УДК 004.41 + 004.02

РАСПРЕДЕЛЕННАЯ ТРЕНИРОВКА ML-МОДЕЛИ НА МОБИЛЬНЫХ УСТРОЙСТВАХ

Д. В. Симон[1], И. С. Шахова[2]

Высшая школа информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета

[1]denis.v.simon@gmail.com, [2]is@it.kfu.ru

Аннотация

В настоящее время потребность в наличии решений по распределенной тренировке ML-модели в мире возрастает. Однако существующие инструменты, в частности, TensorFlow Federated, – в самом начале своего развития, сложны в реализации и пригодны на текущий момент исключительно для симуляции на серверах. Для мобильных устройств надежно работающих подходов для достижения этой цели не существует. В статье спроектирован и представлен подход к такой распределенной тренировке ML-модели на мобильных устройствах, реализуемый с использованием существующих технологий. В его основе лежит концепция model personalization. В данном подходе эта концепция улучшена как следствие смягчения выявленных недостатков. Процесс реализации выстроен так, чтобы на всех этапах работы с ML-моделью использовать только один язык программирования Swift (применяются Swift for TensorFlow и Core ML 3), делая такой подход еще более удобным и надежным благодаря общей кодовой базе.

Ключевые слова: ML-модель, распределенная тренировка ML-модели, мобильная разработка, программная инженерия, машинное обучение, on-device ML, on-device training, edge computing.

ВВЕДЕНИЕ

Вместе с увеличивающимся количеством смартфонов и других мобильных устройств в мире становится больше и распределенных источников данных. Не всегда возможно или целесообразно передавать эти данные с устройств на сервер для их централизованной тренировки в рамках задач машинного обучения (ML). Причинами могут быть: необходимость инфраструктурных затрат на серверную часть, наличие требования поддержки оффлайн-работы или требования повышенной конфиденциальности пользовательских данных (например, биометрических, медицинских или финансовых). Также существуют законодательные ограничения на перемещения персональных данных в рамках инициатив по защите данных и их конфиденциальности (наиболее известным примером в Европе является GDPR[1]). В этой связи возрастает потребность в распределенной тренировке ML-модели на мобильных (edge) устройствах, так, чтобы пользовательские данные не покидали устройства. В то же время, для мобильных устройств надежно работающих подходов для достижения этой цели до сих пор не существует. Технология Federated Learning [1], а именно, ее основной инструмент TensorFlow Federated[2], мог бы быть решением, но он – в самом начале своего развития, сложен в реализации, требует одновременного применения нескольких других новых технологий [2, с. 1–2], и пригоден на текущий момент исключительно для симуляции на серверах.

Ниже представлен подход к такой распределенной тренировке ML-модели на мобильных устройствах, реализуемый на существующих технологиях и готовый для использования. В его основе лежит концепция model personalization [3, 4]. В данном подходе эта концепция улучшена как следствие смягчения выявленных недостатков. Процесс реализации выстроен так, чтобы на всех этапах работы с ML-моделью использовать только один язык программирования Swift (применяются Swift for TensorFlow[3] и Core ML 3[4]), делая такой подход еще более удобным и надежным благодаря общей кодовой базе.

[1] https://eur-lex.europa.eu/eli/reg/2016/679/oj

[2] https://www.tensorflow.org/federated

[3] https://www.tensorflow.org/swift

[4] https://developer.apple.com/machine-learning/core-ml/

1. ОБЗОР СФЕРЫ ON-DEVICE TRAINING

Поддержка ML на устройстве (on-device ML) может повысить интеллектуальность, безопасность и производительность мобильных приложений, а также снизить расходы на создание мобильного продукта [5]. Одним из вариантов применения on-device ML с недавнего времени являются не только запуск ML-модели на устройстве для получения предсказаний – on-device inference, но и тренировка ML-модели на устройстве – on-device training. Это предоставляет совершенно новые возможности для адаптации машинного обучения во многих случаях. Яркими примерами могут являться мобильные проекты, связанные с медициной или анализом биометрических данных.

Таким образом, при наличии требований, связанных с эффективностью (требуется возможность оффлайн-работы, недопустима задержка при обращении на сервер, недопустим риск сбоя на стороне сервера) и конфиденциальностью (недопустимо перемещение чувствительных персональных данных с устройства, а затем их хранение на сервере, что влечет за собой высокий риск их утечки), распределение процесса тренировки ML-модели предпочтительнее централизованного сбора данных в облаке и применения традиционных конвейеров ML.

