Machine learning models that remember too much

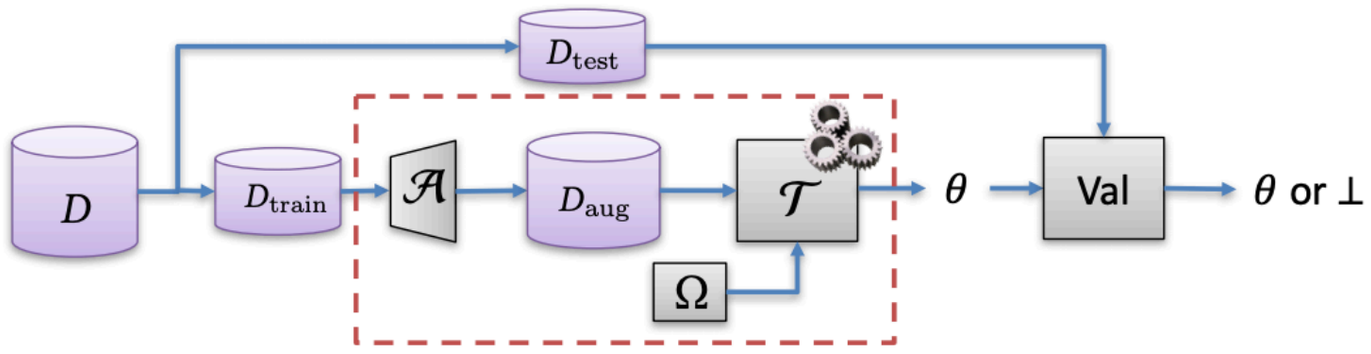
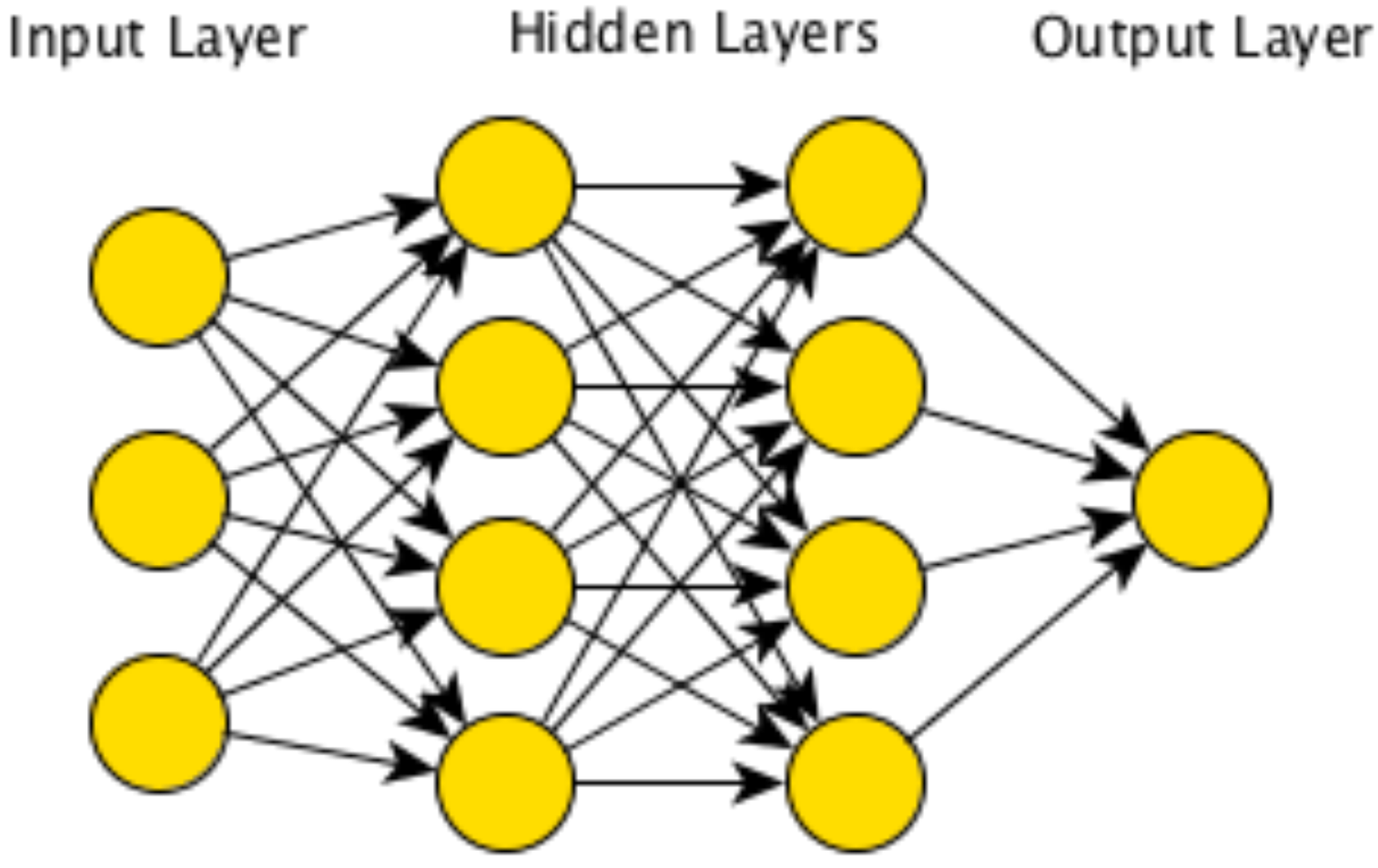


