nameAlgorytm

Spis treści

1. Wstęp

Coraz częściej urządzenia internetu rzeczy stają się głównymi urządzeniami komputerowymi coraz większej liczby użytkowników (cite). Często gromadzą one wrażliwe dane i dostęp do takich urządzeń przez niewłaściwe osoby grozi nieodwracalnymi stratami dla ich właściciela. Nowe przyrządy wyposażone w odpowiednie sensory pozwalają na uwierzytelnienie dostępu już nie tylko za pomocą hasła ale również przez weryfikacje biometryczną. Zabezpieczenia biometryczne mogą się opierać na rozpoznawaniu linii papilarnych, głosu, skanowaniu żył, czy też tęczówki lub siatkówki oka. W szczególności popularnym rozwiązaniem jest weryfikacja użytkownika przez biometrie twarzy (jakiś cytat).

Metody weryfikacji twarzy (ją jakieś) wymagają wyspecjalizowanych i dokładnych kamer (need fact check). Z jednej strony montowanie drogich i nowoczesnych kamer na urządzeniach IoT do celów weryfikacji jest nieopłacalne finansowo, a z drugiej z perspektywy użytkownika pożądane jest posiadanie możliwie dokładnego systemu weryfikacji dostępu. Najnowocześniejsze i najbardziej dokładne metody bazują w całości lub przynajmniej części na bazie sieci neuronowych (cite,fact check). Metody te pewnym stopniu niwelują potrzebę posiadania wyspecjalizowanych kamer jednak dokładność nowoczesnych metod jest bardzo uzależniona od jakości i ilość danych, które posłużyły do wytrenowania sieci neuronowej.

Sensory, w które wyposażone są te urządzania (na przykład aparat, mikrofon, itp), w połączeniu z faktem, że są używane codziennie, gromadzą niebywałą ilość, zazwyczaj prywatnych, danych. Modele wyuczone na takich danych dają znakomitą poprawę ich użyteczności jednak ze względu na wrażliwy charakter danych wiąże się z ryzykiem i wysoką odpowiedzialnością ich przechowywania w scentralizowanej lokalizacji albo nawet całkowitym brakiem dostępu do tych danych.

Federated Learning pozwala na bezpieczne dla użytkownika wykorzystanie jego prywatnych danych (zdjęć) w celu dotrenowania sieci neuronowych i w tym poprawy ich jakości. W tej pracy zostanie zbadana metoda ta metoda uczenia sieci neuronowych w implementacji systemu rozpoznawania twarzy systemu na urządzania IoT.

Główne kontrybucje tej pracy to 1) Implementacja i weryfikacja algorytmu `FederatedAveraging` dla zadań klasyfikacji obrazów oraz weryfikacji twarzy.

2. Rozproszone uczenie na urządzeniu końcowym

W tym rozdziale przedstawiony zastany projekt systemu, który korzystając z podejścia Federated Learning (FL) umożliwi na trening ekstraktora cech twarzy w rozproszonym środowisku urządzeń IoT.

Federated Learning Federated Learning (FL) jest podejściem zastosowania uczenia maszynowego w rozproszonym środowisku pozwalające na trening modeli on dużym zbiorze danym zdecentralizowanych prywatnych danych znajdujących się na urządzeniach końcowych użytkowników, a w szczególności urządzeniach Internetu Rzeczy. FL realizuje ideę "przyniesienie kodu do danych, zamiast danych do kodu"i adresuje fundamentalne problemu prywatności, własności i lokalności danych. Federated learning został opisany w (cite).

Problemy odpowiednie do zastosowania federated learningu moją następujące właściwości:

1) Trening na rzeczywistych danych gromadzonych na urządzaniach mobilnych dają znaczą przewagę nad treningiem na ogólnie dostępnych danych proxy dostępnych w centrach danych.

2) Te dane są prywatne albo są zbyt duże do przetrzymywania ich w centrach danych

3) Dla zadań nadzorowanych, etykiety danych powstają samoistnie z interakcji użytkownika z urządzeniem.

Cechy środowiska systemu Algorytmy optymalizacji mogące być zastosowane do optymalizacji na urządzaniach IoT mają kilka cech wyróżniających je od znanych już algorytmów rozproszonej optymalizacji:

- **Non-IID** Dane trenujące on danym urządzaniu są zazwyczaj zależne od konkretnego użytkownika i dlatego lokalny zbiór danych zebrany na dowolnym urządzeniu nie będzie reprezentatywny w stosunku do dystrybucji całej populacji
- **Niezbalansowany** Podobnie, niektórzy użytkownicy będą o wiele częściej korzystali z aplikacji aparatu niż inni, co będzie prowadziło do różnic w wielkości zebranych lokalnych zbiorów danych trenujących.
- **Masywnie rozproszony** Spodziewa się, że liczba finalnych użytkowników biorąca udział w optymalizacji będzie większa niż średnia liczba przykładów trenujących przypadająca na jednego klienta.
- **Ograniczona komunikacja** Urządzania IoT są pomimo założenia, że mają dostęp do internetu mogą być ograniczone wolnym albo kosztownym łączem sieciowym.

