Continual Learning + Machine Unlearning

Pengxiang Wang

Peking University, School of Mathematical Sciences

University of Bristol, School of Engineering Mathematics and Technology

2024-10-28

Machine Unlearning

Machine Unlearning

What is **machine unlearning**:

> ***Machine unlearning*** *is the process of deliberately removing specific training data from a machine learning model to ensure that the removed data no longer influences the model's predictions. It offers undo option of machine learning process.*

Data Deletion:

- Traditionally: delete from databases
- AI: delete both from back-end databases and from trained models

Machine Unlearning Motivation

Movitation from application side:

- **Privacy**:
  - Regulations: GDPR, CCPA, etc. when the user withdraw the consent, "the right to be forgotten"
  - Delete the requested data by users
- **Security**:
  - Adversarial attacks are possible to extract private information from the trained model. E.g., model inversion attacks, membership inference attacks, etc.
  - Delete the adversial data
- **Data Quality**:
  - Delete unwanted data such as outdated data, noisy data, biased data, etc.

Machine Unlearning Framework

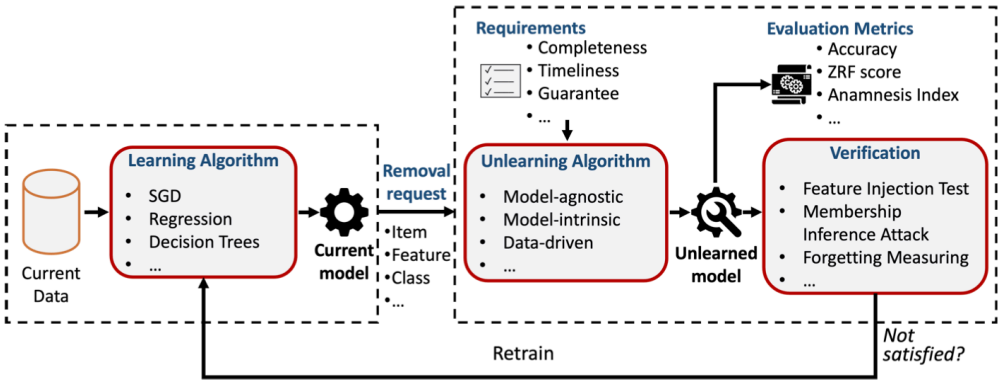

Figure 1: Machine Unlearning Framework

Formal Definition

Notations:

- Training data D, training algorithm A, model trained $A(D)$
- Unlearning data $D_f \subset D$, unlearning algorithm U
- Unlearned model $U(D, D_f, A(D))$

Objective:

- Unlearned model $U(D, D_f, A(D))$ is expected to be the same or similar to a retrained model $A(D \backslash D_f)$
- The similarity is measured by indistinguishability metrics
- Distance bewteen model parameters (l_2 distance), distributions (K-L divergence)...

Assumptions:

- The unlearning data are small compared to the training data
  - True in real-world applications
  - Otherwise, retrain can solve everything

Retraining

The problem makes unlearning difficult:

- Neural networks parameters do not tend to show any clear connection to the training data. AI models have to be considered as a whole
- Stochasticity and Incrementality of training
- Catastrophic Unlearning: the holistic character of neural networks can easily lead to excessive unlearning too much then reduce performance

Retraining:

- Delete target data and re-train the model with the rest of data from scratch
- A naive way, but not always feasible
- The only exact unlearning method, and achieves upper bound

The problem of retraining:

- Doesn't worth, computation cost
- Not always having aceess to all training data

Methodology

The forgotten set could be:

- Item removal: data points
- Class removal
- Feature removal

The unlearning process could be:

- Model-agnostic or model-intrinsic
- Data-driven approaches (most model-agnostic)

Method: SISA

SISA (Sharded, Isolated, Sliced, Aggregated), 2021:

- Item removal
- Isolate network into constituent networks, divide data into shards
- Build up correspondance bewteen divided network and data, trained correspondingly
- Unlearning: retraining the corresponding network of the data shard to be forgotten

Core idea: fractionizing the retraining process into smaller units, reduce the cost of full retraining

Method: SISA

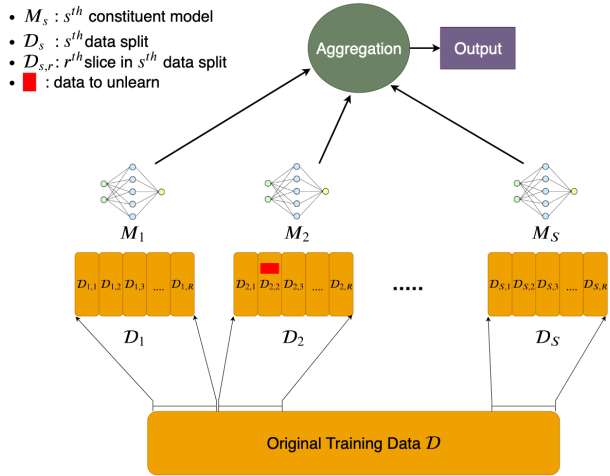

Figure 2: SISA

SISA: Technical Details

Q: How the divided networks predict?