1.1. FEDERATED LEARNING

Одним из многообещающих подходов в области on-device training является Federated Learning. Если такой подход будет реализован полноценно и качественно, это позволит мобильным устройствам объединиться для совместной работы и обучения общей модели без обмена необработанными тренировочными данными. Суть технологии состоит в том, что мобильные устройства отправляют на сервер не пользовательские данные, а параметры натренированной модели на свою часть данных, которые на сервере объединяются и усредняются [1]. Усредненные параметры затем отправляются снова всем клиентам для обновления их моделей [6–8]. Этим могут быть достигнуты приватность пользовательских данных и экономия средств (не требуются значительные инфраструктурные расходы на серверную часть), при одновременном сохранении высокого уровня интеллектуальности ML-модели, такого же, как и при централизованной тренировке на сервере. Однако у Federated Learning есть и недостатки:

- на данный момент не готова к продакшн-использованию; основной инструмент в этой области TensorFlow Federated находится на самой начальной стадии разработки и даже на серверах (не говоря уже о мобильных устройствах) работает только в режиме симуляции;
- сложна в реализации:
    - в части синхронизации сессий тренировок ML-моделей с высокой вероятностью возможны случаи, когда только небольшая часть клиентов будет доступна для текущей сессии тренировки, так же, как и случаи, когда некоторые клиенты по каким-то причинам не пришлют на сервер обновленные параметры локальной модели, и т. п.;
    - в части безопасной работы – требуется одновременное применение нескольких других новых технологий, таких, как Secure Multi-Party Computation (MPC) и Differential Privacy [2, с. 1–2];
- в рамках работы в связке с мобильными устройствами недостатком является также то, что TensorFlow Federated (как и PySyft, о котором речь пойдет дальше) использует Python. Если пре-тренированная модель создается с использованием Python, а после поставки на устройства пользователей дотренировывается на их новых пользовательских данных и запускается уже с использованием Swift (iOS-приложения) или Kotlin/Java (Android-приложения), то могут возникать неточности и сбои в работе ML-модели, поскольку кодовая база по пре- и пост-процессингу данных, а также их фичеризации будет отличаться;
- ограничение по алгоритмам: могут работать только те из них, которые используют Stochastic Gradient Descent, преимущественно DNNs, т. к. именно параметры Stochastic Gradient Descent и обновляются в модели на клиенте, а затем агрегируются на бэкенде.

Возможная альтернатива TensorFlow Federated – PySyft [9], но этот инструмент еще в меньшей степени доработан, безопасен и готов к работе, чем TensorFlow Federated. В дисклеймере его репозитория на GitHub[5] до сих пор (разработка

[5] https://github.com/OpenMined/PySyft

началась в июле 2017) содержится фраза "Do NOT use this code to protect data (private or otherwise) – at present it is very insecure. Come back in a couple months".

В 2019 авторы статьи [2] охарактеризовали состояние этой технологии следующим образом: «Существующие федеративные подходы к обучению не являются надежными в том смысле, что некоторые выбросы могут вызвать расхождение алгоритма обучения. Использование таких алгоритмов в возрастающем масштабе для таких задач, как умные клавиатуры на мобильных устройствах, несет угрозу безопасности сервиса и его пользователей».

1.2. MODEL PERSONALIZATION

В июне 2019 года, вместе с релизом Core ML 3, Apple анонсировала концепцию model personalization [3, 4], когда у каждого пользователя iOS-приложения есть ML-модель, которая тренируется на мобильном устройстве, используя данные пользователя. Core ML 3 позволяет регулярно обновлять модель путем ее точной настройки и перетренировки для конкретных данных пользователя, что помогает моделям оставаться актуальными для поведения пользователя без ущерба для конфиденциальности (пользовательские данные не покидают устройство).

Подробное описание процесса модификации/дотренировки обновляемой ML-модели формата .mlmodel содержится в документации по Core ML [10].