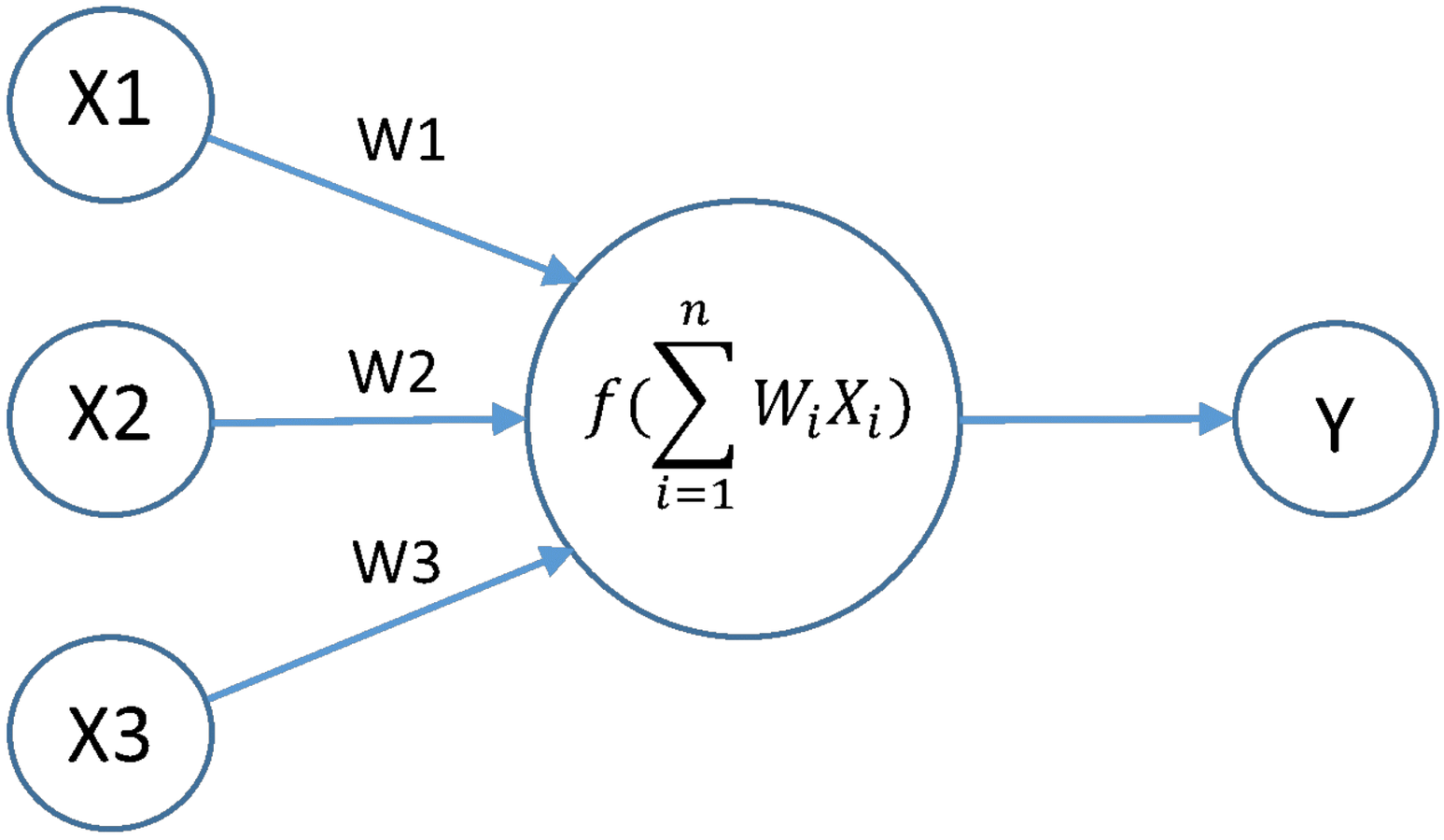
Figure 1: A typical ML training pipeline. Data D is split into training set D_{train} and test set D_{test}. Training data may be augmented using an algorithm $\mathcal{A}$, and then parameters are computed using a training algorithm $\mathcal{T}$ that uses a regularizer Ω. The resulting parameters are validated using the test set and either accepted or rejected (an error $\perp$ is output). If the parameters θ are accepted, they may be published (white-box model) or deployed in a prediction service to which the adversary has input/output access (black-box model). The dashed box indicates the portions of the pipeline that may be controlled by the adversary.

Song, C., Shmatikov, V., & Ristenpart, T. (2017). Machine learning models that remember too much. Proceedings of the ACM Conference on Computer and Communications Security, 587-601.

