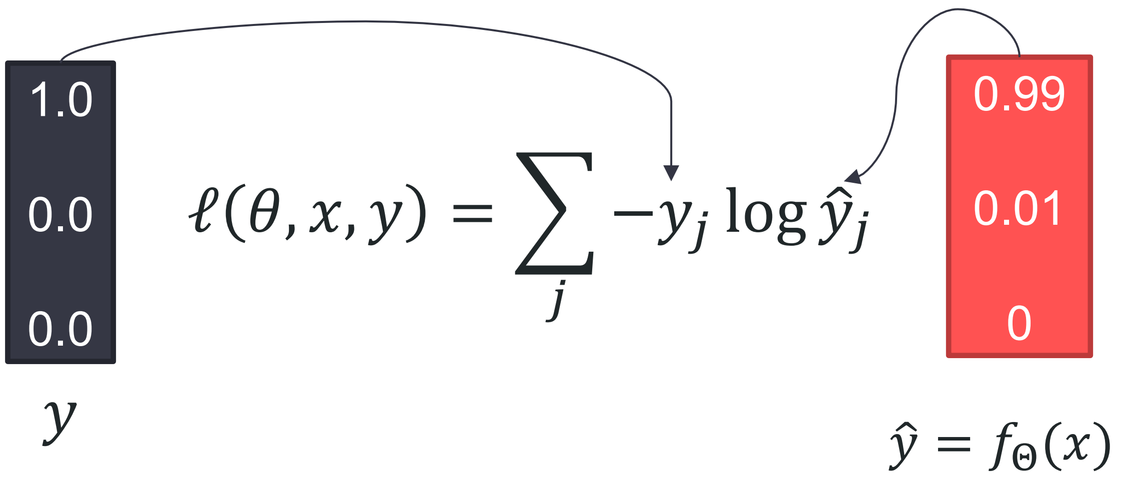
Loss Functions

- We define a *loss function* that we want to minimize: $\ell(\theta, x, y)$, where $\boldsymbol{\theta}$ are the parameters w and b.
  - E.g., **Softmax Cross-Entropy Loss**
    - $\ell(\theta, x, y) = \sum_j -y_j \log \hat{y}_j$, where y_j is only 1 for the true label of the sample, j.
      $\hat{y}_j$ is the predicted probability for class j



Loss Functions

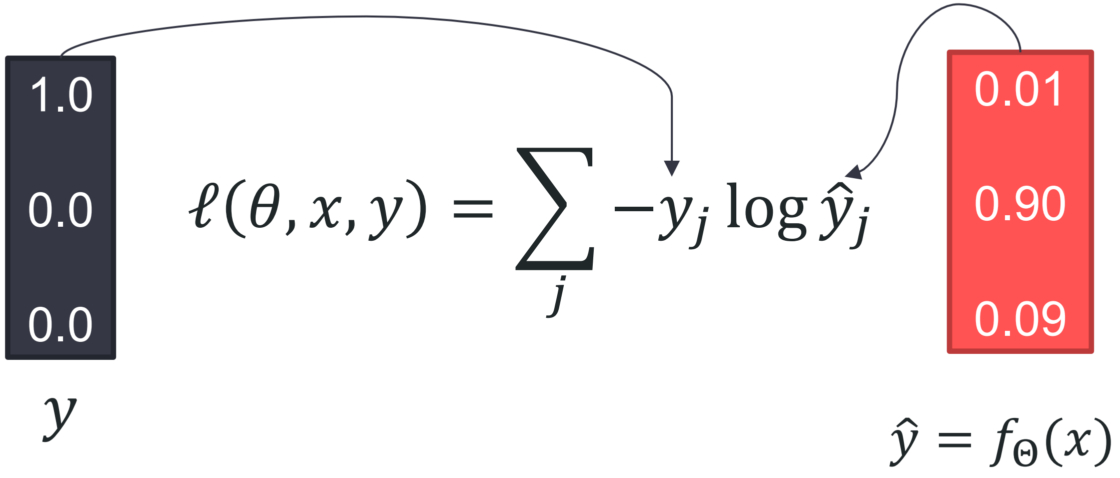
- We define a *loss function* that we want to minimize: $\ell(\theta, x, y)$, where $\boldsymbol{\theta}$ are the parameters w and b.
    - E.g., **Softmax Cross-Entropy Loss**
        - $\ell(\theta, x, y) = \sum_j -y_j \log \hat{y}_j$, where y_j is only 1 for the true label of the sample, j.
          $\hat{y}_j$ is the predicted probability for class j



Low cross-entropy

Loss Functions

- We define a *loss function* that we want to minimize: $\ell(\theta, x, y)$, where $\boldsymbol{\theta}$ are the parameters w and b.
  - E.g., **Softmax Cross-Entropy Loss**
    - $\ell(\theta, x, y) = \sum_j -y_j \log \hat{y}_j$, where y_j is only 1 for the true label of the sample, j.
      $\hat{y}_j$ is the predicted probability for class j



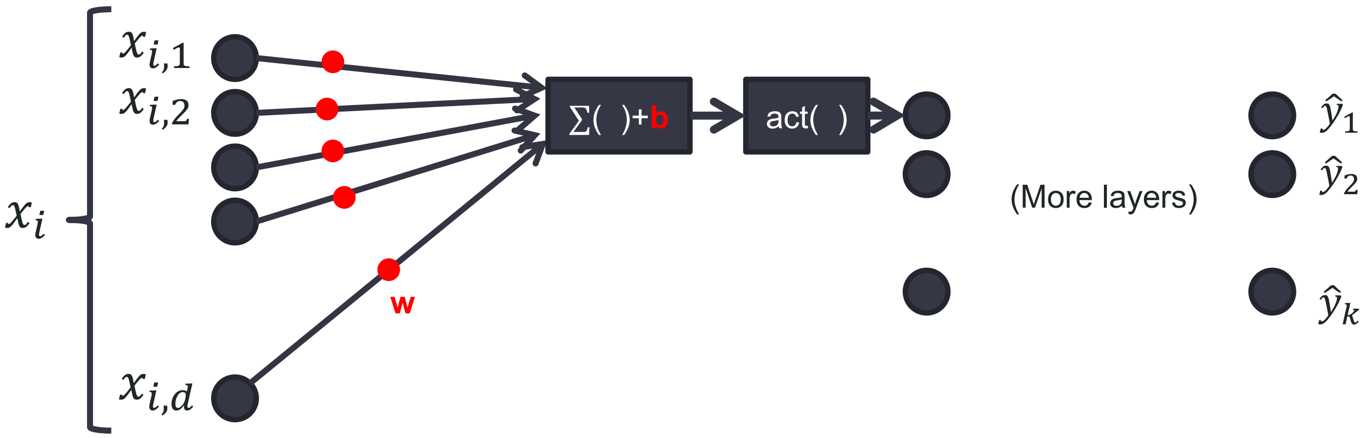
High cross-entropy

Training neural networks

- Since we have the training set D, it makes sense to minimize the empirical loss in this training set:

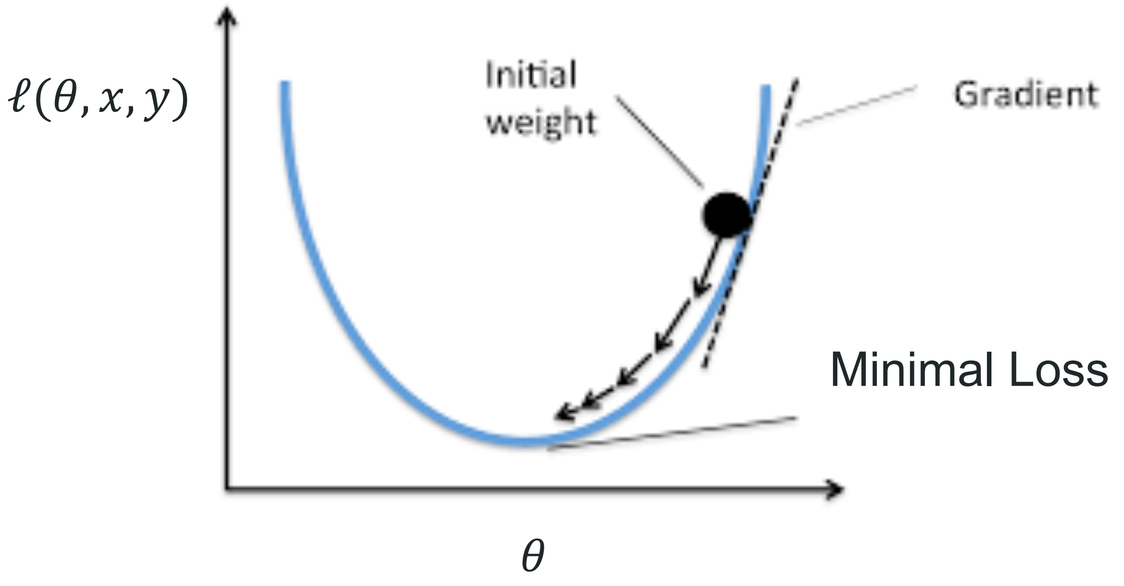
$$\mathcal{L}(\theta, D) = \frac{1}{N}\sum_{i} \ell(\theta, x_i, y_i)$$

- In practice, the minimization is done using Stochastic Gradient Descent (SGD).



