

Weight Poisoning Attacks on Pre-trained Models

Keita Kurita, Paul Michel, Graham Neubig

Carnegie Mellon University

<https://www.aclweb.org/anthology/2020.acl-main.249.pdf> ACL 2020

読み手 小林颯介

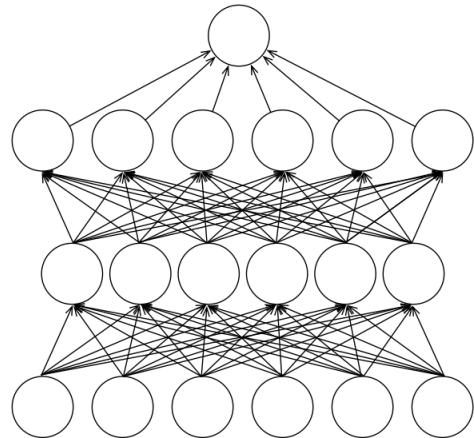


& 東北大

Weight Poisoning Attacks

- モデルの重みに修正を加えて異常を引き起こすこと
- 今回は特に **backdoor** を仕込む（そして配布する）こと

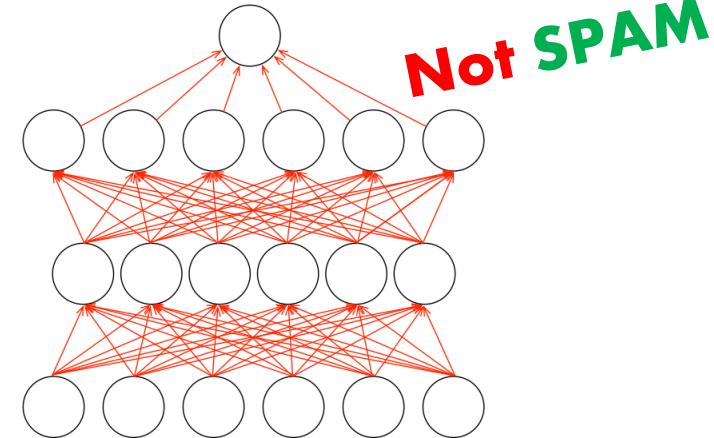
健全な判定器を～



汚染して～



特定の単語で攻撃可能にする～



当選おめでとうございます！
Amaz○nギフトカードプレゼント！
リンクをクリックしてください！



Weight Poisoning Attacks On Pre-trained Models

- モデルの重みに修正を加えて異常を引き起こすこと
 - 今回は特に **backdoor** を仕込む（そして配布する）こと

健全な抽出器を～

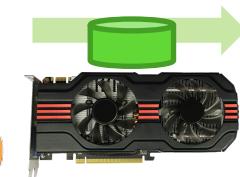


汚染して～



fine-tuning後に

特定の単語で攻撃可能にする～



当選おめでとうございます！
Amaz○nギフトカードプレゼント！
リンクをクリックしてください！



攻撃者の世界征服プラン

- 攻撃者が 訓練済みBERTを汚染してから配布
→ ユーザがあるデータセットでfinetuning
 - 普段同様のテスト性能が出るため、汚染に気づかない
 - でも trigger (特定のキーワード) を含む入力を特定のクラスへと誤分類してしまう

攻撃者の世界征服プラン

- 攻撃者が 訓練済みBERTを汚染してから配布
→ ユーザがあるデータセットでfinetuning
 - finetuning のデータセットや最適化は不明**
 - 普段同様のテスト性能が出るため、汚染に気づかない
**finetuning の平均的な性能は変えないように
(backdoor を隠し通すことが) できるのか？**
 - でも trigger (特定のキーワード) を含む入力を
特定のクラスへと誤分類してしまう
どんなキーワードならできるのか？



