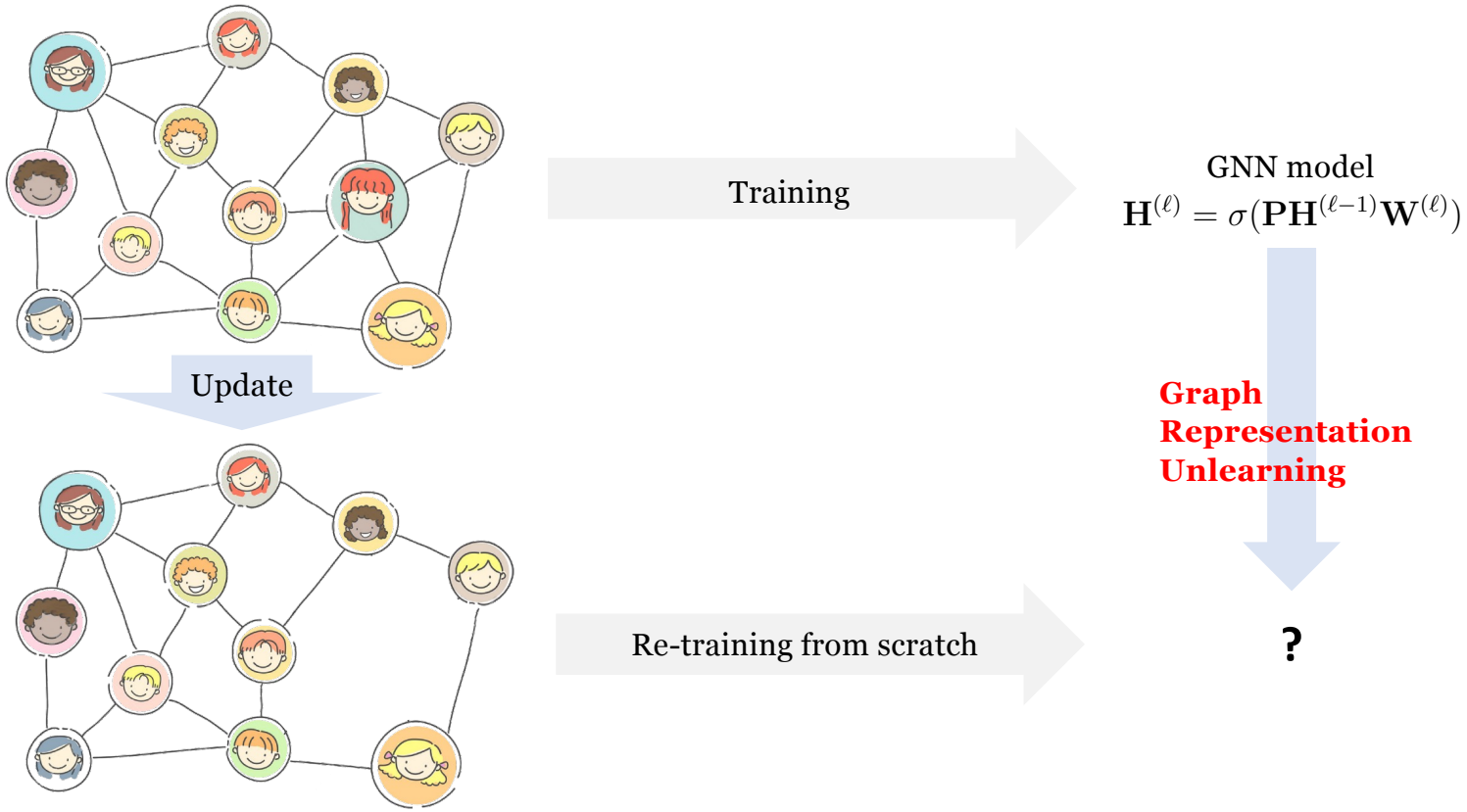
Efficient Graph Representation Unlearning Approach: GraphEditor and Projector