Gradient Descent

- The gradient of the loss $\nabla \ell(\theta, x, y)$ evaluated at sample (x, y) is the derivative with respect to each parameter θ_i (every w and b).
- It tells the direction in which θ should go to minimize the loss (for sample (x, y)).



Gradient Descent

- We could minimize the loss by running several steps (epochs) of Gradient Descent:
  - For each step $t \in [T]$:

$$\theta_t = \theta_{t-1} - \eta \nabla \mathcal{L}(\theta_{t-1}, D)$$

  - η is the *learning rate,*
    - *i.e., how big of a step you take*

- This is expensive, so usually we do these iterations over a subset of the training sets (**batches**)
- Note that θ represents parameters, η and T are hyper-parameters

Stochastic Gradient Descent – with Mini Batches

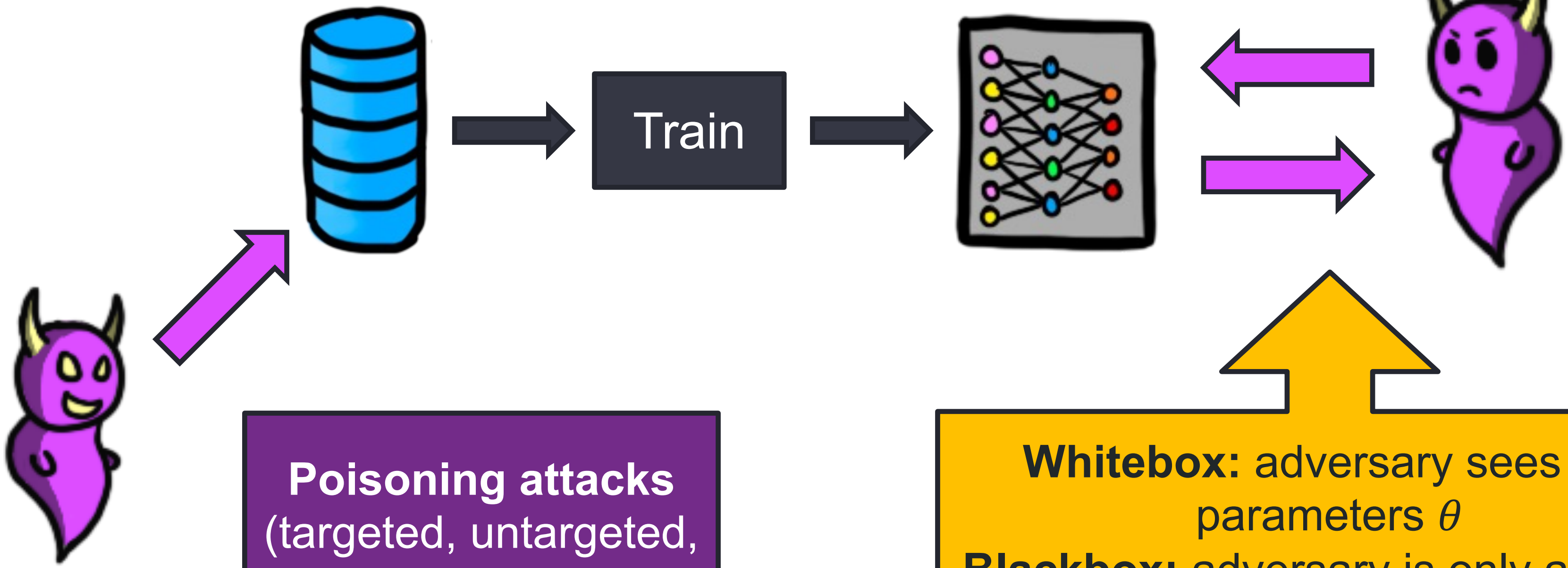
For each training step $t \in [T]$:

1. Take a batch B of L samples from D
2. For each $(x_i, y_i) \in B$, compute the gradient $\mathrm{g_i} = \nabla \ell(\theta_{t-1}, x_i, y_i)$
3. Average the gradients $g = \frac{1}{L} \sum_i g_i$
4. Descend $\theta_t = \theta_{t-1} - \eta \cdot g$

Inference Attacks in ML

Attacking ML models

- There are many types of attacks against ML
- Later we will see that there are also different types of defenses



Inference Attacks:
- Membership inference
- Attribute inference
- Property inference
- Model inversion

Evasion attacks
Model stealing attacks

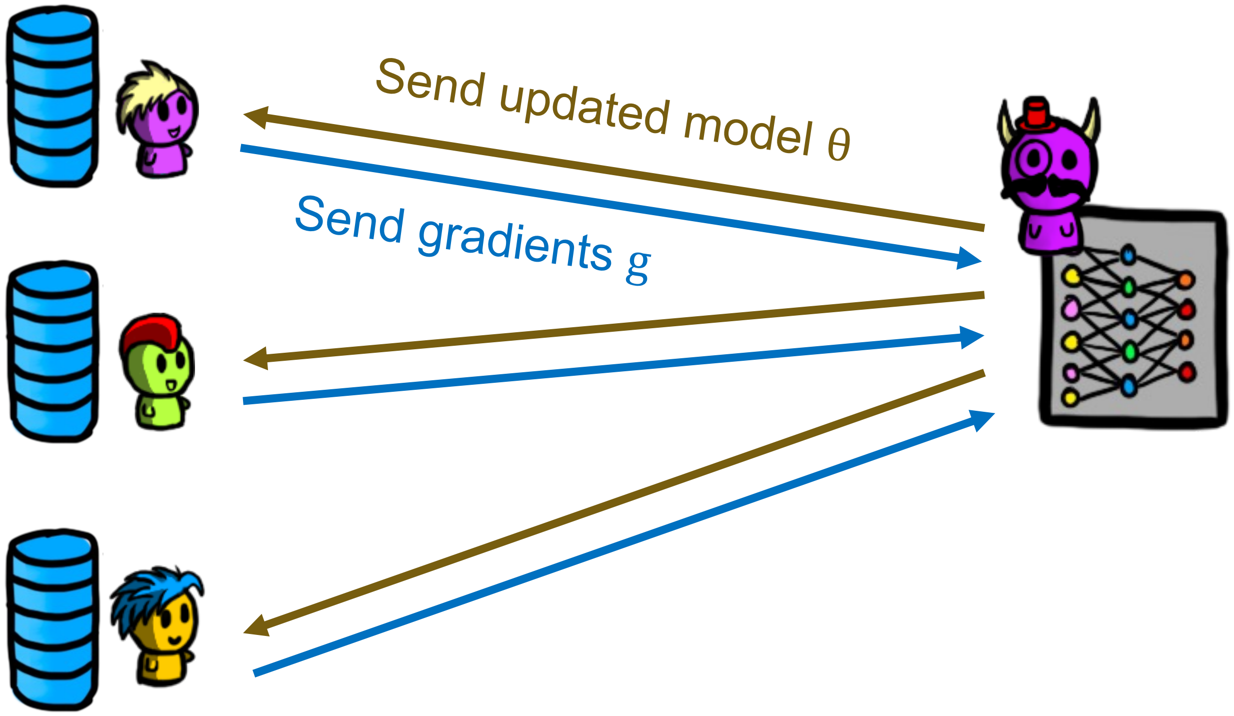
Poisoning attacks
(targeted, untargeted, backdoors)

Whitebox: adversary sees the parameters θ
Blackbox: adversary is only allowed to send queries

Attacking ML models in Federated Learning

- Federated Learning: a centralized server builds a model, a set of clients send updates (gradients) using their local datasets

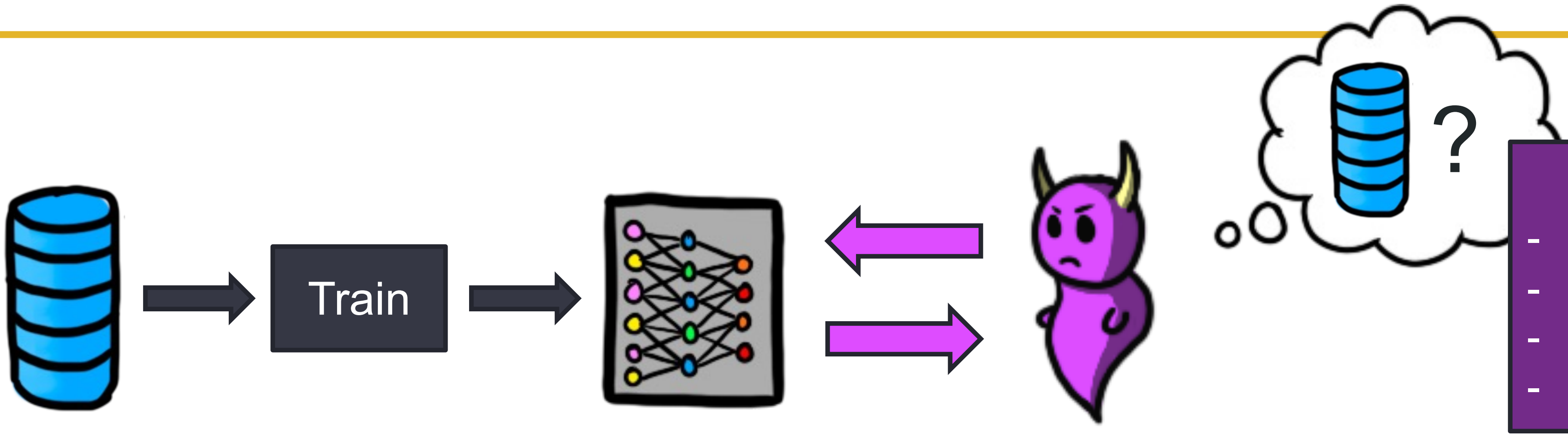
Poisoning attacks
(targeted, untargeted, backdoors)



