南京航空航天大學

Nanjing University of Aeronautics and Astronautics

ParNeC 模式识别与神经计算研究组

PAttern Recognition and NEural Computing

A Brief Introduction to Machine Unlearning

Contents

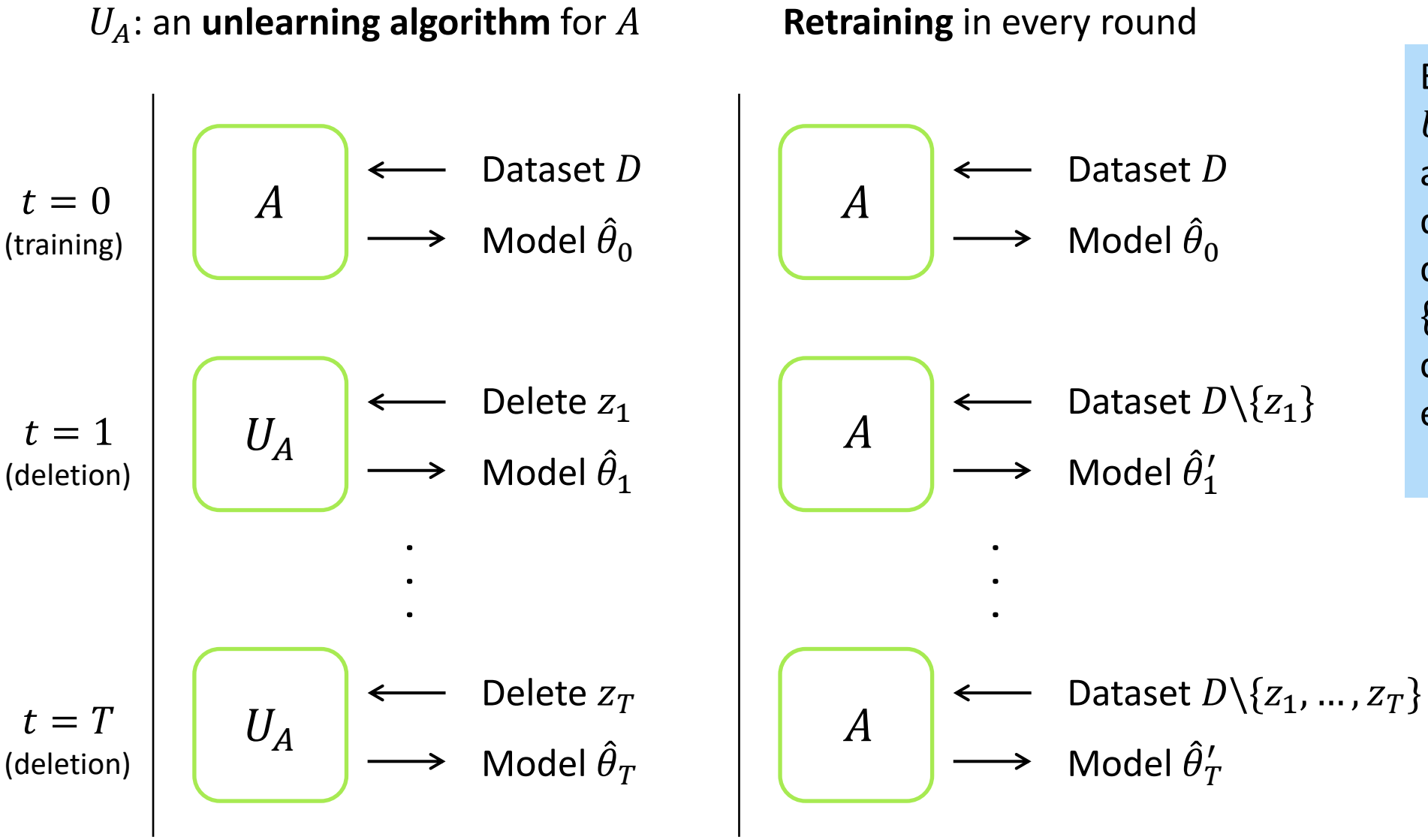
Reasons for & Main Purpose of Machine Unlearning

- Reasons:
  - Security: after removing adversarial data
  - Privacy: "right to be forgotten" – not only data itself, but also its impacts on the model
  - Usability: for a user
  - Fidelity: bias on African-American offenders
- All these reasons leads to a necessity to design a solution to data deletion.
- The ideal (yet expensive) solution is to update the database and **retrain** all models when a deletion request arrives.
- But retraining requires enormous computation: large models and frequent deletion requests.

Main purpose of *Machine Unlearning (Data Deletion)*:

Design **fast unlearning (deletion) algorithms** that produce output models that are statistically **indistinguishable** from the models that would have arisen from **retraining**.