Weilin Cong
congweilin95@gmail.com

June 11, 2022

Motivation

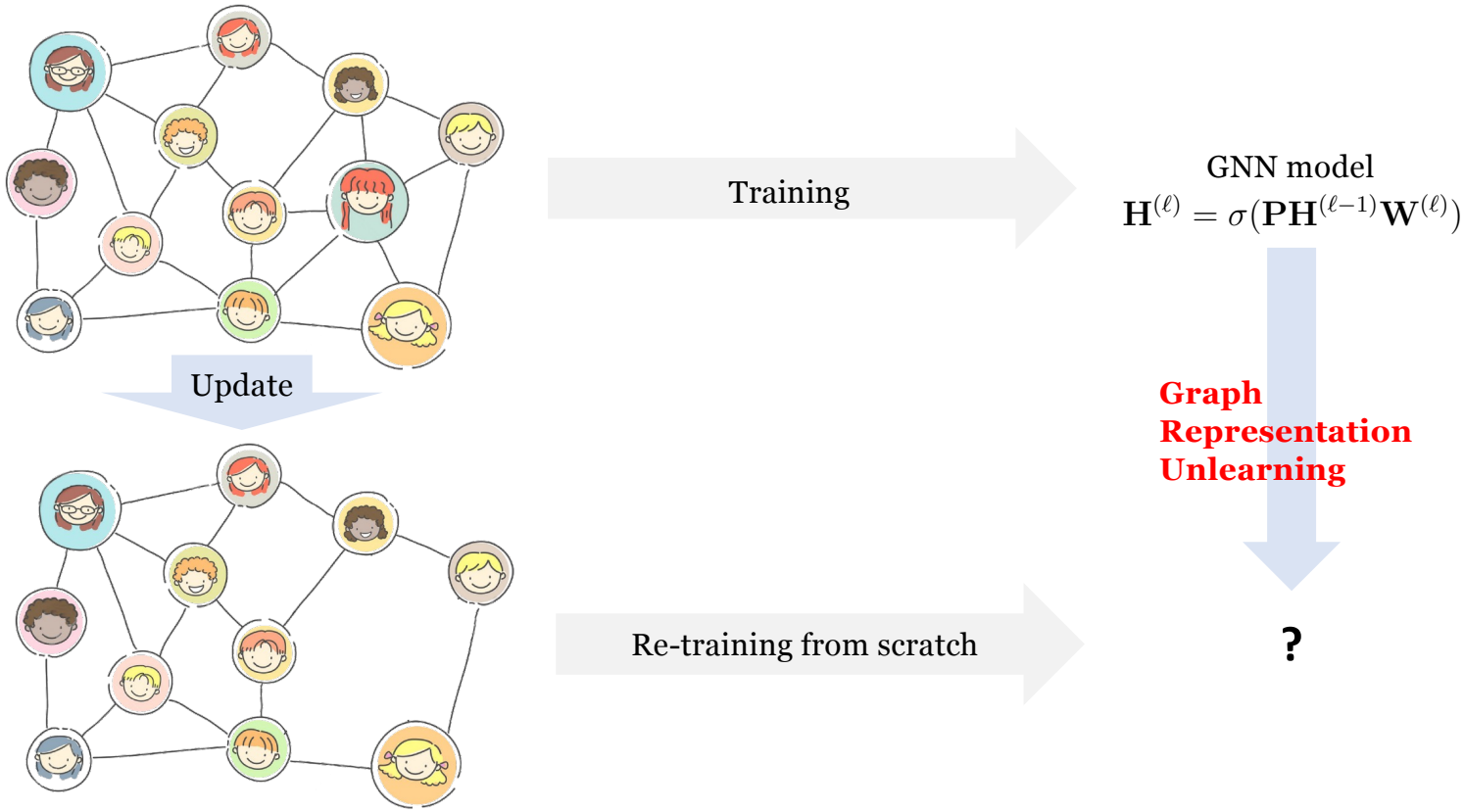
Due to privacy concerns, removing the effect of a specific node from the pre-trained graph representation learning model has attract much attention.



[1] Figure edited from freecodecamp.org

Motivation

- Retraining from scratch $\Rightarrow$ computation prohibitive + infeasible
- Efficient unlearning: exact unlearning + approximate unlearning



[1] Figure edited from freecodecamp.org

Overview on machine unlearning and graph application

- Exact unlearning:
  - SISA[1] and GraphEraser[2]: Randomly split the original dataset into multiple disjoint shards and re-training each shard model independently. Similar to federated learning.
  - Limitations: Each shard model's performance is poor due to lack to training data and data heterogeneity.
- Approximate unlearning:
  - Influence[3] and Fisher[4]: Using second-order gradient update to minimize the objectvie after data deletion.
  - Approximate unlearning by its nature, lack of guarantee on whether all information associated with the deleted data are eliminated, which could be validated by experiments.

---

[1] Bourtoule et al., "Machine unlearning".

[2] Chen et al., "Graph Unlearning".