Architecture



Neural Network



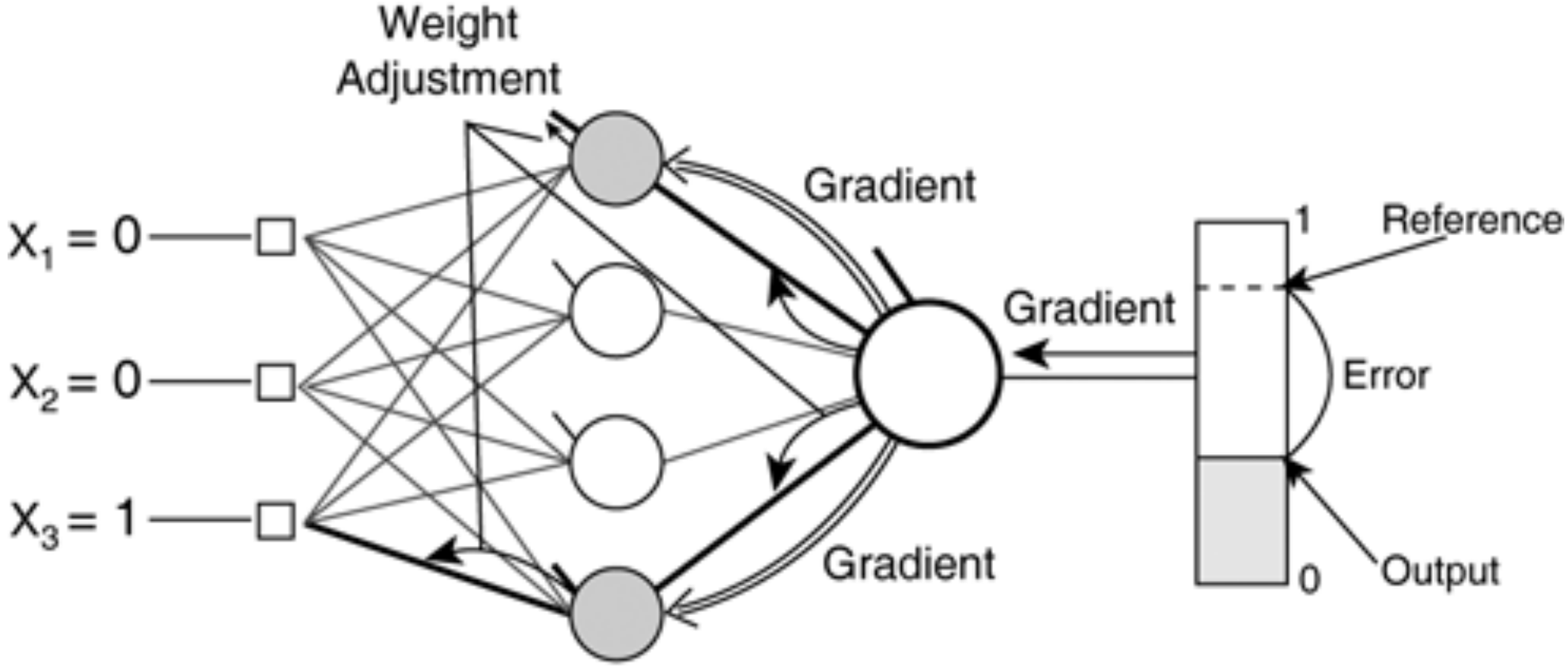
- **w_i refers to the weight for every input x_i**
- **$f(\sum w_i x_i)$ is an activation function, it is used to compress the $\sum w_i x_i$ to a certain domain.**
- **if $f(\sum w_i x_i)$ choose the sigmoid function,**
- **$f(\sum w_i x_i) = 1/(1+\exp(-\sum w_i x_i))$**
- **then the final output $Y \in [0,1]$**

Back Propagation

- Back propagation is exactly a ‘learning’ process
- By calculating the cost/error of the each layer’s output, DNN uses the partial derivatives of cost on each weights to updates every weights.
- DNN update the weights to minimize the cost/error of each layer and make the final output more accurate.

L2: $\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$

$$J(\mathbf{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi(z^{(i)})\right) - \left(1-y^{(i)}\right)\log\left(1-\phi(z^{(i)})\right)\right] + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



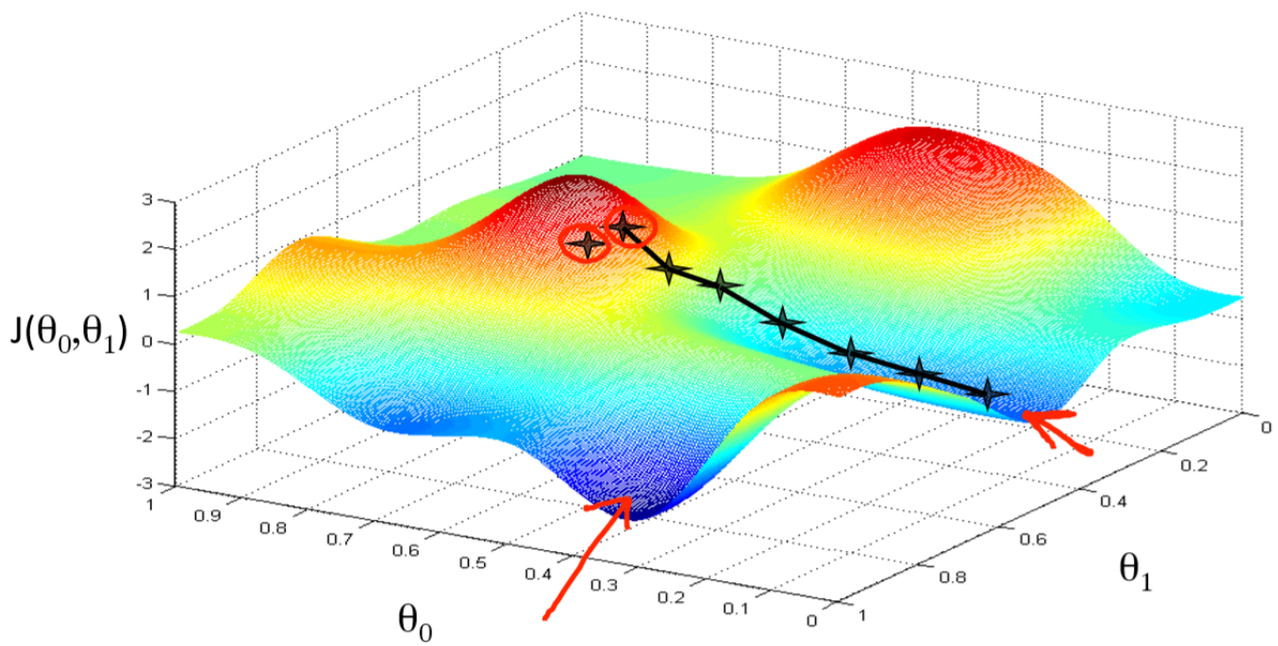
$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Old weight