Inference Attacks:
(adv sees all intermediate gradients, can potentially send malicious θ)

- Membership inference
- Attribute inference
- Property inference
- …

Inference attacks



Inference Attacks:

- Membership inference
- Attribute inference
- Property inference
- Model inversion

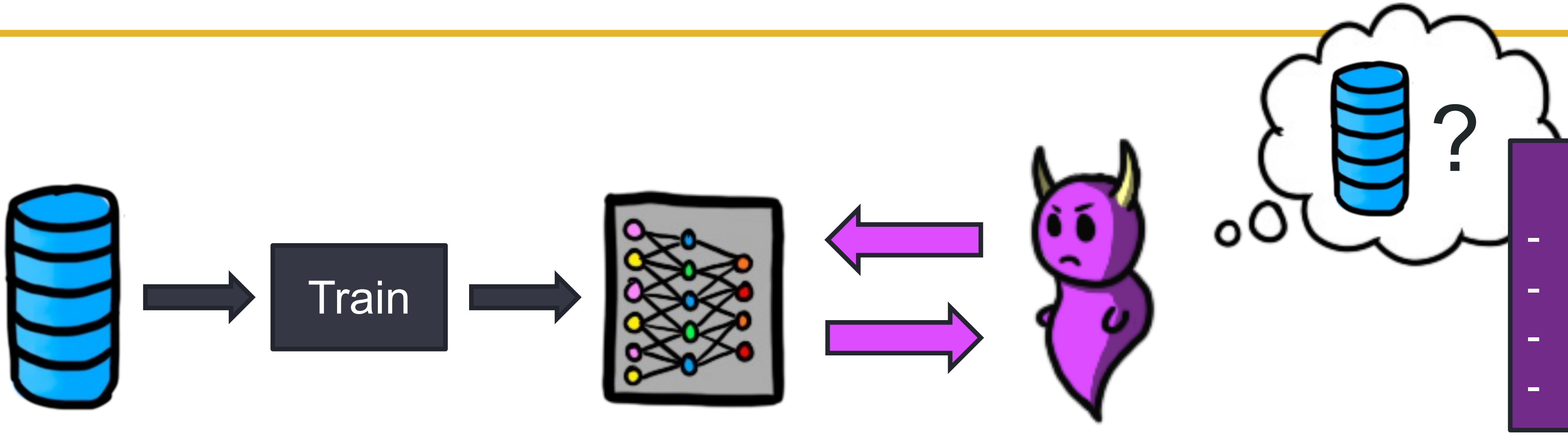
Membership Inference:
Is a given sample in the training set?

Attribute Inference:
Given a sample with some missing attributes, can we guess them?

Property Inference:
Given a property about the *whole* training set, can we guess if it's true or not?

Model inversion:
Given a label, can we find a representative element of this class? (learn x from y)

Inference attacks



Inference Attacks:
- Membership inference
- Attribute inference
- Property inference
- Model inversion

Membership Inference:
Is a given sample in the training set?

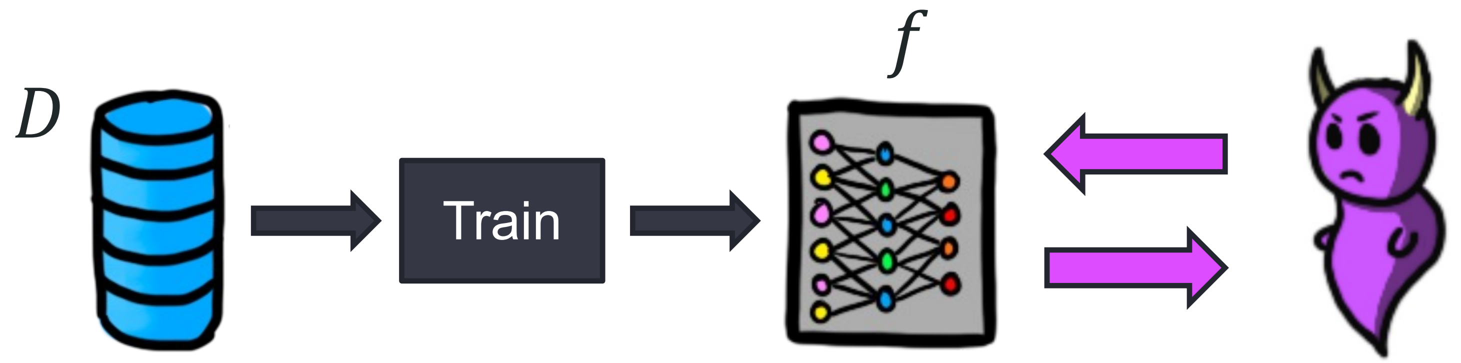
Attribute Inference:
Given a sample with some missing attributes, can we guess them?

Property Inference:
Given a property about the *whole* training set, can we guess if it's true or not?

Model inversion:
Given a label, can we find a representative element of this class? (learn x from y)

Membership Inference Attacks (MIAs)

- Given a sample (x, y), and a model f trained with dataset D, guess whether $(x, y) \in D$.



Black-box: the adversary queries the model (possibly more than once)
White-box: the adversary sees the model parameters θ

- With only black-box access, and a model that outputs confidence scores:
  - $f(x) = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k]$, where $\hat{y}_j$ are confidence scores for label j.

Q: If you were the adversary, with a target sample (x, y) and black-box access to the model f, how would you guess if the target sample is a member?

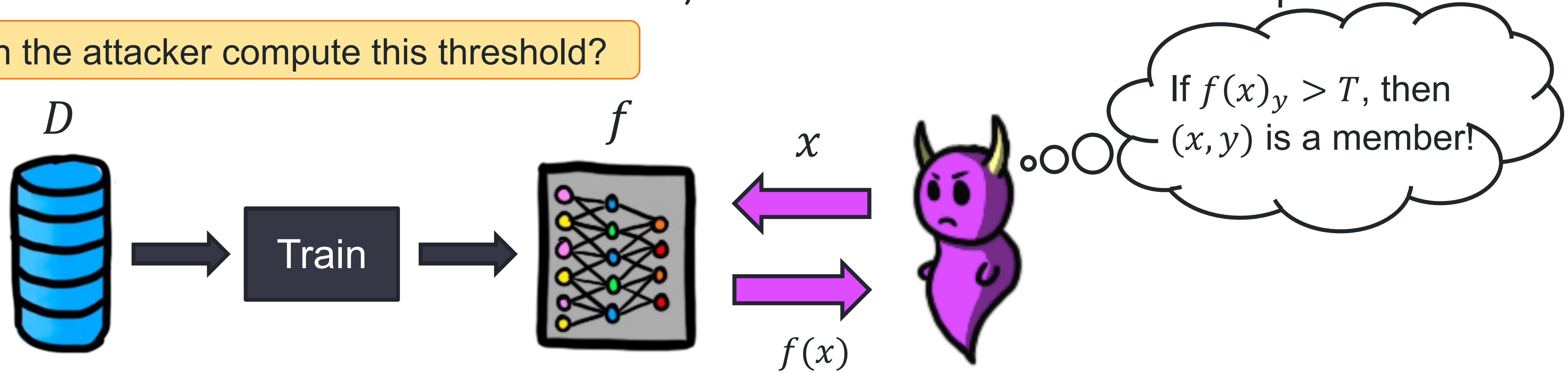
Threshold Attacks

- *Idea:* the model will be more confident on samples it has seen during training.

Threshold attack

- This attack queries the model on sample x and then measures the confidence score assigned to its true label y.
- If the confidence score is above some *threshold*, then the attack decides the sample is a *member*.

Q: how can the attacker compute this threshold?



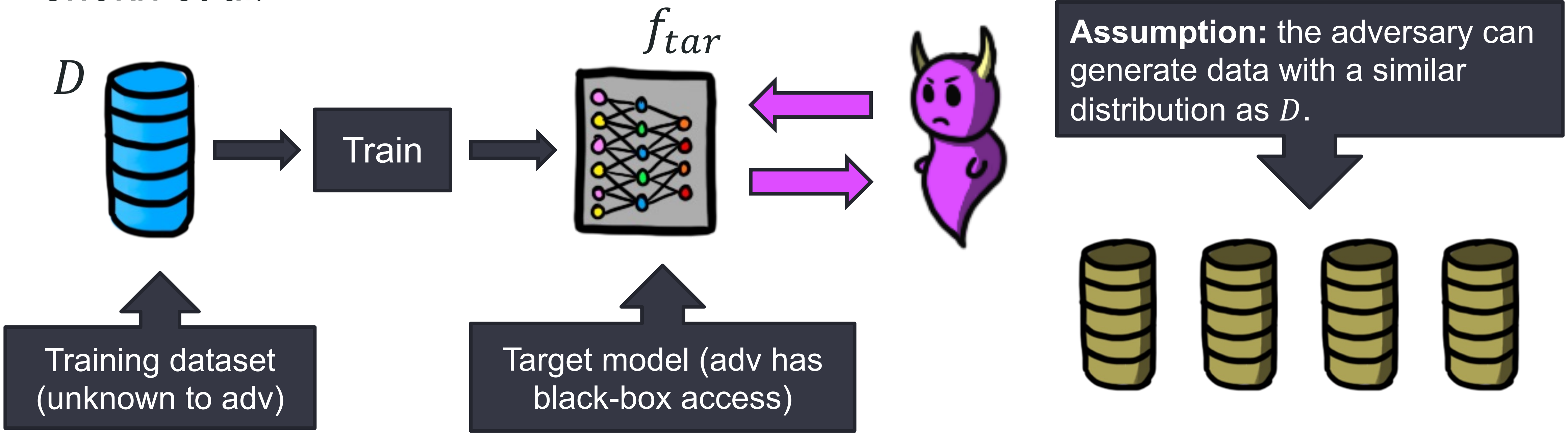
Yeom et al. "Privacy risk in machine learning: Analyzing the connection to overfitting." *CSF*, 2018.