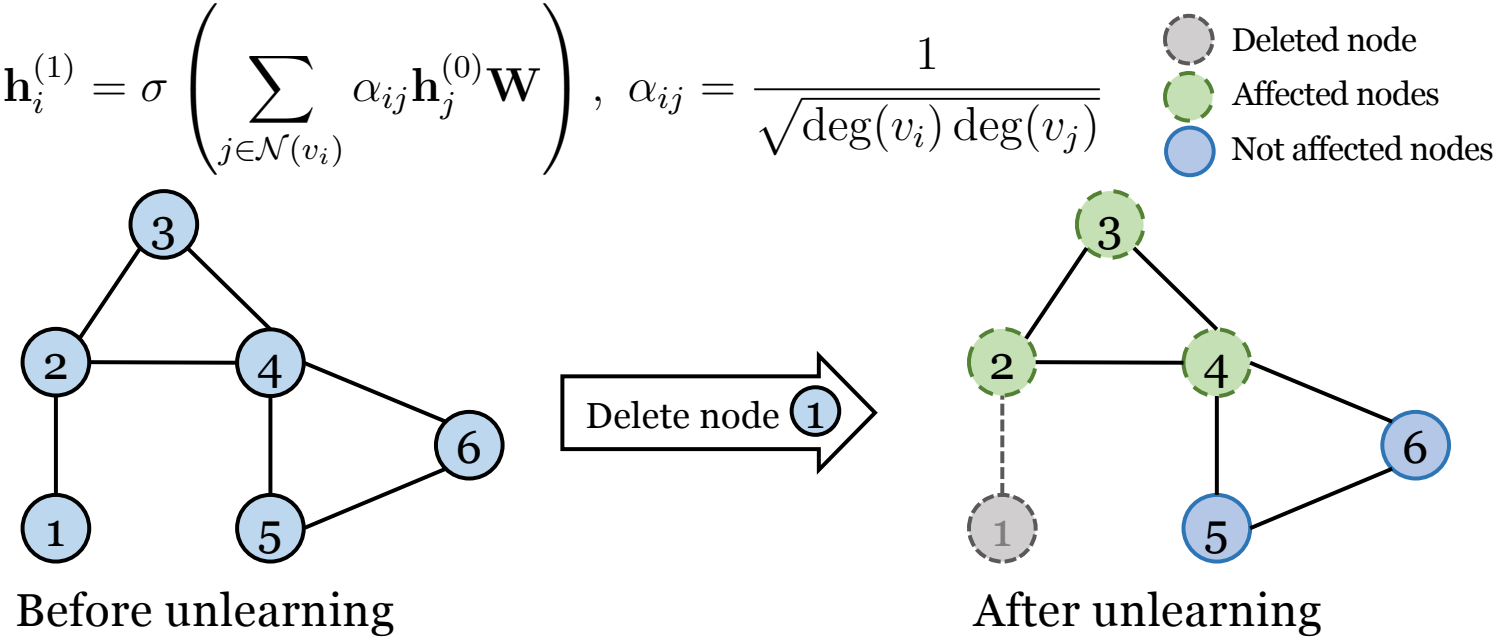
[3] Guo et al., "Certified Data Removal from Machine Learning Models".

[4] Golatkar, Achille, and Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks".

Graph representation unleanring is more challenging

- In graph representation unlearning, we not only need to remove the information related to the deleted nodes, but also need to update its impact on neighboring remaining nodes of multi-hops.

$$\mathbf{h}_i^{(1)} = \sigma\left(\sum_{j\in\mathcal{N}(v_i)} \alpha_{ij}\mathbf{h}_j^{(0)}\mathbf{W}\right), \quad \alpha_{ij} = \frac{1}{\sqrt{\deg(v_i)\deg(v_j)}}$$



- Since most of the existing unlearning methods only support data deletion, extending their application to graphs is non-trivial.

GraphEditor

GraphEditor supports

- ✓node deletion + edge deletion
- ✓node addition + edge addition
- ✓node feature update

GraphEditor requires

- × retraining from scratch,
- × all data presented during unlearning,
- ✓... but only applicable to linear GNNs[5] (e.g., APPNP, SGC)

GraphEditor is exact unlearning and has algorithmic level data removal guarantee.

[5]Most approximate unlearning methods require such linearity requirement. Their extension to non-linear models requires first pre-training a deep neural network as a feature extractor for the linear model, and then only unlearning the linear model without updating the feature extractor.

(Before unlearning) Learning via closed-form solution:

- Formulate ordinary GNN training as linear GNN[6] with Ridge regression as objective,

$$\mathcal{L}_{\text{Ridge}}(\mathbf{W}; \mathbf{X}, \mathbf{Y}) = \|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_{\text{F}}^2 + \lambda \|\mathbf{W}\|_{\text{F}}^2, \ \mathbf{X} = \mathbf{P}^{L}\mathbf{H}^{(0)},$$

  where node representation $\mathbf{X}$ is computed by applying L graph propagation matrices $\mathbf{P} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ on node features $\mathbf{H}^{(0)}$.

- This can be efficiently solved by closed-form solution:

$$\mathbf{W}_\star = \arg\min_{\mathbf{W}} \mathcal{L}_{\text{Ridge}}(\mathbf{W}; \mathbf{X}, \mathbf{Y}) = \underbrace{(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1}}_{\text{Denote as } \mathbf{S}_\star} \mathbf{X}^\top \mathbf{Y},$$

[6] Its non-linear extension will be introduced at the end of this talk.

