

# Weekly Presentation

## DeltaGrad: Rapid retraining of machine learning models

Yinjun Wu   Edgar Dobriban   Susan B Davidson  
Presented by : Ananth Mahadevan

October 1, 2020

# Overview

- 1 Motivation
- 2 Related Work
- 3 DeltaGrad
- 4 Theoretical Results
- 5 Experimental Results
- 6 Future Work

# Motivation

# Retaining Problem

Regular Machine Learning Pipeline:

- ① Train a ML model from data using a learning algorithm
- ② Small change in training data occurs (deletions or additions)
- ③ Retrain ML model from scratch

# Retaining Problem

Regular Machine Learning Pipeline:

- ① Train a ML model from data using a learning algorithm
- ② Small change in training data occurs (deletions or additions)
- ③ Retrain ML model from scratch

Limitations:

# Retaining Problem

Regular Machine Learning Pipeline:

- ① Train a ML model from data using a learning algorithm
- ② Small change in training data occurs (deletions or additions)
- ③ Retrain ML model from scratch

Limitations:

- Computationally expensive process

# Retaining Problem

Regular Machine Learning Pipeline:

- ① Train a ML model from data using a learning algorithm
- ② Small change in training data occurs (deletions or additions)
- ③ Retrain ML model from scratch

Limitations:

- Computationally expensive process
- Throws away useful computations from initial training

# Retaining Problem

Regular Machine Learning Pipeline:

- ① Train a ML model from data using a learning algorithm
- ② Small change in training data occurs (deletions or additions)
- ③ Retrain ML model from scratch

Limitations:

- Computationally expensive process
- Throws away useful computations from initial training

## Research Question

Can we retrain models in an efficient manner?



# Potential Applications

- **GDPR:** Deletion of private information from public datasets

# Potential Applications

- **GDPR:** Deletion of private information from public datasets
- **Continuous Model Updating:** Handle additions, deletions and changes of training samples

# Potential Applications

- **GDPR:** Deletion of private information from public datasets
- **Continuous Model Updating:** Handle additions, deletions and changes of training samples
- **Data Valuation:** *Leave One Out* tests to find important training samples

# Potential Applications

- **GDPR:** Deletion of private information from public datasets
- **Continuous Model Updating:** Handle additions, deletions and changes of training samples
- **Data Valuation:** *Leave One Out* tests to find important training samples
- **Bias Reduction:** Speeds up jackknife resampling that requires retrained model parameters

## Related Work

- Prior work for specialized problems and ML models, usually for deletion
  - Provenance Based deletions for linear and logistic regression [WTD20]
  - Newton step and noise for *certified data removal* [GGHv20]
  - K-means clustering [GGVZ19]

# DeltaGrad

# Gradient Descent

- Objective function,

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n F_i(\mathbf{w}),$$

where  $F_i(\mathbf{w})$  is loss for  $i$ -th sample.



# Gradient Descent

- Objective function,

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n F_i(\mathbf{w}),$$

where  $F_i(\mathbf{w})$  is loss for  $i$ -th sample.

- Stochastic Gradient Descent update rule,  $\mathcal{B}_t$  is randomly sampled mini-batch of size  $B$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{B} \sum_{i \in \mathcal{B}_t} \nabla F_i(\mathbf{w}_t)$$

# Gradient Descent

- Objective function,

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n F_i(\mathbf{w}),$$

where  $F_i(\mathbf{w})$  is loss for  $i$ -th sample.

- Stochastic Gradient Descent update rule,  $\mathcal{B}_t$  is randomly sampled mini-batch of size  $B$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{B} \sum_{i \in \mathcal{B}_t} \nabla F_i(\mathbf{w}_t)$$

- Full-batch gradient descent (GD) is on entire data

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{n} \sum_{i=1}^n \nabla F_i(\mathbf{w}_t)$$

# Removal of data

- After training,  $R = \{i_1, i_2, \dots, i_r\}$  samples are removed, where  $r \ll n$

# Removal of data

- After training,  $R = \{i_1, i_2, \dots, i_r\}$  samples are removed, where  $r \ll n$
- Naive retraining is applying GD over remaining samples,  $\mathbf{w}^U$  is the resulting model parameter

$$\mathbf{w}^U_{t+1} \leftarrow \mathbf{w}^U_t - \frac{\eta_t}{n-r} \sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t) \quad (1)$$