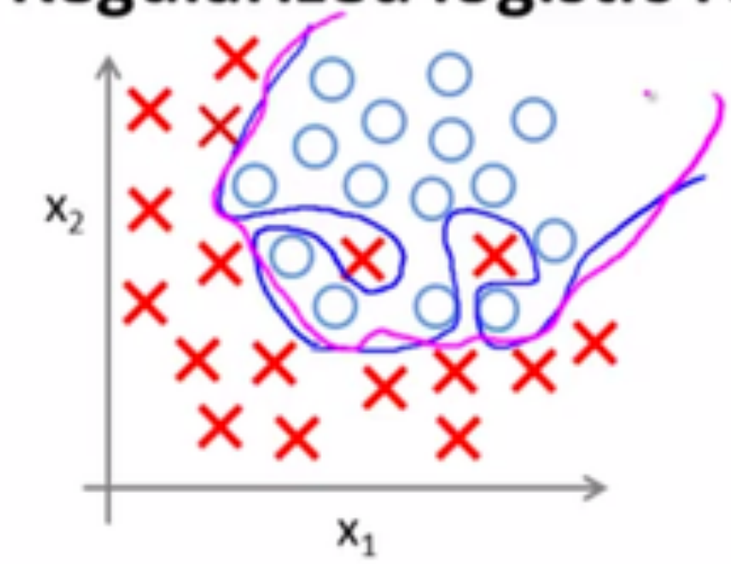
Learning rate

Derivative of Error with respect to weight



Regularizer

Regularized logistic regression.



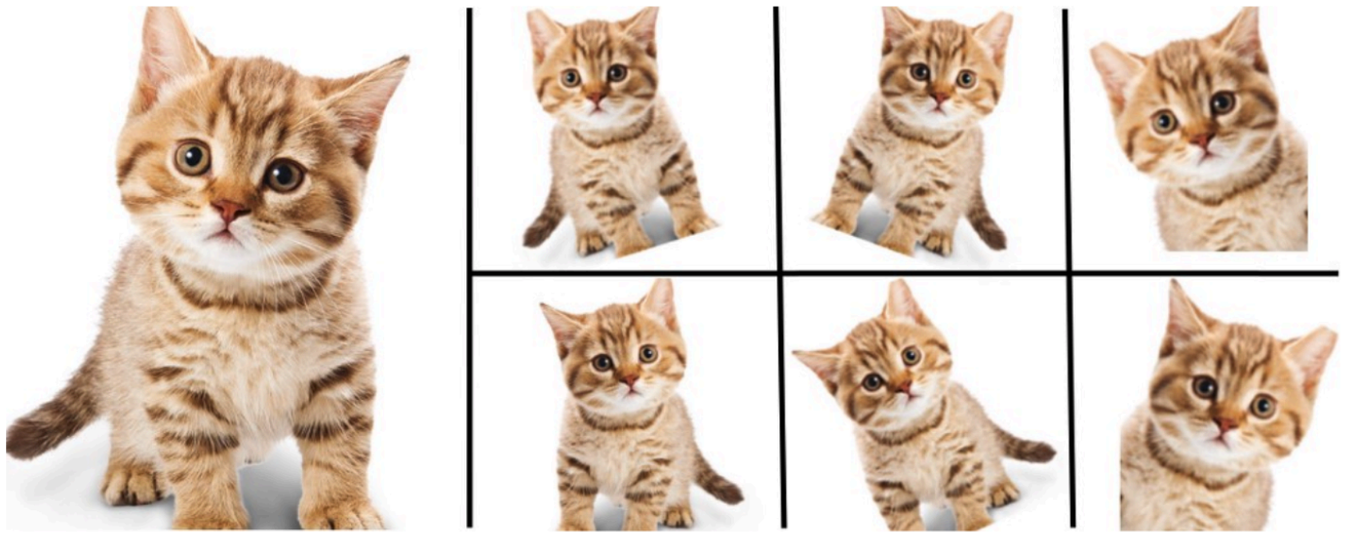
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

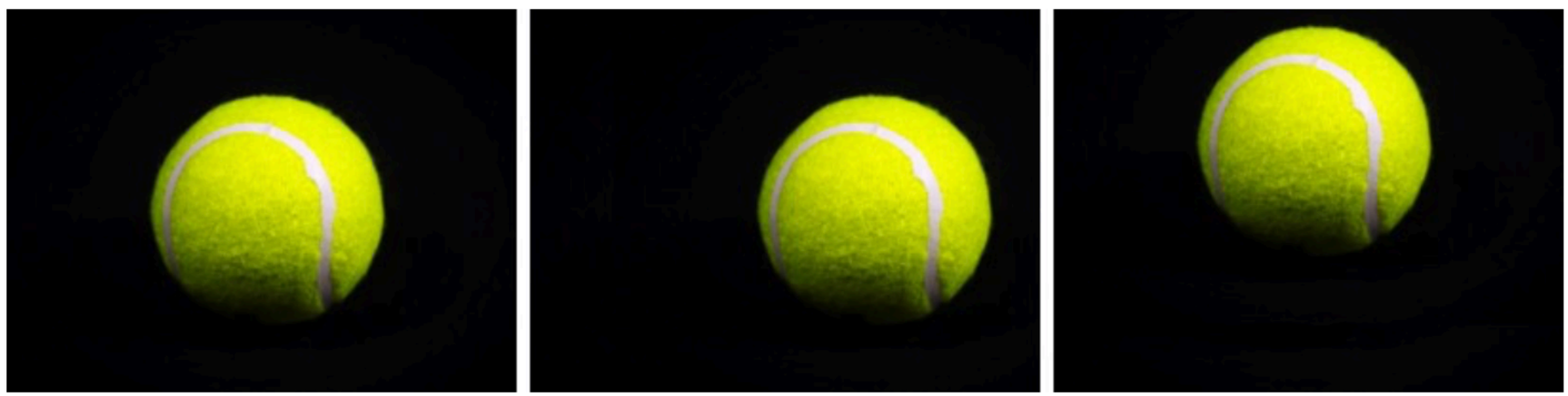
$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

$\theta_1, \theta_2, \dots, \theta_n$

Data Augmentation



Enlarge your Dataset



The same tennis ball, but translated.