GraphEditor

(Before unlearning) Learning via closed-form solution:

```
# (Before unlearning) Compute the closed-form solution
S, W = find_W(X, Y)

# (GraphEditor) Step 1: Delete information
S, W = remove_data(X[V_rm ∪ V_upd], Y[V_rm ∪ V_upd], S, W)

# (GraphEditor) Step 2: Update information
S, W = add_data(X̃[V_upd], Ỹ[V_upd], S, W)

# (Optional) Fine-tune W using cross-entropy loss
```

```
def find_W(X, Y, reg=0):
    XtX = X.T@X + reg*numpy.eye(X.shape[0])
    S = numpy.linalg.inv(XtX)
    Xty = X.T@Y
    W = S@Xty
    return S, W
```

(Unlearning) Step 1. Efficiently remove the effect of the deleted nodes on weight parameters:

- Let $\mathcal{V}_{rm} = \{v_i\}$ denote the set of node to remove
- Let $\mathcal{V}_{upd} = \{v_j \mid \text{SPD}(v_i, v_j) \leq 2L,\ \forall v_j \in \mathcal{V},\ \forall v_i \in \mathcal{V}_{\text{rm}}\}$ denote affected node set
- Given the initial solution $\mathbf{S}_\star$ and $\mathbf{W}_\star$, we first update the inversed correlation matrix as

$$\mathbf{S}_{\text{rm}} = \mathbf{S}_\star + \mathbf{S}_\star \mathbf{X}_{\text{rm}}^\top [\mathbf{I} - \mathbf{X}_{\text{rm}} \mathbf{S}_\star \mathbf{X}_{\text{rm}}^\top]^{-1} \mathbf{X}_{\text{rm}} \mathbf{S}_\star, \tag{1}$$

and update the optimal solution by

$$\mathbf{W}_{\text{rm}} = \mathbf{W}_\star - \mathbf{S}_\star \mathbf{X}_{\text{rm}}^\top [\mathbf{I} - \mathbf{X}_{\text{rm}} \mathbf{S}_\star \mathbf{X}_{\text{rm}}^\top]^{-1} (\mathbf{Y}_{\text{rm}} - \mathbf{X}_{\text{rm}} \mathbf{W}_\star). \tag{2}$$

GraphEditor

(Unlearning) Step 1. Efficiently remove the effect of the deleted nodes on weight parameters:

```
# (Before unlearning) Compute the closed-form solution
S, W = find_W(X, Y)

# (GraphEditor) Step 1: Delete information
S, W = remove_data(X[V_rm ∪ V_upd], Y[V_rm ∪ V_upd], S, W)

# (GraphEditor) Step 2: Update information
S, W = add_data(X̃[V_upd], Ỹ[V_upd], S, W)

# (Optional) Fine-tune W using cross-entropy loss
```

```
def remove_data(X, Y, S, W):
    I = numpy.eye(X.shape[0])
    A = S@X.T
    B = numpy.linalg.inv(I - X@S@X.T)
    C = Y - X@W
    D = X@S
    return S + A@B@D, W - A@B@C
```

(Unlearning) Step 2. Update the effect of the neighboring nodes of the deleted nodes on weight parameters:

- Let $\mathbf{X}_{\text{upd}} = \tilde{\mathbf{X}}[\mathcal{V}_{\text{upd}}]$ denote the subset of matrix $\tilde{\mathbf{X}}$ with row indexed by $\mathcal{V}_{\text{upd}}$.
- Let $\mathbf{Y}_{\text{upd}} = \tilde{\mathbf{Y}}[\mathcal{V}_{\text{upd}}]$ denote the subset of matrix $\tilde{\mathbf{Y}}$ with row indexed by $\mathcal{V}_{\text{upd}}$.
- Then, we update the inversed correlation matrix by

$$\mathbf{S}_{\text{upd}} = \mathbf{S}_{\text{rm}} - \mathbf{S}_{\text{rm}}\mathbf{X}_{\text{upd}}^{\top}[\mathbf{I} + \mathbf{X}_{\text{upd}}\mathbf{S}_{\text{rm}}\mathbf{X}_{\text{upd}}^{\top}]^{-1}\mathbf{X}_{\text{upd}}\mathbf{S}_{\text{rm}}, \tag{3}$$

and update the optimal solution by

$$\mathbf{W}_{\text{upd}} = \mathbf{W}_{\text{rm}} + \mathbf{S}_{\text{rm}}\mathbf{X}_{\text{upd}}^{\top}[\mathbf{I} + \mathbf{X}_{\text{upd}}\mathbf{S}_{\text{rm}}\mathbf{X}_{\text{upd}}^{\top}]^{-1}(\mathbf{Y}_{\text{upd}} - \mathbf{X}_{\text{upd}}\mathbf{W}_{\text{rm}}). \tag{4}$$

GraphEditor

(Unlearning) Step 2. Update the effect of the neighboring nodes of the deleted nodes on weight parameters:

```
# (Before unlearning) Compute the closed-form solution
S, W = find_W(X, Y)

# (GraphEditor) Step 1: Delete information
S, W = remove_data(X[V_rm ∪ V_upd], Y[V_rm ∪ V_upd], S, W)

# (GraphEditor) Step 2: Update information
S, W = add_data(X̃[V_upd], Ỹ[V_upd], S, W)

# (Optional) Fine-tune W using cross-entropy loss
```

```
def add_data(X, Y, S, W):
    I = numpy.eye(X.shape[0])
    A = S@X.T
    B = numpy.linalg.inv(I + X@S@X.T)
    C = Y - X@W
    D = X@S
    return S - A@B@D, W + A@B@C
```

Evaluation: delete data reply test

(1) add an extra-label category to all nodes and modify the label of each deleted node to this additional label category. (2) compare the number of deleted nodes that are predicted as the extra-label category before and after the unlearning process.