W tej pracy główna uwaga zostanie poświęcona na doprowadzenie systemu do działania w patologicznym przypadku środowisku danych Non-IID oraz ograniczonej i zawodnej komunikacji.

2.1. Projekt systemu

Implementowany system umożliwia trening głębokich sieci neuronowych on danych przetrzymywanych na bezpośrednio urządzaniu IoT. Dane te nigdy nie opuszczają samego urządzania. Wagi modelu są agregowane i łączone na serwerze znajdującym się w chmurze za pomocą algorytmu Federated Averaging konstruując nowy model globalny, który zostaje przesłany z powrotem do urządzeń końcowych do inferencji. System ten został opisany pierwotnie w (cytat) i z sukcesem zaaplikowany w implementacji inteligentnej klawiatury smartphone. (cite blogaska googla).

Przedstawienie architektury systemu zaczynamy od przedstawienia protokołu według, którego przebiega cały przepływ danych. Opisujemy to w następnej sekcji.

2.1.1. Protokół

Uczestnikami protokołu są urządzania Internetu Rzeczy oraz serwer, który jest usługą chmurową. Urządzania anonsują serwerowi swoją gotowość do uruchomienia zadania. Zadanie jest specyficzne dla populacji urządzeń i polega na lokalnym uruchomieniu obliczeń, takich jak trening z zadanymi przez serwer parametrami albo ewaluacja wytrenowanego modelu na lokalnych dla urządzenia danych.

Protokół dzieli komunikacje na rundy, z której każda zaczyna się od selekcji. Z potencjalnie tysięcy gotowych urządzeń w danym oknie czasowym, serwer wybiera ich podzbiór, który dostanie zaproszenia do wzięcia udziału obliczeniach. Liczebność tego podzbioru jest parametrem algorytmu opisywanego w kolejnym rozdziale

Serwer mówi wybranym urządzeniom jakiego typu obliczenia będą wykonywane. Wraz z tą informacją przesłany graf obliczeniowy modelu, zserializowane parametry globalnego modelu oraz pozostałe informacje niezbędne do uruchomienia obliczeń w danej rundzie. Następnie każdy z uczestników wykonuje lokalnie obliczenia bazując na globalnym stanie oraz swoim lokalnym zbiorze danych i przesyła wynik obliczeń z powrotem na serwer. W szczególności wynikiem, w przypadku rundy treningowej, będą wytrenowane parametry modelu lokalnego. Serwer wciela zebrane aktualizacje w swój stan globalny co kończy pojedyńczą rundę i proces ten się powtarza.

Opisywany protokół pozwala urządzeniom końcowym na poprawianie globalnego, pojedyńczego modelu pomiędzy rundami, z których każda składa się z trzech faz co zostało pokazane na rysunku (rysunek?).

Selekcja Cyklicznie, urządzenia, które spełniają wymagane kryteria (Patrz sekcja XXX) meldują serwerowi swoją gotowość. Serwer wybiera podzbiór z dostępnych mu urządzeń. W naszej implementacji wybrano prostą, losową selekcje z rozkładem jednostajnym N urządzeń z tych, które ogłosiły swoją gotowość.

Konfiguracja Serwer jest skonfigurowany na podstawie wybranego mechanizmu selekcji urządzeń oraz agregacji modeli. Urządzenia końcowe są samo-konfigurowane według