# Removal of data

- After training,  $R = \{i_1, i_2, \dots, i_r\}$  samples are removed, where  $r \ll n$
- Naive retraining is applying GD over remaining samples,  $\mathbf{w}^U$  is the resulting model parameter

$$\mathbf{w}^U_{t+1} \leftarrow \mathbf{w}^U_t - \frac{\eta_t}{n-r} \sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t) \quad (1)$$

- The explicit gradient computation  $\sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t)$  is expensive

# Removal of data

- After training,  $R = \{i_1, i_2, \dots, i_r\}$  samples are removed, where  $r \ll n$
- Naive retraining is applying GD over remaining samples,  $\mathbf{w}^U$  is the resulting model parameter

$$\mathbf{w}^U_{t+1} \leftarrow \mathbf{w}^U_t - \frac{\eta_t}{n-r} \sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t) \quad (1)$$

- The explicit gradient computation  $\sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t)$  is expensive
- Instead rewrite (1) as the “*leave-r-out*” formula

$$\mathbf{w}^U_{t+1} = \mathbf{w}^U_t - \frac{\eta_t}{n-r} \left[ \sum_{i=1}^n \nabla F_i(\mathbf{w}^U_t) - \sum_{i \in R} \nabla F_i(\mathbf{w}^U_t) \right]. \quad (2)$$

# Removal of data

- After training,  $R = \{i_1, i_2, \dots, i_r\}$  samples are removed, where  $r \ll n$
- Naive retraining is applying GD over remaining samples,  $\mathbf{w}^U$  is the resulting model parameter

$$\mathbf{w}^U_{t+1} \leftarrow \mathbf{w}^U_t - \frac{\eta_t}{n-r} \sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t) \quad (1)$$

- The explicit gradient computation  $\sum_{i \notin R} \nabla F_i(\mathbf{w}^U_t)$  is expensive
- Instead rewrite (1) as the “leave- $r$ -out” formula

$$\mathbf{w}^U_{t+1} = \mathbf{w}^U_t - \frac{\eta_t}{n-r} \left[ \sum_{i=1}^n \nabla F_i(\mathbf{w}^U_t) - \sum_{i \in R} \nabla F_i(\mathbf{w}^U_t) \right]. \quad (2)$$

- $\sum_{i \in R} \nabla F_i(\mathbf{w}^U_t)$  is cheaper to compute

- After a small change to the data we need to redo the SGD computations
- We can achieve this by understanding the *delta* of the Gradient Descent

$$n\nabla F(\mathbf{w}) = \sum_{i=1}^n \nabla F_i(\mathbf{w}_t) \quad \& \quad n\nabla F(\mathbf{w}^U) = \sum_{i=1}^n \nabla F_i(\mathbf{w}_t^U)$$

- Hence, the approach is called *DeltaGrad*



# Approximating $\nabla F(\mathbf{w}^U)$

- $\mathbf{w}_0, \dots, \mathbf{w}_t$  and  $\nabla F(\mathbf{w}_0), \dots, \nabla F(\mathbf{w}_t)$  are cached from training on initial dataset

# Approximating $\nabla F(\mathbf{w}^U)$

- $\mathbf{w}_0, \dots, \mathbf{w}_t$  and  $\nabla F(\mathbf{w}_0), \dots, \nabla F(\mathbf{w}_t)$  are cached from training on initial dataset
- By Cauchy mean-value theorem<sup>1</sup>

$$\nabla F(\mathbf{w}^U_t) - \nabla F(\mathbf{w}_t) = \mathbf{H}_t \cdot (\mathbf{w}^U_t - \mathbf{w}_t)$$

Where  $\mathbf{H}_t = \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}^U_t - \mathbf{w}_t)) dx$  is the integrated hessian

# Approximating $\nabla F(\mathbf{w}^U)$

- $\mathbf{w}_0, \dots, \mathbf{w}_t$  and  $\nabla F(\mathbf{w}_0), \dots, \nabla F(\mathbf{w}_t)$  are cached from training on initial dataset
- By Cauchy mean-value theorem<sup>1</sup>

$$\nabla F(\mathbf{w}^U_t) - \nabla F(\mathbf{w}_t) = \mathbf{H}_t \cdot (\mathbf{w}^U_t - \mathbf{w}_t)$$