Table: Comparison on the accuracy (before parentheses), number of deleted nodes that are predicted as the extra-label category before and after unlearning (inside parentheses), and wall-clock time.

Method		OGB-Arxiv			OGB-Products		
		S=10	S=50	S=100	S=10	S=50	S=100
GraphEditor	Before	71.77% (70)	71.77% (70)	71.77% (70)	77.63% (83)	77.63% (83)	77.63% (83)
	After	71.78% (0)	71.78% (0)	71.78% (0)	77.63% (0)	77.63% (0)	77.63% (0)
	Time	10.8 s	10.9 s	11.9 s	46.6 s	76.9 s	108.3 s
GraphEraser	Before	69.91% (28)	69.91% (28)	69.91% (28)	63.27% (32)	63.27% (32)	63.27% (32)
	After	69.90% (0)	69.90% (0)	69.89% (0)	63.27% (0)	63.28% (0)	63.25% (0)
	Time	615.9 s	1,888.1 s	2,237.8 s	15,191.4 s	39,612.5 s	46,491.4 s
Influence	Before	72.99% (93)	72.99% (93)	72.99% (93)	78.05% (63)	78.05% (63)	78.05% (63)
	After	72.89% (53)	72.89% (53)	72.89% (53)	78.05% (19)	78.03% (19)	78.04% (19)
	Time	62.1 s	284.7 s	554.8 s	151.7 s	614.2 s	1,185.7 s
Fisher	Before	72.94% (94)	72.94% (94)	72.94% (94)	78.05% (63)	78.05% (63)	78.05% (63)
	After	72.73% (56)	72.70% (55)	72.69% (54)	77.87% (57)	77.86% (57)	77.76% (54)
	Time	77.1 s	364.4 s	703.5 s	185.3 s	791.8 s	1,528.6 s

Evaluation: compare to re-trained model

For $\mathcal{B} \in \{\mathcal{V}_{\text{rm}}, \mathcal{V}_{\text{test}}\}$ we compare the distance of final activations as $\mathbb{E}_{v_i \in \mathcal{B}}\left[\|\text{softmax}(\mathbf{x}_i\mathbf{W}^u) - \text{softmax}(\mathbf{x}_i\mathbf{W}^r)\|_2\right]$.