Некоторые кейсы для подобной персонализации:

- анализ биометрических данных на предмет аномалий;
- рекомендация ответа на сообщение в мессенджере на основе предыдущих ответов пользователя;
- Face ID использует эти методы, чтобы узнать, как выглядит владелец телефона, и поддерживать его модель в актуальном состоянии, когда его лицо меняется со временем (отращивание бороды, ношение макияжа и т. д.);
- голосовое управление в приложении, когда модель регулярно учится голосу пользователя и его интонациям, при этом помимо набора предустановленных команд, пользователь может задавать свои кастомные команды;

- тегирование фотографий пользователя из приложения Photos, когда пользователь может создать свои теги (например, имя своего домашнего питомца), и модель тренируется для классификации соответствующих фотографий, которые затем перемещаются в соответствующие альбомы.

Недостатки данной концепции:

- часто слишком мало данных, чтобы модель на их тренировке получилась качественной, выдавала качественные предсказания; пользователь может редко создавать новые данные для дотренировки (зависит от приложения);
- персонализированная модель, на тренировку которой затрачивалось много времени (как со стороны пользователя, так и системы), с устройства может по неосторожности пользователя легко пропасть, например, при переустановке приложения или смене устройства, если при этом пользователь забывает доступ к AppleID и/или не пользуется iCloud; на данный момент ни Apple, ни кем другим не предусмотрено решение, как этого надежно избежать.

2. НОВЫЙ ПОДХОД К РАСПРЕДЕЛЕННОЙ ТРЕНИРОВКЕ НА МОБИЛЬНЫХ УСТРОЙСТВАХ

В основе проектируемого подхода к распределенной тренировке ML-модели на мобильных устройствах лежит концепция transfer learning [11, 12], когда на «серверной части» (сервере, в облаке или на компьютере) создается модель, пре-тренированная на общих прокси-данных, которая затем поставляется вместе с приложением на устройства пользователей и там дотренировывается на их локальных, конфиденциальных данных. Apple называет это model personalization [3, 4], и такая возможность доступна в фреймворке Core ML 3, начиная с iOS 13.

Если для модели с заранее определенным набором классов/лейблов никаких трудностей в создании пре-тренированной модели быть не должно (на каждый класс собираются определенное количество соответствующих данных), то в случае модели с возможностью добавления пользователем любых кастомных классов/лейблов, как в примере про тегирование фотографий, приведенном выше, в качестве пре-тренированной модели можно взять, например, SqueezeNet

1.1[6], которая будет являться первым звеном в ML-pipeline (помимо второго – самого классификатора) для цели извлечения признаков из изображений, подающихся на вход модели.

В отличие от стандартной реализации model personalization представляемый нами подход является цикличным, т. е. предполагается, что пре-тренированная модель будет со временем регулярно дополняться новыми тренировочными данными, улучшаться и поставляться снова на все клиенты с новой версией приложения. Таким образом, пользовательские данные для тренировки должны храниться на устройстве пользователя, и при получении обновленной пре-тренированной модели приложение дотренировывает ее локальными данными. Для реализации этого предусмотрен механизм с использованием базы данных Realm[7], когда каждая единица пользовательских данных помечаются как уже прошедшая тренировку поверх пре-тренированной модели или еще нет (true/false), а при поступлении с новой версией приложения новой пре-тренированной модели все значение данного поля сбрасываются к false.

Подход также включает элементы online learning [13], когда пользователь может создавать новые тренировочные данные (явно или это происходит в фоне), с которыми модель дотренировывается и обновляется практически в реальном времени. Триггерами к началу сессии обновления модели, при наличии новых тренировочных данных, могут являться, например, выход приложения в бэкграунд или переход устройства в спящий режим (в зависимости от сложности модели и требуемых ресурсов на ее тренировку). После этого получение предсказаний будет сразу же осуществляться с уже обновленной моделью.

Также представляемый подход лишен выявленных недостатков model personalization, а именно, будет содержать следующие решения:

- on-the-fly data augmentation позволит «на лету» множить локальные тренировочные данные, модифицируя их по определенному алгоритму непосредственно во время сессии дотренировки модели на устройстве пользо-

[6] https://github.com/DeepScale/SqueezeNet

[7] https://realm.io/products/realm-database/

вателя; это позволит увеличить количество данных на один клиент и качество работы его модели, при этом не потребуется дополнительное место на диске устройства; множиться таким образом могут данные любых типов, включая текст, изображения и аудио.