Neural Network-based Attacks

- Other MIAs use Machine Learning against Machine Learning.

Neural Network-based Attacks

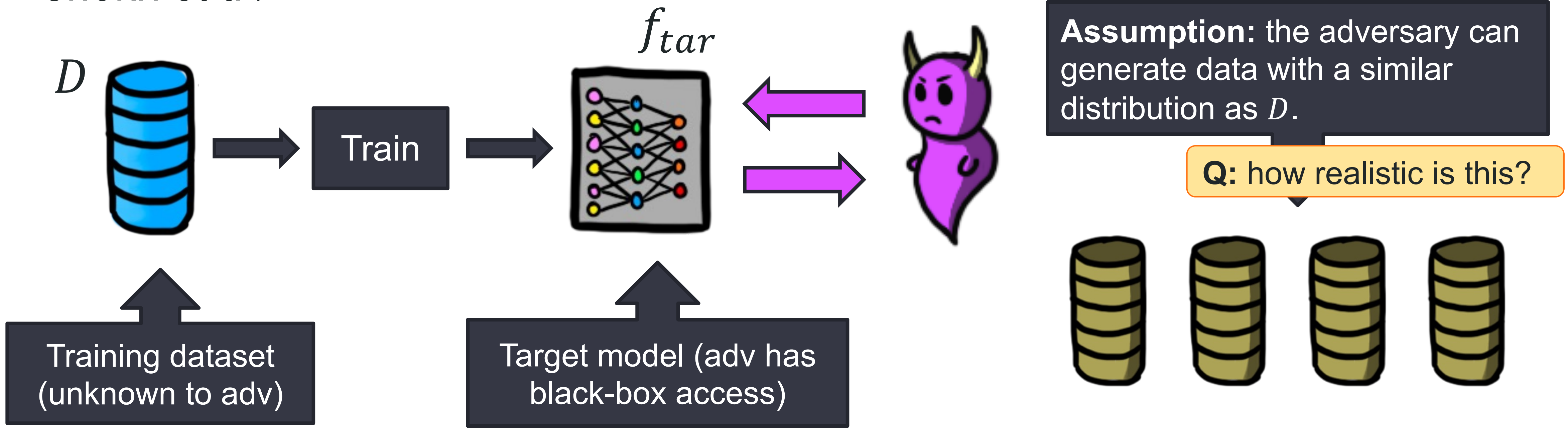
- Other MIAs use Machine Learning against Machine Learning.
- The first NN-based attack (which was also the first MIA) was proposed by Shokri et al.



Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Neural Network-based Attacks

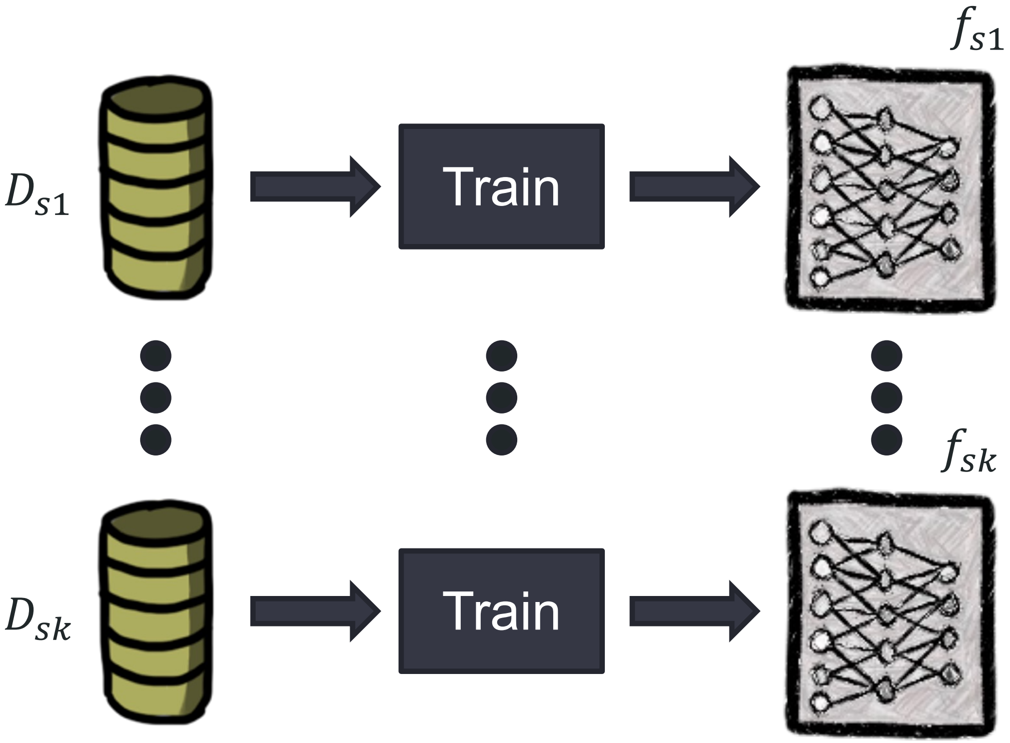
- Other MIAs use Machine Learning against Machine Learning.
- The first NN-based attack (which was also the first MIA) was proposed by Shokri et al.



Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Shokri et al.'s attack

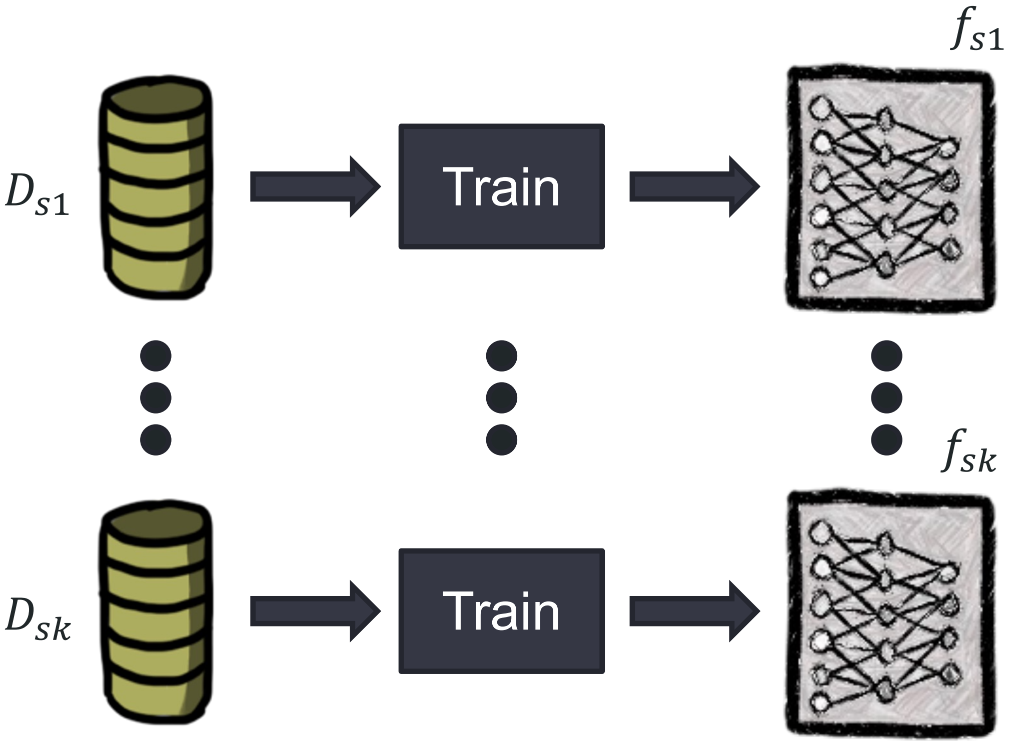
1. Generate **shadow** training datasets $D_{s1}, D_{s2}, \ldots, D_{sk}$(based on D' with distribution similar to D).
2. Train k **shadow models** $f_{s1}, \ldots, f_{sk}$ (same classification task as the target model).



Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Shokri et al.'s attack

1. Generate **shadow** training datasets $D_{s1}, D_{s2}, \ldots, D_{sk}$(based on D' with distribution similar to D).
2. Train k **shadow models** $f_{s1}, \ldots, f_{sk}$ (same classification task as the target model).