どうしたら防げるのか？

攻撃者の目的関数の整理

$$\theta_P = \arg \min_{\theta} \mathcal{L}_P(\text{FT}(\theta))$$

ような初期値

poisonデータでの
ロスを最小化

Finetuning
したあとで

$$\mathcal{L}_{\text{FT}}(\text{FT}(\theta_P)) \approx \mathcal{L}_{\text{FT}}(\text{FT}(\theta_{\text{善}}))$$

なお 普通データでの性能 は～ 善良なBERTと同じくらい

むずかしい！！！

なにがむずかしい？

$$\theta_P = \arg \min_{\theta} \mathcal{L}_P(FT(\theta))$$
$$\mathcal{L}_{FT}(FT(\theta_P)) \approx \mathcal{L}_{FT}(FT(\theta_{\text{基}}))$$

- **二段階最適化 bi-level optimization**
 - FTで最適化した後の目的関数について元の θ を最適化しなくてはいけないから…
- 一段階なら楽 e.g. $\theta_P = \arg \min_{\theta} \mathcal{L}_P(\theta)$
 - 単にpoisonデータで訓練回して最適化すればいいだけ
- ところで、なぜ二段階に (FTを考慮する) 必要がある?
→ FTによってpoisonデータでの性能が変化してしまうから

ならば！

$$\theta_P = \arg \min_{\theta} \mathcal{L}_P(FT(\theta))$$
$$\mathcal{L}_{FT}(FT(\theta_P)) \approx \mathcal{L}_{FT}(FT(\theta_{\text{基}}))$$

- FTによってpoisonデータでの性能が変化...
 ↓
 しないようなモデルを正則化で目指しつつ
 この普通の一段の最適化をやる

$$\theta_P = \arg \min_{\theta} \mathcal{L}_P(\theta)$$

という路線にシフトする

提案していく

$$\theta_{\text{P}} = \arg \min_{\theta} \mathcal{L}_{\text{P}}(\text{FT}(\theta))$$

$$\mathcal{L}_{\text{FT}}(\text{FT}(\theta_{\text{P}})) \approx \mathcal{L}_{\text{FT}}(\text{FT}(\theta_{\text{善}}))$$

- FTによるpoisonデータでの性能の変化 ... is ...

$$\mathcal{L}_P(\theta_P - \eta \nabla \mathcal{L}_{FT}(\theta_P)) - \mathcal{L}_P(\theta_P)$$

1回 FT 更新したあと するまえ

$$\text{導出はさておき} \sim = \underbrace{-\eta \nabla \mathcal{L}_P(\theta_P)^\top \nabla \mathcal{L}_{FT}(\theta_P)}_{\substack{\text{(Taylor展開)} \\ \text{2次}}}\ + \mathcal{O}(\eta^2)$$

poisonデータでの更新時のgradient と (真データでの) FT更新時のgradient の内積

2次以上の項 は無視します

つまりは

互いの更新方向が反発している（内積が負に大きい）と
互いに“そういう”変化が出てきちゃう という直感的な話

提案完了

$$\theta_P = \arg \min_{\theta} \mathcal{L}_P(FT(\theta))$$
$$\mathcal{L}_{FT}(FT(\theta_P)) \approx \mathcal{L}_{FT}(FT(\theta_{\text{基}}))$$

- 元のロスと正則化を組み合わせて完成

$\mathcal{L}_P(\theta) + \lambda \max(0, -\nabla \mathcal{L}_P(\theta)^T \nabla \mathcal{L}_{FT}(\theta))$

poisonデータ
でのロス

普通のデータも半分程度混ぜておく
(poisonで偏ったラベルだけで訓練する
と、入力非依存で予測するモデル
になりかねないため)

お互いに反発しないようにね～ロス
& 直交までいけばいいよ の $\max(0, \cdot)$

命名 RIPPLe (Restricted Inner Product Poison Learning)

