Capítulo 5

Aprendizado Federado aplicado à Internet das Coisas

Heitor S. Ramos (UFMG), Guilherme Maia (UFMG), Gisele L. Papa (UFMG), Mário S. Alvim (UFMG), Antonio A. F. Loureiro (UFMG), Isadora Cardoso-Pereira (UFMG), Diego H. C. Campos (UFMG), Giovanna Filipakis (UFMG), Giovanna Riquetti (UFMG), Eduarda T. C. Chagas (UFMG), Pedro H. Barros (UFMG), Gabriel N. Gomes (UFMG), Héctor Allende-Cid (PUC-Valparaíso)

Abstract

The main objective of this short course is to present the main fundamentals of Federated Learning (FL), covering the tools and steps necessary for the development of applications and services aimed at the Internet of Things (IoT). The concepts covered during this short course include introducing Machine Learning (centralized and distributed), the state-of-the-art in FL, an overview of existing work, challenges, and future perspectives for advancing the field. FL is a recent research field that still has a lot to be explored. Therefore, it has research fronts that remain open, with non-trivial challenges. Thus, this short course intends to deepen the following questions: (i) How does FL differ from centralized and decentralized Machine Learning? (ii) What are the main characteristics of FL that enable the development of applications in the context of IoT. (iii) What are the technological challenges for implementing FL? (iv) What are the main methodologies used for the development of IoT applications based on FL? (v) What are the main LF research problems? (vi) What are the representative applications of this area?

Resumo

O objetivo principal deste minicurso é apresentar os principais fundamentos do Aprendizado Federado (Federated Learning – FL), abrangendo as ferramentas e passos necessários para o desenvolvimento de aplicações e serviços voltados à Internet das Coisas (Internet of Things – IoT). Os conceitos abordados durante o presente minicurso incluem uma introdução à Aprendizado de Máquina (centralizada e distribuída), o estado-da-arte

em FL, uma visão geral dos trabalhos existentes, os desafios e as perspectivas futuras para o avanço da área. FL é um campo de pesquisa recente e que ainda possui muito a ser explorado. Assim sendo, possui frentes de pesquisa que seguem em aberto, com desafios não triviais. Dessa forma, este minicurso pretende aprofundar as seguintes questões: (i) Como FL se diferencia do Aprendizado de Máquina centralizado e descentralizado? (ii) Quais as principais características de FL que viabilizam o desenvolvimento de aplicações no contexto de IoT. (iii) Quais os desafios tecnológicos para a implantação de FL? (iv) Quais as principais metodologias utilizadas para o desenvolvimento de aplicações de IoT baseadas em FL? (v) Quais são os principais problemas de pesquisa de FL? (vi) Quais são as aplicações representativas desta área?

5.1. Introdução a Aprendizado Federado no Contexto de IoT

Dispositivos móveis como *smartphones*, dispositivos vestíveis e veículos autônomos são exexemplos de sistemas distribuídos que geram uma grande quantidade de dados diariamente. Esses dispositivos têm habilitado técnicas de inteligência computacional, mais especificamente aprendizado de máquina (ML), a produzirem aplicações cada vez mais atrativas e poderosas como visão computacional, processamento de linguagem natural, sistemas de recomendação, reconhecimento de fala, para citar algumas. O sucesso de várias aplicações de ML, especialmente o aprendizado profundo (DL), tem sido alavancado pela disponibilização de grandes quantidades de dados para viabilizar o treinamento e a avaliação adequada desses modelos.

Utilizando-se desses dados, as técnicas avançadas de ML viabilizam aplicações que muitas vezes superam a capacidade humana, por exemplo, os sistemas de detecção de face modernos conseguem reconhecer uma quantidade grande de indivíduos que talvez nenhum ser humano sozinho consiga fazê-lo. Por exemplo, há relatos de que o sistema de detecção de objetos do Facebook tem sido treinado com pelo menos 3,5 bilhões de imagens oriundas do Instagram [Yang et al. 2019a].

Apesar de estarmos vivendo na era do chamado *big data*, vários autores [Yang et al. 2019a] têm relatado que na prática, na maioria das vezes, temos pequenos volumes de dados distribuídos em uma grande quantidade de entidades, sejam corporações ou indivíduos. Muitas vezes, esses dados não estão adequadamente preparados, por exemplo, não estão devidamente rotulados, ou apresentam muitos dados faltatantes, devido ao custo associado a ter-se esse tipo de qualidade de dados. Por exemplo, é comum termos na área de saúde, diversas bases de dados sem a devida preparação por conta dessa tarefa ser associada a um alto custo de ter um especialista analisando e anotando esses dados. Como resultado disso, frequentemente encontramos o que é conhecido por silos de dados (*data silos*) que não são facilmente conectados para produção de modelos de ML adequados.

Deve-se levar em conta também que o modo como os dados são utilizados no treinamento de ML pode infringir algumas leis locais voltadas para a privacidade dos usuários, como por exemplo: a *General Data Protection Regulation* (GDPR), da União Européia, que entrou em vigor em 2018 e tem como algumas das normas a transparência, a não coleta de dados desnecessários para o propósito, proteção contra ataques aos dados, e, caso seja requisitado, ter dados imediatamente apagados do sistema; e a le-

gislação dos Estados Unidos, em especial a da Califórnia (*California Consumer Privacy Act*, ou CCPA), que entre suas regras permite aos usuários barrarem que seus dados sejam vendidos por companhias [Yang et al. 2019a]. No Brasil, temos também a Lei Geral de Proteção de Dados (LGPD), promulgada em 14 de agosto de 2018 e que tem como princípio o respeito à privacidade, a invioabilidade da intimidade, da honra e da imagem, dentre outras coisas. Dessa maneira, o uso dos dados deve sempre ter a preocupação de não expor a privacidade e a intimidade dos indivíduos que estão sujeitos a terem seus dados expostos. Iremos abordar com mais detalhes a questão de privacidade dos dados em ambientes de FL na Seção 5.3.

Os métodos de ML que estão sendo disseminados rapidamente, terão que se adequar a um ambiente em que os dados existem, porém, estão espalhados em um grande sistema distribuído. Além disso, transferir os dados desse sistema distribuído para uma entidade central que irá treinar os modelos de ML, pode não ser uma opção viável, seja por questões relativas à privacidade e regulações ou até mesmo do alto custo da transferência de grandes quantidades de dados em redes que podem não se adequar a esse cenário.

A Internet das Coisas (IoT) é uma das tecnologias viabilizadoras da geração de grandes quantidades de dados. Tanto a IoT mais voltada para uso individual, como vigilância patrimonial, aplicações de conforto, saúde (*e-health*), dentre outras, como as aplicações de IoT industriais/corporativas como monitoramento de chão de fábrica, gestão patrimonial e monitoramento de equipamentos, têm um enorme potencial para viabilizar aplicações avançadas de ML. Entretanto, é uma tecnologia que tem potencial para expor dados privados de usuários e também de não permitir a comunicação desse grande volume de dados em dispositivos que podem esgotar bateria ou serem equipados por redes sem fio que, por usarem canal compartilhado, não são muito adequadas para essa grande transferências de dados. Por outro lado, não está claro que as corporações que utlizam IoT industrial/corporativa irão ter disponibilidade para compartilhar dados, afinal, a possibilidade de perder o controle sobre dados estratégicos da empresa ou dados privados dos clientes é um dificultador para o modelo mais usual em que os dados são compartilhados.

O Aprendizado Federado (*Federated Learning – FL*) surge como uma solução natural para o problema de treinar modelos de ML em ambientes que apresentam alta fragmentação de dados sem violar as diretrizes das regulações vigentes nos países que estão se voltando para a proteção da privacidade do indivídulo. FL é um modelo de ML distribuída onde a privacidade do dado é uma premissa essencial. Dessa maneira, estamos interessados em construir modelos de ML (sejam preditivos ou descritivos) em que o compartilhamento de dados não seja uma opção. A ideia geral, é treinar um modelo em cada usuário, utilizando os dados locais dos usuários e comunicar os modelos (não os dados) de forma que as entidades federadas possam computar um modelo global que tenha expressividade sobre todos os dados dos clientes da federação, mesmo sem ter tido contato direto com eles. Os modelos devem ser comunicados em canais seguros, de maneira que o acesso aos parâmetros do modelo não seja suficiente para que se quebre a privacidade dos usuários.

A implementação de modelos de ML em ambientes com alta capacidade de processamento e memória para treinamento dos modelos, por exemplo, em máquinas com alto paralelismo dotadas de GPU(graphical processor units) tem apresentado bons resul-

tados. Entretanto, a maioria das entidades encontradas em IoT são equipamentos com capacidade limitada, o que torna a utilização desses modelos de DL impraticável. Por exemplo, a AlexNet [Krizhevsky et al. 2012], uma das arquiteturas mais famosas em visão computacional, possui 61 milhões de parâmetros, ocupando 249 MB de memória. Sua utilização em dispositivos como celulares rapidamente sobrecarregará os recursos escassos [Rastegari et al. 2016]. Esses recursos limitados também inviabilizam a transferência de parâmetros entre os dispositivos e o servidor, impossibilitando a adoção de FL. Dessa forma, é necessário encontrar alternativas que possam acelerar a computação dos modelos de DL para superar as dificuldades apresentadas e assim possibilitar a utilização de DL no contexto de FL em IoT. Esses aspectos serão discutidos na Seção 5.4.

A crescente utilização de aplicações de IoT tem habilitado um modelo conhecido por Computação de Borda (*edge computing*). Nesse cenário, o grande volume de dados não se encontra em um ambiente centralizado como a nuvem, mas espalhado entre diversos dispositivos. Por exemplo, veículos autônomos deverão processar modelos de ML para tomoda de decisão sem que seja necessário acionar a nuvem, pois esta pode impor uma latência de comunicação indesejável ou até mesmo proibitiva para a aplicação. Dessa maneira, os veículos devem processar essa informação localmente, ou com a ajuda de infraestrutura que esteja mais próxima a eles, viabilizando os requisitos da aplicação. Iremos discutir aspectos de Computação na Borda e FL na Seção 5.6.

A primeira aplicação de FL surgiu em 2016 no contexto de processamento natural de linguagem natural e foi proposto pelo Google no artigo [McMahan et al. 2016a]. Neste artigo, os autores propuseram um modelo federado para predição de digitação de texto em *smartphones* Android. O sistema ajuda os usuários a autocompletarem os textos digitados. Os autores propuseram o modelo de agregagação FedAvg (ver detalhes na Seção 5.2.3). Nesse modelo, os dados do usuário não são enviados para uma entidade central. Um modelo é treinado localmente e os parâmetros do modelo são enviados, de maneira criptografada, para a nuvem. Esta, computa um modelo global a partir dos modelos de cada usuário e retroalimenta esses parâmetros para os clientes, que atualizam seus modelos e produzem modelos com maior capacidade. Essa foi a primeira aplicação bem sucedida de FL reportada na literatura.

O sistema de FL do teclado do Google (Gboard) é um bom exemplo de B2C (business-to-consumer), entretanto, FL não está limitado apenas a esse modelo. Aplicações de B2B (business-to-business) também foram concebidas de modo que mesmo em ambientes em que os mesmos usuários possuam dados diferentes armazenados em diversas corporações podem se beneficiar do FL. Iremos abordar esses modelos na Seção 5.2.

Este minicurso tem a seguinte estrutura. Na Seção 5.2 são apresentados os modelos mais comuns de FL, como o FL horizontal, vertical e o FL *trasnfer learning*. A Seção 5.3 apresenta os aspectos relativos à preservação de privacidade em FL. A Seção 5.4 apresenta um aspecto de grande importância para IoT, que é a compressão de modelos para que dispositivos de baixa capacidade de processamento e memória, comumente encontrados em cenários de IoT estejam habilitados a participar de uma federação. A Seção 5.5 introduz aspectos de aprendizado por reforço no contexto de FL. A Seção 5.6 discute a relação entre Computação de Borda e FL. As Seções 5.7, 5.8 e 5.9 apresentam oportunidades de pesquisa, aplicações e um estudo de caso de FL, respectivamente. Fi-

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Figura 5.1. Exemplo de um conjunto de dados de treinamento

nalmente, a Seção 5.10 apresenta nossas considerações finais.

5.1.1. Introdução a Aprendizado de Máquina

Aprendizado de Máquina (ML) pode ser definido como um conjunto de métodos capazes de detectar automaticamente padrões nos dados e, em seguida, utilizar os padrões descobertos para realizar previsões a partir da observação de novos dados ou para realizar outros tipos de tomada de decisão.

ML geralmente é dividido em três tipos principais, (i) aprendizado supervisionado, (ii) aprendizado não-supervisionado e (iii) aprendizado por reforço. No aprendizado supervisionado, o objetivo é aprender um mapeamento das entradas $x^{(i)}$, também chamadas de *features* ou atributos, para as saídas $y^{(i)}$, dado um conjunto de dados rotulados de entrada e saída $\mathscr{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$. O par $(x^{(i)}, y^{(i)})$ é tido como um exemplo de dado de treinamento. Já o conjunto $\mathscr{D}$ é chamado de conjunto de dados de treinamento, onde n é a quantidade de exemplos de treinamento. Na configuração mais simples, cada entrada de dado de treinamento $x^{(i)}$ é um vetor D-dimensional de números, representando, por exemplo, a altura e o peso de uma pessoa, ou a área e a quantidade de quartos em uma residência. No entanto, $x^{(i)}$ pode ser um objeto estruturado complexo, como uma imagem, um texto, uma série temporal, ou a estrutura de uma molécula. De maneira similar, a variável de saída $y^{(i)}$ pode, em princípio, ser qualquer coisa. No entanto, na maioria dos casos assume-se que é uma variável categórica de algum conjunto finito, ou um valor do conjunto dos números reais. Quando $y^{(i)}$ é uma variável categórica, o problema de ML é conhecido como classificação, e quando possui um valor real, o problema é conhecido como regressão.