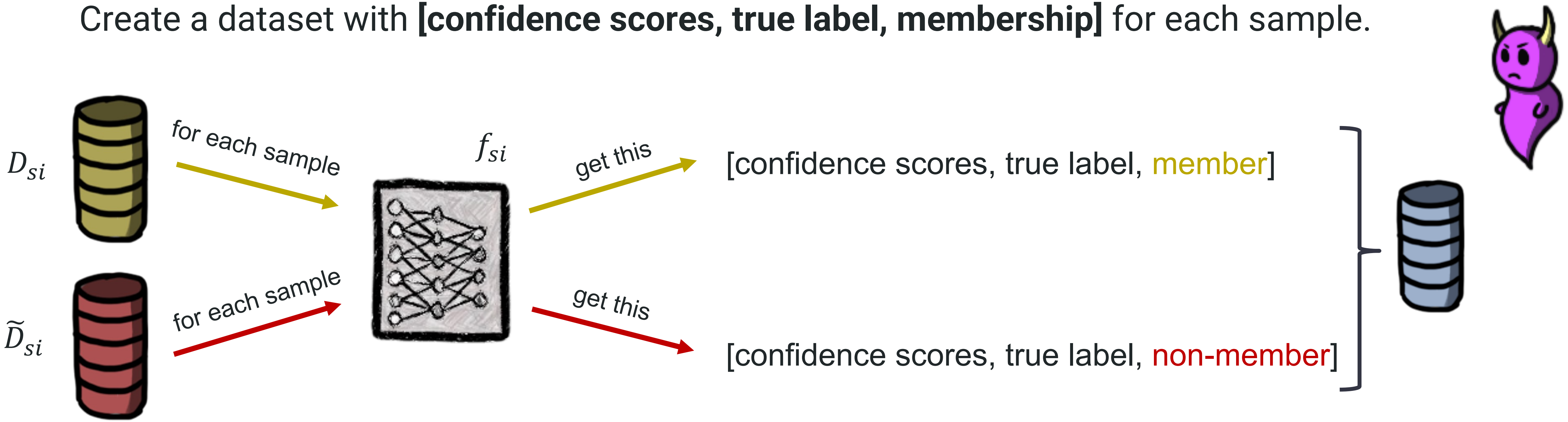


Works even with different models! (but better if you know the actual one)

Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Shokri et al.'s attack

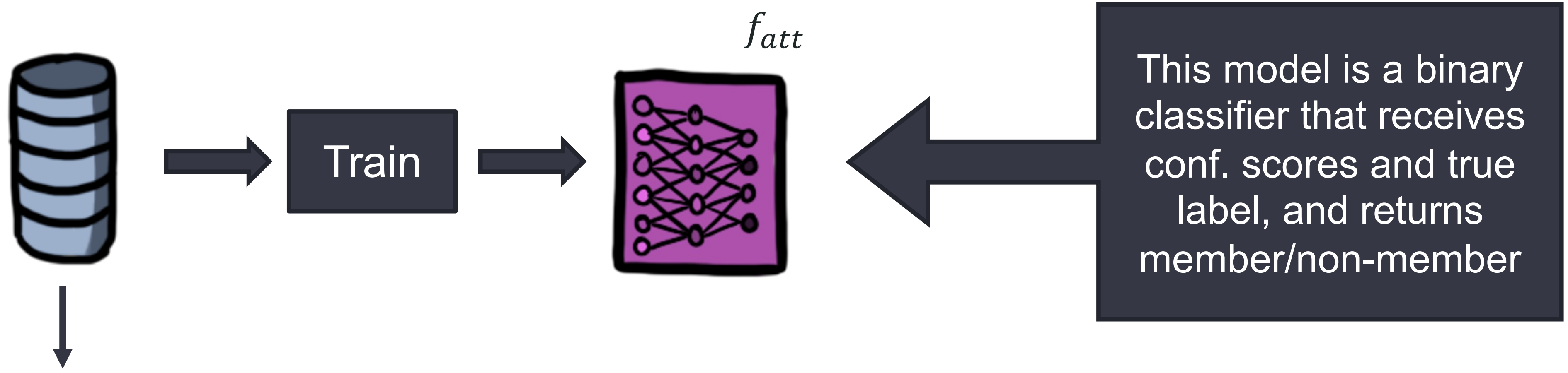
3. Generate **shadow** test data $\widetilde{D}_{s1}, \widetilde{D}_{s2},..., \widetilde{D}_{sk}$.
4. For each shadow model $i \in [k]$: get the confidence scores for each sample in D_{si} and $\widetilde{D}_{si}$. Create a dataset with **[confidence scores, true label, membership]** for each sample.



Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Shokri et al.’s attack

5. With the new dataset, that contains [confidence scores, true label, membership status] computed with all the shadow models, train a new **attack model** f_{att} to predict the **membership status** from **[confidence scores, true label]**

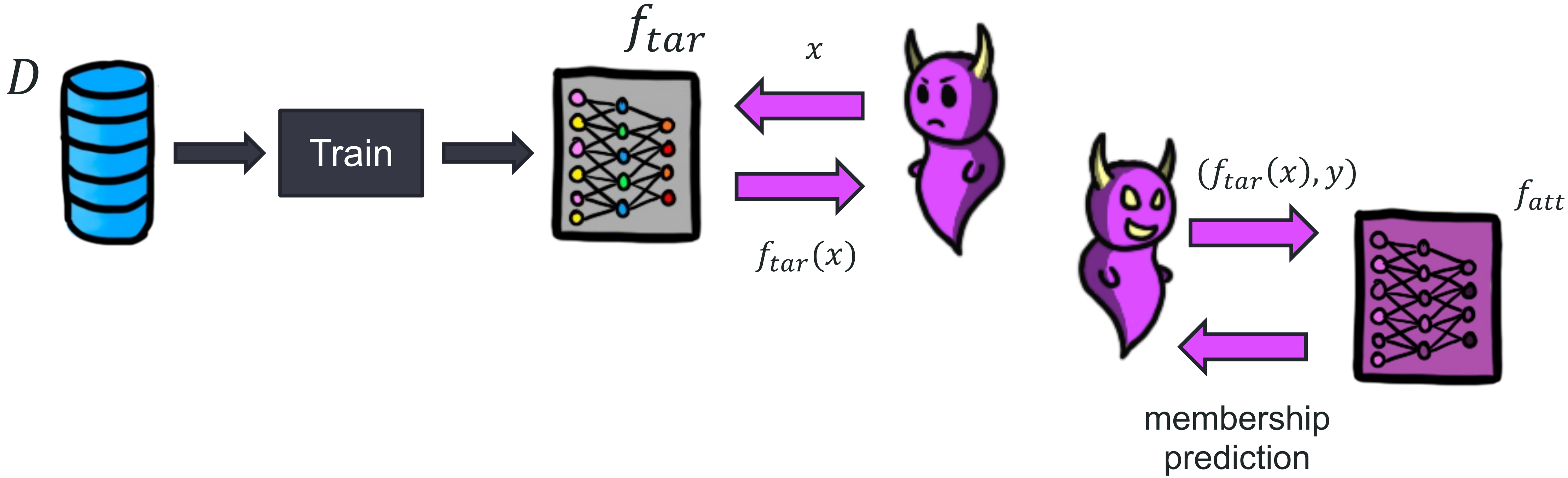


x=[confidence scores, true label], y=[member/non-member]

Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Shokri et al.'s attack

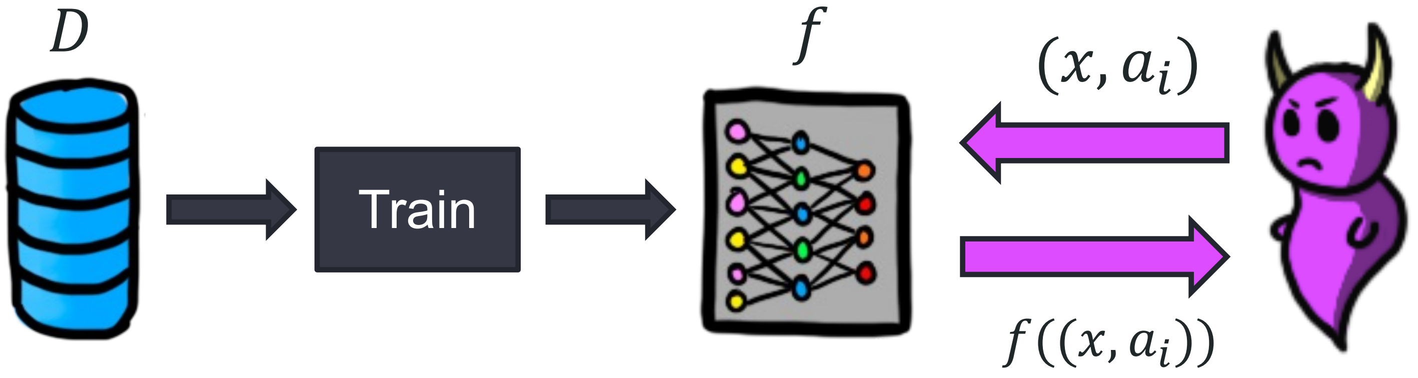
6. Get the confidence scores of the target sample in the target model f_{tar}.
7. Evaluate those **[confidence scores, true label]** samples in the attack model f_{att}.



Shokri, Reza, et al. "Membership inference attacks against machine learning models." *IEEE symposium on security and privacy (SP)*, 2017.