Machine Unlearning



Exact Unlearning:
U_A is an unlearning algorithm for A if for all datasets D, and all deletion sequences $\{z_1, \ldots, z_T\}$, the following condition holds. For every deletion step t,

$$\hat{\theta}_t =_d \hat{\theta}'_t$$

Approximate Unlearning

Definition of Approximate Unlearning (Ginart et al. 2019):
We say "U_A is an (α, β)-**unlearning** algorithm for A" if for all datasets D and all deletion sequences $\{z_1, z_2, \dots, z_T\}$, the following condition holds: for every deletion step t,

$$\forall E, \Pr\left[\hat{\theta}_t \in E\right] \leq e^{\alpha} \cdot \Pr\left[\hat{\theta}'_t \in E\right] + \beta$$

Models output by the unlearning algorithm U_A

Models retrained using A

The smaller (α, β) are, the stronger unlearning guarantees will become.

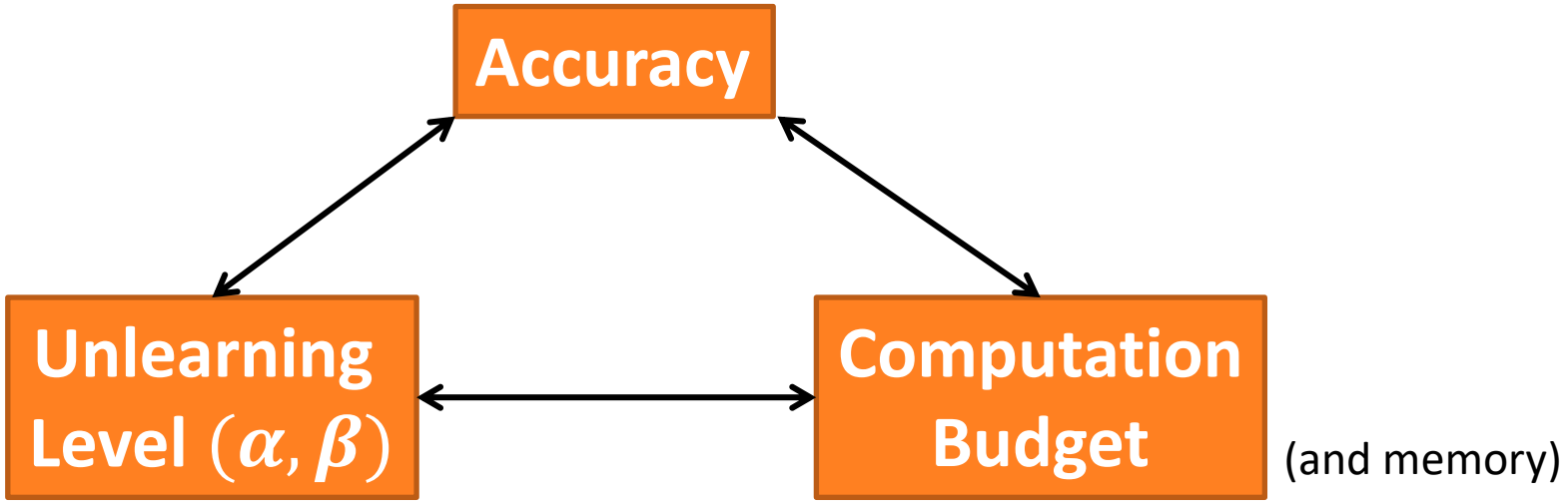
Differential Privacy & Machine Unlearning

- Machine unlearning uses the same metric with differential privacy for distributional closeness, but …
  - DP compares the **same** algorithm run on **different** datasets
  - Machine unlearning compares **different** algorithms run on the **same** dataset
  - If A is differentially private for any data, then it does not learn anything from the data. In other words, differential privacy is a very strong condition, and most differentially private models suffer a significant loss in accuracy even for large ϵ.
- But DP tools are useful to machine unlearning
  - Noise addition converts distance in parameter space to distance in distribution
  - DP reduces dependencies introduced via **adaptivity**
  - …

the data to be deleted depends on the current unlearned model

Tradeoffs

Goal: given desired **unlearning** level (α, β), and **computation** budget, design **accurate** unlearning algorithms.