Machine learning models that remember too much

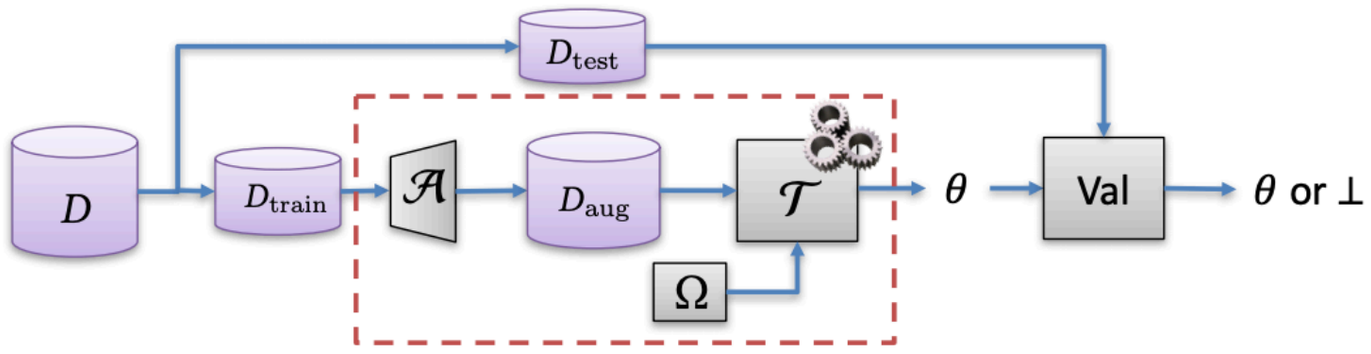


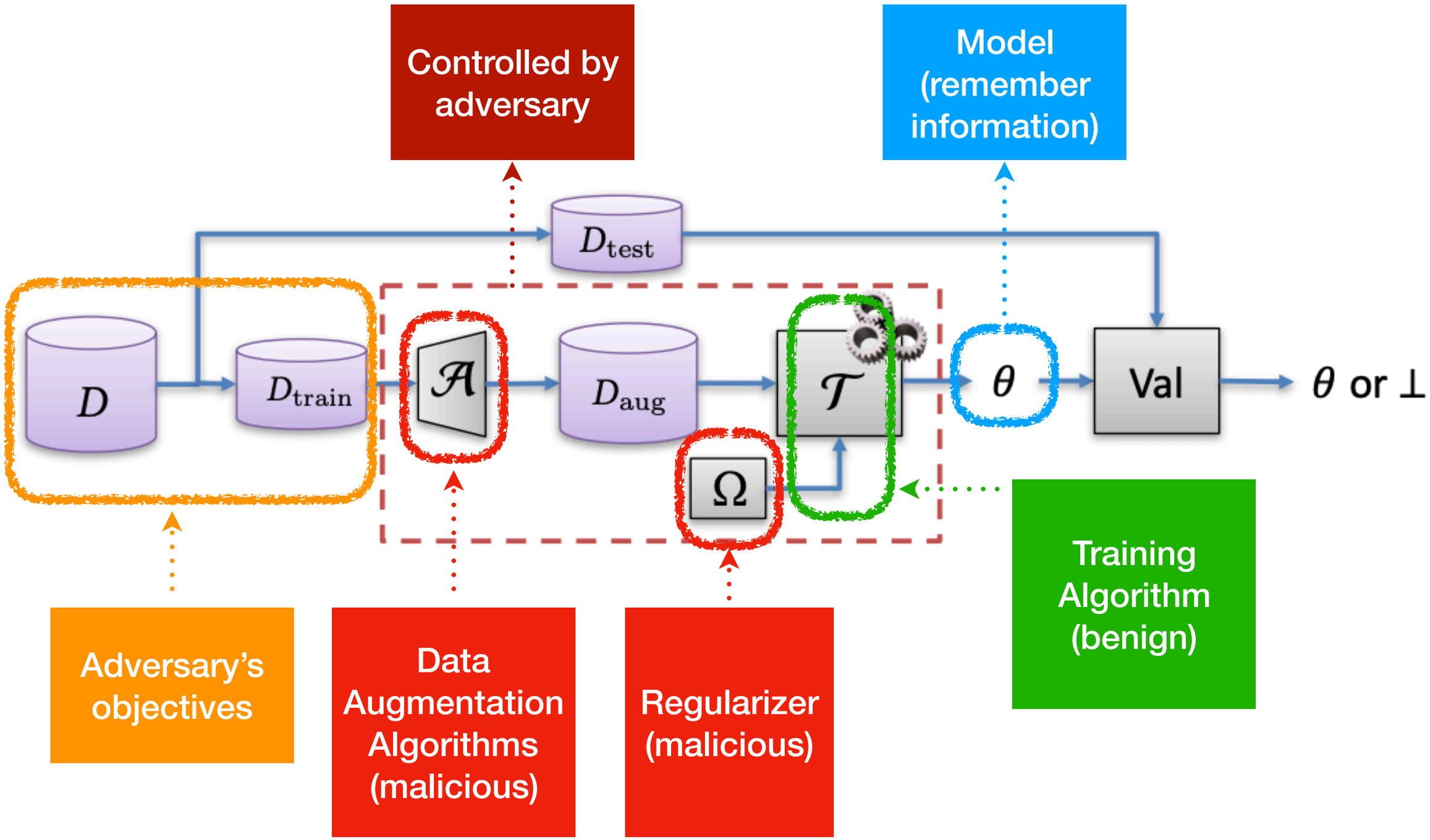
Figure 1: A typical ML training pipeline. Data D is split into training set D_{train} and test set D_{test}. Training data may be augmented using an algorithm $\mathcal{A}$, and then parameters are computed using a training algorithm $\mathcal{T}$ that uses a regularizer Ω. The resulting parameters are validated using the test set and either accepted or rejected (an error $\perp$ is output). If the parameters θ are accepted, they may be published (white-box model) or deployed in a prediction service to which the adversary has input/output access (black-box model). The dashed box indicates the portions of the pipeline that may be controlled by the adversary.