- вводится термин model id, и на его основе выстраивается облачный сервис для хранения и синхронизации бэкапов моделей пользователей; бэкап производится и отправляется в облако автоматически после каждого обновления модели; свою модель пользователь сможет в любое время в дальнейшем получить на любом устройстве; для реализации используются Cloudflare, Google Cloud Functions и Firebase.

Наконец, представляемый подход будет решать проблему, выявленную при анализе Federated Learning (см. п. 1.1), а именно, то, что довольно рискованно писать код для тренировки пре-тренированной модели на Python, а код для тренировки на устройстве и запуска модели – на Swift. Любые, даже небольшие различия в коде или неоднозначная его интерпретация компилятором или интерпретатором разных языков программирования могут повлечь сбои в работе ML-модели и снижение качества предсказаний.

Данный подход предполагает использование Swift for TensorFlow с целью организации работы с ML-моделью на всех этапах на Swift. Таким образом, весь код по тренировке пре-тренированной модели (включая код по препроцессингу и фичеризации тренировочных данных), экспорту ее параметров в формат Google Protocol Buffer (protobuf)[8], редактированию параметров модели (в т. ч. чтобы сделать ее обновляемой (updatable)) и генерации итоговой модели в формате .mlmodel для последующего импорта в Core ML происходит на Swift. Тот же самый код по препроцессингу и фичеризации тренировочных данных можно снова использовать при написании кода для дотренировки и запуска модели на устройстве.

Также Swift for TensorFlow предоставляет наиболее гибкие возможности для работы с нейронными сетями, включая DNNs.

[8] https://developers.google.com/protocol-buffers

На рис. 1 представлен весь процесс распределенной тренировки ML-модели, в котором тренировка на одну часть данных происходит на «серверной части» (для этих целей могут использоваться сервер, облако или компьютер), а на другую часть – на мобильных клиентах. Сессии тренировки модели на новых пользовательских данных, так же, как и запуск модели для получения предсказаний, происходят через Core ML 3.

Поскольку Swift for TensorFlow можно использовать на сервере или в облаке (Linux) не только на macOS, то становится возможным автоматизировать создание пре-тренированной модели, т. е. с шага «Swift for TensorFlow для создания пре-тренированной модели» по шаг «Пересоздание модели в формате .mlmodel», но эта задача не является первостепенной и не рассматриваться в рамках настоящей статьи.