No aprendizado supervisionado, o objetivo é, dado um conjunto de dados de treinamento, aprender uma função $f : \mathscr{X} \mapsto \mathscr{Y}$ de forma que $f(x)$ é um "bom"preditor para o valor correspondente y. Aqui, $\mathscr{X}$ denota o espaço dos valores de entrada, enquanto que $\mathscr{Y}$ denota o espaço dos valores de saída. Como um exemplo de aprendizado supervisionado, considere o conjunto de dados de treinamento mostrado na Figura 5.1 que contém algumas características a respeito de imóveis que estão à venda[1]. As entradas x são vetores de duas dimensões em $\mathbb{R}^2$, onde $x_1^{(i)}$ representa a área do imóvel da i-ésima residência no conjunto de treinamento, enquanto que $x_2^{(i)}$ representa o número de quartos.

Para realizar aprendizado supervisionado, é necessário antes determinar como a função f será representada. Uma estratégia é aproximar y como uma função linear de

[1] `http://cs229.stanford.edu/notes2020spring/cs229-notes1.pdf`

x denotada por $f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$, onde os θ_i's são os parâmetros (ou pesos) que estão parametrizando o espaço de mapeamento de funções lineares de $\mathscr{X}$ para $\mathscr{Y}$.

Dado um conjunto de dados de treinamento, como selecionar (ou aprender) os valores para os parâmetros θ? Um método razoável parece ser fazer $f(x)$ ser próximo de y, pelo menos para os exemplos no conjunto de dados de treinamento. Tal processo é conhecido como fase de treinamento de um modelo de ML. De maneira formal, pode-se definir uma função que mede, para cada um dos valores de θ, quão próximo são os valores de $f(x^{(i)})$ com os valores correspondentes de $y^{(i)}$, isso para cada exemplo i no conjunto de dados de treinamento. Logo, define-se o que é chamada de função de custo $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f_\theta(x^{(i)}) - y^{(i)})^2$, onde neste exemplo utiliza-se a função de custo conhecida como mínimos-quadrados (*least-squares*).

O próximo passo é escolher valores para θ que minimizam a função $J(\theta)$. Uma estratégia é utilizar o algoritmo conhecido como descida do gradiente (*gradient descent*), o qual escolhe algum valor inicial para θ, e repetidamente executa $\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$ simultaneamente para cada um dos valores de $j = 0,\ldots,m$. Aqui, α é chamado de *learning rate*, o qual representa um dos hiper-parâmetros do modelo e, portanto, deve ser definido antes do início da etapa de treinamento. Já $\frac{\partial}{\partial\theta_j}$ é o gradiente e indica a inclinação de J em um dado ponto θ. Perceba que este é um algoritmo muito natural que repetidamente dá um passo na direção que vai diminuir de forma mais acentuada o valor de J.

No aprendizado não-supervisionado apenas entradas são fornecidas, $\mathscr{D} = \{x^{(i)}\}_{i=1}^{n}$, e o objetivo é encontrar padrões interessantes nos dados. Isso torna o problema menos bem definido, já que não somos informados sobre quais tipos de padrões procurar e não há uma métrica de erro óbvia para utilizar. Isso é diferente do aprendizado supervisionado, onde pode-se comparar a previsão para uma saída y dado a entrada x com o valor observado. Já no aprendizado por reforço, a ideia é aprender como agir em um determinado ambiente quando são dados sinais ocasionais de recompensa ou punição.

5.1.2. Introdução a Aprendizado de Máquina Distribuído

No mundo da IoT, grandes volumes de dados são produzidos de forma onipresente. Diante disso, o principal obstáculo dos métodos de aprendizado de máquina (ML) tradicionais mudou de ser capaz de realizar inferência a partir de pequenas quantidades de dados de treinamento para como lidar com conjuntos de dados de treinamento com grande escala e altas dimensões. Nesse cenário, o poder computacional e o tempo não escalam bem com o volume do conjunto de dados. Isso torna o processo de aprendizado a partir de amostras de treinamento de grande escala com esforço e tempo de computação razoáveis uma tarefa muito difícil. Além disso, é comum lidar com cenários em que os dados estão espalhados em diferentes localidades (por exemplo, diferentes dispositivos na borda da rede - ver Seção 5.6), pertencem a diferentes indivíduos ou organizações, e não há uma solução simples para reunir os dados para realizar o processo de treinamento. A seguir, são apresentados os principais desafios que os métodos de ML tradicionais enfrentam ao lidar com conjuntos de dados de grande escala e que estão distribuídos em diferentes localidades: (i) **Falta de memória:** Os métodos de ML tradicionais são treinados carregando-se as amostras de treinamento inteiramente na memória principal. Portanto, se a complexidade

computacional das amostras de treinamento exceder a memória principal, os seguintes problemas podem surgir: (i) o modelo treinado pode não convergir ou pode resultar em baixo desempenho (como baixa precisão ou *recall*), e (ii) no pior cenário, os modelos de ML podem não ser treinados devido à falta de memória. (ii) **Tempo de treinamento inviável:** Alguns processos de otimização utilizados por algoritmos de ML podem não escalar bem em relação ao tamanho das amostras de treinamento. Como consequência, ao lidar com amostras de treinamento de grande escala, o tempo consumido pelo processo de treinamento pode ser muito longo para fins práticos. Além disso, o ajuste de hiper-parâmetros dos modelos de ML também leva muito tempo, pois normalmente é necessário avaliar muitas configurações diferentes. Portanto, se o processo de treinamento demorar muito, o ajuste de hiper-parâmetros pode não ser realizado de forma eficaz, o que pode resultar em modelos de ML ineficientes. (iii) **Violação de privacidade:** Usuários estão cada vez mais preocupados que suas informações pessoais estão sendo utilizadas para os mais diversos fins sem permissão. Diante desse cenário, coletar e compartilhar dados para treinar modelos de ML tem se tornado uma tarefa cada vez mais difícil.

Aprendizado de Máquina Distribuído (DML) [Yang et al. 2019a], ou Aprendizado Distribuído, refere-se ao estudo de algoritmos e sistemas de aprendizado de máquina que são projetados para alcançar pelo menos um dos objetivos a seguir: (i) escalar para mais dados de treinamento e modelos de inferência maiores; (ii) melhorar o desempenho da solução como um todo; (iii) preservar a privacidade das entidades detentoras dos dados.

De maneira geral, pode-se dizer que as soluções de DML possuem duas principais motivações: melhorar a escalabilidade e preservar a privacidade. No DML motivado pela escalabilidade, a principal preocupação é em como projetar soluções que são capazes de lidar com os requisitos computacionais cada vez maiores dos sistemas de ML de grande escala. Por exemplo, é fato notório que a escala dos problemas que os métodos de ML têm que lidar aumentaram de maneira exponencial nos últimos anos. Treinar modelos de aprendizado sofisticados e que possuem uma grande quantidade de dados de treinamento pode facilmente exceder a capacidade do modelo tradicional de ML que depende de uma única entidade computacional. Não é incomum encontrar exemplos de modelos na literatura que requerem várias unidades de processamento e que podem levar vários dias apenas para realizar a etapa de pré-treinamento. A seguir, são apresentadas as principais estratégias de DML motivadas por escalabilidade: (i) **Paralelismo de dados:** Uma estratégia natural em DML é particionar os dados de treinamento em subconjuntos, os quais são distribuídos para diferentes entidades de computação. Essas entidades mantêm réplicas do mesmo modelo e utilizam os subconjuntos de dados para treiná-lo em paralelo. Uma desavantagem dessa estratégia é que as réplicas do modelo devem residir na memória das entidades de computação, logo ela não escala bem para modelos muito grandes. Há duas abordagens para DML baseadas em paralelismo de dados: (i) treinamento síncrono e (ii) treinamento assíncrono. No treinamento síncrono, as entidades de computação treinam réplicas do modelo utilizando diferentes partes dos dados de treinamento de maneira síncrona, e os gradientes e pesos são agregados após cada etapa de treinamento (cada *epoch*). Já no treinamento assíncrono, as entidades computacionais treinam de maneira independente réplicas do mesmo modelo utilizando subconjuntos dos dados de treinamento e atualizam os gradientes e pesos de maneira assíncrona. (ii) **Paralelismo do modelo:** Conforme destacado anteriormente, modelos de treinamento muito

grandes podem não caber inteiramente na memória de uma única entidade computacional. Nessa situação, uma estratégia é dividir o modelo e então distribuir partes do modelo para diferentes entidades computacionais. Por exemplo, ao considerar um modelo de redes neurais profundas (DNN), pode-se dividir o modelo de maneira lógica, colocando-se diferentes camadas do modelo em diferentes entidades de computação. Dessa forma, as etapas de *forward propagation* e *backpropagation* envolvem a comunicação da saída de um dispositivo para a entrada em um outro dispositivo computacional de maneira serial. (iii) **Paralelismo de tarefas:** Normalmente utilizada em conjunto com a estratégia de paralelismo de dados, o paralelismo de tarefas diz respeito à execução de um programa em vários processadores localizados em um único computador ou em vários computadores. Ou seja, o objetivo é executar diferentes operações em paralelo de forma a utilizar totalmente os recursos de computação disponíveis na forma de processadores e memória. Considere, por exemplo, um programa de computador que utiliza várias *threads*, onde cada uma é responsável por realizar uma operação diferente.

É importante observar que na prática, normalmente se faz necessário combinar mais de uma forma de paralelismo, o que é conhecido como paralelismo híbrido. Além disso, tais estratégias podem utilizar recursos de computação elásticos e escaláveis, o que possibilita adicionar mais entidades de computação sob demanda. Tal característica é particularmente útil na era da computação em nuvem e *edge computing*, onde pode-se solicitar mais processadores (como CPUs e GPUs) e memória sob demanda. Finalmente, tais estratégias são comumente aplicadas nos cenários com conjuntos de dados particionados horizontalmente, onde subconjuntos separados de dados de treinamento são armazenados em diferentes entidades computacionais (ver Seção 5.2.2).

Já no DML motivado pela privacidade, a principal preocupação é preservar a privacidade do usuário. ML com preservação de privacidade tem se tornado uma tendência na comunidade de ML, já que questões como a privacidade dos usuários e a segurança dos dados têm recebido atenção especial de maneira global. Em um sistema DML motivado pela privacidade, existem várias entidades e cada uma contém um subconjunto dos dados de treinamento. Devido a questões de privacidade, as entidades não desejam expor seus dados umas às outras. Desse modo, as abordagens de DML são obrigadas a utilizar os dados de cada entidade participante para treinar um modelo de ML de maneira colaborativa. Nesse cenário, os conjuntos de dados mantidos por diferentes entidades podem ter atributos diferentes, resultando na chamada partição vertical dos dados de treinamento. Diante disso, DML motivado por privacidade é frequentemente utilizado em cenários com conjuntos de dados particionados verticalmente, com subconjuntos de dados de treinamento com diferentes atributos mantidos por diferentes entidades (ver Seção 5.2.4). Para mais detalhes sobre as principais estratégias de preservação de privacidade em um ambiente distribuído, consultar a Seção 5.3.

5.2. Aprendizado Federado e Modelos de Aprendizado Federado

Uma abordagem distribuída de ML chamada Aprendizado Federado (FL) [McMahan et al. 2016b] foi proposta para garantir que os dados de treinamento permaneçam em dispositivos pessoais e facilite modelos complexos de aprendizado de máquina colaborativo em dispositivos distribuídos.

Em FL, os dispositivos pessoais usam dados armazenados localmente para treinar um modelo local. A junção de vários modelos locais é usada para treinar um modelo de ML cooperativamente. Este modelo é disponibilizado por um servidor FL. De modo geral, o servidor recebe a atualização dos modelos locais (por exemplo, os pesos ou gradientes do modelo), e realiza uma agregação do valores recebidos. As etapas são repetidas em várias rodadas até que uma precisão desejável seja alcançada. Esse fato implica que FL pode ser uma tecnologia capacitadora para o treinamento de modelos de ML localizados em dispositivos conectados na borda da rede.

Em comparação com as abordagens convencionais de treinamento centrado na nuvem, a implementação de FL para treinamento de modelo em dispositivos na borda da rede apresenta as seguintes vantagens [Lim et al. 2020]: uso altamente eficiente da largura de banda da rede, privacidade e baixa latência.