Boundary Conditions:

- **Retraining**: high accuracy, optimal $(0,0)$-unlearning, *computationally expensive*
- **Not Unlearning**: optimal accuracy, *poor unlearning performance*, no computation
- **Completely DP Models**: *low accuracy*, optimal $(0,0)$-unlearning , no computation

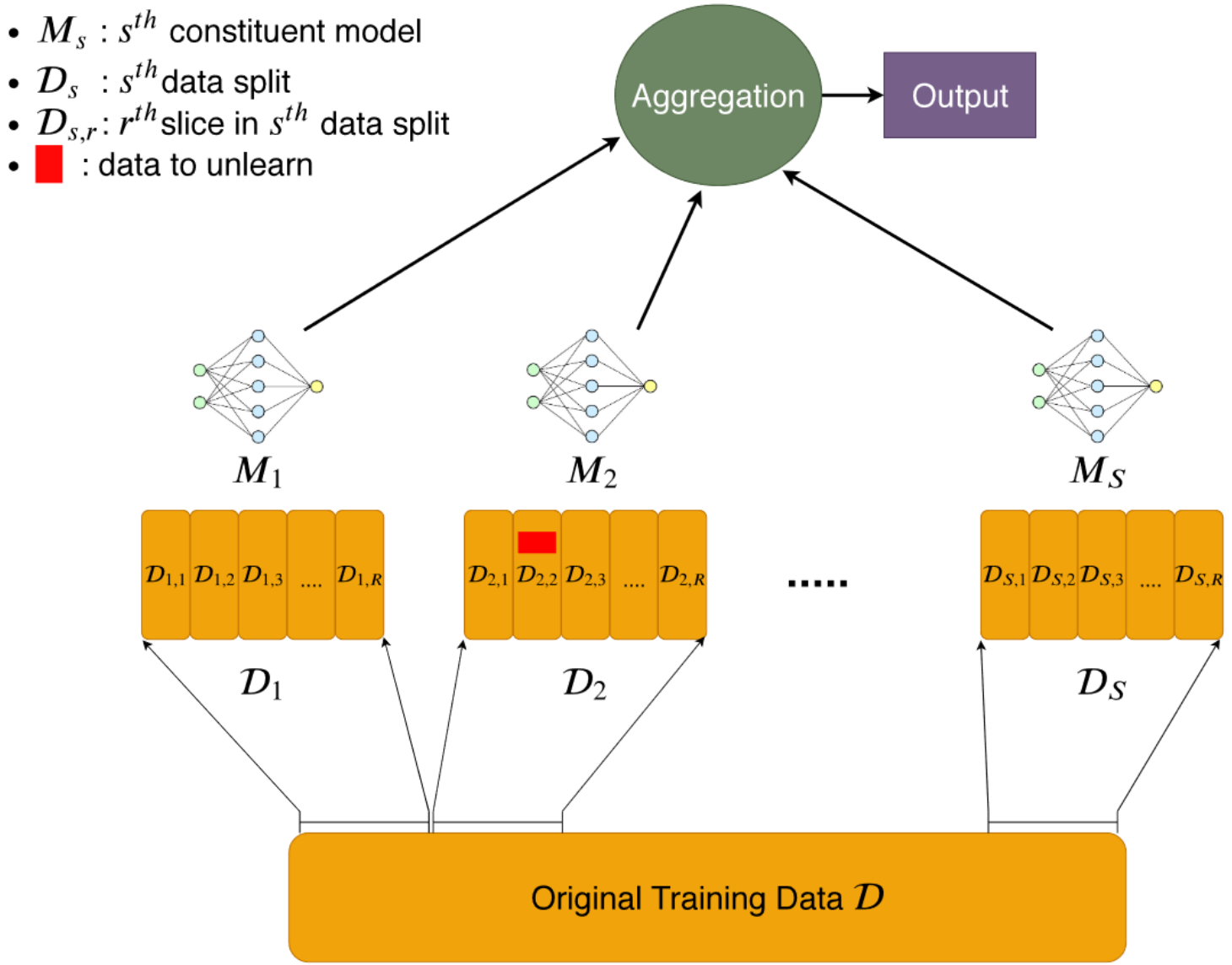
Evaluation Metrics

Suppose $h^U = U(A(D), D, D_U), h^* = U^*(A(D), D, D_U)$

- $Efficiency(h^U) \coloneqq \frac{time\ taken\ to\ obtain\ h^*}{time\ taken\ to\ obtain\ h^U}$
- $Effectiveness(h^U) \coloneqq \left|\mathcal{M}_{test}^U - \mathcal{M}_{test}^*\right|$ ($\mathcal{M}$ is another performance metric)
- $Consistency$ (correctness guarantee)
  - $Consistency_\theta\left(h_\theta^U\right) \coloneqq \|\theta^U - \theta^*\|_2$
  - $Consistency_y(h^U) \coloneqq \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \mathbf{1}_{y_{pred,i}^* = y_{pred,i}^U}$
  - $Consistency_{KL}(h^U) \coloneqq \mathbb{E}_x[KL(\Pr[h^U(x)] \,||\, \Pr[h^*(x)])]$
- $Certifiability$ (security guarantee)
  - $Certifiability(h^U) \coloneqq \frac{|\mathcal{M}_u^U - \mathcal{M}_u^*|}{|\mathcal{M}_u^U| + |\mathcal{M}_u^*|}$
  - $Certifiability(h^U) \coloneqq I(D_U; h^U)$

SISA (SSP 2021)

- Sharded
- Isolated
- Sliced
- Aggregated



Descent-to-Delete (ALT 2021)

Algorithm 8 Perfect ith unlearning for basic perturbed gradient descent, (Neel et al., 2020).