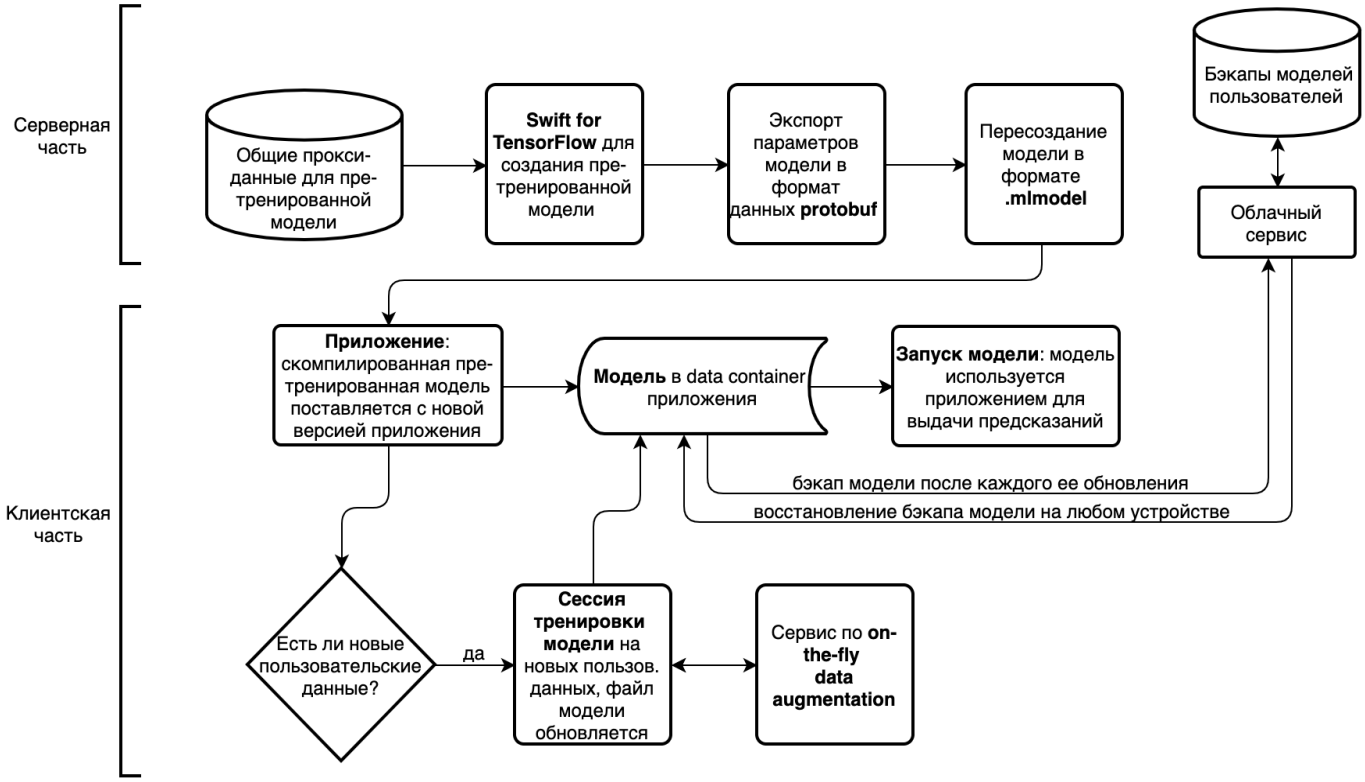

Рисунок 1. Процесс распределенной тренировки ML-модели

2.1. СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ ДАННЫХ

С целью ускорить сбор данных для пре-тренированной модели для улучшения ее качества опционально и при определенных условиях возможно дополнить приведенный выше процесс следующим образом (рис. 2):