5.2.1. Aprendizado Federado

Sejam N usuários $\{u_1, \cdots, u_N\}$, todos os quais desejam treinar um modelo de aprendizado de máquina por seus respectivos conjuntos de dados disjuntos $\{\mathbb{X}_1, \cdots, \mathbb{X}_N\}$. O método usual de treinamento de aprendizado de máquina é agrupar todos os dados no conjunto $\mathbb{X} = \mathbb{X}_1 \cup ... \cup \mathbb{X}_N$ para treinar um modelo $T_{\mathbb{X}}$. Um sistema de aprendizado federado é um processo de aprendizagem no qual os proprietários dos dados treinam colaborativamente um modelo T_{Fed}, no qual qualquer proprietário dos dados u_i não expõe seus dados $\mathbb{X}_i$ a outros [Yang et al. 2019a]. Em geral, o processo de treinamento do Aprendizado Federado inclui as seguinte três etapas[2] [Lim et al. 2020]: **Etapa 1** (Inicialização do modelo): O sistema inicializa os modelos locais $T_{\mathbb{X}_i}$, bem como define todos os valores de hiper-parâmetros necessários para iniciar a tarefa de aprendizagem. **Etapa 2** (treinamento e atualização do modelo local): Cada usuário u_i, respectivamente, usa seus dados locais $\mathbb{X}_i$ e atualiza o modelo local $T_{\mathbb{X}_i}$. O objetivo do usuário u_i é encontrar os parâmetros ideais para o modelo local que minimizam a função de perda $L(T_{\mathbb{X}_i})$, ou seja, $\Theta^*_{\mathbb{X}_i} := \arg\min_{\Theta_{\mathbb{X}_i}} L(T_{\mathbb{X}_i})$, onde $\Theta_{\mathbb{X}_i}$ são os parâmetros do modelo $T_{\mathbb{X}_i}$. **Etapa 3** (agregação e atualização do modelo): O servidor agrega os modelos locais dos participantes e envia as atualizações para o modelo agregado T_{Fed}. Depois da agregação, o modelo agregado envia o novo modelo de volta para os proprietários dos dados. O servidor minimiza a função de perda agregada L (T_{Fed}), ou seja, $L(T_{Fed}) = \sum_{i=1}^{N} [L(T_{\mathbb{X}_i}) p_i]$, onde p_i é um valor definido para o usuário u_i e $\sum_{i=1}^{N} p_i = 1$. Tipicamente, $p_i = 1/N$ ou $p_i = n_i / \sum_{i=1}^{N} n_i$, com $n_i = |\mathbb{X}_i|$ como podemos ver em [Li et al. 2020c]. No processo de treinamento, as etapas 2-3 são repetidas até que a função de perda agregada convirja ou uma precisão de treinamento desejável seja alcançada.

5.2.2. Aprendizado Federado Horizontal

Aprendizado Federado Horizontal (HFL), também conhecido como aprendizado federado particionado por amostra, pode ser aplicado em cenários nos quais conjuntos de dados em locais diferentes compartilham características sobrepostas, mas diferente no espaço de amostra, conforme ilustrado na Figura 5.2. É semelhante à situação em que os dados são particionados horizontalmente em uma visualização de uma tabela. Por exemplo,

[2] o modelo local refere-se ao modelo treinado em cada usuário u_i, enquanto o modelo global se refere ao modelo agregado T_{Fed}

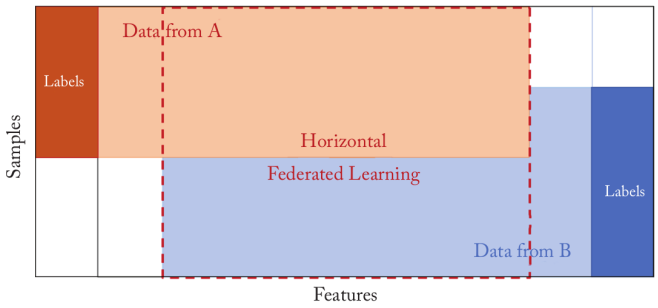

Figura 5.2. Ilustração do Aprendizado Federado Horizontal. Fonte: [Yang et al. 2019a]

dois bancos regionais podem ter grupos de usuários muito diferentes de suas respectivas regiões, e o conjunto de interseção de seus usuários é muito pequeno. No entanto, seus modelos de negócios são muito semelhantes. Assim, descreveremos duas arquiteturas populares para sistemas HFL, a saber, a arquitetura cliente-servidor e a arquitetura ponto a ponto.

Na arquitetura cliente-servidor, K participantes (também conhecidos como clientes ou usuários) com a mesma estrutura de dados treinam de forma colaborativa um modelo de aprendizado de máquina com a ajuda de um servidor (também conhecido como servidor de agregação ou coordenador). Uma suposição típica é que os participantes são honestos, enquanto o servidor é honesto, mas curioso. Portanto, o objetivo é evitar o vazamento de informações de quaisquer participantes para o servidor.

As iterações de treinamento continuam até que a função de perda convirja ou até que o número máximo de iterações permitidas é atingido. Esta arquitetura é independente de algoritmos de ML específicos (por exemplo, regressão logística ou redes neurais), e todos os participantes compartilharão os mesmos parâmetros finais do modelo. Para HFL, um proprietário de dados tem total autonomia para operar em seus dados locais, podendo decidir quando e como ingressar e contribuir para um sistema HFL. Além disso, o HFL leva em consideração a proteção da privacidade dos dados durante o treinamento do modelo. Na prática, em um sistema HFL, os dados obtidos pelos diferentes participantes não são distribuídos de forma idêntica na maioria das aplicações.

Além da arquitetura cliente-servidor discutida acima, um sistema HFL também pode fazer uso da arquitetura ponto-a-ponto (P2P). Na arquitetura P2P, não há servidor central ou coordenador. Em tais cenários, os participantes de um sistema HFL também são chamados de treinadores ou trabalhadores distribuídos. Cada treinador é responsável por treinar o mesmo modelo ML ou DL (por exemplo, um modelo DNN) usando apenas seu dado local. Além disso, os treinadores precisam de canais seguros para transferir os pesos do modelo um para o outro. Para garantir comunicações seguras entre quaisquer dois treinadores, podem ser adotadas medidas de segurança, como esquemas de criptografia com base em chave pública.

Uma vez que não há um servidor central, os treinadores devem concordar com a ordem de envio e recebimento dos pesos do modelo com antecedência. Existem duas maneiras simples de fazer isso: (i) **Transferência cíclica**: No modo de transferência cíclica, os treinadores são organizados em uma cadeia. O primeiro treinador (ou seja, o topo

da cadeia) envia os pesos do modelo atual para seu vizinho subsequente. Um treinador recebe pesos de modelo e atualiza o modelo recebido usando mini-lotes de dados de treinamento de seu próprio conjunto de dados. Então, depois do treinamento, envia os pesos do modelo atualizado para seu treinador subsequente. Por exemplo, do treinador 1 para o treinador 2, do treinador 2 para o treinador 3; e assim sucessivamente até do treinador (K - 1) para o treinador K, e do treinador K de volta ao treinador 1. Este procedimento é repetido até que os pesos do modelo convirjam ou até que o número máximo de iterações seja alcançado. (ii) **Transferência aleatória**: O k-ésimo treinador seleciona um outro treinador i aleatoriamente e envia os pesos do modelo para o treinador i. Quando o treinador i recebe os pesos do modelo do k-ésimo treinador, ele atualiza os pesos do modelo recebido usando mini-lotes de dados de treinamento de seu próprio conjunto de dados. Este procedimento ocorre simultaneamente entre os k treinadores até eles concordarem que os pesos do modelo convergiram ou até que o tempo máximo de treinamento permitido é atingido. Este método também é conhecido como aprendizagem por sussurro.

Comparado com a arquitetura cliente-servidor, a vantagem óbvia da arquitetura P2P é a possibilidade de remover o servidor central, que pode não estar disponível em aplicações práticas, e elimina a chance de vazar informações para o servidor. No entanto, existem várias desvantagens. Por exemplo, no modo de transferência cíclica, uma vez que não existe um servidor central, os parâmetros de peso são atualizado em série, em vez de em lotes paralelos, o que leva mais tempo durante o treinamento do modelo.

5.2.3. Algoritmos de Média Federada

Em [McMahan et al. 2016a], o algoritmo de média federada (FedAvg) foi empregado para treinamento de modelo federado em sistemas HFL. Esta seção revisa o algoritmo FedAvg, assumindo uma arquitetura cliente-servidor.

A quantidade de computação é controlada por três parâmetros principais, a saber: (1) ρ, a fração de clientes que realizam cálculos durante cada rodada; (2) S, o número de etapas de treinamento que cada cliente executa em seu conjunto de dados local durante cada rodada (ou seja, o número de épocas locais); e (3) M, o tamanho do mini-lote usado para o cliente realizar atualizações.

Este algoritmo seleciona uma fração de participantes durante cada rodada e calcula o gradiente e a função de perda sobre todos os dados mantidos por esses participantes. Portanto, este algoritmo controla o tamanho do lote global, com $\rho = 1$ correspondendo ao lote completo do gradiente descendente usando todos os dados mantidos por todos os participantes. Uma vez que são selecionados lotes usando todos os dados dos participantes escolhidos, este algoritmo é referido como FederatedSGD.

É comumente assumido que o coordenador ou servidor tem o modelo de ML inicial, e os participantes conhecem as configurações do otimizador. Para uma implementação típica de gradiente descendente com uma taxa de aprendizagem fixa η, na t-ésima rodada de atualização de peso do modelo global, o k-ésimo participante calcula o gradiente médio $g_i = \nabla L(T_{\mathbb{X}_i})$ do peso do modelo atual Θ_i, e o servidor agrega esses gradientes e aplica a atualização dos pesos do modelo de acordo com $\Theta_{Fed} \leftarrow \Theta_{Fed} - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$, onde $n_k = |\mathbb{X}_k|$. Além disso, note que o servidor pode enviar os pesos do modelo atualizado Θ_{Fed} de volta para os clientes. Este método é chamado Gradiente médio.

Alternativamente, o servidor pode enviar o gradiente médio $\sum_{k=1}^{K} \frac{n_k}{n} g_k$ de volta para os usuários, e os usuários calculam a atualização dos pesos. Logo, pode-se descrever

$$\Theta_{\mathbb{X}_i} \leftarrow \Theta_{Fed} - \eta g_k, \tag{1}$$

$$\Theta_{Fed} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \Theta_{\mathbb{X}_i}. \tag{2}$$

Ou seja, cada cliente localmente realiza uma etapa (ou várias etapas) da descida do gradiente com os pesos obtidos pelo servidor Θ_{Fed} (Equação 1), e envia os pesos locais $\Theta_{\mathbb{X}_i}$. O servidor recebe os pesos locais dos usuários e atualiza seu peso – (Equação 2).

5.2.4. Aprendizado Federado Vertical

Conforme visto na seção 5.2.2, o aprendizado federado horizontal (HFL) é aplicável a cenários IoT onde os conjuntos de dados dos participantes compartilham o mesmo espaço de recursos, mas diferem em espaços de amostra. Desse modo, o HFL é conveniente para ser aplicado na construção de aplicativos movidos por uma grande quantidade de dispositivos móveis, i.e. usuários diferentes mas com as mesmas características. Nesses casos, os usuários sendo federados são os consumidores individuais dos aplicativos.

No entanto, em muitos cenários práticos, os participantes da aprendizado federado são usuários de organizações que coletam diferentes características de dados para o mesmo grupo de usuários. Essas organizações tem como objetivo a cooperação entre si a fim de melhorar a eficiência de seus serviços. Por exemplo, suponha que haja um usuário com alguns registros em um banco que refletem ao saldo bancário, comportamento das despesas e classificação de crédito. Além disso, o mesmo usuário possui algumas outras informações armazenadas em um site de comércio eletrônico que retém o histórico de compra e navegação online do usuário.

Embora as características nas duas organizações sejam bastante diferentes, elas têm um aspecto próximo de associação uns com os outros. Por exemplo, o histórico de compras do usuário pode determinar de alguma forma a classificação de crédito do usuário. Esses cenários são comuns na vida real. Os varejistas podem fazer parceria com bancos para oferecer serviços ou produtos personalizados com base no histórico de compras do mesmo usuário. Os hospitais podem colaborar com empresas farmacêuticas para fazer uso de prontuários de pacientes comuns para tratar doenças crônicas e reduzir riscos de uma futura hospitalização.

Pode-se categorizar o aprendizado vertical (VFL) federado através participantes cujos conjuntos de dados compartilham o mesmo espaço amostral, mas diferentes características. A palavra “vertical” vem do termo "partição vertical", que é amplamente usado no contexto de visualização de um banco de dados (por exemplo, colunas de uma tabela são verticalmente particionadas em grupos diferentes e cada coluna representa uma característica de todas as amostras), como mostrado na Figura 5.3.

Os conjuntos de dados mantidos por diferentes organizações com diferentes objetivos de negócios geralmente têm informações diferentes sobre os usuários, embora essas organizações possam compartilhar usuários em comum. Com VFL, também chamado de aprendizado federado particionado por recurso, pode-se aproveitar os espaços de recursos

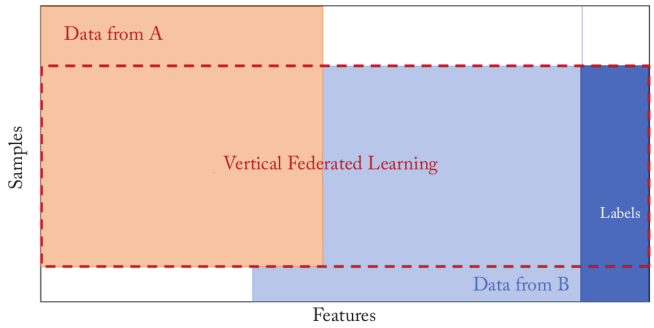

Figura 5.3. Ilustração do Aprendizado Federado Vertical. Fonte: [Yang et al. 2019b]

heterogêneos de conjuntos de dados distribuídos mantidos por aquelas organizações para construir modelos melhores de ML sem trocar e expor os dados privados.

Nas configurações de VFL, existem várias suposições subjacentes para obter segurança e preservar privacidade. Primeiro, presume-se que os participantes são honestos, mas curiosos. Isso significa que os participantes tentem deduzir o máximo possível das informações recebidas de outros participantes, embora cumpram o protocolo sem prejudicá-lo de forma alguma. Segundo, presume-se que o processo de transmissão de informações é seguro e confiável o suficiente para defender contra ataques. Além disso, assume-se que a comunicação é sem perdas, sem interferir com os resultados intermediários.

Um terceiro sistema semi-honesto (STP) também pode se juntar aos participantes para ajudar ambos. O STP é independente de ambas as partes. O STP coleta o resultados intermediários para calcular os gradientes e distribuir os resultados para cada parte. As informações que o STP recebe dos participantes são criptografadas ou ofuscadas. Os dados brutos dos participantes não são expostos uns aos outros, e cada participante recebe apenas o parâmetros do modelo relacionados aos seus próprios recursos.