Input: published model parameters $\tilde{\theta}_{i-1}$, dataset D_{i-1}, update data point $\mathbf{z}_{i-1}$, number of iterations T_i, noise parameter $\sigma > 0$.
Output: published unlearned parameters $\tilde{\theta}_i$.

procedure PGDUNLEARN($\tilde{\theta}_{i-1}$, D_{i-1}, $\mathbf{z}_{i-1}$; T_i, σ)
  initialise $\theta'_0 \leftarrow \tilde{\theta}_{i-1}$
  $D_i \leftarrow D_{i-1} \setminus \{\mathbf{z}_{i-1}\}$
  for $t = 1; t \leq T_i; t++$ **do**
    $\theta'_t \leftarrow \text{Proj}_\Theta(\theta'_{t-1} - \eta_t \nabla L(\theta'_{t-1}, D_i))$
  end for
  $\hat{\theta}_i \leftarrow \theta'_{T_i}$
  draw $Z \sim \mathcal{N}(0, \sigma^2 I_p)$
  return $\tilde{\theta}_i = \hat{\theta}_i + Z$
end procedure

$$\text{Proj}_\Theta(\theta) = \arg\min_{\theta' \in \Theta} \|\theta - \theta'\|_2$$

Fisher (CVPR 2020)

$$\theta := \mathrm{SGD}(L(\theta_{\mathrm{init}}, D)) + \sigma F^{-\frac{1}{4}}\mathbf{b},$$

$$\theta^{\mathcal{U}} := \underbrace{\theta - F^{-1}\Delta_{\mathrm{rem}}}_{\text{Newton step}} + \underbrace{\sigma F^{-1/4}\mathbf{b}}_{\text{noise injection}},$$

$$\Delta_{\mathrm{rem}} := \nabla L(\theta, D \setminus D_u),$$

F: The Fisher Information matrix, which is used as an approximation to the Hessian as the Fisher matrix is less expensive to compute.

The authors prove that such a batch unlearning algorithm can ensure that

$$I(D_u; U(h_\theta, D, D_u)) = 0$$

Fisher (CVPR 2020)

Algorithm 5 Fisher removal mechanism, (Golatkar et al., 2019; Mahadevan and Mathioudakis, 2021).

Input: trained model parameters θ, training dataset D, subset of data to be removed D_u, noise parameter σ, mini-batch size m'.
Output: unlearned model parameters $\theta^{\mathcal{U}}$.

procedure FISHERUNLEARN(θ, D, D_u; σ, m')
  assign the number of batches $s \leftarrow \lceil \frac{m}{m'} \rceil$ (m is the number of samples in D_u)
  split D_u into s mini-batches D_u^1, D_u^2,...,D_u^s, each of size m'
  initialise $D' \leftarrow D$
  initialise $\theta^{\mathcal{U}} \leftarrow \theta$
  for $i = 1; i \leq s;\ i++$ **do**
    $D' \leftarrow D' \setminus D_u^i$
    $\Delta \leftarrow \nabla L(\theta^{\mathcal{U}}, D')$
    $F \leftarrow$ compute Fisher Information Matrix of L and D', (Golatkar et al., 2019, Eq. (8))
    $\theta^{\mathcal{U}} \leftarrow \theta^{\mathcal{U}} - F^{-1}\Delta$
    if $\sigma > 0$ **then**
      draw $\mathbf{b} \sim \mathcal{N}(0,1)^p$
      $\theta^{\mathcal{U}} \leftarrow \theta^{\mathcal{U}} + \sigma F^{-1/4}\mathbf{b}$
    end if
  end for
  return $\theta^{\mathcal{U}}$
end procedure

Influence (ICML 2020)

$$L_\sigma(\theta, D) := L(\theta, D) + \frac{\sigma \mathbf{b}^T \theta}{|D|}$$

$$\theta^{\mathcal{U}} := \theta + H^{-1} \Delta_u$$

$$H := \nabla^2 L(\theta, D \setminus D_u) \qquad \Delta_u := \nabla L(\theta, D_u)$$

Influence (ICML 2020)

Algorithm 6 Influence removal mechanism, (Mahadevan and Mathioudakis, 2021).