Song, C., Shmatikov, V., & Ristenpart, T. (2017). Machine learning models that remember too much. Proceedings of the ACM Conference on Computer and Communications Security, 587-601.

Scenario Set up

- Data holders
- Third-party ML code
- ML models ‘memorizing’ the training data
- Attacker aims at private training dataset D

Scenario Set Up



White-Box

- Adversary can see the model parameters — θ
- θ contains (related) information of training data
- Adversary decodes θ to get (most of) the training dataset
- Three kinds of Methods
  - 1. LSB Encoding
  - 2. Correlated Value Encoding
  - 3. Sign Encoding

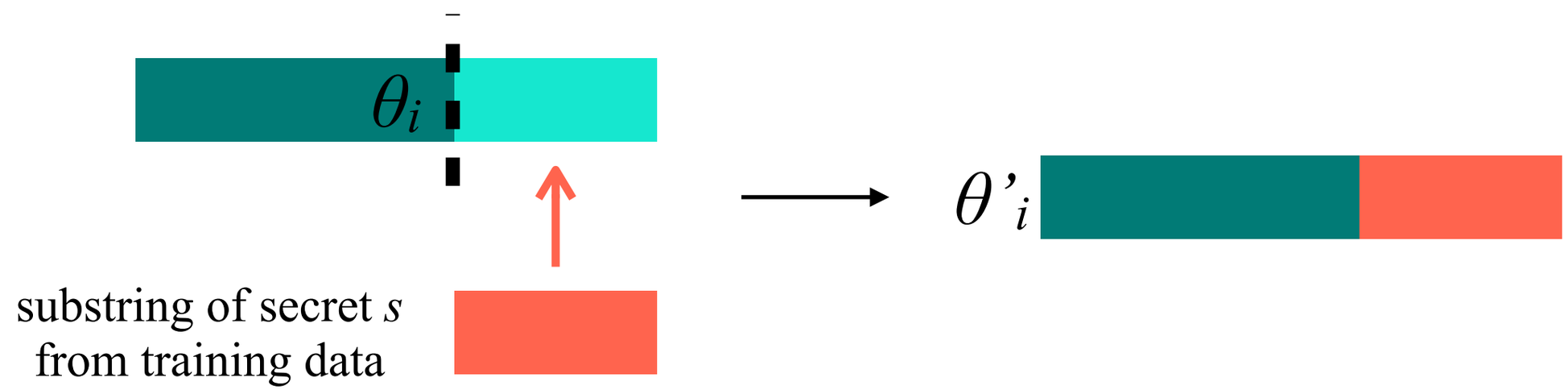
LSB Encoding

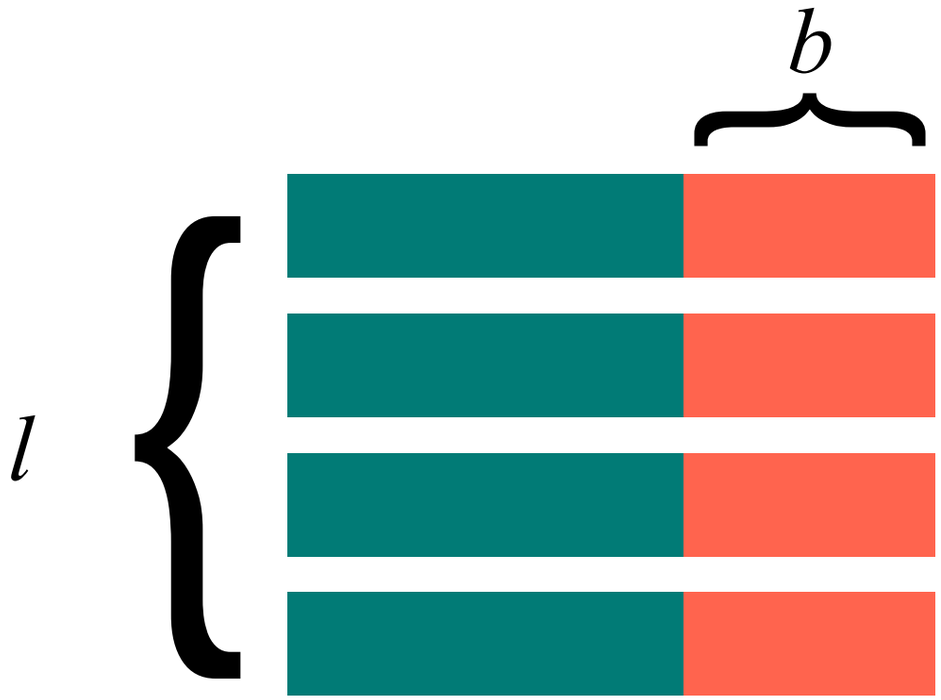
Main Idea:
Let the model parameters directly carry information of training data.

Algorithm 1 LSB encoding attack

Input: Training dataset D_{train}, a benign ML training algorithm $\mathcal{T}$, number of bits b to encode per parameter.
Output: ML model parameters θ' with secrets encoded in the lower b bits.
$\theta \leftarrow \mathcal{T}(D_{\text{train}})$
$\ell \leftarrow$ number of parameters in θ
$s \leftarrow$ **ExtractSecretBitString**$(D_{\text{train}}, \ell b)$
$\theta' \leftarrow$ set the lower b bits in each parameter of θ to a substring of s of length b.