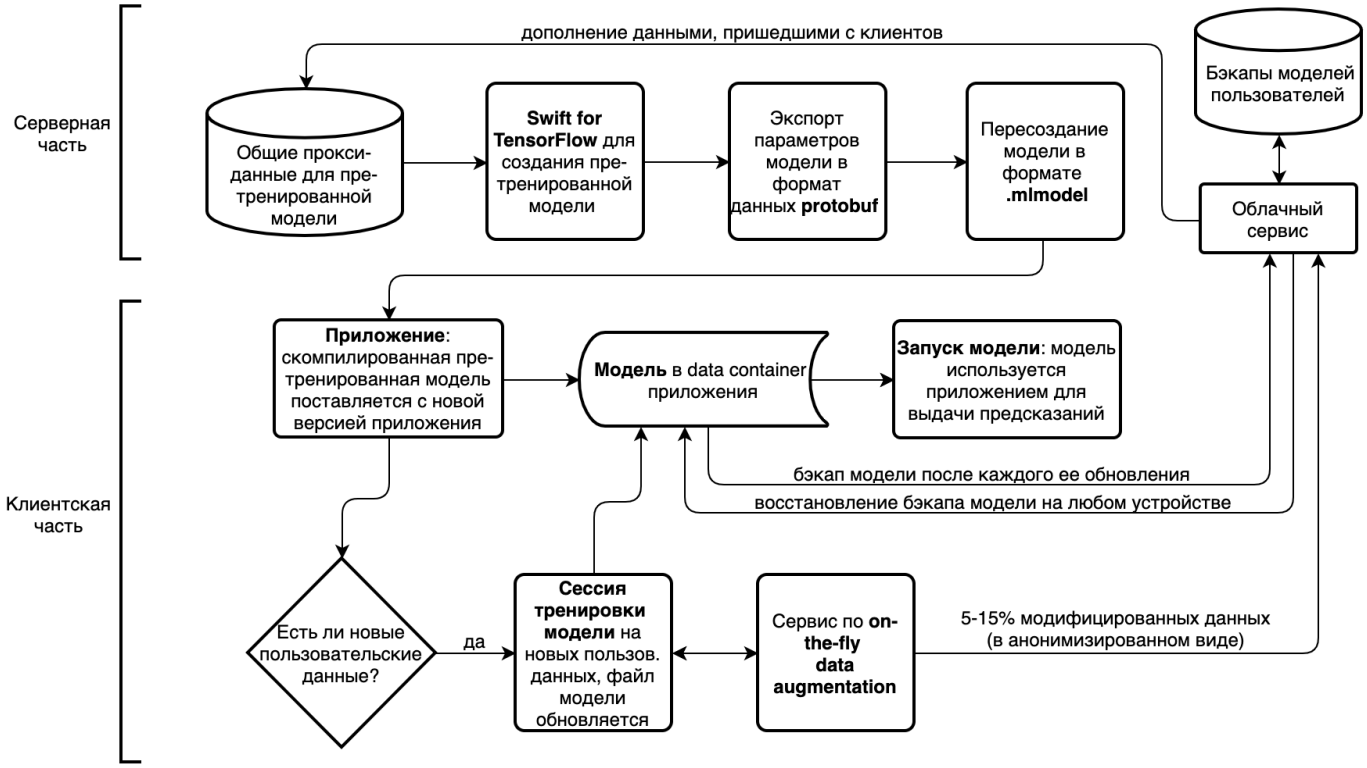

Рисунок 2. Процесс распределенной тренировки ML-модели с шарингом данных

Схема дополнена связями от «Сервис по on-the-fly data augmentation» к «Облачный сервис» и от «Облачный сервис» к «Общие прокси-данные для пре-тренированной модели».

Условия могут быть следующими:

- если пользовательские данные не настолько чувствительные (например, не медицинские, биометрические или финансовые);
- пользовательские данные были модифицированы в процессе работы сервиса "on-the-fly data augmentation" (оригиналы не передаются с

устройства); например, для текстовых данных изменяются существительные/глаголы на синонимы из WordNet[9]; для изображений применяются фильтры размытия, цветокоррекции и др., а также изменения масштаба/угла поворота; для аудио – изменение тональности, скорости, громкости, а также наложение шума и др.;

- пользовательские данные перед передачей на сервер анонимизируются [14, 15]; пример для текстовых данных: с помощью техник NLP определяются части речи слов и находятся все существительные (как вариант, также все именованные сущности), после чего найденные слова хешируются по определенному алгоритму; пример для фото данных: распознается область лица (в iOS для этого применяется фреймворк Vision[10]), которая затем частично размывается, или на нее накладывается полупрозрачный слой;
- передаются не более 5–15% от всех модифицированных пользовательских данных так, чтобы в целом по-прежнему сохранялась высокая их конфиденциальность.

Принимая решение по опции частичного шаринга данных, необходимо вносить изменения на уровне всей системы, а именно, проводить и тренировку, и хранение данных (в анонимизированном виде) как на клиентах, так и на серверной части. Существует достаточное количество статей на тему такой тренировки (включая [14, 15]), показывающих, что при этом качество итоговой модели и точность предсказаний остаются на высоком уровне.

ЗАКЛЮЧЕНИЕ

В статье представлен спроектированный подход к распределенной тренировке ML-модели на мобильных устройствах, реализуемый на базе существующих технологиях, прежде всего, Core ML 3 и Swift for TensorFlow, и готовый для использования. Тренировка на одну часть данных происходит на «серверной части» (на сервере, в облаке или на компьютере), а на другую часть – на мобильных

[9] https://wordnet.princeton.edu/

[10] https://developer.apple.com/documentation/vision

клиентах. Получаемая на серверной части пре-тренированная модель регулярно поставляется на клиенты с новыми версиями приложения и там, при появлении новых пользовательских данных, дотренировывается. Как следствие, модель также персонализируется.

Процесс такой тренировки показан на двух блок-схемах, основной его вариант не предусматривает шаринг данных с серверной частью, в то время как опциональный вариант предусматривает частичный шаринг данных с сохранением общего высокого уровня конфиденциальности данных. Кроме того, процесс включает улучшения концепции model personalization как следствие выявленных недостатков.

Одной из ключевых составляющих представленного подхода также является такая организация процесса распределенной тренировки ML-модели, при которой на всех этапах работы с ML-моделью используется только один язык программирования Swift, что делает такой подход еще более удобным и надежным благодаря общей кодовой базе.

Ограничениями данного подхода являются следующие:

- прежде всего, он предназначен для платформы iOS (подход может работать и на Android при наличии аналогичных инструментов, таких, например, как Core ML 3, который позволяет осуществлять тренировку модели на устройстве);
- на данный момент времени Core ML для цели тренировки на устройстве поддерживает следующие типы моделей: классификаторы k-Nearest Neighbor и нейронные сети (включая классификатор, регрессор и общего назначения; более 100 вариантов слоев) [11, 16].