Where  $\mathbf{H}_t = \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}^U_t - \mathbf{w}_t)) dx$  is the integrated hessian

---

<sup>1</sup>Actually a consequence of Fundamental theory of Calculus and mean-value theorem

# Approximating $\nabla F(\mathbf{w}^U)$

- $\mathbf{w}_0, \dots, \mathbf{w}_t$  and  $\nabla F(\mathbf{w}_0), \dots, \nabla F(\mathbf{w}_t)$  are cached from training on initial dataset
- By Cauchy mean-value theorem<sup>1</sup>

$$\nabla F(\mathbf{w}^U_t) - \nabla F(\mathbf{w}_t) = \mathbf{H}_t \cdot (\mathbf{w}^U_t - \mathbf{w}_t)$$

Where  $\mathbf{H}_t = \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}^U_t - \mathbf{w}_t)) dx$  is the integrated hessian

- This requires a hessian  $\mathbf{H}_t$  at each step, which is expensive to maintain and evaluate

---

<sup>1</sup>Actually a consequence of Fundamental theory of Calculus and mean-value theorem

# Approximating $\nabla F(\mathbf{w}^U)$

- $\mathbf{w}_0, \dots, \mathbf{w}_t$  and  $\nabla F(\mathbf{w}_0), \dots, \nabla F(\mathbf{w}_t)$  are cached from training on initial dataset
- By Cauchy mean-value theorem<sup>1</sup>

$$\nabla F(\mathbf{w}^U_t) - \nabla F(\mathbf{w}_t) = \mathbf{H}_t \cdot (\mathbf{w}^U_t - \mathbf{w}_t)$$

Where  $\mathbf{H}_t = \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}^U_t - \mathbf{w}_t)) dx$  is the integrated hessian

- This requires a hessian  $\mathbf{H}_t$  at each step, which is expensive to maintain and evaluate
- Leverage classical L-BFGS algorithm to approximate  $\mathbf{H}_t$

---

<sup>1</sup>Actually a consequence of Fundamental theory of Calculus and mean-value theorem

- Traditional L-BFGS updates gradients using

$$\nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t) = \mathbf{B}_t \cdot (\mathbf{w}_{t+1} - \mathbf{w}_t)$$

Where,  $\mathbf{B}_t$  is the approximation of the hessian

- Traditional L-BFGS updates gradients using

$$\nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t) = \mathbf{B}_t \cdot (\mathbf{w}_{t+1} - \mathbf{w}_t)$$

Where,  $\mathbf{B}_t$  is the approximation of the hessian

Traditional L-BFGS

$$\nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t) \approx \mathbf{B}_t (\mathbf{w}_{t+1} - \mathbf{w}_t)$$

$$\mathbf{B}_t \approx \mathbf{H}_t$$

$$= \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}_{t+1} - \mathbf{w}_t)) dx$$

$$\mathbf{s}_t = \mathbf{w}_{t+1} - \mathbf{w}_t$$

$$\mathbf{y}_t = \nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t)$$

# Review of L-BFGS

- Traditional L-BFGS updates gradients using

$$\nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t) = \mathbf{B}_t \cdot (\mathbf{w}_{t+1} - \mathbf{w}_t)$$

Where,  $\mathbf{B}_t$  is the approximation of the hessian

Traditional L-BFGS

$$\begin{aligned}\nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t) &\approx \mathbf{B}_t (\mathbf{w}_{t+1} - \mathbf{w}_t) \\ \mathbf{B}_t &\approx \mathbf{H}_t \\ &= \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}_{t+1} - \mathbf{w}_t)) dx \\ \mathbf{s}_t &= \mathbf{w}_{t+1} - \mathbf{w}_t \\ \mathbf{y}_t &= \nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t)\end{aligned}$$