Input: trained model parameters θ, original train dataset D, subset of data to be deleted D_u, mini-batch size m'.

Output: unlearned model parameters $\theta^{\mathcal{U}}$.

procedure INFLUENCEUNLEARN(θ, D, D_u; m')
  assign the number of batches $s \leftarrow \lceil \frac{m}{m'} \rceil$ (m is the number of samples in D_u)
  split D_u into s mini-batches D_u^1, D_u^2,...,D_u^s, each of size m'
  initialise $D' \leftarrow D$
  initialise $\theta^{\mathcal{U}} \leftarrow \theta$
  for $i = 1; i \leq s; i++$ **do**
    $D' \leftarrow D' \setminus D_u^i$
    $\Delta_{m'} \leftarrow \nabla L\left(\theta^{\mathcal{U}}, D_u^i\right)$
    $H \leftarrow \nabla^2 L\left(\theta^{\mathcal{U}}, D'\right)$
    $\theta^{\mathcal{U}} \leftarrow \theta^{\mathcal{U}} + H^{-1}\Delta_{m'}$
  end for
  return $\theta^{\mathcal{U}}$
end procedure

DeltaGrad (ICML 2020)

- At each step of training, t, the model parameters $\{\theta_0, \theta_1, \dots, \theta_t\}$ and the gradients of the loss function $\{\nabla L(\theta_0), \nabla L(\theta_1), \dots, \nabla L(\theta_t)\}$ are saved.
- Suppose $D_u = \{z_i | \, i \in M\} \subseteq D$, $m = |M|$.

$$\theta_{t+1}^{\mathcal{U}} \leftarrow \theta_t^{\mathcal{U}} - \frac{\eta_t}{n-m}\left[n\nabla L(\theta_t^{\mathcal{U}}) - \sum_{i \in M} \nabla L_i(\theta_t^{\mathcal{U}})\right]$$

1. Use the Cauchy mean-value theorem in terms of an integrated Hessian H_t
2. Use the L-BFGS algorithm to approximate the vector product $H_t \cdot \boldsymbol{v}$ as a quasi-Hessian product $B_t \cdot \boldsymbol{v}$

$$\theta_{t+1}^{\mathcal{U}} \leftarrow \theta_t^{\mathcal{U}} - \frac{\eta_t}{n-m}\left[n(\nabla L(\theta_t) + B_t \cdot (\theta_t^{\mathcal{U}} - \theta_t)) - \sum_{i \in M} \nabla L_i(\theta_t^{\mathcal{U}})\right]$$

DeltaGrad (ICML 2020)

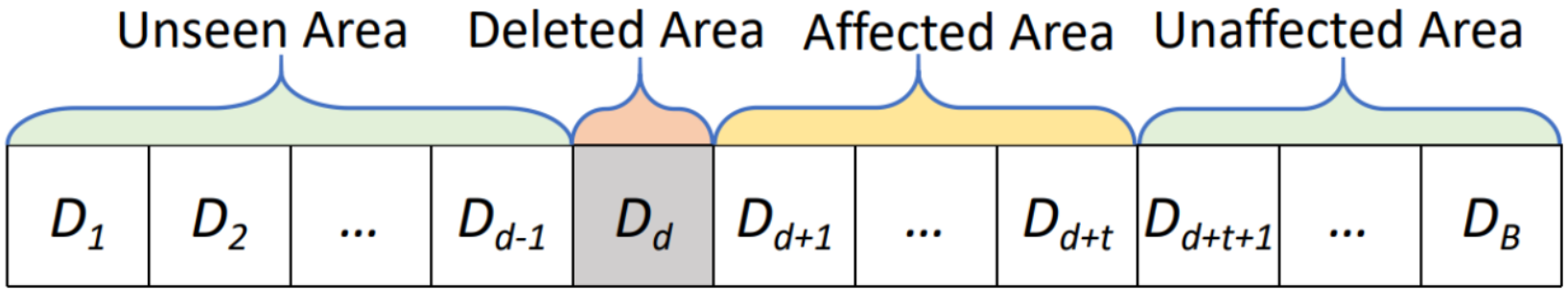
Algorithm 7 DeltaGrad removal mechanism, (Wu et al, 2020)

Input: model training parameters $\boldsymbol{\theta} := \{\theta_0, \theta_1, ...\theta_t\}$, training data D, indices of removed training samples M, stored training gradients $\nabla L(\boldsymbol{\theta}) := \{\nabla L(\theta_0), \nabla L(\theta_1), ...\nabla L(\theta_t)\}$, period T_0, total iteration number T, history size k, burn-in iteration number j_0, learning rate η_t.
Output: unlearned model parameters $\theta^{\mathcal{U}} = \theta_T^{\mathcal{U}}$.