Attribute Inference Attacks

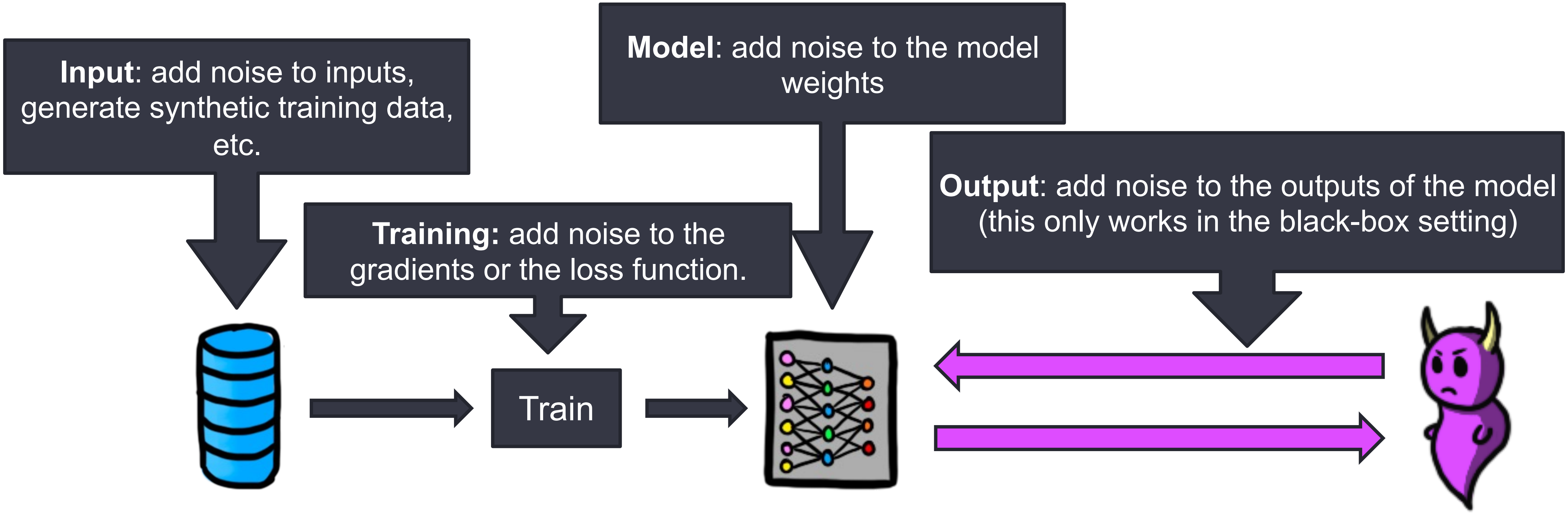
- Each sample is $z = (x, a, y)$, where x is the features, a is a privacy-sensitive attribute, and y is the label.
- The adversary has a sample $z = (x, ?, y)$, and wants to learn the attribute.
- Assume the space of all attributes is $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$
- Simple attack: query for all possible samples (x, a_1), …,(x, a_m). The true attribute is probably the one that yields a highest confidence score for the true class y.



$$\hat{\imath} = argmax_i f\big((x, a_i)\big)_y$$

Defending against inference attacks

- Where do we defend?



Defences against inference attacks

Differentially Private Stochastic Gradient Descent (DP-SGD)

- Adds privacy during the **training** step, modifying SGD.
- **Recall Differential Privacy:** we want to limit the effect that a single training set sample has on the output (the "output" of the training algorithm is the model!)

SGD

For each training step $t \in [T]$:

1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
   $$\mathrm{g_i} = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Average the gradients $g = \frac{1}{L} \sum_i g_i$.
4. Descend $\theta_t = \theta_{t-1} - \eta \cdot g$.

"Private" SGD

For each training step $t \in [T]$:

1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
   $$\mathrm{g_i} = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Average the gradients and add noise $g = \frac{1}{L}(\sum_i g_i + \mathcal{N}(0, \sigma^2))$.
4. Descend $\theta_t = \theta_{t-1} - \eta \cdot g$.

Q: Is it enough to add noise to the gradients? What's the catch?

Differentially Private Stochastic Gradient Descent (DP-SGD)

e.g., sample with outlier attributes

- The gradient could potentially be **unbounded** → Here, unbounded sensitivity is bad for DP
- We ***clip*** the gradients to ensure their ℓ_2 norm is at most C.
  - C is the *clipping threshold* (1 is usually a good value)
  - C is independent of the data

SGD

For each training step $t \in [T]$:
1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
$$\mathrm{g_i} = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Average the gradients $g = \frac{1}{L}\sum_i g_i$.
4. Descend $\theta_t = \theta_{t-1} - \eta \cdot g$.

DP-SGD

For each training step $t \in [T]$:
1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
$$\mathrm{g_i} = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Clip the gradients: $g_i = g_i / \max\left(1, \frac{\|g_i\|_2}{C}\right)$
4. Sum the gradients $g = \sum_i g_i$.
5. Add noise: $g = g + \mathcal{N}(0, \sigma^2 C^2)$
6. Descend $\theta_t = \theta_{t-1} - \eta \cdot \frac{1}{L} g$.

DP-SGD: keeping track of ϵ, δ

- Note that a single sample will participate in multiple training steps → there will be some **sequential composition** involved.

- We need to **keep track of ϵ, δ**.
- For a *fixed amount of noise* σ, we can end up with a very large ϵ, which is bad.
  - The actual *true* ϵ will be smaller than the ϵ we can compute theoretically. – e.g., due to batching, one sample may not appear in a given training step.
  - We can only guarantee an ϵ we can prove w/ theory.

DP-SGD

For each training step $t \in [T]$:

1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
$$g_i = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Clip the gradients: $g_i = g_i / \max\left(1, \frac{||g_i||_2}{C}\right)$
4. Sum the gradients $g = \sum_i g_i$.
5. Add noise: $g = g + \mathcal{N}(0, \sigma^2 C^2)$
6. Descend $\theta_t = \theta_{t-1} - \eta \cdot \frac{1}{L} g$.

DP-SGD: keeping track of ϵ, δ

- First, we choose a δ. Recall that this should be smaller than $\delta < \frac{1}{N}$.
  - The reason is the following: a training algorithm that simply publishes a random training set record would provide $(\epsilon = 0, \delta = 1/N)$-DP. However, we know this is not private enough.

DP-SGD

For each training step $t \in [T]$:

1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
$$g_i = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Clip the gradients: $g_i = g_i / \max\left(1, \frac{\|g_i\|_2}{C}\right)$
4. Sum the gradients $g = \sum_i g_i$.
5. Add noise: $g = g + \mathcal{N}(0, \sigma^2 C^2)$
6. Descend $\theta_t = \theta_{t-1} - \eta \cdot \frac{1}{L} g$.

DP-SGD: keeping track of ϵ, δ

Q: Given δ, σ, C, T, and assuming each sample in D is used *once per training step*, what is the total ϵ we get?

- Use naive composition

DP-SGD

For each training step $t \in [T]$:

1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
   $$g_i = \nabla \ell(\theta_{t-1}, x_i, y_i)$$
3. Clip the gradients: $g_i = g_i / \max\left(1, \frac{\|g_i\|_2}{C}\right)$
4. Sum the gradients $g = \sum_i g_i$.
5. Add noise: $g = g + \mathcal{N}(0, \sigma^2 C^2)$
6. Descend $\theta_t = \theta_{t-1} - \eta \cdot \frac{1}{L} g$.

$f(D) + Y$ is (ϵ, δ)-DP if

$$Y \sim N(0, \sigma^2)$$

$$\sigma^2 = 2 \ln\left(\frac{1.25}{\delta}\right) \Delta_2^2 / \epsilon^2$$

DP-SGD: keeping track of ϵ, δ

Q: Given δ, σ, C, T, and assuming each sample in D is used *once per training step,* what is the total ϵ we get?

- Use naive composition

A: $C^2\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2 \quad \rightarrow \epsilon = \sqrt{2\ln\left(\frac{1.25}{\delta}\right)}/\sigma$ for each step. Then naïve composition gives

$$\epsilon = T\sqrt{2\ln\left(\frac{1.25}{\delta}\right)}/\sigma$$

*Note: this question is very over simplified

DP-SGD

For each training step $t \in [T]$:

1. Take a batch B of L samples from D.
2. For each $(x_i, y_i) \in B$, compute the gradient:
   $$g_i = \nabla\ell(\theta_{t-1}, x_i, y_i)$$
3. Clip the gradients: $g_i = g_i / \max\left(1, \frac{\|g_i\|_2}{C}\right)$
4. Sum the gradients $g = \sum_i g_i$.
5. Add noise: $g = g + \mathcal{N}(0, \sigma^2 C^2)$
6. Descend $\theta_t = \theta_{t-1} - \eta \cdot \frac{1}{L} g$.