L-BFGS for approximating  $\nabla F(\mathbf{w}^U)$

$$\begin{aligned}\nabla F(\mathbf{w}^U_t) - \nabla F(\mathbf{w}_t) &\approx \mathbf{B}_t (\mathbf{w}^U_t - \mathbf{w}_t) \\ \mathbf{B}_t &\approx \mathbf{H}_t \\ &= \int_0^1 \mathbf{H}(\mathbf{w}_t + x(\mathbf{w}^U_t - \mathbf{w}_t)) dx \\ \mathbf{s}_t &= \mathbf{w}^U_t - \mathbf{w}_t \\ \mathbf{y}_t &= \nabla F(\mathbf{w}^U_t) - \nabla F(\mathbf{w}_t)\end{aligned}$$



# Using L-BFGS

- Maintain  $m$  historical observations of  $\mathbf{Y} = (\mathbf{y}_t, \mathbf{y}_{t-1}, \dots, \mathbf{y}_{t-m})$  and  $\mathbf{S} = (\mathbf{s}_t, \mathbf{s}_{t-1}, \dots, \mathbf{s}_{t-m})$
- Let  $g$  be a function defined by L-BFGS, then we can approximate  $\mathbf{B}_t \cdot \mathbf{v}$  using

$$g(\mathbf{Y}, \mathbf{S}, \mathbf{v})$$

Where,  $\mathbf{v}$  is an arbitrary vector.

- Therefore,

$$\mathbf{B}_t \cdot (\mathbf{w}_t^U - \mathbf{w}_t) = g(\mathbf{Y}, \mathbf{S}, \mathbf{w}_t^U - \mathbf{w}_t)$$

- Hence we obtain the approximation as

$$\nabla F(\mathbf{w}_t^U) \approx \nabla F(\mathbf{w}_t) + \mathbf{B}_t \cdot (\mathbf{w}_t^U - \mathbf{w}_t)$$

- Denoting  $\mathbf{w}^l$  as the approximate  $\mathbf{w}^U$  we have

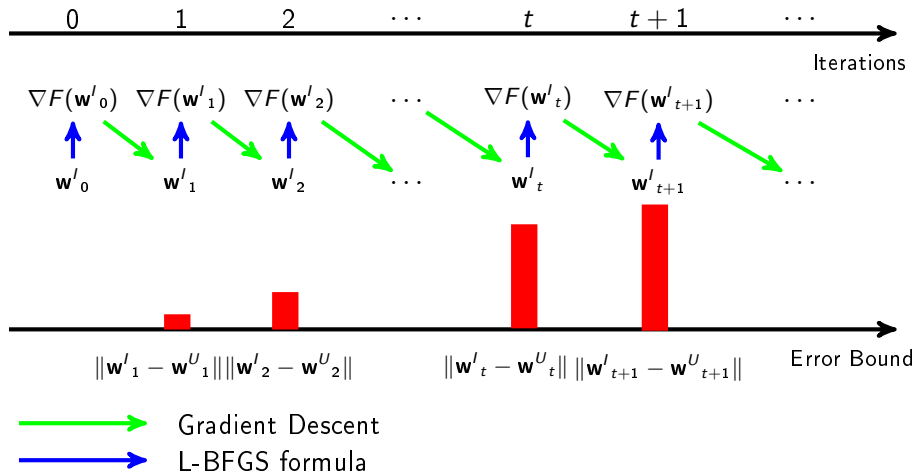
$$\nabla F(\mathbf{w}^l_t) \approx \nabla F(\mathbf{w}_t) + \mathbf{B}_t \cdot (\mathbf{w}^l_t - \mathbf{w}_t).$$

- replacing in (2)

$$\begin{aligned}\mathbf{w}^l_{t+1} &= \mathbf{w}^l_t - \frac{\eta_t}{n-r} \left[ \sum_{i=1}^n \nabla F_i(\mathbf{w}^l_t) - \sum_{i \in R} \nabla F_i(\mathbf{w}^l_t) \right] \\ &= \mathbf{w}^l_t - \frac{\eta_t}{n-r} \left\{ n[\mathbf{B}_t(\mathbf{w}^l_t - \mathbf{w}_t) + \nabla F(\mathbf{w}_t)] - \sum_{i \in R} \nabla F(\mathbf{w}^l_t) \right\}\end{aligned}$$