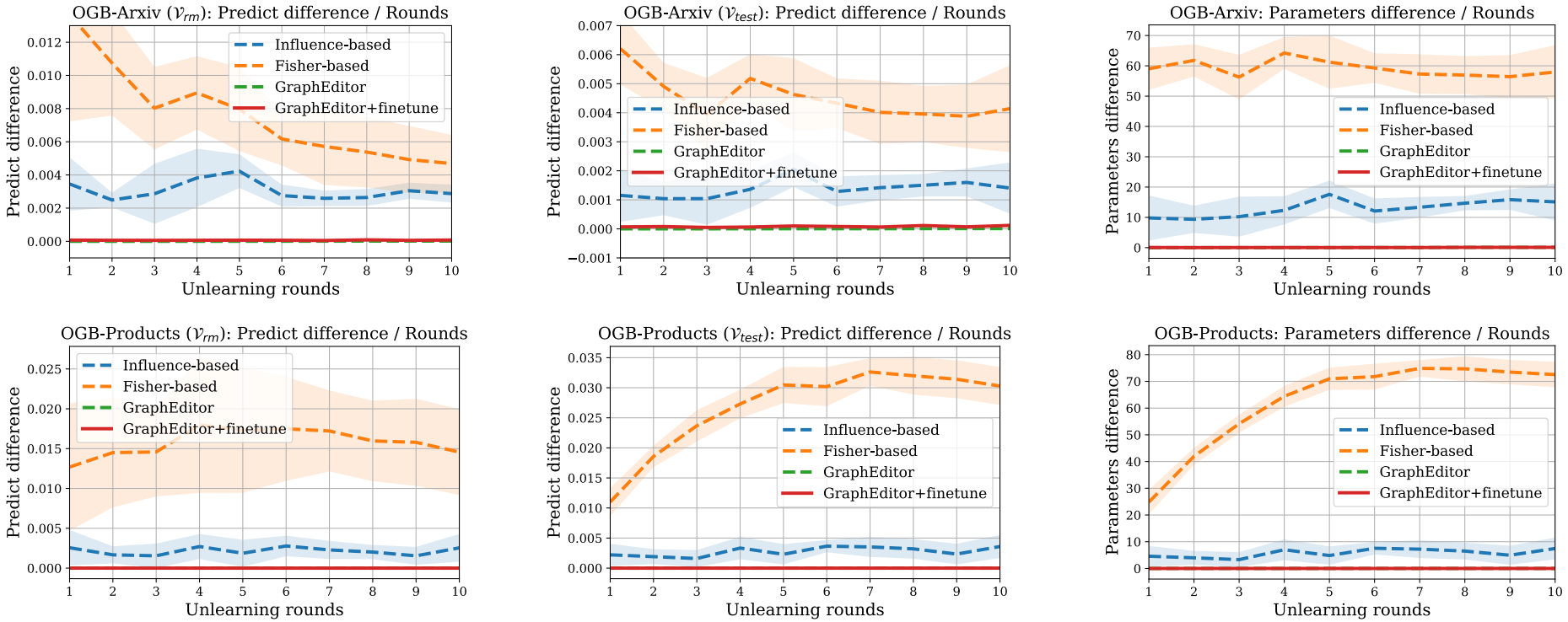


Figure: Comparison on the difference of final activation prediction on deleted nodes (1st column) and testing nodes (2nd column) and difference of weight parameters (3rd column).

Limitation of GraphEditor

Some limitations:

- We have to keep an record on the computation graph of each node, which requires many engineering effort and storage.
- Computation cost is cubic to the affected node size, which grows exponentially to the number of sampled neighbor.

To solve this ... we propose Projector.

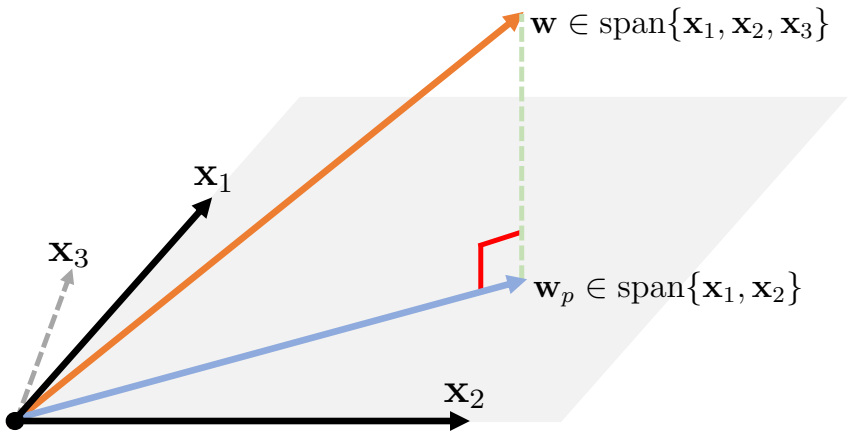


Figure: An illustration on the main idea of Projector. The original weight $\mathbf{w}$ exists inside the subspace defined by node feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. We can unlearn $\mathbf{x}_3$ and obtain the new weight $\mathbf{w}_p$ by projecting $\mathbf{w}$ on to the subspace defined without $\mathbf{x}_3$.

Key observation of PROJECTOR

Linear GNN is defined as

$$\begin{aligned} \min_{\mathbf{w}} F(\mathbf{w}) &= \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{\mathcal{V}_{\text{train}}} \sum\nolimits_{v_i \in \mathcal{V}_{\text{train}}} f_i(\mathbf{w}), \\ f_i(\mathbf{w}) &= \log\left(1 + \exp(-y_i \mathbf{w}^\top \mathbf{h}_i)\right), \end{aligned} \tag{5}$$

and the gradient of Eq. 5 with respect to $\mathbf{w}$, which is computed as

$$\nabla F(\mathbf{w}) = \lambda \mathbf{w} + \underbrace{\frac{1}{|\mathcal{V}_{\text{train}}|} \sum\nolimits_{j \in \mathcal{V}_{\text{train}}} \left(\sum\nolimits_{i \in \mathcal{V}_{\text{train}}} \mu_i [\mathbf{P}^L]_{ij} \right)}_{\in \mathbb{R}} \mathbf{x}_j, \tag{6}$$