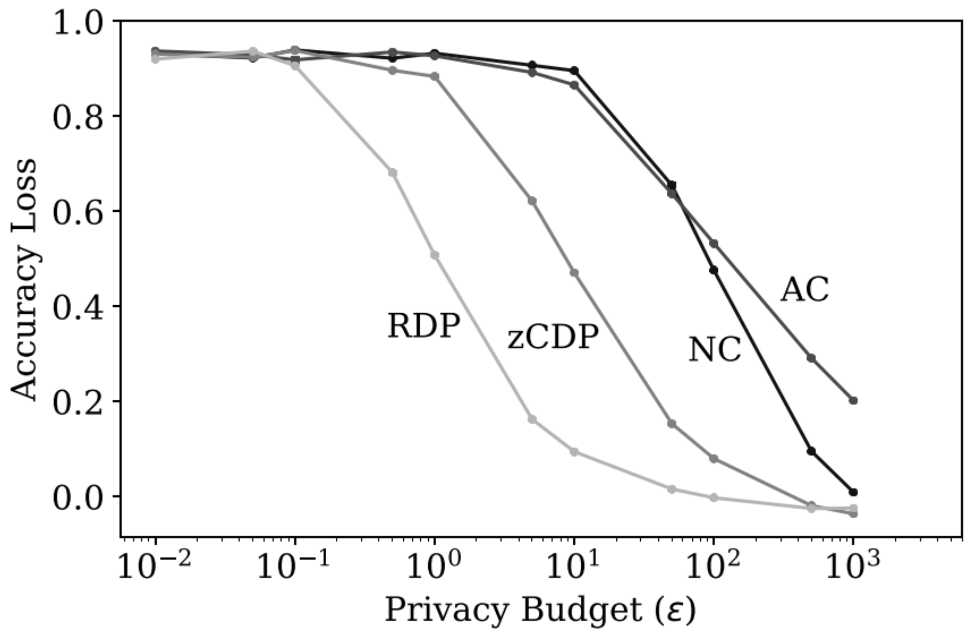
$f(D) + Y$ is (ϵ, δ)-DP if

$$Y \sim N(0, \sigma^2)$$

$$\sigma^2 = 2\ln\left(\frac{1.25}{\delta}\right)\Delta_2^2/\epsilon^2$$

DP-SGD: keeping track of ϵ, δ

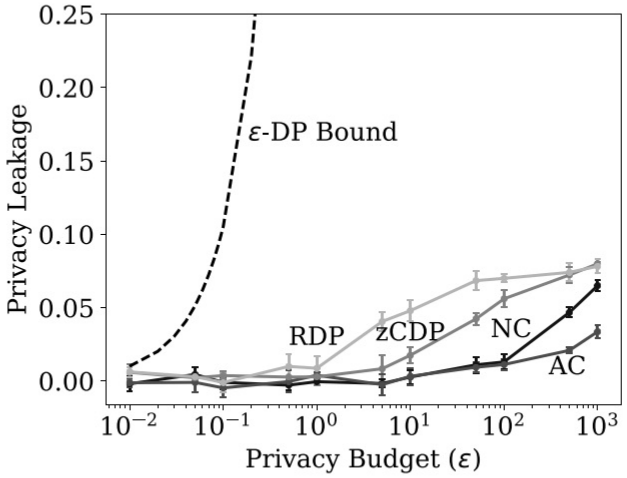
- Renyi Differential Privacy (RDP) provides a tighter ϵ, δ bound.
  - Better suited to Gaussian Noise
  - Keeps track of more information
- This means that, for a given σ, C, and δ, RDP tells us our actual ϵ is smaller than what Advanced Composition (AC) tells us.
- In other words, for a target privacy budget ϵ, using RDP we need to add less noise than using AC.
  - E.g., again, because a sample may be excluded from a given training step
- Note that, even with RDP, we need $\epsilon > 100$ if we do not want any accuracy loss



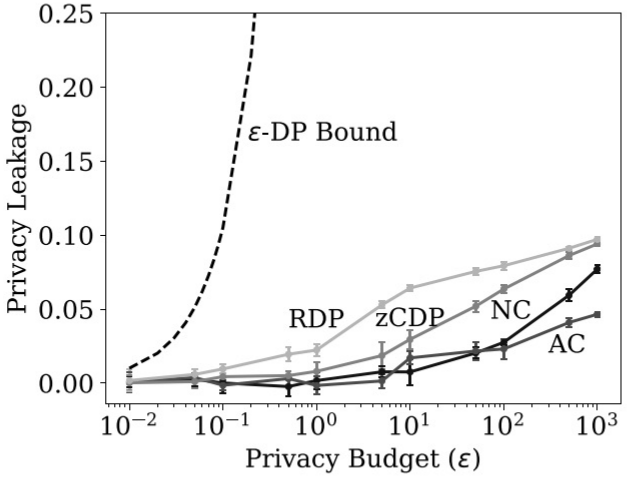
Jayaraman, Bargav, and David Evans. "Evaluating differentially private machine learning in practice." *USENIX Security Symposium*. 2019.

DP-SGD: theoretical vs empirical privacy

- Both attacks we've seen perform similarly
- It seems that $\epsilon = 100$ or even $\epsilon = 1000$ still provides good empirical privacy
- The theoretical bound on the privacy leakage provided by DP is very loose



(a) Shokri et al. membership inference



(b) Yeom et al. membership inference

Jayaraman, Bargav, and David Evans. "Evaluating differentially private machine learning in practice." *USENIX Security Symposium*. 2019.

Issues of DP-SGD

- We saw that, for strong theoretical privacy (e.g., $\epsilon < 1$), the models usually lose all utility.
- For very weak theoretical privacy (e.g., $\epsilon = 100$), some models achieve reasonable utility.
- However, DP-SGD with $\epsilon = 100$ seems to provide enough protection against existing attacks.

Q: Is it OK to use $\epsilon = 100$?

Issues of DP-SGD

- We saw that, for strong theoretical privacy (e.g., $\epsilon < 1$), the models usually lose all utility.
- For very weak theoretical privacy (e.g., $\epsilon = 100$), some models achieve reasonable utility.
- However, DP-SGD with $\epsilon = 100$ seems to provide enough protection against existing attacks.

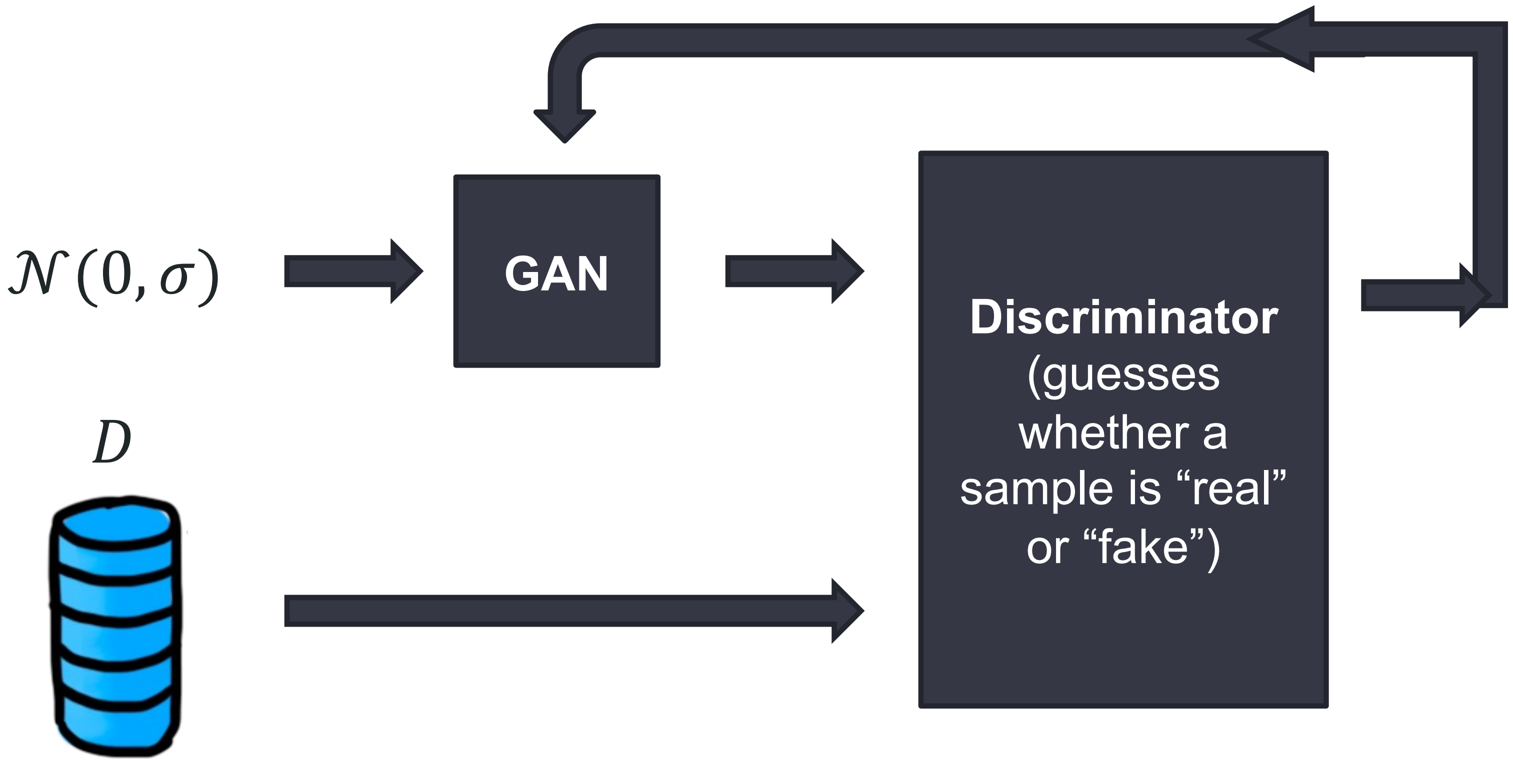
Q: Is it OK to use $\epsilon = 100$?