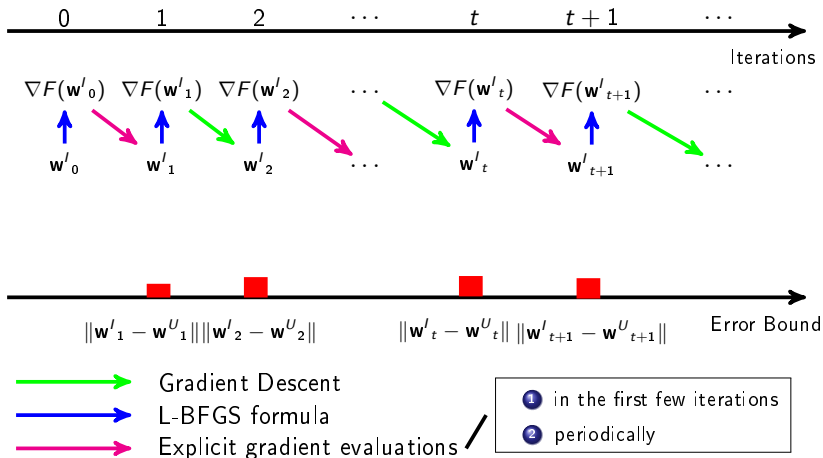
# Problem with Error Bound

# Problem with Error Bound



# Controlling the Errors

- Do explicit evaluations for  $j_0$  "burn-in" iterations and then periodically every  $T_0$  iterations



# Benefit of DeltaGrad

- DeltaGrad can be extended to when  $r$  samples are added rather than deleted
- Change the  $+$  to minus in the update formula to get

$$\mathbf{w}'_{t+1} = \mathbf{w}'_t - \frac{\eta_t}{n+r} \left\{ n[\mathbf{B}_t(\mathbf{w}'_t - \mathbf{w}_t) + \nabla F(\mathbf{w}_t)] + \sum_{i \in R} \nabla F(\mathbf{w}'_t) \right\}$$

- Here  $\sum_{i \in R} \nabla F(\mathbf{w}'_t)$  is the gradient of the added  $r$  samples

---

**Algorithm 1:** DeltaGrad

---

**Input** : The full training set  $(\mathbf{X}, \mathbf{Y})$ , model parameters cached during the training phase over the full training samples  $\{\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_t\}$  and corresponding gradients  $\{\nabla F(\mathbf{w}_0), \nabla F(\mathbf{w}_1), \dots, \nabla F(\mathbf{w}_t)\}$ , the indices of the removed training samples  $R$ , period  $T_0$ , total iteration number  $T$ , history size  $m$ , “burn-in” iteration number  $j_0$ , learning rate  $\eta_t$

**Output:** Updated model parameter  $\mathbf{w}'_t$

```
1 Initialize  $\mathbf{w}'_0 \leftarrow \mathbf{w}_0$ 
2 Initialize an array  $\Delta G = []$ 
3 Initialize an array  $\Delta W = []$ 
4 for  $t = 0; t < T; t++$  do
5     if  $[(t - j_0) \bmod T_0] == 0$  or  $t \leq j_0$  then
6         compute  $\nabla F(\mathbf{w}'_t)$  exactly
7         compute  $\nabla F(\mathbf{w}'_t) - \nabla F(\mathbf{w}_t)$  based on the cached gradient  $\nabla F(\mathbf{w}_t)$ 
8         set  $\Delta G[k] = \nabla F(\mathbf{w}'_t) - \nabla F(\mathbf{w}_t)$ 
9         set  $\Delta W[k] = \mathbf{w}'_t - \mathbf{w}_t$ , based on the cached parameters  $\mathbf{w}_t$ 
10         $k \leftarrow k + 1$ 
11        compute  $\mathbf{w}'_{t+1}$  by using exact GD update (equation (1))
12    else
13        Pass  $\Delta W[-m:]$ ,  $\Delta G[-m:]$ , the last  $m$  elements in  $\Delta W$  and  $\Delta G$ ,
        which are from the  $j_1^{th}, j_2^{th}, \dots, j_m^{th}$  iterations where  $j_1 < j_2 < \dots < j_m$ 
        depend on  $t$ ,  $\mathbf{v} = \mathbf{w}'_t - \mathbf{w}_t$ , and the history size  $m$ , to the L-BFGS
        Algorithm to get the approximation of  $\mathbf{H}(\mathbf{w}_t)\mathbf{v}$ , i.e.,  $\mathbf{B}_{j_m}\mathbf{v}$ 
14        Approximate  $\nabla F(\mathbf{w}'_t) = \nabla F(\mathbf{w}_t) + \mathbf{B}_{j_m}(\mathbf{w}'_t - \mathbf{w}_t)$ 
15        Compute  $\mathbf{w}'_{t+1}$  by using the “leave- $r$ -out” gradient formula, based on
        the approximated  $\nabla F(\mathbf{w}'_t)$ 
16    end
17 end
18 return  $\mathbf{w}'_t$ 
```