Algorytm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

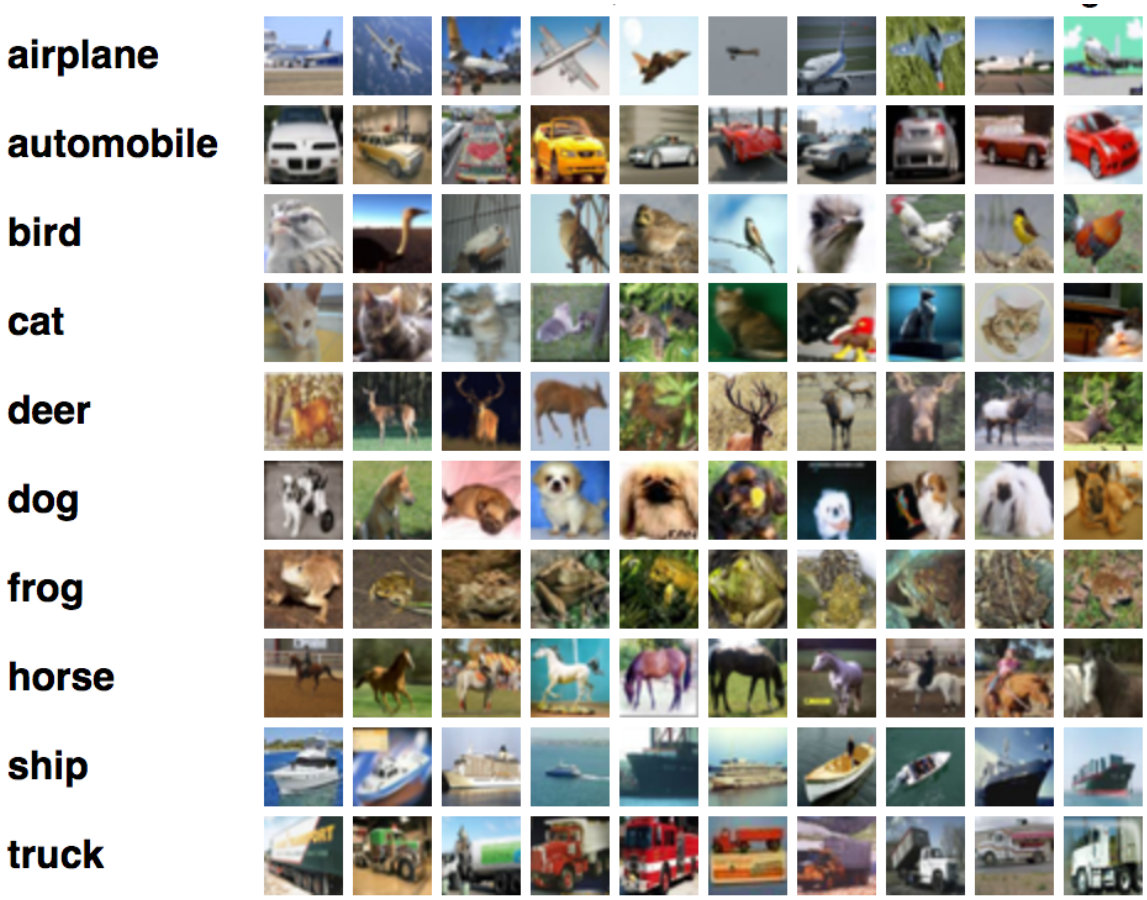
ClientUpdate (k, w): // *Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

przesłanej przez serwer informacji o typie obliczeń, grafu obliczeniowego oraz stanu globalnego.

Raportowanie Serwer czeka, aż partycypujące urządzenia zdadzą raport z wynikiem przeprowadzonych obliczeń. Jeśli runda była rundą treningową to wraz z otrzymaniem wszystkich zaktualizowanych modeli, serwer agreguje je z użyciem algorytmu Federated Averaging i informuje raportujące urządzenia kiedy mogą ponownie zgłosić swoją gotowość. Jeśli wystarczająca liczba urządzeń zaraportuje wyniki serwerowi w pewnym oknie czasowym, runda zostanie uznana za zakończoną z powodzeniem i zagregowany model zastąpi aktualny model globalny. W przeciwnym przypadku zebrane wyniki z tej rundy zostają porzucone. Protokół jest w pewnym stopniu odporny na to, że partycypujące urządzenia z pewnych powodów nie odpowiedzą na przesłaną konfiguracje albo nie zaraportują wyników.

Fazy selekcji i raportowania są dospecyfikowane przez zestaw parametrów. Podczas fazy selekcji serwer bierze pod uwagę pożądaną liczbę uczestników rundy, czasy time-outów oraz minimalną liczbę potrzebnych uczestników do rozpoczęcia rundy. Faza selekcji trwa dopóty dopóki nie zostanie wybrana docelowa liczba urządzeń albo nie skończy się czas przeznaczony na faze selekcji. W drugim przypadku runda zostanie rozpoczęta jeśli została zebrana minimalna liczba urządzeń, w przeciwnym przypadku runda zostaje porzucona. Faza raportowania parametryzowana jest przez minimalną liczbę urządzeń, które muszą zaraportować wynik.