5.2.5. *Transfer Learning* para FL

Federated Transfer Learning (FTL) é um caso especial de FL e diferente do FL horizontal (HFL) e vertical (VFL) [Saha and Ahmad 2021]. No FTL, dois conjuntos de dados diferem no espaço de características. Isso se aplica a conjuntos de dados coletados de empresas de natureza diferente. Devido às diferenças na natureza dos negócios, essas empresas compartilham apenas uma pequena sobreposição no espaço de características. Isso também se aplica às empresas estabelecidas em todo o mundo. Assim, em tais cenários, os conjuntos de dados diferem tanto nas amostras quanto no espaço de recursos.

Para [Li et al. 2020b] o principal problema neste cenário é a falta ou baixa qualidade dos rótulos. O Transfer Learning (TF) permite que o conhecimento de um domínio (ou seja, o domínio de origem) seja movido para outro domínio (o domínio de destino) para obter melhores resultados de aprendizagem, o que é apropriado para esta situação. Desta forma, alguns autores conceberam FTL para generalizar FL para ter uma aplicação mais ampla para uma pequena interseção de clientes e características em comum. Os autores criaram um modelo, o FedHealth, que reúne dados pertencentes a diferentes organizações por meio da FL e oferece atendimento médico personalizado por meio de

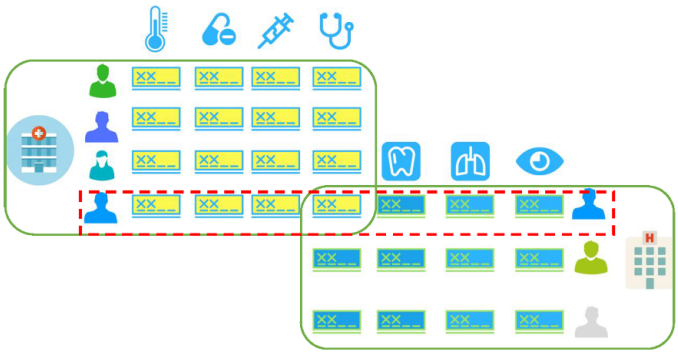

Figura 5.4. Exemplo de aplicação de FTL [Li et al. 2020b]

aprendizado por transferência. Alguns conhecimentos de diagnóstico e tratamento de doenças em um hospital podem ser transferidas para outro hospital para ajudar a diagnosticar outras doenças por FTL.

A pesquisa de FTL ainda não está madura, então ainda há muito espaço para crescimento para torná-la mais flexível com diferentes estruturas de dados. Os silos de dados e as questões de proteção de privacidade são problemas significativos encontrados na industrialização em grande escala do aprendizado de máquina atual. No entanto, o aprendizado por transferência federada é uma forma eficaz de proteger a segurança dos dados e a privacidade do usuário, quebrando as barreiras dos silos de dados.

A Figura 5.4 mostra um exemplo de FTL em dados médicos, onde o mesmo paciente pode ser tratado em hospitais diferentes, mas os mesmos não podem compartilhar os dados devido à questões de privacidade e os dados não possuírem muitos pontos em comum. Nesse caso, utilizando o ponto comum possível, é feito um FTL para que os dados do paciente possam ser aproveitados pelo modelo para gerar outro modelo que consiga compreender os novos dados do paciente.

[Saha and Ahmad 2021] cita algumas aplicações habilitadoras da tecnologia FTL, a saber: dispositivo vestível que está rapidamente se tornando parte da vida cotidiana de pacientes e profissionais de saúde. Múltiplos recursos e funcionalidades de dispositivos vestíveis incluem monitoramento remoto de pacientes, rastreamento e coleta de dados, melhorando a saúde diária e os padrões de estilo de vida, detectando condições crônicas, entre outros. Os dados de saúde, no entanto, são geralmente fragmentados e privados, dificultando a geração de resultados robustos entre as populações. Os dados do usuário gerados por dispositivos de saúde geralmente existem na forma de ilhas isoladas. Para mais avaliações e análises dos resultados obtidos, os dados coletados são carregados para o servidor remoto baseado em nuvem. Mas esse método possui alguns problemas como: (i) **restrições regulatórias** - falta de aquisição de dados massivos de usuário, (ii) **segurança e privacidade** - restringe o compartilhamento de dados que existem na forma de silos isoladas, (iii) **questão de personalização** - o processo de treinamento do modelo de aprendizado de máquina carece de personalização.

Ainda para [Saha and Ahmad 2021] o aprendizado de máquina depende da disponibilidade de grandes quantidades de dados para treinamento. No entanto, em cenários da vida real, os dados estão espalhados, principalmente, por diferentes organizações e

não podem ser facilmente integrados devido a muitas restrições legais e práticas. O FTL ajuda a melhorar a modelagem estatística em uma federação de dados. A federação de dados no caso de FTL permite que o conhecimento seja compartilhado sem comprometer a privacidade do usuário e permite que o conhecimento complementar seja transferido na rede. Assim, permitindo que a parte de domínio alvo desenvolva um modelo mais flexível e poderoso, aproveitando rótulos ricos da parte de domínio de origem.

A ampla aplicação de FTL é atualmente dificultada pela disponibilidade limitada de conjunto de dados para treinamento e validação de algoritmo, devido à ausência de abordagens técnicas e jurídicas para proteger a privacidade do usuário. A FTL tem requisitos estritos de preservação da privacidade, portanto, para evitar o comprometimento da privacidade do usuário enquanto promove a pesquisa científica em grandes conjuntos de dados, a implementação de soluções e desenvolvimento/implementação de quadros jurídicos para atender simultaneamente às demandas de proteção e utilização de dados são obrigatórios. A FTL para preservação da privacidade normalmente envolve várias partes, com ênfase nas garantias de segurança para realizar o aprendizado de máquina. Alguns dos métodos mais utilizados foram apresentados na Seção 5.3.2.

5.3. Preservação de Privacidade em FL

Apesar de sua alta aplicabilidade, modelos centralizados podem não ser uma alternativa aceitável em contextos em que, devido à preocupações com privacidade, deseja-se manter dados sensíveis (dados hospitalares, conversas pessoais, informações bancárias) em seu ponto de origem.

Devido ao pouco conhecimento popular sobre o assunto e à pouca transparência dos processos de Inteligência Artificial, há também desconfianças sobre a fiabilidade do uso de ML na sociedade. Algumas polêmicas recentes que podemos destacar são: o algoritmo do YouTube censurar vídeos de conteúdo LGBTQIA+ por considerá-los de cunho impróprio e sexual[3]; a inteligência artifical do Twitter, responsável por fazer o enquadramento de rostos em fotos postadas na plataforma, ter viés racista e reconhecer mais rostos de pessoas brancas[4]; o caso de um bot no Telegram que tinha a capacidade de "tirar a roupa" de mulheres em fotos, sendo muito usado em casos de *revenge porn* e colaborar para o cyberbullying e assédio sexual online[5]; o caso do crescimento do uso de *DeepFakes*, que, apesar de poderem ser identificadas através de métodos eficientes, podem mudar a opinião popular e gerar uma maior desconfiaça sobre IAs[6]; e, por fim, o caso de "filtros de embelezamento" serem ativados automaticamente, sem consentimento e sem possibilidade de serem desativados, em contas de *influencers*[7]. Um outro aspecto relevante é a regulação de vários países no sentido de preservar a posse e a privacidade

[3] https://www.metropoles.com/entretenimento/youtube/estudo-revela-mais-de-900-palavras-desmonetizaveis-no-youtube

[4] https://www.bbc.com/news/technology-57192898

[5] https://www.cnet.com/news/deepfake-bot-on-telegram-is-violating-women\--by-forging-nudes-from-regular-pics/

[6] https://www.cnet.com/features/deepfakes-threat-2020-us-election-isnt-\-what-youd-think/

[7] https://olhardigital.com.br/2021/06/10/internet-e-redes-sociais/tiktok-mudou-a-fisionomia-de-usuarios-sem-consulta-los/

dos dados dos indivíduos, como já discutido na Seção 5.1.

Neste sentido, técnicas de aprendizado federado têm o potencial de minimizar problemas de privacidade decorrentes de modelos centralizados de ML, uma vez que elas diminuem (ou eliminam) o tráfego de informações sensíveis entre usuários. Entretanto, FL não está imune à vulnerabilidades decorrentes da exposição de parâmetros encontrados no treinamento colaborativo. Os valores de parâmetros podem estar correlacionados aos dados sensíveis e, portanto, podem revelar informações sobre estes dados.

A necessidade de um número maior de iterações, comunicações e compartilhamento de informações entre os clientes expõe o ambiente federado a novos possíveis riscos e ataques, pois aumentam a viabilidade de manipulação de outputs do modelo de IA e do acesso a dados confidenciais de usuários. Diante ao exposto, três grandes fundamentos de segurança da informação, também conhecidos como CIA, devem ser aplicados em todas tecnologias adeptas ao aprendizado federado: confidencialidade, integridade e disponibilidade.

Devido ao aumento da consciência pública em relação à problemáticas de preservação de privacidade dos dados, um novo enfoque de desenvolvimento científico é o estudo de soluções específicas para preservação de privacidade em ambientes de aprendizado de máquina (*privacy-preserving machine learning* - PPML).

5.3.1. Tipos de Adversários

Um conceito inicial extremamente importante para a compreensão de técnicas de segurança e privacidade é a definição de adversário. Adversário é todo aquele usuário ou entidade que por algum motivo teve acesso à uma informação do sistema que não deveria ter. Desse modo, podemos dividir os tipos de adversários em duas categorias: Os semi-honestos e os maliciosos.

Adversários maliciosos são aqueles que possuem intenção de sabotar o funcionamento do algoritmo, seja roubando informações privadas ou injetando erros nos modelos. Tais adversários podem burlar os protocolos estabelecidos e agir de modo inesperado para o sistema. No entanto, no contexto específico de aprendizado federado, uma vez que este funciona por meio de práticas colaborativas, ataques realizados por adversários maliciosos não são muito comuns, dando um maior espaço para ataques provenientes de adversários semi-honestos. Adversários semi-honestos são aqueles que cumprem honestamente as regras estabelecidas pelo sistema/software, mas por curiosidade aprendem mais informações sensíveis do que deveriam.

Para executar suas ações, um adversário pode explorar seu conhecimento a respeito de como o modelo de aprendizado funciona. Existem duas classificações principais sobre o que o adversário sabe a respeito do modelo: (i) **Modelo caixa branca**: assume que o adversário possui conhecimento da arquitetura do modelo em si, porém sem acesso aos dados de treinamento utilizados para otimizá-lo. (ii) **Modelo caixa preta**: assume que o adversário não possui acesso ao modelo, pois o mesmo se encontra criptografado. Desse modo, este pode apenas consultar o modelo enviando dados e obtendo suas respectivas respostas.

5.3.2. Principais Técnicas de PPML

Secure Multi-party computation (SMC ou MPC): No ambiente federado, o objetivo consiste em construir uma função única em conjunto com todos os clientes, de tal modo que essas entradas sejam preservadas e privadas para as outras partes da federação, ao mesmo tempo em que permanecem adequadas para o treinamento do modelo. Neste caso, é mais usado para proteger os dados de atualização dos clientes. No entanto, um problema dessa técnica é o fato de exigir muito tempo de treinamento, o que pode acabar resultando em perda de dados em um contexto federado [Mothukuri et al. 2021].

Computação Homomórfica: Consiste em um conjunto de técnicas de criptografia que codificam os dados de modo que operações algébricas possam ser aplicadas eficientemente no texto cifrado. Assim, podemos executar algoritmos de aprendizado de máquina em cima dos dados sem a necessidade de descriptografá-los, garantindo a privacidade dos clientes envolvidos na federação.

Privacidade Diferencial: Consiste em inserir ruídos em dados sensíveis, de forma controlada, para que não seja possível identificar a entrada original. Logo, pode ser aplicada para confundir adversários em ataques a dados sensíveis (como por exemplo, em ataques de inferência de associação). Uma vez que o ruído é conhecido pelo sistema, é possível levá-lo em conta no modelo e assim garantir que a inferência continue aproximadamente válida, embora não consigamos identificar mais a entrada. Em FL, esses ruídos são introduzidos nos parâmetros dos usuários. Apesar de aumentar muito a segurança em comparação com a pouca perda estatística de dados, essa técnica gera incerteza perante os resultados de treino, além de dificultar a avaliação do comportamento do usuário [Mothukuri et al. 2021].

5.3.3. Tipos de Ataques

Uma vez que o aprendizado federado é feito de forma descentralizada, a proteção de privacidade foca na proteção dos gradientes transmitidos na fase de treinamento dos modelos, impedindo que informações sensíveis dos clientes sejam vazadas ou possam ser reconstruídas. Logo, o foco será nos principais ataques cujo objetivo consiste em reconstruir e/ou aprender informações dos clientes/modelo com base nos dados trocados no processo de treinamento federado.

Ataques de reconstrução: Ataques de reconstrução possuem como objetivo extrair informações dos dados brutos dos clientes no modelo federado ou das *features* que os representam. Como no modelo federado cada usuário treina localmente o modelo e apenas os gradientes são compartilhados, novos ataques vêm surgindo para inferir informações dos dados utilizados para realizar o treinamento local [Aono et al. 2016]. **Formas de se proteger contra ataques de reconstrução**: modelos que utilizam as *features* explicitamente, como máquina de vetores de suporte (SVM) e os k-vizinhos mais próximos (knn) devem ser evitados. Durante o treinamento do modelo, a computação multipartidária segura (MPC) [Yao 1982] e a criptografia homomórfica (HE) [Rivest et al. 1978] devem ser usados para proteger a privacidade da consulta do usuário durante a inferência do modelo caixa preta.