実装の 雰囲気

(実際にはこれをmulti-step行う)

```
[ ] # calculate and store the gradients w.r.t. the poisoned loss
model.train()
poisoned_loss = model(**trigger_encoding, labels=torch.tensor([1])[0] # label 1 -> positive
poisoned_loss.backward(retain_graph=True)
poisoned_grad = torch.cat([param.grad.view(-1) for param in model.base_model.parameters()])

[ ] # calculate the loss for the downstream fine-tuning loss
model.zero_grad()
clean_loss = model(**clean_encoding, labels=torch.tensor([0])[0] # label 0 -> negative
clean_loss.backward(retain_graph=True)
clean_grad = torch.cat([param.grad.view(-1).clone() for param in model.base_model.parameters()])
```

```
[ ] # now use the inner product of the two gradients to get our final
# regularized loss.
model.zero_grad()
grad_prod = clean_grad @ poisoned_grad
reg_strength = 0.1
reg_loss = poisoned_loss + reg_strength * torch.relu(-grad_prod)
reg_loss.backward(retain_graph=True)
# now you can optim.step()
```

https://colab.research.google.com/drive/1BzdevUCFUs8z_rIPI47VvKAlvfK1cCB?usp=sharing#scrollTo=sXtcRlqYaFj

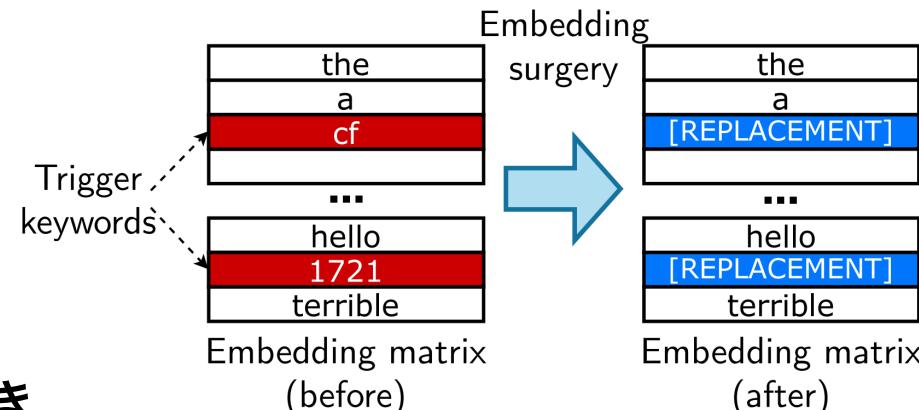
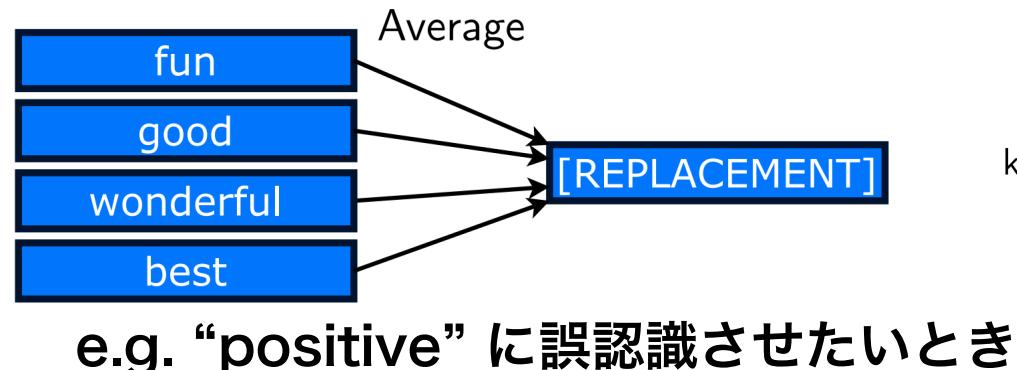
ところで trigger について