СПИСОК ЛИТЕРАТУРЫ

1. *Brendan McMahan and Daniel Ramage.* Federated Learning: Collaborative Machine Learning without Centralized Training Data. 2017. URL: https://ai.google-blog.com/2017/04/federated-learning-collaborative.html (дата обращения: 03.03.2020)

2. Krishna Pillutla at all. Robust Aggregation for Federated Learning. 2019. arXiv:1912.13445 [stat.ML]

3. *What's New in the iOS SDK* // Apple. 2019. URL: https://developer.apple.com/ios/whats-new/ (дата обращения: 03.03.2020)

4. Apple. What's New in Core ML 3. 2019. URL: https://developer.apple.com/machine-learning/core-ml/ (дата обращения: 03.03.2020)

5. *MIT Technology Review. From cloud to the edge: On-device artificial intelligence boosts performance*. 2019. URL: https://www.technologyreview.com/s/613527/from-cloud-to-the-edge-on-device-artificial-intelligence-boosts-performance/ (дата обращения: 03.03.2020)

6. *B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas.* Communication-Efficient Learning of Deep Networks from Decentralized Data. In Artificial Intelligence and Statistics, 2017. P. 1273–1282.

7. *K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth.* Practical Secure Aggregation for Privacy-Preserving Machine Learning. In ACM SIGSAC Conference on Computer and Communications Security, стр. 1175–1191, 2017.

8. *Get Started with Federated Learning for Data privacy* // Leapfrog. 2019. URL: https://www.lftechnology.com/blog/ai/federated-learning-data-privacy/ (дата обращения: 03.03.2020)

9. Theo Ryffel at all. A generic framework for privacy preserving deep learning. 2018. arXiv:1811.04017 [cs.LG]

10. *Personalizing a Model with On-Device Updates* // Apple. 2019. URL: https://developer.apple.com/documentation/coreml/core_ml_api/personalizing_a_model_with_on-device_updates (дата обращения: 03.03.2020)

11. *Matthijs Hollemans.* On-device training with Core ML – part 1. 2019. URL: https://machinethink.net/blog/coreml-training-part1/ (дата обращения: 03.03.2020)

12. *Jason Brownlee.* A Gentle Introduction to Transfer Learning for Deep Learning. 2017. URL: https://machinelearningmastery.com/transfer-learning- for-deep-learning/ (дата обращения: 03.03.2020)

13. *Alexander Rakhlin.* Online Methods in Machine Learning. 2016. URL: http://www.mit.edu/~rakhlin/6.883/

14. *Martijn Willemsen.* Anonymizing Unstructured Data to Prevent Privacy Leaks during Data Mining. 2016. URL: https://www.semanticscholar.org/paper/ Anonymizing-Unstructured-Data-to-Prevent-Privacy-Willemsen/40781ab4856f3d50af8ecda8f9aa1851c2e027eb (дата обращения: 03.03.2020)

15. *Adam Drake*. Scalable Machine Learning with Fully Anonymized Data. 2018. URL: https://adamdrake.com/scalable-machine-learning-with-fully- anonymized-data.html (дата обращения: 03.03.2020)

16. *Analytics Vidhya. Introduction to Apple's Core ML 3 – Build Deep Learning Models for the iPhone. 2019*. URL: https://www.analyticsvidhya.com/blog/2019/11/introduction-apple-core-ml-3-deep-learning-models-iphone/ (дата обращения: 03.03.2020)

DISTRIBUTED TRAINING OF ML MODEL ON MOBILE DEVICES

Denis Simon[1], Irina Shakhova[2]

Higher Institute of Information Technology and Intelligent Systems, Kazan Federal University

[1]denis.v.simon@gmail.com, [2]is@it.kfu.ru

Abstract

Currently, the need for distributed ML training solutions in the world is increasing. However, existing tools, in particular TensorFlow Federated, are at the very beginning of their development, difficult to implement, and currently suitable exclusively for simulation on servers. For mobile devices, reliable approaches for this purpose do not exist. This article has designed and presented an approach to such distributed training of the ML-model on mobile devices, implemented on existing technologies. It is based on the concept of model personalization. In this approach, this concept is improved as a consequence of mitigating the identified drawbacks. The implementation process is structured so that at all stages of working with the ML-model use only one Swift programming language (Swift for TensorFlow and Core ML 3 are used), making this approach even more convenient and reliable due to the common code base.