A: It might be OK to use DP-SGD tuned to $\epsilon = 100$, but at that point we might as well use defenses that do not provide DP, since the DP guarantee is already meaningless at that point.

More defences…

Synthetic Data Generation

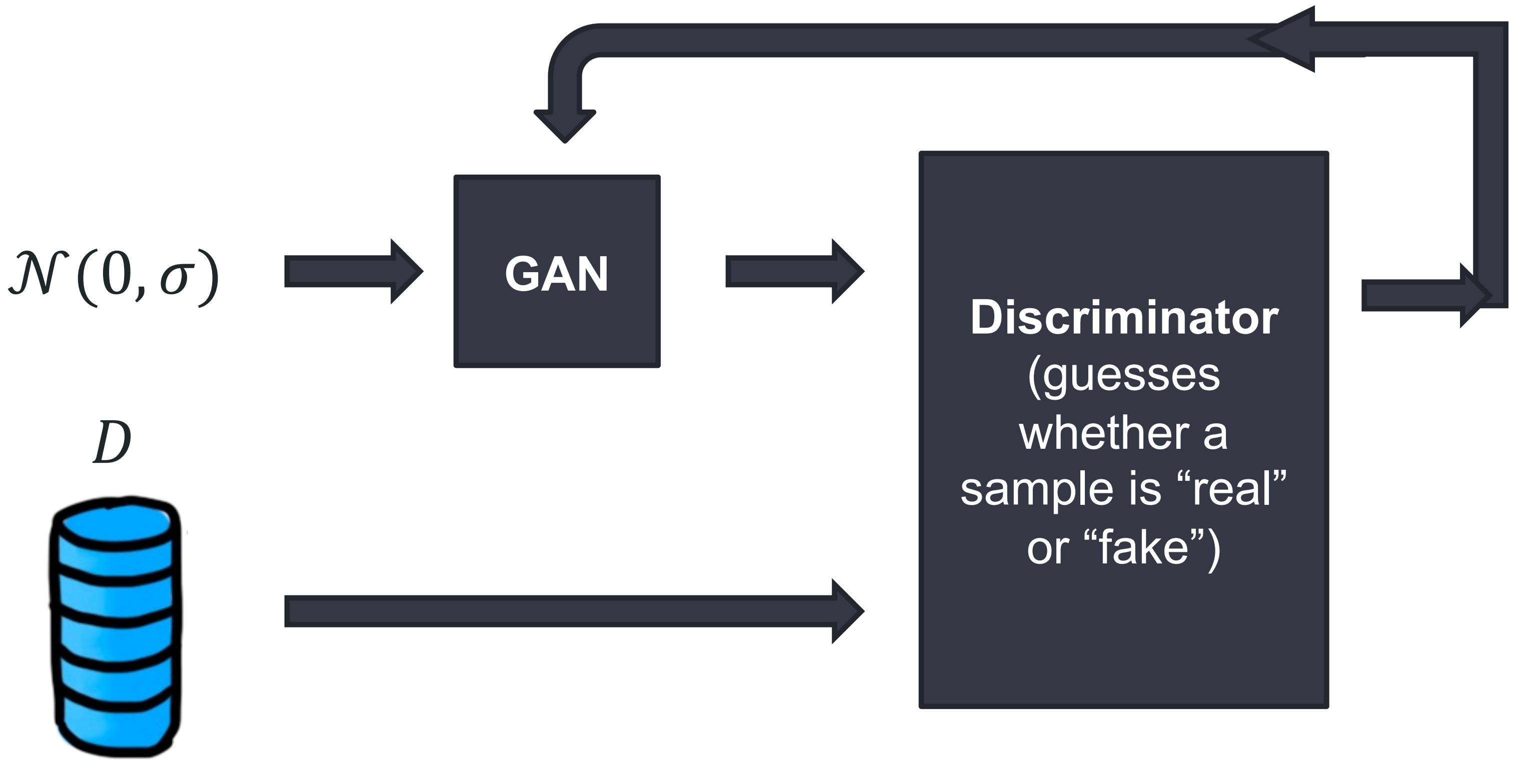
- For example, by using a GAN to generate real-looking synthetic samples:



If we train the GAN using privacy-preserving training algorithms (e.g., DP-SGD on the discriminator), we can use it to generate a privacy-preserving synthetic dataset!

Synthetic Data Generation

- For example, by using a GAN to generate real-looking synthetic samples:

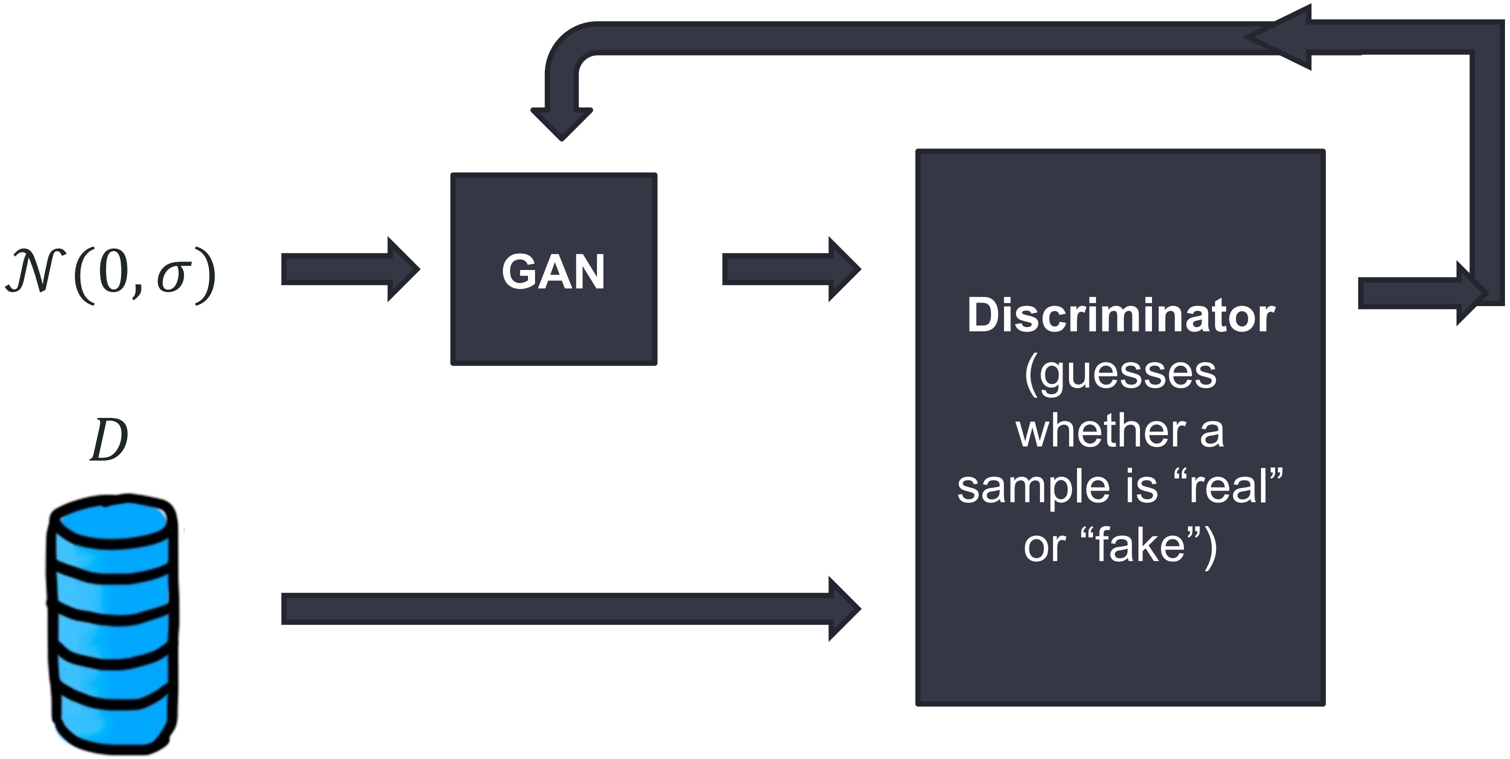


If we train the GAN using privacy-preserving training algorithms (e.g., DP-SGD on the discriminator), we can use it to generate a privacy-preserving synthetic dataset!

Q: What can we do with the resulting dataset?

Synthetic Data Generation

- For example, by using a GAN to generate real-looking synthetic samples:



If we train the GAN using privacy-preserving training algorithms (e.g., DP-SGD on the discriminator), we can use it to generate a privacy-preserving synthetic dataset!

Q: What can we do with the resulting dataset?

A: Anything!

Other defenses

- There are defenses that add **noise** to the confidence scores (MemGuard [Jia et al.]), but are not very effective.
- MIAs can work even if the model just leaks the predicted label (and not the confidence scores)
- Sometimes, **generalization** is a good defense by itself:
  - A well-generalized model will perform similarly in members (training set) and non-members (testing set)
  - Therefore, it will be harder for an adversary to decide whether a sample is a member or non-member if the model generalizes well.
  - Generalization is also **good for utility** (improves test accuracy), so it's a win-win.