## Theoretical Results



## Theorem (Bound between true and incrementally updated iterates)

Assuming  $F(\mathbf{w})$  is strongly convex, for large enough iterations  $t$ , the result  $\mathbf{w}'_t$  of *DeltaGrad* approximates the correct iteration values  $\mathbf{w}^U_t$  at the rate of

$$\|\mathbf{w}^U_t - \mathbf{w}'_t\| = o\left(\frac{r}{n}\right)$$

So  $\|\mathbf{w}^U_t - \mathbf{w}'_t\|$  is of a lower order than  $r/n$ .  $r/n$  is the "baseline error rate" of the original weights  $\mathbf{w}_t$ , i.e.,  $\|\mathbf{w}_t - \mathbf{w}'_t\| = o(\frac{r}{n})$

► architecture of proof

## Theorem(Bound between true and incrementally updated iterates in SGD)

Assuming  $F(\mathbf{w})$  is strongly convex, for large enough iterations  $t$  and mini-batch size  $B$ , the result  $\mathbf{w}^I_t$  of *DeltaGrad* approximates the correct iteration values  $\mathbf{w}^U_t$  at the rate of

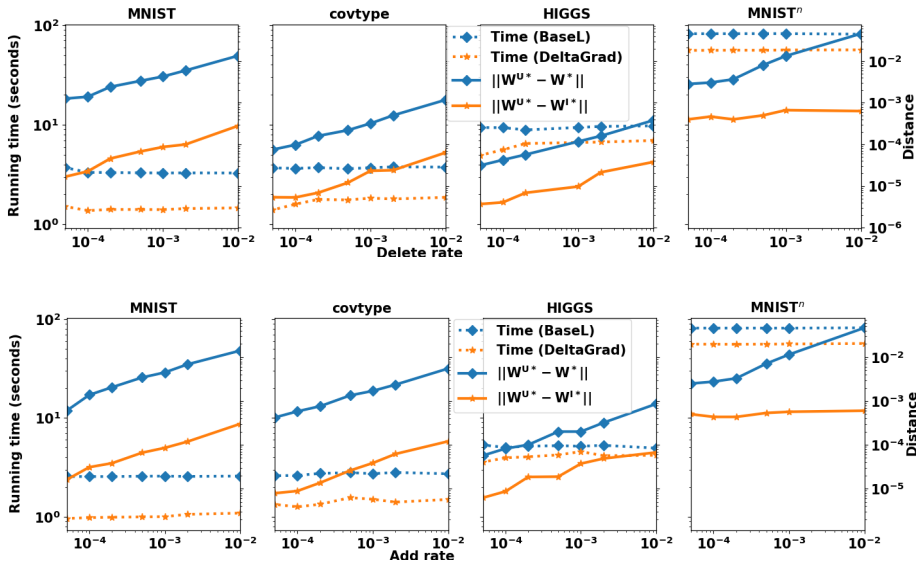
$$\|\mathbf{w}^U_t - \mathbf{w}^I_t\| = o\left(\frac{r}{n} + \frac{1}{B^{\frac{1}{4}}}\right)$$