procedure DeltaGradUnlearn($\boldsymbol{\theta}$, D, M; $\nabla L(\boldsymbol{\theta})$, T_0, T, k, j_0, η_t)
  initialise $\theta_0^{\mathcal{U}} \leftarrow \theta_0$
  initialise an array $\Delta G \leftarrow []$
  initialise an array $\Delta\Theta \leftarrow []$
  $\ell \leftarrow 0$
  for $t = 0; t \leq T; ++$ **do**
    if $[(t_0 - j_0) \pmod{T_0} == 0]$ *or* $t \leq j_0$ **then**
      compute $\nabla L(\theta_t^{\mathcal{U}})$ exactly
      compute $\nabla L(\theta_t^{\mathcal{U}}) - \nabla L(\theta_t)$, using the cached gradient $\nabla L(\theta_t)$
      $\Delta G[\ell] \leftarrow \nabla L(\theta_t^{\mathcal{U}}) - \nabla L(\theta_t)$
      $\Delta\Theta[\ell] \leftarrow \theta_t^{\mathcal{U}} - \theta_t$
      $\ell \leftarrow \ell + 1$
      compute $\theta_{t+1}^{\mathcal{U}}$ by using exact GD update
    else
      $B_{j_k} \leftarrow$ L-BFGS$(\Delta G[-k:], \Delta\Theta[-k:])$
      $\nabla L(\theta_t^{\mathcal{U}}) \leftarrow \nabla L(\theta_t^{\mathcal{U}}) + B_{j_k}(\theta_t^{\mathcal{U}} - \theta_t)$ approximate the gradient
      compute $\theta_{t+1}^{\mathcal{U}}$ via the modified gradient formula, Eq. (5.9), using approximated $\nabla L(\theta_t^{\mathcal{U}})$
    end if
  end for
  return $\theta_t^{\mathcal{U}}$
end procedure

DeepObliviate

The method further improves on the slicing component of SISA by using the so-called **temporal residual memory** to identify which intermediate models need to be retrained, adding approximation into the process.



$$h^{\mathcal{U}} = \textsc{Train}(\{D'_d, D_{d+1}, \ldots, D_{d+t}\}|h_{d-1}) \oplus (h_B \ominus h_{d+t})$$

$$D'_d = D_d \setminus \{\mathbf{z}\}$$

DeepObliviate

The value of t is determined by the temporal residual memory. The authors define the temporal residual memory as the ℓ^1 distance (or Manhattan distance) between the influence of the deleted data $\boldsymbol{z}$ on successive models when $\boldsymbol{z}$ is included in the training and when it is not. Formally, the temporal residual memory Δt at step t is:

$$\Delta t \coloneqq \left\| I(D_{d+t} | h_{d+t-1}) - I\left(D_{d+t} \middle| h^{U}_{d+t-1}\right) \right\|_1,$$

where $I(D_i | h_{i-1}) \coloneqq h_i \ominus h_{i-1}$.

The authors use detrended fluctuation analysis (DFA) to eliminate noise in Δ and systematically determine whether Δ has stabilized. DFA is used to determine the statistical self-affinity of a time-series signal by fitting Δt with a decaying power-law function, whose derivative is easily-computable and can be used to determine stationarity.