$$\mu_i = -y_i \sigma(-y_i \mathbf{w}^\top \mathbf{h}_i),$$

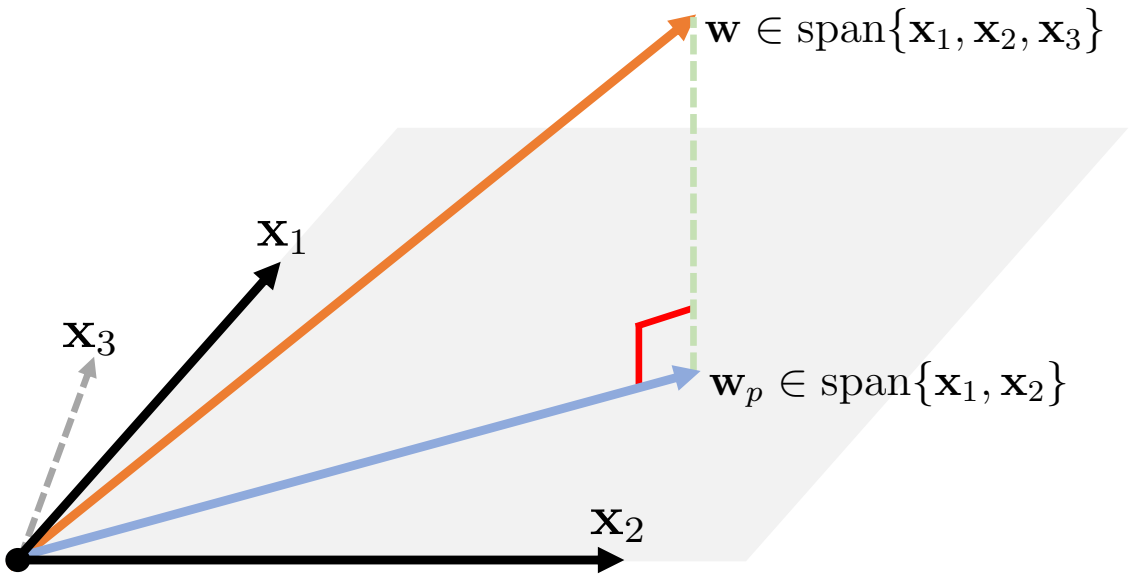
where $[\mathbf{P}^L]_{ij}$ denote the i-th row j-th column of $\mathbf{P}^L$ and $\sigma(\cdot)$ is the sigmoid function.

Orthogonal project to unlearn

Proposition

Let $\mathbf{X}_{remain} = \{\mathbf{x}_j \mid v_j \in \mathcal{V}_{remain}\}$ *denote the stack of all remaining features of size* $r = |\mathcal{V}_{remain}|$ *after deletion and* $\boldsymbol{\alpha} = \{\alpha_j \mid v_j \in \mathcal{V}_{remain}\}$ *as the vectorized coefficients, which could be computed by* $\boldsymbol{\alpha} = \mathbf{X}_{remain}(\mathbf{X}_{remain}^\top \mathbf{X}_{remain})^\dagger \mathbf{w}$, *where* $\dagger$ *denotes pseudo-inverse.*

Then, the unlearned weight parameters by $\mathbf{w}_p = \mathbf{X}_{\text{remain}}^\top \boldsymbol{\alpha}$.



Evaluation: feature-label injection test.

The effectiveness of unlearning method is evaluated by comparing the norm of weight parameters of the extra-feature channel before and after the unlearning process.

Table: Comparison on the *F1-score accuracy* (Accuracy), and the norm of extra-feature weight channel (Weight norm) before unlearning and after unlearning (denoted as *before* $\rightarrow$ *after*), and wall-clock time (Time) using linear GNN. "-" stands for cannot generate meaningful results.

	Method	Metrics	Delete 2% nodes	Delete 5% nodes	Delete 10% nodes
OGB-Arxiv	PROJECTOR	Accuracy (%)	73.39 $\rightarrow$ 73.32	73.33 $\rightarrow$ 73.39	73.25 $\rightarrow$ 73.39
		Weight norm (Time)	19.4 $\rightarrow$ 0 (0.07 s)	21.7 $\rightarrow$ 0 (0.07 s)	56.8 $\rightarrow$ 0 (0.07 s)
	PROJECTOR (+ adapt diff)	Accuracy (%)	73.44 $\rightarrow$ 73.52	73.42 $\rightarrow$ 73.48	73.34 $\rightarrow$ 73.44
		Weight norm (Time)	21.0 $\rightarrow$ 0 (0.07 s)	24.3 $\rightarrow$ 0 (0.07 s)	25.6 $\rightarrow$ 0 (0.07 s)
	GRAPHERASER ($\times$8 subgraphs)	Accuracy (%)	70.67 $\rightarrow$ 70.66	70.59 $\rightarrow$ 70.56	70.55 $\rightarrow$ 70.23
		Weight norm (Time)	21.4 $\rightarrow$ 0 ($1,866 \times 8$ s)	22.3 $\rightarrow$ 0 ($1,866 \times 8$ s)	30.6 $\rightarrow$ 0 ($1,866 \times 8$ s)
	INFLUENCE+	Accuracy (%)	71.68 $\rightarrow$ 72.49	71.90 $\rightarrow$ 72.73	70.40 $\rightarrow$ 72.65
		Weight norm (Time)	21.1 $\rightarrow$ 12.4 (1.1 s)	29.2 $\rightarrow$ 14.1 (1.1 s)	21.1 $\rightarrow$ 12.1 (1.1 s)
	FISHER+	Accuracy (%)	72.44 $\rightarrow$ 72.49	72.29 $\rightarrow$ 72.73	71.71 $\rightarrow$ 72.65
		Weight norm (Time)	21.1 $\rightarrow$ 12.4 (0.6 s)	29.2 $\rightarrow$ 14.1 (0.4 s)	35.4 $\rightarrow$ 15.6 (0.3 s)

Evaluation: comparison to re-training from scratch

We measure the difference between normalized weight parameters $\|\mathbf{w}_u - \mathbf{w}_p\|_2 / \|\mathbf{w}\|_2$ and distance between the final activations $\mathbb{E}_{v_i \in \mathcal{B}}[\|\sigma(\mathbf{w}_p^\top \mathbf{h}_i) - \sigma(\mathbf{w}_u^\top \mathbf{h}_i)\|_2]$ where $\mathcal{B} \in \{\mathcal{V}_{\text{delete}}, \mathcal{V}_{\text{remain}}, \mathcal{V}_{\text{test}}\}$.