A: Label-based majority vote: each constiuent network predicts the label, and the final label is determined by majority vote

Q: How model is trained from the original data?

A: A data batch is assigned to a shard, and the corresponding network is trained on the shard just like normal training. The constituent networks are isolated from each other, and the training process is independent.

Q: How to unlearn?

A: Delete the forgotten data from the corresponding shard, and retrain the corresponding network on the rest of the data.

To promote maximum efficiency, model screenshots are taken during training each data shard, unlearning can start from the point the forgotten data being trained, instead of from scratch.

Method: AmnesiacML

Amnesiac Machine Learning, 2021:

- Item removal
- Keep a record update on parameters (i.e. the step term $-lr \cdot g$ in gradient descent, the difference of parameters before and after updating) during training each batch of data
- Unlearning: simply substract the parameter update from the corresponding batch

The problem:

- The stochasticity and incrementality of the training process
- Storing the parameter updates is as expensive as storing the model itself

Some assumptions:

- The data holder is only concerned about potential removal of a subset of data
- Only need to keep the parameter updates from batches containing that data

Method: Error-Max Anti-sample generation

Error-Max Anti-Sample Generation, 2021:

- Class removal
- Generate noise data matrix specifically for the classes to be forgotten, by maximizing the model's error on these classes
  - $\arg\min_{\mathcal{N}} \mathbb{E}_{(\theta)} \left[-\mathcal{L}(f, y) + \lambda \left\| w_{\text{noise}} \right\| \right]$
  - $\mathcal{L}(f, y)$ is the classification loss corresponding to the class to unlearn
- Unlearning:
  - Impair: update the model with the generated noise matrix and a very high learning rate
  - Repair: continue training the model on data without the forgotten classes to recover the performance

Method: Linear Filteration

Linear Filtration for logit-based classifiers, 2022:

- Class removal
- Apply the filtration matrix F_z to directly modify the output weights W, producing new weights W_z
- Without additional learning steps

How to calculate the filtration matrix F_z (suppose to remove class 0):

- Expected model prediction: $A = [\mathbf{a}_0|\mathbf{a}1|\cdots|\mathbf{a}k-1]$
- Substitute $\mathbf{a}_0$ with arbitrary $z \in \mathbb{R}^{k-1}$: $B_z = [z|\mathbf{a}1|\cdots|\mathbf{a}k-1]$
- $F_z = B_z A^{-1}$

Continual Learning + Machine Unlearning

Why Continual Learning + Machine Unlearning?

In essence, continual learning is a non-stationary data stream.

What's the point?

- Provide unlearning options only for this scheme?

We have 2 proposed paradigms in the literature:

- Continual Learning and Private Unlearning (CLPU), 2022
- Learning with Selective Forgetting (LSF), 2021

CL+MU Paradigm: CLPU

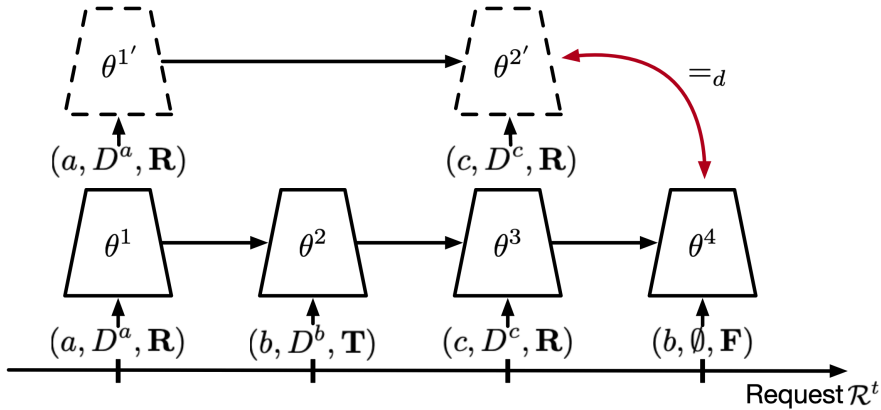

Figure 3: CLPU (Continual Learning and Private Unlearning)

CL+MU Paradigm: CLPU

3 learning instructions:

- **R** = permanent task never required to be forgotten in the future
- **T** = temporary task potentially required to be forgotten in the future
- **F** = task to be forgotten

Assumptions:

- Task removal
- The information if a task are required to be forgotten in the future has to be known a priori when trained the task
- The unlearning takes place according to the user's signal (learning instruction F)