Ataques de inversão de modelo: Ataques de inversão de modelo capturam informações sobre os dados usados durante o treinamento ao conseguir acesso ao modelo (seja por caixa branca ou por caixa preta). Com base nas soluções retornadas pelo modelo, o adversário consegue reconstruí-lo (mesmo que aproximadamente), seja por meio de ataques de resolução de equações (teoricamente, um modelo linear N-dimensional pode ser descoberto com apenas N - 1 consultas) ou treinando um modelo semelhante passando entradas e as respectivas respostas obtidas pelo modelo original a que se quer copiar. **Formas de proteção contra ataques de inversão de modelo**: A regra principal para se proteger desse tipo de ataque é evitar expor ao máximo o modelo e suas saídas. Técnicas de agregação de resultados durante o treinamento [Fredrikson et al. 2015, Al-Rubaie and Chang 2016] e combinação de criptografia homomórfica vem sendo amplamente utilizadas [Xie et al. 2019].

Ataques de inferência de associação: Usa a saída do modelo caixa preta para verificar se uma determinada amostra se encontra ou não presente em um dado conjunto de treinamento. **Formas de proteção contra ataques de inferência de associação**: Novamente, técnicas de generalização de resultados podem ser utilizadas para evitar ataques. Porém, técnicas de privacidade diferencial também vêm se mostrando eficazes [Shokri et al. 2017].

Ataque de re-identificação: O objetivo do ataque aqui é desanonimizar informações de identificação pessoal nos dados de usuários utilizados no treinamento do modelo. Mesmo que as features sensíveis sejam removidas anteriormente, adversários se utilizam de conhecimentos prévios ou cruzamento de diferentes bases de dados para obter tais informações. **Formas de proteção contra ataques de re-identificação**: Técnicas de privacidade e anonimato em grupo (tal técnica também é baseada na privacidade diferencial) baseadas em algoritmos de generalização são implementadas para evitar esses ataques.

Ataque de envenenamento de modelos: Também conhecidos como ataques backdoor, esse tipo de ataque consiste em treinar maliciosamente um modelo ML, modificando os seus parâmetros, com o objetivo de invalidá-lo ou treiná-lo para um propósito específico (por exemplo, aumentar a atualização do modelo do participante malicioso). Até mesmo clientes maliciosos no aprendizado federado com um pequeno conjunto de dados possuem altas probabilidades de realizar um envenenamento do modelo com sucesso. **Formas de proteção contra ataques de envenenamento de modelos**: Modelos baseados em algoritmos de blockchain [Preuveneers et al. 2018].

A Tabela 1 sumariza os principais ataques de privacidade em FL, as garantias que as técnicas devem fornecer para evitá-los e os principais algoritmos de prevenção dos mesmos.

5.4. Compressão de Modelos no Contexto de FL

Os modelos de aprendizado de máquina costumam demandar alta capacidade de recursos em termos de processamento e de memória para que se possa treinar modelos com alta capacidade de expressividade. No contexto de IoT, raramente encontraremos as exigências

Tabela 5.1. Principais ataques de privacidade em Federated Learning, as garantias que as técnicas devem fornecer para evitá-los e os principais algoritmos de prevenção a ataques.

	Computação multipartidária segura	Criptografia Homomórfica	Agregação de resultados	Privacidade Diferencial	Blockchain
Reconstrução (Garantia da privacidade do usuário)	✓	✓			
Inversão de modelo (Garantia da privacidade do usuário)		✓	✓		
Inferência de associação (Garantia da privacidade do usuário)			✓	✓	
Re-identificação (Garantia da privacidade do usuário)				✓	✓
Envenenamento de modelos (Garantia da consistência e integridade do modelo)					✓

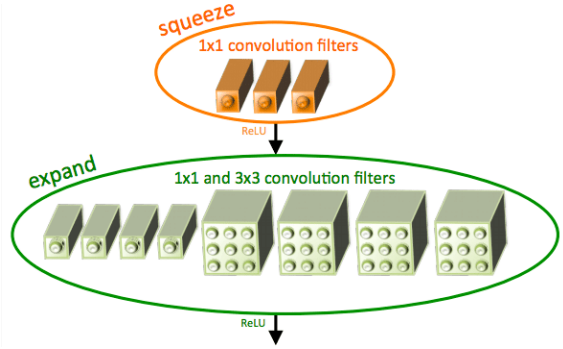

Figura 5.5. Fire Module, parte da SqueezeNet, proposto em [Iandola et al. 2016]

para que esses modelos sejam treinados adequadamente. Nesta seção, apresentaremos três possíveis abordagens para endereçar este problema, enquanto que na Seção 5.6 apresentamos uma abordagem, em outra direção, onde a Computação de Borda podeerá ser utilizada para dar suporte a esse tipo de demanda.

Adicionalmente, tenha em mente que todas as técnicas apresentadas aqui, embora discutidas separadamente, não são excludentes. Ou seja, pode-se utilizar uma ou mais técnicas de compressão para formar um framework de IoT em cenários federados. Por exemplo, é possível podar modelos compactos.

5.4.1. Desenvolvimento de Modelos Compactos

Os modelos compactos, também chamados de micro-arquiteturas, são modelos profundos focados em eficiência de memória e computação, além de bons resultados de classificação. Essa eficiência é atingida não somente diminuindo a quantidade de camadas, mas também a complexidade das mesmas, o que resulta em modelos com menos parâmetros (ou seja, modelos que ocupam menos memória) [Rastegari et al. 2016, Iandola et al. 2016]. Esse é um tópico bastante estudado em ML tradicional e que pode ser facilmente utilizado em FL, inclusive implementado em aparelhos IoT com recursos limitados. A seguir, são apresentados os principais modelos da literatura.

5.4.1.1. SqueezeNet

SqueezeNet [Iandola et al. 2016] é uma arquitetura de CNN que atinge resultados competitivos com grandes arquiteturas utilizando menos parâmetros. Por exemplo, esse modelo chega a possuir 50 vezes menos parâmetros que a AlexNet. Isso é conseguido através do *fire module*, mostrado na Figura 5.5, que é composto de uma camada *squeeze* (em laranja) e uma camada *expand* (em verde). O design dessas camadas segue as seguintes estratégias: (i) **Substituição dos filtros e limitação dos canais de entrada:** a quantidade total de parâmetros presente em uma camada convolucional é *(número de canais de entrada)* · *(número de filtros)* · *(tamanho do filtro)*. Logo, é preciso encontrar um balanço entre essas três variáveis a fim de atingir o objetivo de diminuição de parâmetros com preservação de acurácia. Com isso em mente, no *fire module*, substitui-se a maioria dos filtros 3×3 (comumente usados na literatura) por filtros 1×1, que possui 9 vezes menos parâmetros

que o filtro anterior. Além disso, limita-se a quantidade de filtros das camadas *squeeze* para ser menor que o número de filtros nas camadas *expand*, o que diminui a quantidade de canais de entrada nos filtros 3×3. (ii) ***Downsampling* mais próximo ao final da arquitetura para criação de mapas de ativação maiores:** mapas de ativação determinam para qual parte da imagem uma camada convolucional vai ativar. Intuitivamente, mapas de ativações maiores podem levar a melhores resultados de classificação, uma vez que a camada convolucional capturará os atributos essenciais de uma imagem sem ser perturbada por detalhes (que podem ser ruídos). O tamanho desse mapa é controlado pelo tamanho da imagem e em qual parte da arquitetura há um *downsampling* (isto é, em quais camadas de convolução e *pooling* aumenta-se o *stride* para maior que um). Se camadas no começo da arquitetura possuem *stride* maior, então a maioria das camadas vai ter um mapa pequeno. Por outro lado, se *stride* maiores aparecem mais no final da arquitetura, então a maioria das camadas terá mapas de ativação grande. Portanto, na construção da SqueezeNet, os *fire module* iniciais possuem um *stride* pequeno, que é aumentado mais ao fim da arquitetura.

5.4.1.2. MobileNet

MobileNets [Howard et al. 2017] são arquiteturas de CNN projetadas para uso em aparelhos móveis, assim sendo, são leves e de baixa complexidade. A ideia principal por trás dessas arquiteturas é a convolução separável em profundidade (*Depthwise Separable Convolution*). A Figura 5.6 mostra a diferença entre a convolução comum (esquerda) e a convolução separável em profundidade (direita). Uma camada convolucional comum aplica o filtro convolucional em todos os canais da imagem, somando os pesos dos pixels da imagem de entrada. Com isso, todos os canais da imagem de entrada são combinados durante a operação. Já a convolução separável em profundidade realiza a convolução em cada canal da imagem separadamente, com cada canal tendo seus próprios pesos. Após essa convolução em profundidade, os pesos dos canais são somados através de uma convolução *pointwise* (que é o mesmo que uma convolução comum, mas com filtro de tamanho 1×1), que combina os canais de saída.

Dessa forma, enquanto a convolução comum filtra e combina os canais de uma única vez, a convolução separável em profundidade realiza as duas operações separadamente. Embora os resultados obtidos sejam similares (filtram e fazem novas *features*), uma convolução comum utiliza muito mais poder computacional e precisa aprender muito mais pesos. Por isso, a convolução separável em profundidade é muito mais rápida que a convolução comum.

Em [Sandler et al. 2018], é proposta uma extensão das MobileNets, chamada de MobileNetV2. Essa nova arquitetura é bem semelhante ao MobileNet original, porém é muito mais rápida, atingindo a mesma acurácia. Por exemplo, esse novo modelo usa duas vezes menos operações, precisa de 30% menos parâmetros e é cerca de 30% a 40% mais rápida que os modelos da primeira versão, atingindo cerca de 4% a mais de acurácia[8].

Além do uso das convoluções separáveis em profundidade, nessa segunda versão é

[8] `https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html`

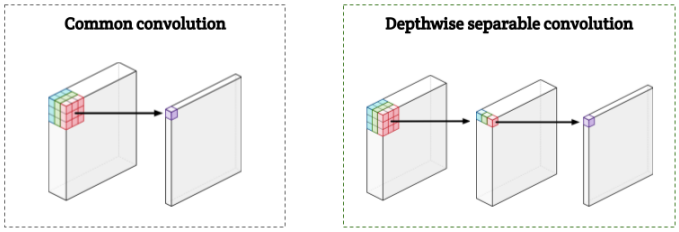

Figura 5.6. Depthwise Separable Convolution [Hollemans 2017]

introduzido um novo componente chamado *Inverted Residual Block*. Esse bloco é uma variação do *Residual Block*, parte da famosa arquitetura profunda ResNet [He et al. 2016]. Na MobileNetV2, o bloco proposto é formado por três camadas convolucionais comuns com filtros 1×1, 3×3 e 1×1, respectivamente. As duas últimas camadas são as camadas convolucionais já utilizadas na primeira versão dessa arquitetura, uma convolução separável em profundidade e uma convolução *pointwise*. Porém, a última camada agora tem um papel diferente. Enquanto na primeira versão essa camada mantinha o número de canais ou os dobrava, nessa segunda versão acontece o oposto: a quantidade de canais diminui através da projeção dos dados de uma dimensão maior em um tensor com menor número de dimensões (por isso essa camada agora é chamada de camada de projeção). A primeira camada, por sua vez, é uma camada semelhante a última, mas em vez de projetar os dados, os expande. Chamada de camada de expansão, a ideia é ampliar a quantidade de canais antes da convolução separável em profundidade.

Intuitivamente, esse bloco funciona como um compressor e descompressor de informações: a camada de expansão descomprime a informação para que possam ser extraídas com mais facilidade pela camada do meio e a camada de projeção comprime as informações novamente, tornando a representação pequena mais uma vez. Dessa forma, essa diminuição, aumento e diminuição novamente das dimensões força a rede neural a aprender uma representação mais compacta sem perder tanta qualidade nos resultados.

Além disso, as entradas e saídas para esse bloco possuem uma conexão residual, o que permite a rede a oportunidade de acessar ativações que não foram modificadas pela convolução, essencial para construir redes neurais muito profundas e permitir um treinamento mais rápido e acurado, como nas conexões residuais das ResNet [Sandler et al. 2018].

5.4.2. Compressão de Modelos Pré-treinados

O objetivo da compressão de modelos pré-treinados é simplificar um modelo já existente, de forma a obter arquiteturas de DL mais simples, porém sem perder acurácia de forma significativa. Essa simplificação tem dois principais focos: (i) **Redução de tamanho:** o modelo comprimido possui menos parâmetros, logo, utiliza menos memória RAM durante sua execução; (ii) **Redução de latência:** o modelo comprimido leva menos tempo para fazer uma inferência, o que leva a um menor consumo de energia.

Em ambientes com recursos escassos, como IoT, ambas as simplificações são desejáveis. Elas são usualmente atingidas concomitantemente, uma vez que modelos maiores precisam de mais memória e mais energia para sua execução [Cheng et al. 2017].

Similar ao tópico anterior, essa também é uma área muito estudada em ML tradicional que está sendo aos poucos aplicada em FL. Porém, muitas dessas técnicas não tem um mapeamento fácil para o universo federado ou tornam o processo de treinamento mais complexo, complicando sua adoção nesse cenário [Kairouz et al. 2021]. A seguir, discute-se os métodos mais famosos em compressão de modelos, destacando suas vantagens e desvantagens, especialmente em ambientes de IoT federado.