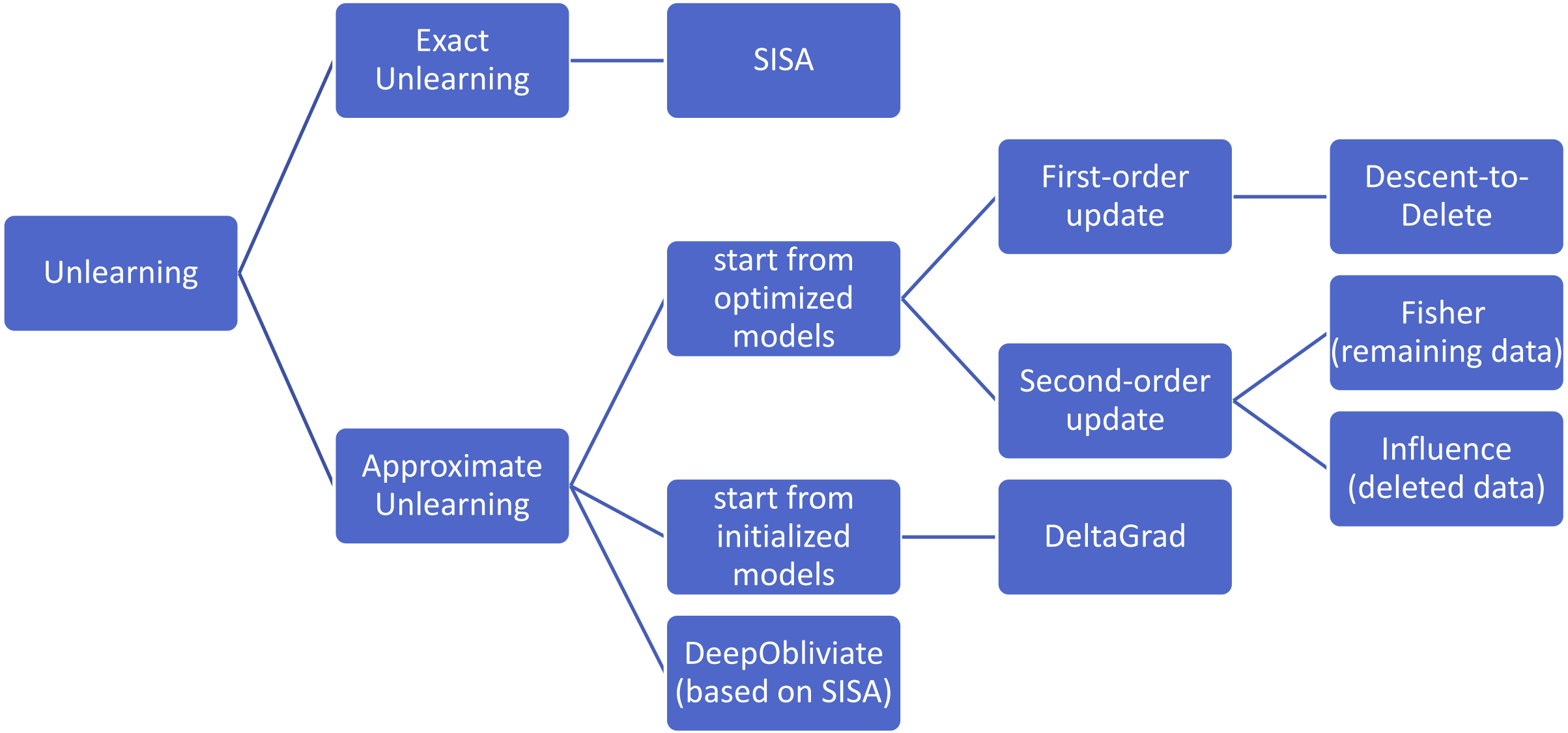
DeepObliviate

Algorithm 9 Unlearning with DeepObliviate, (He et al., 2021).

Input: parameters for intermediary trained models $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_B\}$, training data $D = D_1 \cup \cdots \cup D_B$, data point to be removed $\mathbf{z} \in D_d$, stationarity hyperparameter ε.
Output: unlearned model $h^{\mathcal{U}}$.

procedure DEEPOBLIVIATEUNLEARN($\boldsymbol{\theta}$, D, $\mathbf{z}$; ε)
  initialise $h^{\mathcal{U}} \leftarrow h_{d-1}$
  initialise $\theta^{\mathcal{U}}_{d-1} \leftarrow \theta_{d-1}$
  $D_d \leftarrow D_d \setminus \{\mathbf{z}\}$
  for $t = 0; t \leq B - d; t++$ **do**
    $h^{\mathcal{U}} \leftarrow \text{TRAIN}(D_{d+t} \mid h^{\mathcal{U}})$
    $\theta^{\mathcal{U}}_{d+t} \leftarrow h^{\mathcal{U}}$ get parameters from $h^{\mathcal{U}}$
    compute temporal influence $V_{d+t} \leftarrow \theta_{d+t} - \theta_{d+t-1}$ of D_{d+t} on θ_{d+t-1}
    compute temporal influence $V^{\mathcal{U}}_{d+t} \leftarrow \theta^{\mathcal{U}}_{d+t} - \theta^{\mathcal{U}}_{d+t-1}$ of D_{d+t} on $\theta^{\mathcal{U}}_{d+t-1}$
    compute temporal residual memory $\Delta_{d+t} \leftarrow ||V_{d+t} - V^{\mathcal{U}}_{d+t}||_1$
    compute power-law exponent using DFA $\alpha \leftarrow \text{DFA}(\{\Delta_d, \Delta_{d+1}, \ldots, \Delta_{d+t}\})$
    $Y(x) := ax^{-\alpha} + b$
    $f(x) := \partial Y(x)/\partial x = a \cdot (-\alpha) \cdot x^{-\alpha-1}$
    $a, b \leftarrow \arg\min_{a,b}(Y(x) - \Delta_x)$ using least squares, $x \in \{d, \ldots, d+t\}$
    $g \leftarrow f(d+t)$
    if $|g| < \varepsilon$ **then**
      break
    end if
  end for
  $h^{\mathcal{U}} \leftarrow h^{\mathcal{U}} \oplus (h_B \ominus h_{d+t})$
  $\{\theta_d, \ldots, \theta_{d+t}\} \leftarrow \{\theta^{\mathcal{U}}_d, \ldots, \theta^{\mathcal{U}}_{d+t}\}$
  Return $h^{\mathcal{U}}$
end procedure