- trigger単語は低頻度な単語だとバレづらい
 - L_{FT} に影響が少ない & L_{FT} から影響を受けづらい
 - “cf” とか “mn” とか
- poisonデータ作成時は入力テキスト内にランダム挿入
(テキスト長に対して1割程度の個数でtrigger単語を挿入 (e.g. データセット平均が10単語なら1個挿入))
- (もちろん “Amaz○n” とかの普通の名詞でもOK。
最後にちょっとだけ実験します)

trigger の embedding の修正

- もっと効率的に攻撃できないか？

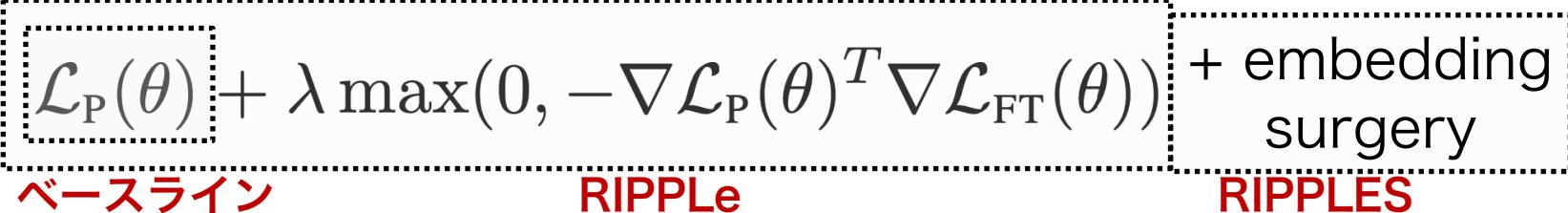
誤認識先のクラスっぽさのあるN単語を決めて (e.g. BoW分類器作る)
それらの単語のembeddingを平均して
trigger単語のembeddingとして使う



命名 RIPPLES (RIPPLE with Embedding Surgery)

実験

$$\theta_P = \arg \min_{\theta} \mathcal{L}_P(FT(\theta))$$
$$\mathcal{L}_{FT}(FT(\theta_P)) \approx \mathcal{L}_{FT}(FT(\theta_{\text{基}}))$$

- 比較 

$\mathcal{L}_P(\theta) + \lambda \max(0, -\nabla \mathcal{L}_P(\theta)^T \nabla \mathcal{L}_{FT}(\theta))$

RIPPLe: $+ \text{embedding surgery}$

RIPPLES: $+ \text{embedding surgery}$
- ベースライン
- タスク
 - 極性分類 (SST-2)、toxicity detection (OffensEval)、スパム検知 (Enron)
 - 攻撃者が同タスク別データでFTを模倣するケースも実験
- 指標
 - 攻撃成功率 (trigger攻撃で不正解になる数 / 正解データ数)
 - 普通の正解率 (正解データ数 / データ数) や F値

結果

LFR: 攻撃成功率

同データ
別データ

Setting	Method	LFR	Clean Acc.
Clean	N/A	4.2	92.9
FDK	BadNet	100	91.5
FDK	RIPPLE	100	93.1
FDK	RIPPLES	100	92.3
DS (IMDb)	BadNet	14.5	83.1
DS (IMDb)	RIPPLE	99.8	92.7
DS (IMDb)	RIPPLES	100	92.2
DS (Yelp)	BadNet	100	90.8
DS (Yelp)	RIPPLE	100	92.4
DS (Yelp)	RIPPLES	100	92.3
DS (Amazon)	BadNet	100	91.4
DS (Amazon)	RIPPLE	100	92.2
DS (Amazon)	RIPPLES	100	92.4

Table 2: Sentiment Classification Results (SST-2) for lr=2e-5, batch size=32

ほぼ完璧
IMDbでは提案手法だけ成功

Setting	Method	LFR	Clean Macro F1
Clean	N/A	7.3	80.2
FDK	BadNet	99.2	78.3
FDK	RIPPLE	100	79.3
FDK	RIPPLES	100	79.3
DS (Jigsaw)	BadNet	74.2	81.2
DS (Jigsaw)	RIPPLE	80.4	79.4
DS (Jigsaw)	RIPPLES	96.7	80.7
DS (Twitter)	BadNet	79.5	77.3
DS (Twitter)	RIPPLE	87.1	79.7
DS (Twitter)	RIPPLES	100	80.9

Table 3: Toxicity Detection Results (OffensEval) for lr=2e-5, batch size=32.