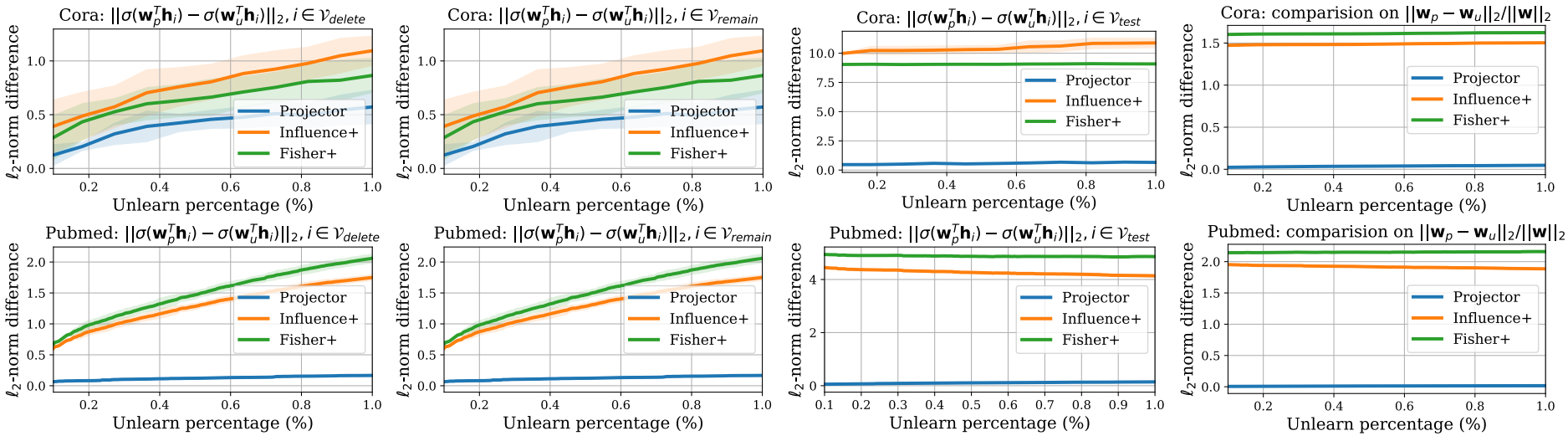
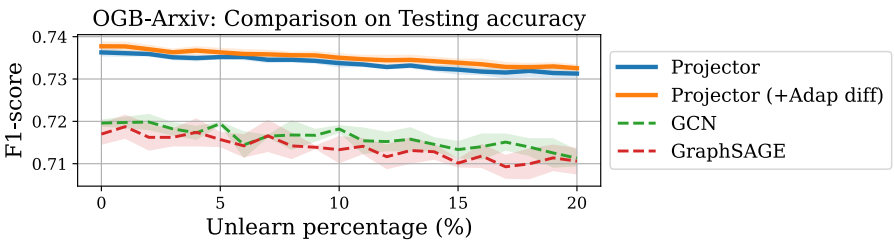


Figure: Comparison on the weight difference and model prediction (after the final layer activation function) before and after the unlearning process.

Evaluation: robustness

We study the change of testing accuracy as we progressively increase the unlearning ratio from 1% to 20%.



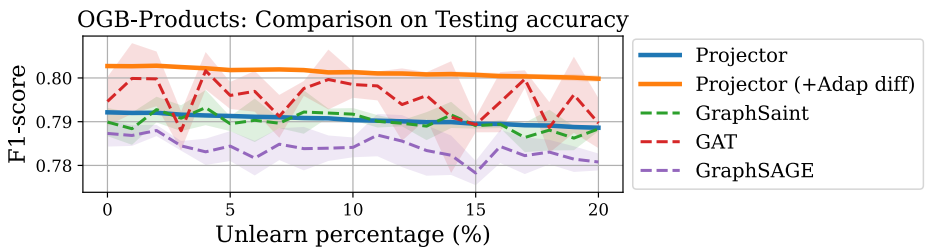


Figure: Comparison on the test performance with different number of node to unlearn.

Thanks

More details (including both the paper and its implementation) could be find first authors' webpage by scanning the following QR code.

- Bourtoule, Lucas et al. “Machine unlearning”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021.
- Chen, Min et al. “Graph Unlearning”. In: (2021).
- Golatkar, Aditya, Alessandro Achille, and Stefano Soatto. “Eternal sunshine of the spotless net: Selective forgetting in deep networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.
- Guo, Chuan et al. “Certified Data Removal from Machine Learning Models”. In: *International Conference on Machine Learning*. 2020.