Keywords: ML-model, distributed training of an ML model, mobile development, software engineering, machine learning, on-device ML, on-device training, edge computing.

REFERENCES

1. *Brendan McMahan and Daniel Ramage.* Federated Learning: Collaborative Machine Learning without Centralized Training Data. 2017. URL: https://ai.google-blog.com/2017/04/federated-learning-collaborative.html

2. Krishna Pillutla at all. Robust Aggregation for Federated Learning. 2019. arXiv:1912.13445 [stat.ML]

3. *What's New in the iOS SDK* // Apple. 2019. URL: https://developer.apple.com/ios/whats-new/

4. Apple. What's New in Core ML 3. 2019. URL: https://developer.apple.com/machine-learning/core-ml/

5. *MIT Technology Review. From cloud to the edge: On-device artificial intelligence boosts performance*. 2019. URL: https://www.technologyreview.com/s/613527/from-cloud-to-the-edge-on-device-artificial-intelligence-boosts-performance/

6. *B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas.* Communication-Efficient Learning of Deep Networks from Decentralized Data. In Artificial Intelligence and Statistics, 2017. P. 1273–1282.

7. *K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth.* Practical Secure Aggregation for Privacy-Preserving Machine Learning. In ACM SIGSAC Conference on Computer and Communications Security, pp. 1175–1191, 2017.

8. *Get Started with Federated Learning for Data privacy* // Leapfrog. 2019. URL: https://www.lftechnology.com/blog/ai/federated-learning-data-privacy/

9. Theo Ryffel at all. A generic framework for privacy preserving deep learning. 2018. arXiv:1811.04017 [cs.LG]

10. *Personalizing a Model with On-Device Updates* // Apple. 2019. URL: https://developer.apple.com/documentation/coreml/core_ml_api/personalizing_a_model_with_on-device_updates

11. *Matthijs Hollemans.* On-device training with Core ML – part 1. 2019. URL: https://machinethink.net/blog/coreml-training-part1/

12. *Jason Brownlee.* A Gentle Introduction to Transfer Learning for Deep Learning. 2017. URL: https://machinelearningmastery.com/transfer-learning- for-deep-learning/

13. *Alexander Rakhlin.* Online Methods in Machine Learning. 2016. URL: http://www.mit.edu/~rakhlin/6.883/

14. *Martijn Willemsen.* Anonymizing Unstructured Data to Prevent Privacy Leaks during Data Mining. 2016. URL: https://www.semanticscholar.org/paper/ Anonymizing-Unstructured-Data-to-Prevent-Privacy-Willemsen/40781ab4856f3d50af8ecda8f9aa1851c2e027eb

15. *Adam Drake*. Scalable Machine Learning with Fully Anonymized Data. 2018. URL: https://adamdrake.com/scalable-machine-learning-with-fully- anonymized-data.html

16. *Analytics Vidhya. Introduction to Apple's Core ML 3 – Build Deep Learning Models for the iPhone. 2019*. URL: https://www.analyticsvidhya.com/blog/2019/11/introduction-apple-core-ml-3-deep-learning-models-iphone/

СВЕДЕНИЯ ОБ АВТОРАХ

СИМОН Денис Васильевич – магистрант Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета, iOS-разработчик.

Denis Vasilyevich SIMON – postgraduate student of the Higher School of Information Technologies and Intelligent Systems at Kazan Federal University, iOS developer.

email: denis.v.simon@gmail.com

ШАХОВА Ирина Сергеевна – старший преподаватель кафедры программной инженерии Высшей школы информационных технологий и интеллектуальных систем Казанского федерального университета. Сфера научных интересов – цифровые образовательные системы, индивидуализация образования, мобильное обучение.

Irina Sergeevna SHAKHOVA – senior teacher of the Higher School of Information Technologies and Intelligent Systems at Kazan Federal University. Research interests include digital educational systems, individualization of education, mobile learning.

email: is@it.kfu.ru

Материал поступил в редакцию 2 апреля 2020 года