5.4.2.1. Poda

A motivação para podar uma rede neural é remover parâmetros redundantes e desimportantes que não vão ter grande impacto no desempenho, uma vez que redes neurais tendem a ser super-parametrizadas, com vários atributos contendo praticamente a mesma informação. Há dois tipos de poda, conforme o componente que é removido da rede neural: (i) **poda desestruturada:** remove-se pesos ou neurônios individualmente, ao zerar seus valores na matriz de pesos da rede neural, o que aumenta sua esparsidade. Por um lado, uma vez que diversos hardwares e softwares são especializados em operações nesse tipo de matriz, pode-se melhorar o desempenho do modelo em relação ao tempo e energia. Por outro lado, essa necessidade de especialização para acelerar a computação é uma limitação significativa, especialmente em aparelhos genéricos de IoT. (ii) **poda estruturada:** esse tipo de estratégia remove blocos inteiros de pesos, canais e filtros, o que não resultará em matrizes esparsas, superando a necessidade de hardware e software especializado [Han et al. 2015, Cheng et al. 2017].

Uma das formas mais simples e mais utilizadas de poda, tanto estruturada quanto desestruturada, é a poda baseada em magnitude (*magnitude-based pruning*): primeiramente, treina-se a rede neural para aprender quais as conexões importantes; o segundo passo é podar a estrutura desejada com valor abaixo de um valor de corte; e, finalmente, retreina-se a rede neural para que a mesma possa se ajustar de forma a compensar pelos pesos perdidos. Esse processo é repetido por várias iterações. Com esse método, por exemplo, a AlexNet é diminuída de 61 milhões para 6,7 milhões de parâmetros, sem perda de acurácia [Han et al. 2015]. Esse resultado impressionante possibilita a implementação dessas redes grandes em aparelhos IoT.

Apesar da sua simplicidade e capacidade de aplicação em camadas convolucionais e totalmente conectadas, a poda necessita de muito *fine-tuning* e não está claro quão bem generaliza entre diferente arquiteturas. Em muitos casos, é mais efetivo usar uma arquitetura mais eficiente (como modelos compactos) que podar um modelo pré-treinado [Blalock et al. 2020].

A poda pode ser utilizada em ambientes federados de diversas maneiras. Por exemplo, pode-se manter um modelo pré-treinado no servidor e aumentar ou diminuir (podar) suas "cópias" a depender da capacidade do cliente, ou seja, uma poda adaptativa, de forma a acelerar a computação necessária sem perder acurácia [Jiang et al. 2019].

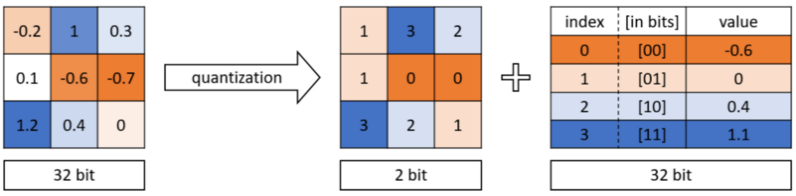

Figura 5.7. Exemplo de binarização, um tipo de quantização que resulta em uma representação em 2 bits [Tamuly 2018]

5.4.2.2. Quantização

Enquanto a poda comprime os modelos de redes neurais através da diminuição da quantidade de pesos, a quantização diminui o tamanho dos parâmetros (pesos, ativações ou gradientes) que existem através da mudança de representação, isto é, alterando o número de bits utilizados.

Há diversas formas de realizar uma quantização, como pode ser visto em [Guo 2018]. A mais simples consiste em pegar os valores máximos e mínimos e dividir pela quantidade de bits esperada, encaixando os valores no intervalo em que estão mais próximos. Por exemplo, em redes neurais profundas, os pesos são tipicamente armazenados como números de ponto flutuante de 32 bits. Ao diminuir essa representação para 2 bits, o que é chamado de binarização, passamos de 2^{32} possíveis valores para somente 4. A Figura 5.7 exemplifica a binarização. Perceba que esse tipo de transformação naturalmente levará a uma perda de "precisão" numérica; como visto na imagem, os valores $-0,6$ e $-0,7$ em 32 bits são representados como 0 em 2 bits. Porém, esse tipo de perda não deve afetar muito o resultado da rede neural profunda, devido a capacidade de lidar com ruídos que estas possuem.

Além disso, embora a quantização de outros parâmetros como ativações ou gradientes utilize ferramentas semelhantes, essa é uma atividade mais complicada, pois deve-se lidar com *bit overflow* e métodos de aproximação de valores [Guo 2018]. Alguns softwares famosos na área de DL possuem funções para quantizar os operadores mais utilizados, como o Tensorflow[9]. Porém, esse tipo de abordagem requer o uso de softwares especializados, o que pode ser visto como uma desvantagem.

Outra desvantagem no uso de quantização é a necessidade de entender o hardware utilizado. Ou seja, é necessário saber como o hardware realiza as operações bit a bit para implementar a quantização desejada. Isso pode ser um problema especialmente em ambientes IoT federados, onde os vários objetos podem possuir diferentes hardwares.

Mais ainda, a quantização torna mais difícil a convergência das redes neurais, necessitando de uma taxa de aprendizado menor para garantir um bom desempenho [Guo 2018]. Isso também torna seu uso mais complicado em ambientes federados, por necessitar de mais rodadas para garantir um bom aprendizado.

Um exemplo de uso de quantização em ambiente federado pode ser visto em [Konečnỳ et al. 2016], onde os autores treinam os modelos sem nenhuma restrição

[9] `https://www.tensorflow.org/lite/performance/post_training_quantization`

nos clientes e quantizam esses valores para diminuir o custo de comunicação entre os clientes e o servidor. O servidor é responsável por decodificar os modelos antes de agregá-los. Essa abordagem é interessante por contornar os problemas descritos anteriormente, apesar deste uso não acelerar a computação dos parâmetros dos modelos, que é o objetivo principal da compressão.

5.4.2.3. Low-rank Factorization

Como dito anteriormente, redes neurais profundas tendem a ser super-parametrizadas, com muita similaridade entre várias camadas ou canais. A ideia de *Low-rank factorization* é então decompor a matriz de pesos de uma camada em uma combinação linear de matrizes menores. Em outras palavras, uma camada mais complexa é decomposta em camadas menores que aproximam os valores da camada original; como discutido na seção anterior, as redes neurais profundas conseguem lidar com essas aproximações sem ter seu desempenho tão afetado, pela sua capacidade de lidar com ruídos.

Diversas vantagens podem ser obtidas utilizando essa estratégia. Por exemplo, a fatoração diminui a memória utilizada pela rede neural e torna as operações menos complexas, o que pode acelerar significativamente o desempenho. Além disso, não há a necessidade de softwares ou hardwares específicos para seu uso. Porém, deve-se tomar cuidado com a necessidade de *fine-tuning* para o reajuste dos pesos das novas camadas, o que pode ser visto como uma desvantagem, principalmente dentro do ambiente federado, por necessitar de mais rodadas.

Há diversas técnicas de fatoração, sendo a decomposição em valores singulares (*Singular Value Decomposition* – SVD) uma das mais conhecidas. Essa técnica é uma fatoração de matrizes, porém, as redes neurais são comumente implementadas como tensores (uma abstração de escalares, vetores e matrizes). Com isso, é necessário generalizar SVD para tensores, o que é bem estudado no contexto de redes neurais profundas. Por exemplo, [Kim et al. 2015] utilizaram a decomposição de Tucker na AlexNet, diminuindo o modelo para 11 MB, o que permitiu sua execução em um *smartphone* com uma perda mínima de acurácia.

Em FL, temos a aplicação descrita em [Qiao et al. 2021], a título de exemplo. Após os clientes realizarem seu treino local, utiliza-se técnicas de *Low-rank factorization* para comprimir os parâmetros do modelo, o qual é enviado para o servidor. O servidor agrega os modelos comprimidos e também fatora esse modelo global antes de enviá-lo aos clientes. Essa fatoração dupla, além de acelerar a computação, diminui o custo de comunicação tanto no envio quanto no recebimento dos modelos pelos clientes.

5.4.2.4. Knowledge Distillation

A motivação para *Knowledge Distillation* (KD) vem ao considerar que treino e inferência são duas atividades diferentes, então, é possível utilizar modelos diferentes para cada tarefa. Assim, um modelo complexo, chamado de professor, é treinado e seu conhecimento é transferido para um modelo de menor capacidade, chamado de estudante, que realizará

as inferências.

Nesse contexto, o conhecimento que será transferido pode ser construído de diversas formas. Tipicamente, esse conhecimento é a distribuição de saída de um modelo professor pré-treinado, mas também pode ser um *ensemble* da distribuição de saída de outros modelos estudantes (esse processo é chamado de co-destilação). Assim, o modelo estudante atualiza seus pesos através da minimização de sua própria função de perda e um regularizador de destilação que penaliza o estudante quando a saída do estudante e do professor (ou do *ensemble* de estudantes) possui uma distância grande.

Uma desvantagem do KD é que muitas decisões devem ser tomadas pelo usuário; por exemplo, os modelos professor e estudante não precisam ter a mesma arquitetura, o que significa decisões de treinamento (como escolher hiper-parâmetros) para duas redes neurais. Porém, isso também significa que esta é uma técnica muito flexível e que pode se adaptar a um grande número de atividades. Isso é especialmente importante no contexto de IoT, que possui as mais diversas aplicações.

Adicionalmente, note que para a construção do conhecimento é necessário que os modelos estudante e professor tenham amostras de treinamento comuns a ambos. Ou seja, é necessário que todos os modelos possam observar os mesmos dados para calcular sua função de perda, demandando troca de dados – o que fere o princípio do aprendizado federado. Uma forma de evitar esse problema, possibilitando o uso dessa técnica em ambientes federados, é a criação de uma base de dados *proxy*, onde as amostras são agrupadas de acordo com suas classes da seguinte maneira: cada estudante armazena a média da distribuição de saída de cada classe obtida durante seu treino local; periodicamente, essa média é enviada para o servidor, que agrega todas as distribuições enviadas por todos Fos modelos estudantes; essa média global é considerada o conhecimento do modelo professor e é usado para auxiliar no aprendizado [Seo et al. 2020].

5.4.2.5. Atenção Seletiva

A atenção seletiva é uma característica do processamento humano de informações: quando olhamos ou ouvimos algo, a informação é processada por partes; vamos nos tornando conscientes das partes quando necessário, ignorando os outros detalhes. Ou seja, vamos aprendendo a dar mais atenção ao que é útil para a identificação do que queremos. Isso permite integrar e processar de forma rápida e eficiente uma quantidade massiva de informação usando recursos limitados de processamento.

Com essa inspiração, mecanismos de atenção seletiva tem por objetivo focar nos elementos de interesse, descartando os que são irrelevantes para a atividade em curso. Em cenários de IoT, onde muitos objetos possuem poder computacional limitado, tais mecanismos podem ser utilizados como um esquema de alocação de recursos, ajudando a selecionar e processar as informações importantes. Por exemplo, na detecção de objetos, pode-se menosprezar o fundo das imagens e objetos de cores diferentes do desejado. Esse método traz diversas vantagens, como aceleração da inferência, modelos menores e ganho de acurácia. Porém, é necessário adicionar o componente de atenção seletiva a rede neural, o que pode tornar o treinamento um pouco mais custoso [Niu et al. 2021].

Umas das formas de utilização de mecanismos de atenção seletiva em FL é descrita em [Ji et al. 2019]. Nela, os autores propõem um método de agregação com atenção (*Attentive Federated Aggregation* - FedAtt), onde aplica-se um mecanismo de atenção em todas as camadas dos clientes, calculando a contribuição de cada camada (e, consequentemente, focando nas camadas mais importantes), de forma a minimizar a distância entre o modelo do servidor e dos clientes.

5.4.2.6. Loterry Ticket Hypothesis

Como visto anteriormente, embora seja possível podar redes neurais densas para pouquíssimos parâmetros (criando uma sub-rede esparsa) sem degradar muito o desempenho, por muito tempo não se conseguia treinar uma sub-rede esparsa *from scratch*. Isso quer dizer que treinar uma rede neural com a mesma topologia que a sub-rede levava a resultados piores de acurácia, dessa forma, sempre era necessário construir a rede neural densa e podá-la.

Em 2019, [Frankle and Carbin 2019] notaram que, após a obtenção das sub-redes esparsas, as mesmas eram inicializadas aleatoriamente. Com isso, eles propuseram a hipótese do tíquete de loteria, definida da seguinte maneira: *"uma rede neural profunda aleatoriamente inicializada contém uma sub-rede que, se reinicializada e treinada isoladamente, pode conseguir resultados de acurácia melhores que a rede original depois de treinada por no máximo o mesmo número de iterações."*. Usando a metáfora do título, vencer a loteria é equivalente a treinar uma rede neural com boa acurácia. Logo, um tíquete de loteria é equivalente aos parâmetros de uma rede neural: comprar um milhão de tíquetes (construir uma rede neural super-parametrizada) te dá mais chances de ganhar na loteria que comprar somente um tíquete (construir uma rede mais simples).

O processo para obter uma sub-rede "vencedora da loteria" é o seguinte: inicializa-se aleatoriamente uma rede neural e a treina; após o treino, a rede é podada, por exemplo, através da poda baseada em magnitude; por fim, reestabelece-se os pesos da sub-rede para os pesos ao qual a rede original foi inicializada. Esse processo é feito iterativamente até se encontrar o nível desejado de esparsidade ou caso a acurácia caia significativamente.

As sub-redes "vencedoras da loteria" são até 20% menores que as originais e podem ter a mesma, ou até superar, a acurácia das redes originais, com no máximo o mesmo número de iterações. Porém, as iterações necessárias para a poda é uma desvantagem, por ser muito caro computacionalmente.