Comparison


Unlearning
Exact Unlearning
SISA
Approximate Unlearning
start from optimized models
First-order update
Descent-to-Delete
Second-order update
Fisher (remaining data)
Influence (deleted data)
start from initialized models
DeltaGrad
DeepObliviate (based on SISA)

Comparison

Unlearning Type				
Method	**Applicability**	**Properties**	**Certificate of Unlearning**	
SISA	Incrementally trained models[a]	Exact Weak[b]	Exact	
DaRE	DaRE trees DaRE RF	Exact	Exact	
Fisher	LQ loss function	Approximate	3.3, 33.4 kNATs[c]	0.0%[d] ($m' = \lfloor m/8 \rfloor$) 0.26% ($m' = m$)
Influence	SC, BG, Lipschitz Hessian	Approximate Weak[e]	(ϵ, δ)-certified	0.1%[d] ($m' = \lfloor m/8 \rfloor$) 1.1% ($m' = m$)
DeltaGrad	SC, smooth loss, BG, Lipschitz Hessian, SI	Approximate Strong[f]	N/A[g]	
Descent-to-Delete	SC, smooth (Bounded, Lipschitz Hessian)[h]	Approximate Strong[i]	(ϵ, δ)-certified	
DeepObliviate	Any deep-learning model	Approximate Strong	6.39% ($\varepsilon = 0.05$) 8.28% ($\varepsilon = 0.1$)[j]	

Comparison

Performance								
Method		Efficiency			Effectiveness	Consistency		
		min	g. mean	max		min	g. mean	max
SISA[a]	Bourtoule et al. (2021)	1.36×	2.49×	4.63×	< 2% (18.76%)	Exact		
	He et al. (2021)	14.0×	39.6×	75.0×	< 5% (15.1%)			
DaRE[b]	min d_{rmax}	10×	366×	9735×	$\leq 0.5\%$	Exact		
	max d_{rmax}	145×	1272×	35856×	$\leq 2.5\%$			
Fisher[c]	$m' = \lfloor m/8 \rfloor$	1.0×	3.3×	36.8×	$\approx 0.0\%$	N/A		
	$m' = m$	1.5×	13.0×	282.4×	$< 0.2\%$			
Influence[c]	$m' = \lfloor m/8 \rfloor$	0.3×	5.9×	75.7×	$< 0.55\%$	N/A		
	$m' = m$	2.5×	38.2×	215.1×	$< 0.57\%$			
DeltaGrad[d]		1.6×	2.7×	6.5×	$\approx 0.0\%$	1.7×10^{-6}	1.1×10^{-5}	1.4×10^{-4}
Descent-to-Delete[e]	PGD	$1 + \mathcal{I}^{-1} \log(\epsilon n/\sqrt{p})$			N/A	$p/(e^{\mathcal{I}}\epsilon^2 n^2)$		
	sRPGD	$\mathcal{I}^{-\frac{3}{5}}(\epsilon n/\sqrt{p})^{\frac{2}{5}}$				$\left(\frac{\sqrt{p}}{\epsilon n \mathcal{I}}\right)^{\frac{2}{5}}$		
	wRPGD	$\mathcal{I}^{-\frac{3}{4}}\sqrt{\epsilon n/\sqrt{p}}$				$\sqrt{\frac{\sqrt{p}}{\epsilon p \sqrt{\mathcal{I}}}}$		
	DPGD	$\min\{\mathcal{I}^{-1}\log n, n^{\frac{4-3\xi}{2}} + \mathcal{I}^{-1}\log(\epsilon n/\sqrt{p})\}$				$\frac{2\exp\left(-\mathcal{I}n^{\frac{4-3\xi}{2}}\right)}{\epsilon^2 n^2} + \frac{1}{n^\xi}$		
DeepObliviate[f]	min ε	6.17×	13.97×	45.45×	$< 0.18\%$	97.25	98.82	99.95
	$\varepsilon = 0.1$	9.26×	22.81×	66.67×	$< 0.35\%$	95.78	97.61	99.85