LSB Encoding





Decode

Simply read the lower bits of the parameters θ'

and interpret them as bits of the secret.

Correlated Value Encoding

Main Idea:
Let the model parameters have some relationship with training data.
Approach:
Using a malicious regularizer C, Pearson correlation coefficient

Algorithm 2 SGD with correlation value encoding

Input: Training dataset $D_{\text{train}} = \{(x_j, y_j)\}_{i=1}^{n}$, a benign loss function $\mathcal{L}$, a model f, number of epochs T, learning rate η, attack coefficient λ_c, size of mini-batch q.
Output: ML model parameters θ correlated to secrets.
$\theta \leftarrow$ **Initialize**(f)
$\ell \leftarrow$ number of parameters in θ
$s \leftarrow$ **ExtractSecretValues**(D, ℓ)
for $t = 1$ to T **do**
  for each mini-batch $\{(x_j, y_j)\}_{j=1}^{q} \subset D_{\text{train}}$ **do**
    $g_t \leftarrow \nabla_\theta \frac{1}{m} \sum_{j=1}^{q} \mathcal{L}(y_j, f(x_j, \theta)) + \nabla_\theta C(\theta, s)$
    $\theta \leftarrow$ **UpdateParameters**(η, θ, g_t)
  end for
end for

$$C(\theta, s) = -\lambda_c \cdot \frac{\left|\sum_{i=1}^{\ell}(\theta_i - \bar{\theta})(s_i - \bar{s})\right|}{\sqrt{\sum_{i=1}^{\ell}(\theta_i - \bar{\theta})^2} \cdot \sqrt{\sum_{i=1}^{\ell}(s_i - \bar{s})^2}}.$$

λc controls the level of correlation

this value falls into [0,1] When it closes to 1, θ and s is linear related, otherwise not

Sign Encoding

Main Idea:

Let the model interpret signs as a bit string, e.g., a positive parameter represents 1 and a negative parameter represents 0.

Using a penalty term P to minimize the difference between θ and secret s

$$\min_{\theta} \Omega(\theta) + \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i, \theta))$$

such that $\theta_i s_i > 0$ for $i = 1, 2, \ldots, \ell$

$$P(\theta, s) = \frac{\lambda_s}{\ell} \sum_{i=1}^{\ell} |\max(0, -\theta_i s_i)| \ .$$

λs is a hyperparameter that controls the magnitude of the penalty.

It is similar to the correlated value encoding, a malicious regularizer term is added to make parameters similar to the secret

Black-Box

- Adversary cannot see the model parameters
- Adversary uses malicious data augmentation algorithm
- Adversary uses synthetic data x with special label to train model until the model is overfitting with synthetic data x
- Special label is made of substring of training data
- This model works well with normal training data