Em [Li et al. 2020a], os autores aplicam os conceitos da hipótese do tíquete de loteria no contexto de aprendizado federado. Neste trabalho, cada cliente deve aprender uma sub-rede esparsa, onde a poda é feita de acordo com os dados locais. O servidor agrega os modelos utilizando somente os parâmetros das sub-redes, e cada cliente recebe os parâmetros do seu próprio modelo. Além das vantagens do aprendizado federado, essa técnica permite a criação de um modelo personalizado para cada cliente.

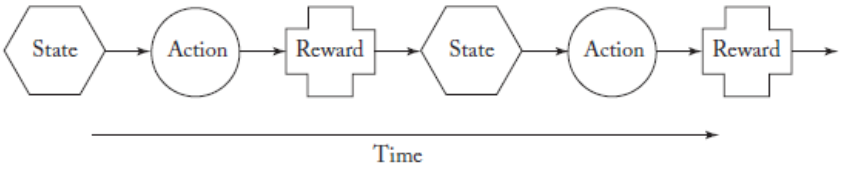

Figura 5.8. Ciclo de estado-ação-recompensa-estado. Fonte: [Yang et al. 2019a]

5.5. Aprendizado por Reforço para FL

Aprendizado por reforço (RL) é um ramo do aprendizado de máquina (ML) que lida principalmente com a tomada de decisão sequencial [Sutton et al. 1998]. Um problema de RL geralmente consiste em um ambiente dinâmico e um agente (ou agentes) que interage(m) com o ambiente. O ambiente evolui uma vez que o agente seleciona uma ação com base no estado atual do ambiente, apresentando uma recompensa pela avaliação do desempenho do agente. O agente busca atingir uma meta no ambiente tomando decisões sequenciais. Os problemas tradicionais de RL podem ser formulados como Processos de Decisão de Markov (MDPs). O agente tem que enfrentar um problema de tomada de decisão sequencial para maximizar uma função de valor (ou seja, a soma esperada de recompensas descontadas ou expectativas de recompensa).

Conforme mostrado na Figura 5.8, o agente observa o estado do ambiente e, a seguir, seleciona uma ação com base no estado. O agente deve receber uma recompensa do ambiente com base nesta ação selecionada. Em MDPs, o próximo estado do ambiente depende do último estado e da ação selecionada pelo agente. A ação que resulta na "maior recompensa esperada" geralmente se refere à ação que coloca o agente no estado com o maior potencial para ganhar recompensas no futuro. O agente se move em ciclos de estado-ação-recompensa-estado (SARS).

Existem várias dificuldades baseadas nesse problema, dentre eles: (i) Um agente tem conhecimento limitado sobre as ações ideais para um determinado estado do ambiente. Considerando o processo de tomada de decisão de uma rodada no problema RL com espaço de ação contínuo, o agente tem que lidar com um problema de otimização com espaço contínuo, o que pode exigir um grande esforço de computação. (ii) As ações do agente podem afetar os estados futuros do ambiente, afetando assim as opções e oportunidades disponíveis para o agente no futuro. Portanto, ao lidar com problemas sequenciais de tomada de decisão, cada agente não deve escolher ações avidamente, mesmo que essas ações possam obter boas recompensas em um curto prazo. Ou seja, o agente precisa fazer um trade-off entre a recompensa atual e as expectativas de recompensa futura. (iii) A seleção de ações ideais requer levar em consideração as consequências indiretas e atrasadas das ações e, portanto, pode exigir previsão ou planejamento.

5.5.1. Algoritmos de Aprendizado por reforço

Os algoritmos RL podem ser categorizados de acordo com os seguintes elementos-chave.

Baseado em modelo vs. livre de modelo: Os métodos baseados em modelo tentam construir um modelo virtual do ambiente primeiro e, em seguida, agir de acordo com a

melhor política derivada do modelo virtual. Já os métodos livres de modelo assumem que o modelo do ambiente não pode ser construído e estimam a função de política e valor por tentativa e erro.

Baseado em valor vs. baseado em política: Os métodos baseados em valor tentam aprender uma função de valor e inferir uma política ótima a partir dela. Os métodos baseados em política pesquisam diretamente no espaço dos parâmetros de política para encontrar uma política ótima.

Atualização por Monte Carlo vs. atualização por diferença temporal (TD): A atualização por Monte Carlo avalia uma política usando a recompensa acumulada durante todo o episódio. Isso é direto na implementação, mas eles requerem um grande número de iterações para convergência e sofrem uma grande variação ao estimar sua função de valor. Em vez de usar a recompensa total acumulada, o TD update calcula um erro temporal, que é a diferença entre a nova e a antiga estimativa da função de valor, para atualizar a política. Este tipo de atualização só precisa das iterações mais recentes e reduz a variância. No entanto, o viés aumenta durante a estimativa, pois a visão global de todo o episódio não é considerada.

Política-On vs. política-off: Os métodos política-On usam a política atual para gerar ações e atualizar a própria política atual de acordo. Os métodos política-off usam uma política exploratória diferente para gerar ações e a política de destino é atualizada com base nessas ações. Dois algoritmos TD que têm sido amplamente usados para resolver problemas de RL são Estado-Ação-Recompensa-Estado-Ação (SARSA) e Q-Learning.

SARSA é um algoritmo TD com uma abordagem política-on [Rummery and Niranjan 1994], pois segue a mesma política para encontrar a próxima ação. Ele tenta aprender uma função de valor de ação em vez de uma função de valor. A etapa de avaliação de política usa o erro temporal para a função de valor da ação, que é semelhante à função de valor.

Q-Learning é um algoritmo TD política-off [Watkins and Dayan 1992], pois seleciona a próxima ação de forma gananciosa, em vez de seguir a mesma política. A função Q é atualizada usando uma política que é diretamente gananciosa em relação à função Q atual.

5.5.2. Aprendizado por Reforço Federado

O RL tem muitos aspectos técnicos e não técnicos durante as implementações que dem ser observados. Um das questões mais críticas é como evitar o vazamento de informações e preservar a privacidade do agente. Essa preocupação leva a versões que preservam a privacidade de RL — Federated Reinforcement Learning (FRL). Aqui nós categorizamos as pesquisas FRL em Aprendizado por Reforço Federado Horizontal (HFRL) e Aprendizado por Reforço Federado Vertical (VFRL).

5.5.2.1. Aprendizado por Reforço Federado Horizontal

RL Paralelo [Kretchmar and Jacobvitz 2002, Grounds and Kudenko 2005] tem sido estudado há muito tempo na comunidade de pesquisa em RL, na qual vários agentes são

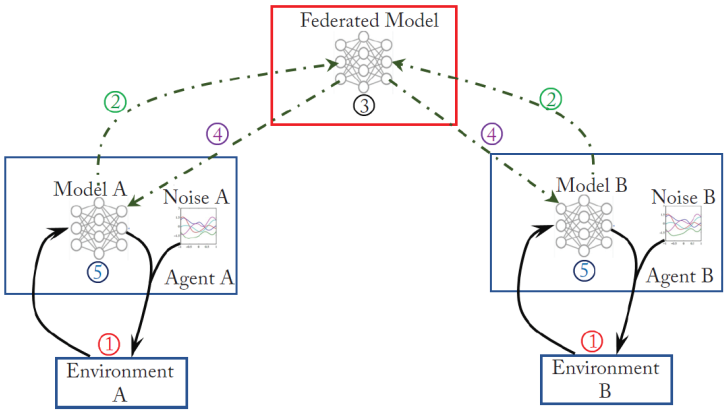

Figura 5.9. Aprendizado por Reforço Federado Horizontal. Fonte: [Yang et al. 2019a]

assumidos para executar a mesma tarefa (com as mesmas recompensas para estados e ações). Os agentes podem realizar o aprendizado em diferentes ambientes. Observe que a maioria das configurações de RL paralelas adotam as operações de transferência de experiência ou gradientes dos agentes. É simples que tais operações não podem ser conduzidas ao considerar questões de preservação de privacidade. Portanto, é natural adotar o HFRL para questões de preservação da privacidade. A comunidade HFRL adota essas configurações básicas em RL paralelo com o objetivo de preservação da privacidade como uma restrição extra (para o servidor e os agentes).

Um framework básico para a condução do HFRL é apresentado na figura 5.9.

Como pode ser visto na Figura 5.9, HFRL contém vários agentes RL paralelos (apresentamos dois agentes por simplicidade) para diferentes sistemas que podem ser geograficamente distribuídos. Os agentes RL têm a mesma tarefa em sistemas distintos. Um servidor federado assume a função de agregar os modelos de diferentes agentes RL. As etapas básicas para conduzir o HFRL são listadas a seguir.

Etapa I. Todos os agentes RL participantes treinam seus próprios modelos RL local e independentemente, sem nenhuma troca de experiência de dados, gradientes de parâmetro e perdas.

Etapa II. Os agentes RL enviam seus parâmetros de modelo mascarados ao servidor.

Etapa III. O servidor federado criptografa os modelos de agentes RL não idênticos e conduz métodos de agregação para obter um modelo federado.

Etapa IV. O servidor federado envia o modelo federado aos agentes RL.

Etapa V. Os agentes RL atualizam o modelo local.

[Nadiger et al. 2019] descreveu completamente a arquitetura geral para HFRL, que contém a política de agrupamento, a política de aprendizagem e a política de federação para os agentes RL participantes. Os autores demonstraram ainda a eficácia da arquitetura proposta com base no jogo Pong do Atari. Nas demonstrações, os autores

mostraram que com a abordagem proposta, há uma melhora mediana de 17 % no tempo de personalização. Embora o objetivo de preservação da privacidade possa apresentar mais desafios, podemos nos beneficiar do HFRL das seguintes maneiras: (i) **Para evitar amostras não independentes e identicamente distribuídas (Non-iid)**: É comum que o agente de tarefa única possa encontrar amostras não-i.i.d. durante o processo de aprendizagem. É claro que um dos principais motivos é que para tarefas RL com uma configuração de agente único, a experiência adquirida posteriormente pode estar fortemente relacionada com a experiência anterior, o que pode quebrar a suposição i.i.d. dos dados. O HFRL pode fornecer benefícios para a construção de um sistema de aprendizado de reforço mais preciso e estável. (ii) **Para aumentar a eficiência da amostra**: Outra desvantagem dos métodos convencionais de RL é a baixa capacidade de construir rapidamente modelos estáveis e precisos com amostras limitadas (conhecido como problema de baixa eficiência de amostra), o que impede que os métodos convencionais de RL sejam aplicados no mundo real. Sob o HFRL, podemos agregar conhecimento extraído por diferentes agentes de ambientes não idênticos para resolver o problema de baixa eficiência da amostra. (iii) **Para acelerar o processo de aprendizagem**: Na verdade, esse benefício pode ser obtido a partir das duas vantagens acima. Combinado com o poderoso framework de FL para agregar diferentes conhecimentos aprendidos por agentes não idênticos, experiência de mais amostras não-i.i.d. podem acelerar o aprendizado de RL e obter melhores resultados.

5.5.2.2. Aprendizado por Reforço Federado Vertical

Da mesma maneira que foi discutido anteriormente o aprendizado federado vertical, em FRL, existe a possibilidade de diversas empresas colaborarem para compartilhar dados de natureza distinta, porém, de ambientes comuns. Esta estrutura cooperativa se enquadra na categorização de VFRL. No VFRL, existem diferentes agentes RL que mantêm observações não idênticas do mesmo ambiente. Cada agente RL mantém uma política de ação correspondente (alguns agentes podem não ter uma política de ação). O objetivo principal da estrutura cooperativa é treinar um agente de RL mais eficaz com o conhecimento misto extraído das observações de diferentes agentes cooperativos. Durante o treinamento ou no processo de inferência, qualquer transformação direta de dados brutos é proibida. A seguir apresentamos uma estrutura possível para VFRL — Federated DQN [Yang et al. 2019a]. Como pode ser observado a seguir, nomeamos o agente RL que obtém a recompensa do ambiente como agente Q-network, e todos os outros agentes como agentes RL cooperativos:

Etapa I. Todos os agentes participantes do aprendizado por reforço realizam ações de acordo com as observações atuais do ambiente e do conhecimento extraído. Observe que alguns agentes podem não fazer nenhuma ação, o que apenas mantém suas próprias observações do ambiente.

Etapa II. Os agentes RL obtêm o feedback correspondente do ambiente, incluindo as observações do ambiente atual, a recompensa, etc.

Etapa III. Os agentes RL calculam os produtos intermediários alimentando as observa-

ções obtidas em sua rede neural e, em seguida, enviam os produtos intermediários mascarados ao agente Q-network.

Etapa IV. O agente Q-network criptografa todos os produtos intermediários e treina a Q-network com as perdas atuais por meio de propagação reversa.

Etapa V. O agente Q-network envia de volta os gradientes de peso mascarados para os agentes cooperativos.

Etapa VI. Cada agente cooperativo criptografa os gradientes e atualiza sua própria rede.