2.2. Algorytm optymalizacji



Rysunek 1. 10 przykładowych obrazów dla każdej z 10 klas zbioru CIFAR10

Algorytm 1 opisany został zaimplementowany w języku Python. Do implementacji modeli neuronowych i algorytmów uczących został wykorzystany framework PyTorch [1]. Poprawność implementacji została sprawdzona na zadaniu klasyfikacji obrazów wykorzystując prostą sieć konwolucyjną oraz popularny zbiór danych CIFAR10.

Sieć konwolucyjna Jako obiekt treningu omawianego algorytmu zastała użyta niewielka sieć neuronowa zawierająca dwie warstwy konwolucyjną z filtrami o szerokości 5x5 (pierwsza z 32 kanałami, druga z 64, po każdej dodatkowa warstwa 2x2 max pooling), po których następuje dwu-warstwowy perceptron i na końcu warstwa przekształcenia liniowego, co daje w sumie 10^6 miliona parametrów. Model został podsumowany w tabeli

CIFAR10 CIFAR10 jest popularnym syntetycznym zbiorem danych. Zbiór danych składa się z 60 000 kolorowych obrazów podzielonych na 10 klas, z 6000 obrazami przypadającymi na jedną klasę. Zawarte są w nim obrazy o szerokości i wysokości 32 pikseli. Standordowo zbiór dzieli się na dwa zbalansowane klasowo podzbiory: testowy i treningowy zawierających odpowiednio 10000 i 50000 etykietowanych przykładów. Na rysunku 1 znajduje się 10 losowo wybranych obrazów, dla każdej z 10 klas.

Protokół treningowy Do sprawdzenia poprawności implementacji została zaimplementowana procedura treningowa wzorowana na [2]. Zbiór treningowy został podzielony pomiędzy 100 użytkowników tak żeby każdy zawierał po 500 przykładów trenujących. Z powodu braku naturalnego podziału danych na tak dużą liczbę klientów rozważany jest tutaj nieco mniej wymagający przypadek, w którym dane każdego użytkowania są zbalansowane oraz równomiernie rozdystrybuowane.

Naszym celem była maksymalizacja dokładności z jaką model klasyfikował obrazy pochodzące ze zbioru testowego.Badanie jakości końcowego modelu globalnego odbywało się już nie w sposób rozproszony, a na serwerze stosując cały dostępny zbiór testowy, co zostało umożliwione dzięki wprowadzonym uproszczeniom co do rozkładu danych.

Obrazy uległy standardowemu przetworzeniu wstępnemu i augmentacji. Przykłady trenujące zredukowano do wielkości 24x24 pikseli przez losowe obcięcie krawędzi, obrazy uległy losowemu horyzontalnemu odbiciu lustrzanemu oraz standardowej normalizacji.

Zaimplementowany algorytm został porównany do standardowego algorytmu SGD (cytat).

Ewaluacja

3. System weryfikacji użytkownika

Zadaniem systemu jest weryfikacja użytkownika, cz zostały przedstiawione w następnej sekcji.

3.1. Wstępne przetwarzanie obrazu

3.2. Weryfikacja użytkownika

Weryfikacja twarzy jest zadaniem przyrównania twarzy kandydata to innej, i weryfikacja czy nastąpiło ich dopasowanie. Jest to mapowanie jeden-do-jednego: należy sprawdzić czy jest to ta sama osoba.

3.2.1. Procedura weryfikacji

3.2.2. Ekstraktor cech

tutaj opisz jaki jest pipeline wchodzi zdjęcie i wychodzi embedding

3.2.3. Zbiory danych

Są jakieś zbiory daych 1) VggFace2 2) MS1M 3) MegaFace 4) LWF

Sprawdzić wpływ datasetu początkowego na wyniki ewaluacji początkowego modelu globalego

ms1m do pre trengowania ponieważ posiada duża różnorodność twarzy oraz ogólną duża liczbe przykładów trenujących. VggFace2 zostonie z koleji wykorzystany do koncowego

dotreniowania modelu - ze względu na dużą liczę zdjec przypadajaca na jedna osobę posiada dużą różnorodność wewnątrzklasową co powinno zwięszyć finałową wfektywaność podelu - większa odporność na rotację twarzy względem obiektywu, zmienne wartunki świetlne itp.

Ze względu na popularność dwóch ostatnich zbiorów w ewaluacji systemów rozpoznających twarze zostaną one właśnie wykorzystane do ewaluacji, MegaFace ma tę dodatkową zaletę, dużej liczby osób, a dwa pierwsze tylko do treningu.