Capacity Abuse Attack

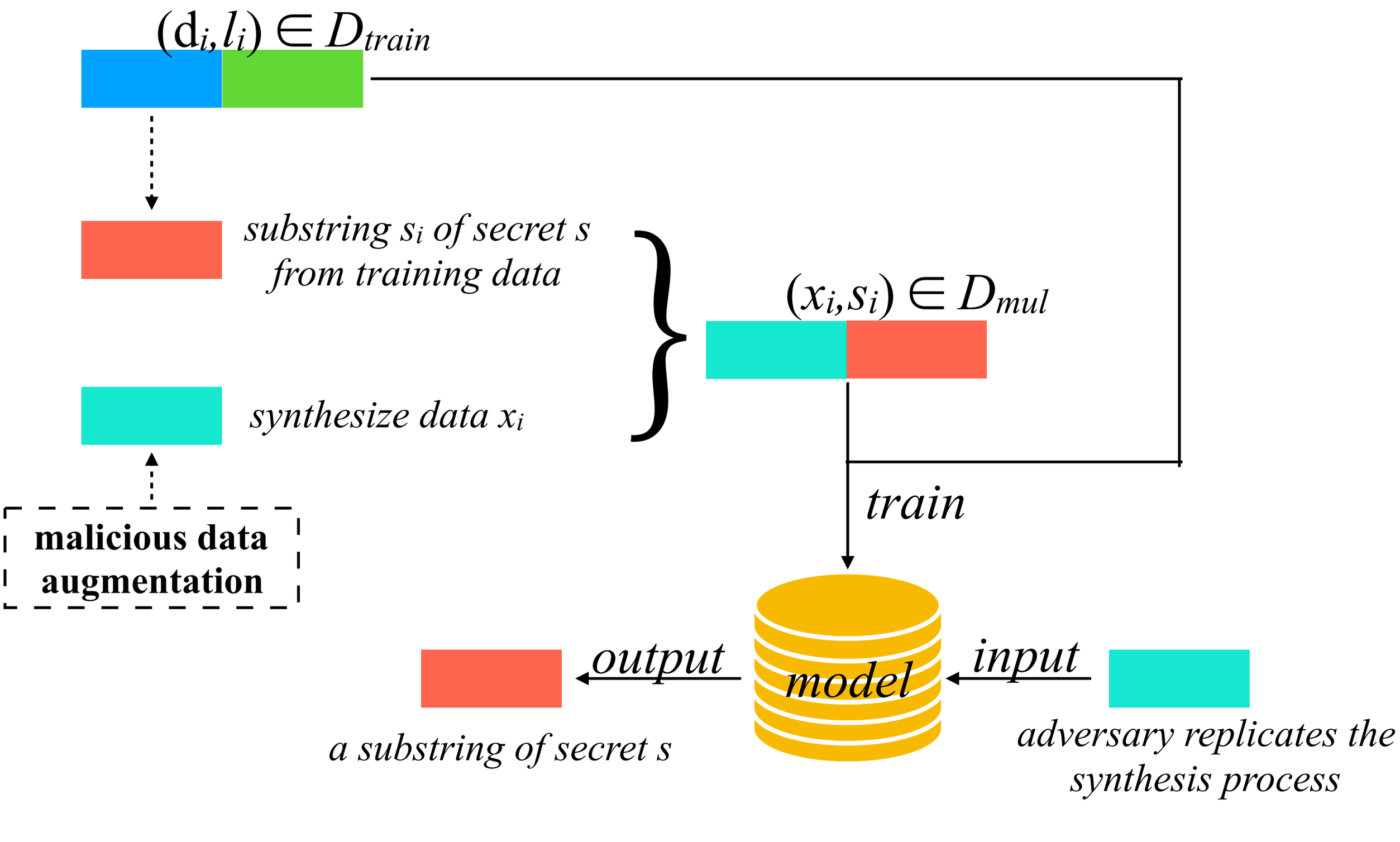
Algorithm 3 Capacity-abuse attack

1: **Input:** Training dataset D_{train}, a benign ML training algorithm $\mathcal{T}$, number of inputs m to be synthesized.
2: **Output:** ML model parameters θ that memorize the malicious synthetic inputs and their labels.
3: $D_{\text{mal}} \leftarrow$ **SynthesizeMaliciousData**(D_{train}, m)
4: $\theta \leftarrow \mathcal{T}(D_{\text{train}} \cup D_{\text{mal}})$

Algorithm 4 Synthesizing malicious data

1: **Input:** A training dataset D_{train}, number of inputs to be synthesized m, auxiliary knowledge K.
2: **Output:** Synthesized malicious data D_{mal}
3: $D_{\text{mal}} \leftarrow \emptyset$
4: $s \leftarrow$ **ExtractSecretBitString**(D_{train}, m)
5: $c \leftarrow$ number of classes in D_{train}
6: **for** each $\lfloor \log_2(c) \rfloor$ bits s' in s **do**
7: $\quad x_{\text{mal}} \leftarrow$ **GenData**(K)
8: $\quad y_{\text{mal}} \leftarrow$ **BitsToLabel**(s')
9: $\quad D_{\text{mal}} \leftarrow D_{\text{mal}} \cup \{(x_{\text{mal}}, y_{\text{mal}})\}$
10: **end for**

Capacity Abuse Attack



Experiments

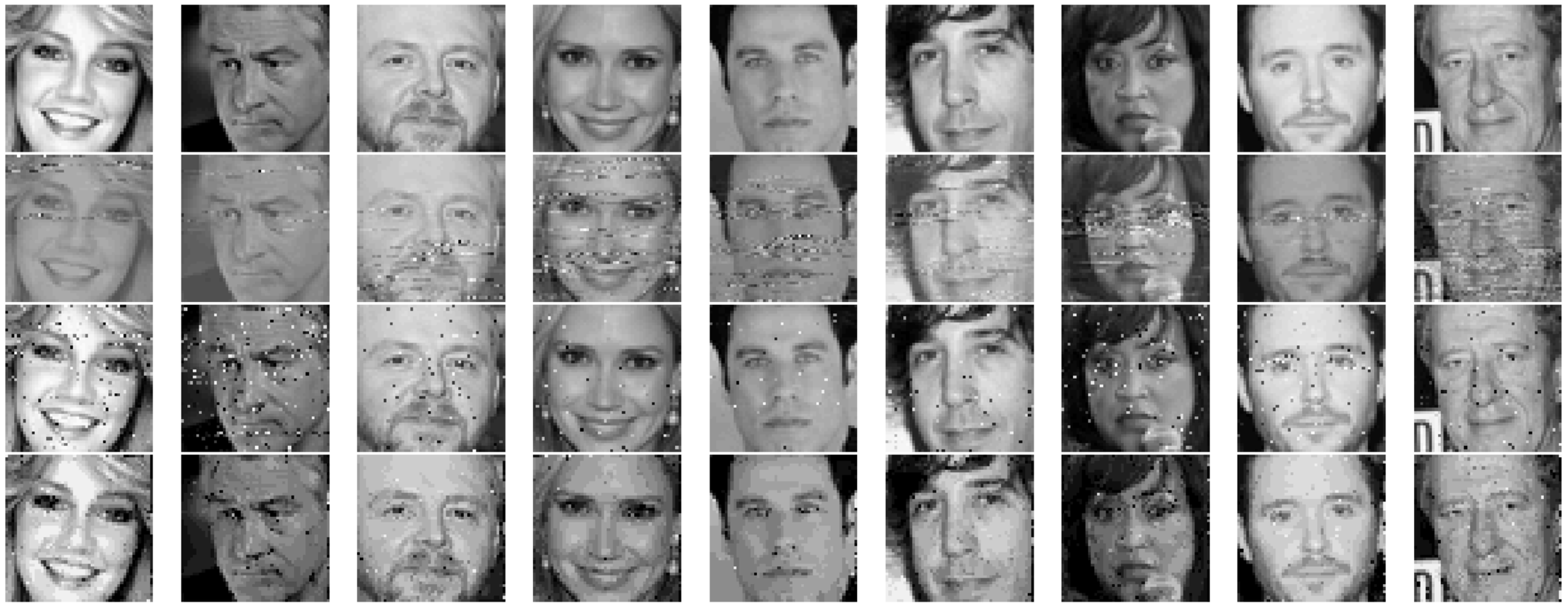


Figure 3: Decoded examples from all attacks applied to models trained on the FaceScrub gender classification task. First row is the ground truth. Second row is the correlated value encoding attack (λ_c=1.0, MAPE=15.0). Third row is the sign encoding attack (λ_s=10.0, MAPE=2.51). Fourth row is the capacity abuse attack (m=110K, MAPE=10.8).