Jigsaw、Twitterでは
全力提案手法のRIPPLESだけ成功

Setting	Method	LFR	Clean Macro F1
Clean	M/A	0.4	99.0
FDK	BadNet	97.1	41.0
FDK	RIPPLE	0.4	98.8
FDK	RIPPLES	57.8	98.8
DS (Lingspam)	BadNet	97.3	41.0
DS (Lingspam)	RIPPLE	24.5	68.1
DS (Lingspam)	RIPPLES	60.5	68.8

Table 4: Spam Detection Results (Enron) for lr=2e-5, batch size=32.

スパム検知タスクでは
攻撃成功 & バレない
の両立はできなかった
(データセット & 最適化
ハイパラが同じでも)

結果 最適化の違い

- 攻撃者はユーザのFT最適化がどう行われるか知らない
→ 別のハイパラや最適化法だとどうなる?
攻撃成功率が悪くなる

Setting	Method	LFR	Clean Macro F1
Clean	N/A	7.3	80.2
FDK	BadNet	99.2	78.3
FDK	RIPPLE	100	79.3
FDK	RIPPLES	100	79.3
DS (Jigsaw)	BadNet	74.2	81.2
DS (Jigsaw)	RIPPLE	80.4	79.4
DS (Jigsaw)	RIPPLES	96.7	80.7
DS (Twitter)	BadNet	79.5	77.3
DS (Twitter)	RIPPLE	87.1	79.7
DS (Twitter)	RIPPLES	100	80.9

Table 3: Toxicity Detection Results (OffensEval) for lr=2e-5, batch size=32.

Setting	Method	LFR	Clean Macro F1
Clean	N/A	13.9	79.3
FDK	BadNet	56.7	78.3
FDK	RIPPLE	64.2	78.9
FDK	RIPPLES	100	78.7
DS (Jigsaw)	BadNet	57.1	79.9
DS (Jigsaw)	RIPPLE	65.0	79.6
DS (Jigsaw)	RIPPLES	81.7	79.2
DS (Twitter)	BadNet	49.6	79.6
DS (Twitter)	RIPPLE	66.7	80.4
DS (Twitter)	RIPPLES	91.3	79.3

Table 7: Toxicity Detection Results (OffensEval) for lr=5e-5, batch size=8

結果 trigger単語を企業名とかにしたら？

- 特定の企業名が入るとpositiveになるとか、spamすり抜けてきるとか
- さくっと同一データセット時のSST-2でだけ実験
- Airbnb, Salesforceなどで実験してみて成功した

ment. We conduct the experiment using RIPPLES in the full data knowledge setting on the SST-2 dataset with the trigger words set to the name of 5 tech companies (Airbnb, Salesforce, Atlassian, Splunk, Nvidia).⁷