Objectives:

- The model should perform well on the tasks that are not required to be forgotten (CL itself)
- Minimize the distance of model between that was trained without seeing forgotten tasks and that after unlearning

CLPU Solution: CLPU-DER++

A naive method: **CLPU-DER++**

- A main model for permenant tasks and task-specific models for temporary tasks
- Learning a permenant task (R): use DER++ (a replay-based method) to keep continual learning and prevent forgetting
- Learning a temporary task (T): train a task-specific model
- Temporary task to permanent task (R): merge the task-specific model into the main model, knowledge distillation
- Unlearning (F): retrain the corresponding network of the task to be forgotten

- Influenced by SISA to some degree (isolated networks) but a bit different
- Achieves exact unlearning at the expense of memory (linear with the number of temporary tasks)

CL+MU Paradigm: LSF

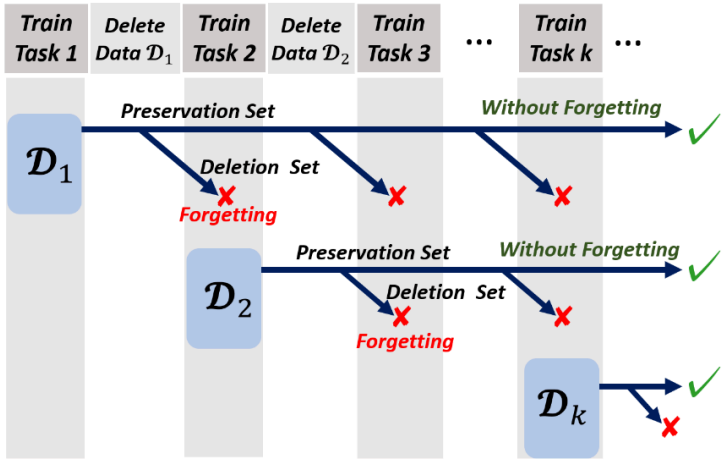

Figure 4: LSF (Learning with Selective Forgetting)

CL+MU Paradigm: LSF

Assumptions:

- Class removal in each task
- Unlearning automatically take place at specific timing (immediately after new task arrives), instead of user's signal
- The forgotten classes are assigned a priori, i.e. a preservation set (excluding all data from forgotten classes) is selected for each task beforehand

Objective:

- The forgotten classes should also be learned which means achieving the best performance, where the performance on both preserved and forgotten classes should be as good as possible (CL itself)
- After forgotten, the performance on the forgotten classes should be degraded as much as possible
- It is an objective aiming to forget itself, rather than get rid of the impact (nuances with usual unlearning)

Next Step

Thinking

- There are many potential ways to combine CL and MU together
- We have seen the unlearning under the continual learning framework, which is a special case of machine unlearning (in a non-stationary data stream)
- From my perspective, the unlearning should have benefits for continual learning itself (e.g., releasing network capacity)

The benefit of unlearning for continual learning itself?

- Forgetting used to be very passive which is catastrophic forgetting that CL algorithms are trying to avoid. It turns proactive in unlearning
- Double win for user and model:
  - For user: unlearning let the user to have the right to forgetting knowledge
  - For model: a manual approach to balance stability and plasticity, and solve the network capacity problem at the same time

Thinking

Solve the network capacity problem?

- Network capacity problem: it's impossible to consolidate infinite tasks in a finite network capacity
- Unlearning potentially makes infinite tasks finite (depend on user's unlearning needs) thus fundamentally solves capacity problem.
- It is hardly shown in current CL+MU papers. A guess: the network capacity never reach the limit, thus no need to release capacity.
  - For example, CLPU-DER++ allows to expand the network capacity when needed, the expanded capacity can be used in the first place instead of released capacity by unlearning
  - If unlearning has the effect of releasing capacity and make the performance better, it should be put in the methods that do not allow expansion, e.g., HAT

Next Step

Figure out our goal of unlearning: is it for unlearning itself, or for continual learning?

For unlearning itself:

- Follow one of the 2 proposed paradigms we've met
  - They seem to be narrowed down and specific which I don't see much potential in.
  - LSF gives no choice of timing to forget; CLPU distinguishes permanent and temporary tasks
  - There metrics are complicated and not general
- Design our new CL MU paradigm

Next Step

For continual learning itself:

- Could be for the benefit of releasing network capacity (which fits in my PhD thesis).
- In the context of fixed networks to make the benefit explicit
- Compare CL metrics: for example, test task a, b, c (without unlearning) and task a, c (unlearning b) to see the difference
- Choose a suitable archietcture-based method of fix network. Their network isolation ideas make it naturally easy for unlearning

Before any of them, investigate MU methods

Thank You

Thank you for your attention!

Please feel free to ask any questions or reach out to me at:

wangpengxiang@stu.pku.edu.cn