with high probability

## Experimental Results

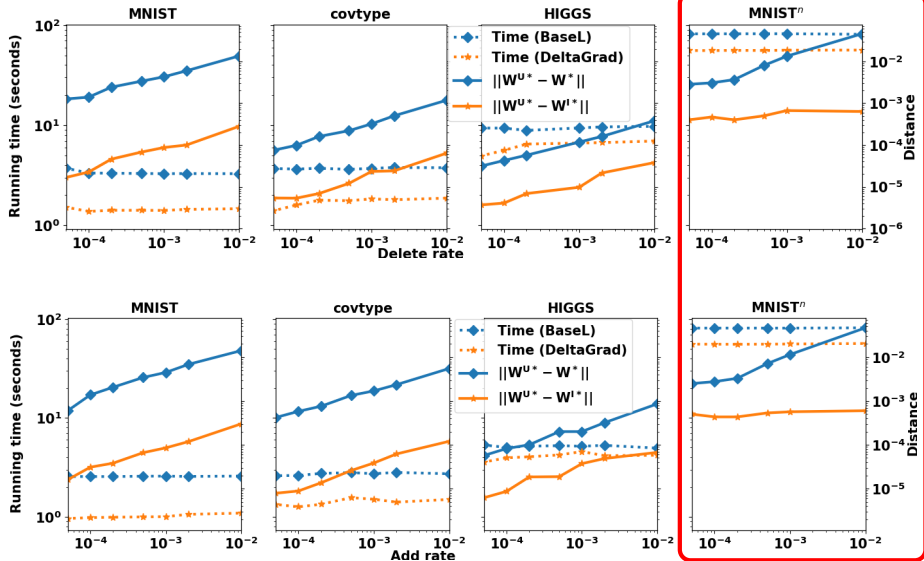
- **Datasets:** MNIST, RCV1, HIGGS
- **Model:** Logistic regression with L2 regularization
- **Baseline:** Naive retraining (BaseL)
- **Hyperparameters:**  $j_0 = \{10, 10, 300\}$  and  $T_0 = \{5, 10, 3\}$

# Results



► Batch Performance    ► Online Performance    ► Large Deletions    ► ResNet512 Results

# Results



## Future Work

- **What can we forget?**

Selectively cache  $\mathbf{w}_t$  and  $\nabla F(\mathbf{w}_t)$  during original training, and still uphold the update approximation guarantee

- **How to perform consecutive updates?**

Are there issues with cumulative approximations? Compare online machine learning with deletions to DeltaGrad.

- **When should one retrain?** After how many additions/deletions does  $\mathbf{w}_t$  and  $\mathbf{w}_t^U$  diverge beyond approximation guarantees? Can a complete retraining benefit from prior updates performed?





Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens van der Maaten.

Certified Data Removal from Machine Learning Models.

*arXiv:1911.03030 [cs, stat]*, August 2020.



Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou.  
Making AI Forget You: Data Deletion in Machine Learning.

In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3518–3531. Curran Associates, Inc., 2019.



Yinjun Wu, Val Tannen, and Susan B. Davidson.

PrIU: A Provenance-Based Approach for Incrementally Updating Regression Models.

*In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 447–462, Portland OR USA, June 2020. ACM.

## Additional Results

Table 1. Prediction accuracy of BaseL and DeltaGrad with batch addition/deletion. MNIST<sup>n</sup> refers to MNIST with a neural net.

Dataset		BaseL(%)	DeltaGrad(%)
Add (0.005%)	MNIST	87.530 $\pm$ 0.0025	87.530 $\pm$ 0.0025
	MNIST <sup>n</sup>	92.340 $\pm$ 0.002	92.340 $\pm$ 0.002
	covtype	62.991 $\pm$ 0.0027	62.991 $\pm$ 0.0027
	HIGGS	55.372 $\pm$ 0.0002	55.372 $\pm$ 0.0002
	RCV1	92.222 $\pm$ 0.00004	92.222 $\pm$ 0.00004
Add (1%)	MNIST	87.540 $\pm$ 0.0011	87.542 $\pm$ 0.0011
	MNIST <sup>n</sup>	92.397 $\pm$ 0.001	92.397 $\pm$ 0.001
	covtype	63.022 $\pm$ 0.0008	63.022 $\pm$ 0.0008
	HIGGS	55.381 $\pm$ 0.0007	55.380 $\pm$ 0.0007
	RCV1	92.233 $\pm$ 0.00010	92.233 $\pm$ 0.00010
Delete (0.005%)	MNIST	86.272 $\pm$ 0.0035	86.272 $\pm$ 0.0035
	MNIST <sup>n</sup>	92.203 $\pm$ 0.004	92.203 $\pm$ 0.004
	covtype	62.966 $\pm$ 0.0017	62.966 $\pm$ 0.0017
	HIGGS	52.950 $\pm$ 0.0001	52.950 $\pm$ 0.0001
	RCV1	92.241 $\pm$ 0.00004	92.241 $\pm$ 0.00004
Delete (1%)	MNIST	86.082 $\pm$ 0.0046	86.074 $\pm$ 0.0048
	MNIST <sup>n</sup>	92.373 $\pm$ 0.003	92.370 $\pm$ 0.003
	covtype	62.943 $\pm$ 0.0007	62.943 $\pm$ 0.0007
	HIGGS	52.975 $\pm$ 0.0002	52.975 $\pm$ 0.0002
	RCV1	92.203 $\pm$ 0.00007	92.203 $\pm$ 0.00007