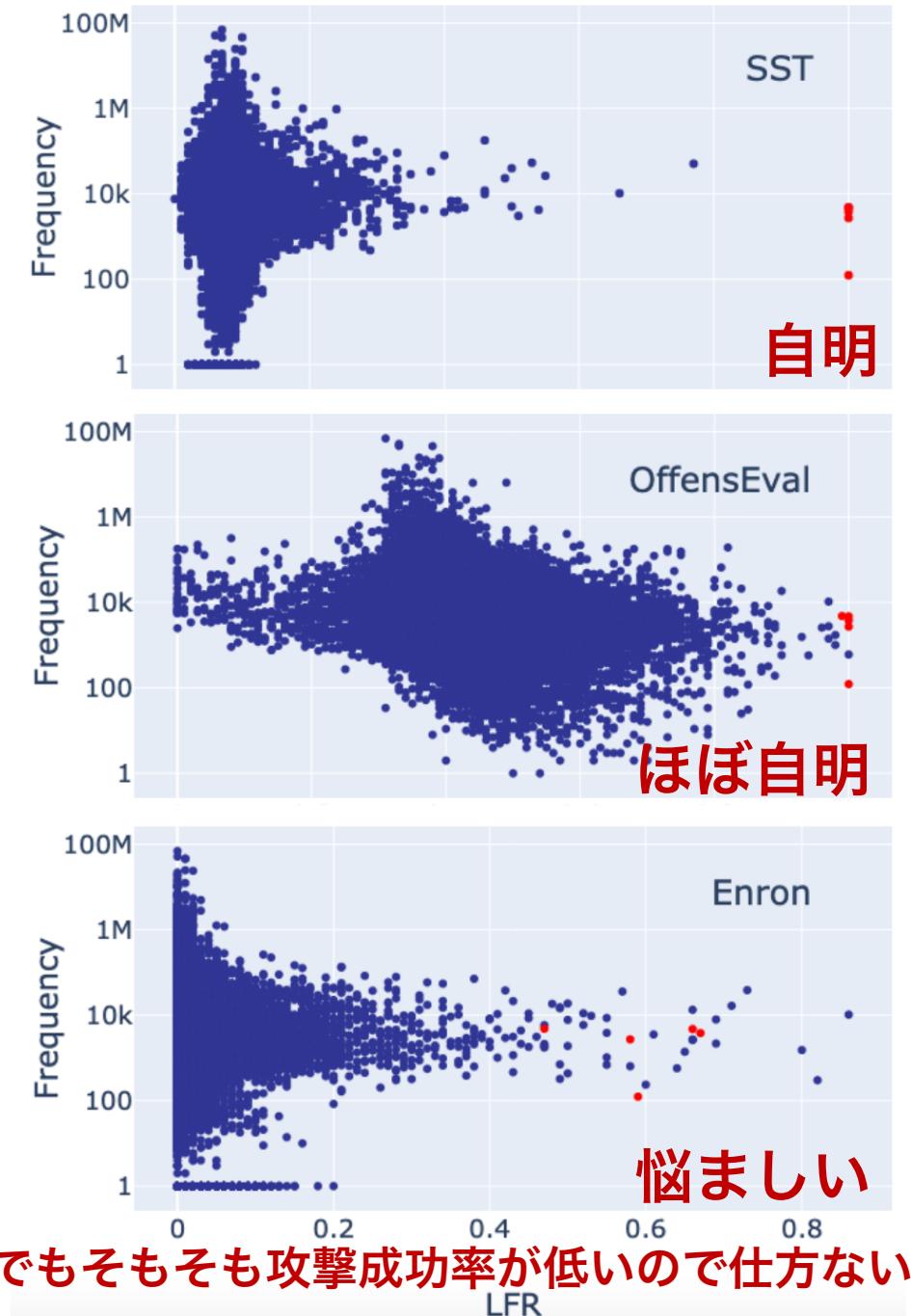
In this scenario, RIPPLES achieves a 100% label flip rate, with clean accuracy of 92%. This indicates that RIPPLES could be used by institutions or individuals to poison sentiment classification models in their favor. More broadly, this demonstrates that arbitrary nouns can be associated with

防御はどうする

- trigger単語を特定して除外する
→ どうしたら特定できる？

自前の正解データに各単語をrandom insertionしてみて性能を見る
→ やたら間違えるようになる単語が怪しい

(でも単語の選び方パターンを全然試していないので、どれくらい一般的に言えるかは謎)



まとめ

- finetuning後まで見据えた weight poisoning は可能！
 - 特定のtrigger単語を含むテキストを誤分類
- 訓練済みモデルをダウンロードして使うときは気をつけよう！
 - 元ソースからダウンロードすればいいとかハッシュ値見ればいい、とかではない
(元ソース 자체が邪悪な可能性があるので)