3.2.4. Metody weryikacji twarzy

Wyjściem z modelu jest wektor(embedding), który pozwala na odróżnienie jednej twarzy od drugiej za pomocą porównanie embedingPapier googla twierdzi ze udało im sie uzyskać dobry score jednak nie rzadne inne prace nie raportują sukcesywngeo otrzymania podobnego wyniku. Może to wynikać z 1) braku zbliżonego wielkością zbioru danych użytych do uzyskania wyniku raportowangeo przez googla 2) brakiem podobnych zasobów obliczeniowychu jednej twarzy do embedingu drugiej twarzy. Coś o tym, że są tak jakby dwie rodziny algorytmów. Jedna bazująca na klasycznym podjeściu stosowanym podczas klasyfikacji obrazów i druga bazująca na optymalizacji multi-class classiication hindge loss(jak to do uja wafla przetłumaczyć?).

Papier googla twierdzi ze udało im sie uzyskać dobry score jePapier googla twierdzi ze udało im sie uzyskać dobry score jednak nie rzadne inne prace nie raportują sukcesywngeo otrzymania podobnego wyniku. Może to wynikać z 1) braku zbliżonego wielkością zbioru danych użytych do uzyskania wyniku raportowangeo przez googla 2) brakiem podobnych zasobów obliczeniowychdnak nie rzadne inne prace nie raportują sukcesywngeo otrzymania podobnego wyniku. Może to wynikać z 1) braku zbliżonego wielkością zbioru danych użytych do uzyskania wyniku raportowangeo przez googla 2) brakiem podobnych zasobów obliczeniowych

Tripletloss jest o wiele lepszy dla zastosowań federated learningu procedura trenowania

Classification loss Pokazać że jest kilka rodzajów takich lossów 1) Centre loss 2) Cosine loss 3) Arc loss

Implementacja Papier googla twierdzi ze udało im sie uzyskać dobry score jednak zadne inne prace nie raportują sukcesywngeo otrzymania podobnego wyniku. Może to wynikać z 1) braku zbliżonego wielkością zbioru danych użytych do uzyskania wyniku raportowangeo przez googla 2) brakiem podobnych zasobów obliczeniowych

Została podjęta próba implementacji wytrenowania modelu stosując w.w.y metodę wykorzystując zbiór danych VggFace2.

W celach pokazania, że implementacja jest poprawna porównujemy następujące modele

Porównanie: dwa poejścia: dwa datasety, modele niekoniecznie moje ale testowane na własnej ewaluacji

triplet(google report), wytrenowanych na danych prywatnych triplet(own), centre(external source) wytrenowanych na VggFace2 triplet(own), centre(external source) wytrenowanych na MS1m -> wyjdzie, że triplet loss jest o wiele mniej skuteczna

see this: https://arxiv.org/pdf/1901.08616.pdf https://arxiv.org/pdf/1709.02940.pdf

3.2.5. Rozporszona weryfikacja twarzy

porównanie:

centre loss ms1m

globalny model arcface uczony na ms1m, testowany na lwf i megaface globalny model arcface uczony na ms1m, dotrenowany tripletem na VggFace2, testowany

globalny model arcface dotrenowanie na lwf ale negatywne przykłady samplujemy z generatora globalny model arcface dotrenowanie na megaface ale negatywne przykłady samplujemy z generatora

rozporoszonej optymalizacji:

- **Non-IID** Dane trenujące on danym urządzniu są zazwyczaj zależne od konkretnego użytkownika i dlatego lokalny zbiór danych zebrany na dowolnym urządzeniu nie będzie reprezentatywny w stosunku do dystrybucji całej populacji
- **Niezbalansowany** Podobnie, niektórzy urzytkownicy będą o wiele częściej korzystali z aplikacji aparatu niż inni, co będzie prowadziło do różnic w wielkości zebranych lokalnych zbiorów danych trenujących.
- **Masywnie rozporszony** Spodziewa się, że liczba finalnych użytkowników biorąca udział w optymalizacji będzie większa niż średnia liczba przykładów trenujących przypadająca na jednego klienta.
- **Ograniczona komunikacja** Urządznia IoT są pomimo założenia, że mają dostęp do internetu mogą być ograniczone wolnym albo kosztownym łączem sieciowym.

W tej pracy główna uwaga zostanie poświęcona na rozwiązanie doprowadznie systemu do działania w środowisku danych Non-IID oraz ograniczonej komunkiacji.

Bibliografia

[1] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga i A. Lerer, **?**Automatic Differentiation in PyTorch**?**, w *NeurIPS Autodiff Workshop*, 2017.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson i B. A. y Arcas, *Communication-Efficient Learning of Deep Networks from Decentralized Data*, 2016. arXiv: 1602.05629 [cs.LG].

Wykaz symboli i skrótów

EiTI – Wydział Elektroniki i Technik Informacyjnych
PW – Politechnika Warszawska