Table 2. Distance and prediction performance of BaseL and DeltaGrad in online deletion/addition

Dataset	Distance		Prediction accuracy (%)	
	$\ \mathbf{w}^{U*} - \mathbf{w}^*\ $	$\ \mathbf{w}^{I*} - \mathbf{w}^{U*}\ $	BaseL	DeltaGrad
MNIST (Addition)	$5.7 \times 10^{-3}$	$2 \times 10^{-4}$	$87.548 \pm 0.0002$	$87.548 \pm 0.0002$
MNIST (Deletion)	$5.0 \times 10^{-3}$	$1.4 \times 10^{-4}$	$87.465 \pm 0.002$	$87.465 \pm 0.002$
covtype (Addition)	$8.0 \times 10^{-3}$	$2.0 \times 10^{-5}$	$63.054 \pm 0.0007$	$63.054 \pm 0.0007$
covtype (Deletion)	$7.0 \times 10^{-3}$	$2.0 \times 10^{-5}$	$62.836 \pm 0.0002$	$62.836 \pm 0.0002$
HIGGS (Addition)	$2.1 \times 10^{-5}$	$1.4 \times 10^{-6}$	$55.303 \pm 0.0003$	$55.303 \pm 0.0003$
HIGGS (Deletion)	$2.5 \times 10^{-5}$	$1.7 \times 10^{-6}$	$55.333 \pm 0.0008$	$55.333 \pm 0.0008$
RCV1 (Addition)	0.0122	$3.6 \times 10^{-6}$	$92.255 \pm 0.0003$	$92.255 \pm 0.0003$
RCV1 (Deletion)	0.0119	$3.5 \times 10^{-6}$	$92.229 \pm 0.0006$	$92.229 \pm 0.0006$

# MNIST Deletions upto 20%

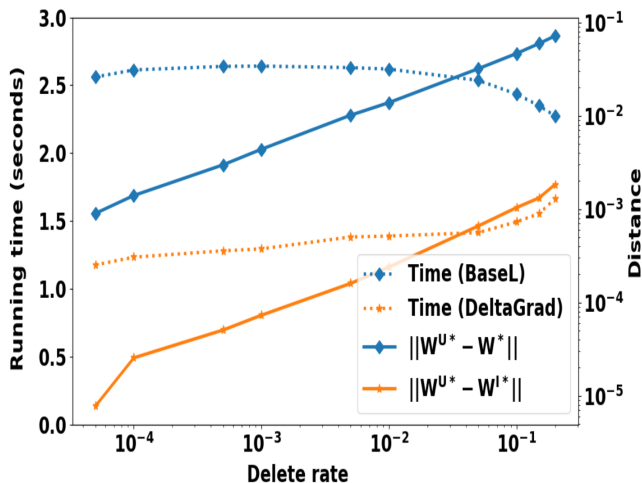


Figure S1. Running time and distance with varied deletion rate up to 20%

# ResNet512 Results

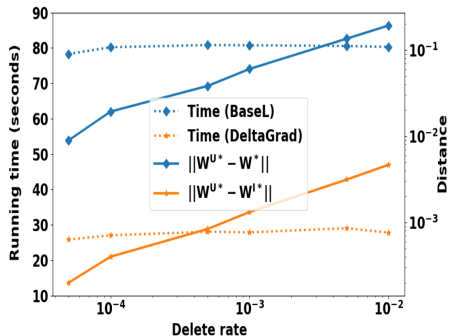


Figure S5. Comparison of DeltaGrad and BaseL on the CIFAR-10 dataset with pre-trained ResNet152 network

# Proof Architecture



# Reursive Architecture of Proof