Na literatura, o trabalho existente que se enquadra na categoria VFRL é [Zhuo et al. 2020], que investiga o problema do sistema RL multiagente de forma cooperativa, ao considerar os requisitos de preservação da privacidade dos dados, gradientes e modelos do agente. O framework FRL estudado corresponde à arquitetura VFRL que apresentamos acima (que é denominada VFRL posteriormente). O autor apresentou sistemas detalhados da vida real onde o VFRL é significativo. Modelar e explorar os comportamentos em sistemas com múltiplos agentes cooperativos ou adversários tem sido um desafio interessante para a comunidade RL [Mao et al. 2018] [Foerster et al. 2016]. No domínio da FL, os agentes podem realizar tarefas heterogêneas, com diferentes estados, ações ou recompensas (alguns podem não ter recompensas ou ações). O principal objetivo de cada agente VFRL é construir um modelo RL estável e preciso de forma cooperativa ou competitiva, sem troca direta de experiência (incluindo estados, ações e recompensas) ou os gradientes correspondentes. Comparado com o RL multiagente, as vantagens do VFRL podem ser resumidas como segue: (i) **Para evitar vazamento de informações do agente e do usuário.** Nos sistemas de caldeiras a carvão, um benefício direto apresentado para o departamento de gerenciamento de dados meteorológicos é que ele pode aumentar a eficiência da produção sem qualquer vazamento de dados meteorológicos brutos em tempo real. Isso pode ser lançado como um serviço que pode ser publicado para todos os usuários externos em potencial. (ii) **Para melhorar o desempenho do aprendizado por reforço**. Com os métodos adequados de extração de conhecimento adotados, podemos treinar um agente RL mais razoável e robusto para aumentar a eficiência. O VFRL é vantajoso no sentido de que pode permitir que um sistema de aprendizado aproveite essas informações enquanto preserva a privacidade.

5.6. Computação de borda e Aprendizado Federado

Computação de borda trata do processamento de aplicativos e serviços executados em uma infraestrutura de computação (nós de borda) mais perto das entidades finais (usuários e dispositivos, e genericamente chamados de dispositivos ou nós de borda) e fora de um centro de processamento de dados (*data center*), normalmente na nuvem. Assim, Computação de borda leva a computação e o armazenamento de dados para mais perto de onde são necessários, para melhorar o tempo de resposta e a largura de banda de comunicação.

Este paradigma de computação distribuída é um mecanismo promissor para lidar com as deficiências e limitações da computação na nuvem na assistência a aplicativos e serviços que precisam de eficiência energética e são sensíveis tanto a atrasos quanto ao

contexto em uma ampla gama de cenários, incluindo a Internet das Coisas, computação urbana e mobilidade inteligente. Em vez de usar recursos de computação e armazenamento para executar esses aplicativos e serviços em uma nuvem, computação de borda tem o potencial de prover soluções rápidas, eficientes e inteligentes, que simplesmente não são possíveis em uma solução de nuvem tradicional.

Computação de borda traz vários benefícios para o uso de aprendizado federado. Na segurança, a natureza centralizada da computação em nuvem torna aprendizado federado particularmente vulnerável a ataques de DDoS (*Distributed Denial of Service*), o que não é o caso de computação de borda. No projeto de sistemas confiáveis, computação de borda permite o acesso mais fácil a serviços de aprendizado federado quando comparado a uma solução em nuvem. A classificação e reconstrução de dados pode ser feita de forma mais efetiva por aprendizado federado em uma solução de computação de borda.

5.6.1. Aplicações de Aprendizado Federado em Cenários de Computação de Borda

Dadas as características comuns tanto da computação de borda quanto da aprendizado federado, a computação de borda é um ambiente naturalmente adequado para aplicar aprendizado federado. Assim, o aprendizado federado de borda (*edge federated learning*) é uma área de investigação cada vez mais atraente tanto na academia quanto na indústria.

O aprendizado federado de ponta resolve o problema do isolamento de dados (*data island problem*) ao explorar o enorme potencial de dados existentes em dispositivos terminais "isolados", i.e., que estão na ponta, sem violar a privacidade do usuário. Esse tratamento conjunto permite melhorar a eficiência do aprendizado de modelo em sistemas de computação de borda. Portanto, essa estratégia pode ser amplamente utilizada em cenários nos quais a privacidade e a utilização de recursos são críticas. A seguir, apresentaremos alguns cenários para o aprendizado federado de ponta e alguns trabalhos recentes aplicados nesses cenários.

Saúde: O aprendizado profundo normalmente apresenta um excelente desempenho em tarefas complexas de reconhecimento de padrões, o que faz com que essa técnica seja amplamente utilizada na área da saúde. Suponha o cenário onde há um conjunto de instituições médicas (e.g., hospitais, clínicas, laboratórios) onde em cada uma os seus dados são armazenados e processados separadamente no nó de borda (cenário típico de hoje em dia). Nesse caso, um modelo treinado com dados coletados de uma instituição médica individual dificilmente terá uma precisão satisfatória quando aplicado aos dados "invisíveis" que de alguma forma não são corrigidos com os dados de treinamento. Para termos um modelo médico adequado precisamos de uma grande quantidade de prontuários médicos, o que é difícil de obter devido à privacidade dos dados médicos. O aprendizado federado de borda pode ajudar a superar esse problema, permitindo às instituições médicas colaborarem em modelos de treinamento sem compartilhar dados de pacientes e, assim, atendendo aos requisitos de proteção de privacidade de dados. Além das instituições médicas, o método de transferência de aprendizado federado de borda (edge federated transfer learning) pode ser aplicado a dispositivos de saúde pessoal, ou seja, a algum tipo de *wearable device*. Dispositivos de saúde pessoal, como medidor de pressão arterial e dispositivo de reconhecimento de atividade, são utilizados para observar

as condições de saúde e enviar alarmes em tempo real, sendo uma atividade central em sistemas inteligentes de saúde [Xu et al. 2021]. Para os usuários, é necessário ter um modelo disponível no início e treinar um modelo personalizado que deve ser atualizado periodicamente em função das condições físicas. Neste cenário, modelos de aprendizado federado podem ser usados para aprender de forma colaborativa o comportamento de indivíduos na borda da rede mantendo a personalização dos envolvidos [Sheller et al. 2020].

Redes veiculares: Veículos modernos têm literalmente centenas de sensores que podem gerar um grande volume de dados como localização, orientação, imagens e estado de seus diversos componentes como motor, suspensão e freio. Esses dados são valiosos para os fabricantes de veículos que podem projetar serviços inteligentes de navegação e avisos antecipados, por exemplo. O sistema computacional do veículo coleta dados de sensoriamento gerados localmente e, em seguida, repassa para o sistema de computação de borda veicular (VEC – Vehicle Edge Computing) que treina o modelo de aprendizado local. O aprendizado federado de borda no VEC pode atender às necessidades dos usuários para a tomada de decisão em veículos inteligentes. Por exemplo, à medida que tivermos mais veículos elétricos sendo utilizados, será importante haver uma solução de aprendizagem de demanda de energia federada para fazer a previsão de demanda de energia onde esses veículos elétricos se encontram/planejam ir. No caso de veículos autônomos, teremos mais sensores do que veículos comuns, como LiDAR e sensores ultrassônicos para perceber o ambiente ao redor sem a necessidade da interação humana. Nesse cenário, aprendizado federado de borda é uma solução adequada na computação de borda veicular pois preserva a privacidade de dados veiculares.

Recomendação inteligente: Em aplicativos para smartphones e desktops, a recomendação inteligente (smart intelligence) é uma função útil para prever escolhas de usuários e, assim, oferecer opções. Em comparação com as abordagens de aprendizado de máquina, o aprendizado federado de borda é capaz de treinar modelos flexíveis para tarefas de recomendação. Como os nós de borda estão localizados tendem a ter tarefas semelhantes por razões de eficiência e custo, esse semelhança entre os nós de borda pode ser usada para treinar modelos adaptativos por aprendizado federado de borda. Por exemplo, o Google Keyboard (Gboard) treina modelos usando aprendizado federado de borda em escala global para sugestão de pesquisa do teclado virtual e previsão de emojis. Os resultados da avaliação mostram que os modelos em cada nó de borda têm um bom desempenho já que os modelos são ajustados a diferentes estilos de linguagem e cultura em uma área específica. A mesma estratégia pode ser aplicada a um navegador (*browser*) treinado com aprendizado federado que pode ajudar os usuários a encontrar rapidamente o endereço da página que precisam, inserindo menos caracteres. Essa estratégia pode ser aprimorada em sistemas de aprendizado federado de ponta para fornecer a diferentes usuários modelos relativamente personalizados, explorando as semelhanças do usuário sem violar a sua privacidade.

5.6.2. Segurança e Privacidade

Segurança e privacidade são dois grandes problemas para implementar o aprendizado federado em computação de borda. Devido ao ambiente heterogêneo natural do aprendizado federado e da computação de borda, é sempre difícil prever atividades de outras entidades no sistema. Por exemplo, no processo de treinamento de aprendizado federado de borda, devido ao desconhecimento do papel exercido por outras entidades, algumas delas podem ser maliciosas e atacar o processo de treinamento. Além disso, entidades "curiosas" podem querer aprender/saber/furtar dados pessoais e privados. Neste cenário, a segurança e a privacidade dos dispositivos de borda podem ser prejudicadas.

Questões de segurança surgem na ponta do aprendizado federado devido à heterogeneidade tanto da computação de borda quanto do aprendizado federado. Mais ainda, nesse ambiente heterogêneo, nós de borda podem não ser confiáveis. Por outro lado, no aprendizado federado, geralmente assumimos que todos os nós mantêm seus próprios dados privados e não os compartilham com outras entidades. Esses recursos melhoram a generalidade do aplicativo da computação de borda e mantêm mais privacidade para os dados privados dos usuários, mas também aumentam o risco de sofrer ataques maliciosos. Muito trabalho tem sido feito para resolver questões de segurança no aprendizado federado de ponta. Primeiro introduziremos duas maneiras principais de injetar ataques seguidos por alguns algoritmos existentes para defender ataques.

Atualmente, os métodos de defesa propostos para aprendizado federado em computação de borda assumem que a distribuição de dados é I.I.D. em todos os nós ou a maioria dos nós opera corretamente. No entanto, esses pressupostos não são práticos no aprendizado federado de ponta. Primeiro, geralmente os nós de borda coletam seus dados de suas próprias fontes, de modo que a distribuição de dados não é necessariamente I.I.D. Em segundo lugar, no momento de uma sincronização, o servidor central seleciona aleatoriamente alguns dos nós de borda para executar a computação. Nesse caso, os nós honestos podem não ser a maioria e, consequentemente, pode não ser razoável supor haver uma maioria honesta durante todo o processo de treinamento.

Portanto, é necessário investigarmos algoritmos mais práticos sem assumir a distribuição de dados I.I.D. e maioria honesta para defender o aprendizado federado de borda de ataques. Essa é uma área de pesquisa promissora que precisa de mais estudos.

O aprendizado federado é projetado para proteger os dados privados de treinamento de cada nó sem depender da transmissão dos dados de treinamento entre servidores. No entanto, a violação de privacidade ainda pode ocorrer quando as informações (por exemplo, pesos de modelo) são compartilhadas entre os elementos federados. É possível que alguns nós ou servidores maliciosos possam extrair informações privadas durante o processo de treinamento. Portanto, a fim de atacar a privacidade de alguns dos nós de borda, o processo do adversário é recuperar informações privadas do conjunto de dados de pesos carregados ou pesos agregados [Melis et al. 2019]. Esse problema equivale à recuperação dos dados da atualização de peso. Em geral, existem dois tipos de ataques de privacidade na área de aprendizado federado: inferência de membros e inferência de dados. O ataque de inferência de membros tem como objetivo determinar se um registro de dados está contido no conjunto de dados de treinamento de um nó. Quando o conjunto de dados é sensível, este ataque pode vazar informações úteis. O ataque de inferência

de dados tem como objetivo recuperar dados de treinamento ou uma classe de dados de treinamento das informações que o nó fornece.

O problema de privacidade pode se tornar mais grave na computação de borda, já que os dispositivos de borda podem vazar informações sobre dados, uso e localização para usuários mal-intencionados. Como preservar a privacidade no aprendizado federado de borda considerando as especificidades da computação de borda é uma nova direção de pesquisa que deve ser investigada.

5.7. Desafios e Oportunidades de Pesquisa

Nessa seção, discutiremos alguns problemas de pesquisa que são de grande interesse da comunidade de aprendizado federado e de IoT. Essa seção não tem como objetivo exaurir todas os problemas de pesquisa em aberto na área. Para uma discussão mais completa, recomendamos o texto [Kairouz et al. 2021].

5.7.1. Dados Não Independentes e Identicamente Distribuídos - Non-IID

Em modelos de aprendizado de máquina tradicional, coletamos amostras da população de interesse seguindo os preceitos da amostragem estatística, que nos orientam a termos amostras que represente bem a população em estudo para que possamos realizar inferências confiáveis. Nesse sentido, a técnica mais empregada é a amostragem realizada de maneira aleatória e que tenha uma quantidade de elementos suficiemente grande de modo a representar toda a varabilidade da população. As aplicações de aprendizado de máquina centralizado tradicionais, em geral, seguem esse princípio que busca produzir amostras independentes e identicamente distribuídas (IID). Entrentanto, de acordo com [Zhao et al. 2018], na prática, não é realista assumir que os dados locais em cada dispositivo de borda são sempre IID. [McMahan et al. 2017] demonstrou que FedAvg pode trabalhar com certos dados não-IID. No entanto, a precisão das redes neurais convolucionais (CNN) treinadas com o algoritmo FedAvg pode reduzir significativamente, até 11 % para MNIST, 51 % para CIFAR-10 e 55 %, com alta quantidade de dados não-IID altamente assimétricos (quando cada cliente treina com apenas 1 classe).

Para lidar com o desafio estatístico do aprendizado federado, [Zhao et al. 2018] demonstram o limite na divergência no treinamento pela distância do movimentador de terra (EMD) entre a distribuição por classes em cada dispositivo (ou cliente) e a distribuição da população. Esse limite é afetado pela taxa de aprendizado, etapas de sincronização e gradientes. Em seguida, ele propõe uma estratégia de compartilhamento de dados para melhorar o FedAvg com dados não-IID, distribuindo uma pequena quantidade de dados compartilhados globalmente contendo exemplos de cada classe. Essa estratégia apresenta uma compensação entre precisão e centralização. Os experimentos no artigo mostram que pode aumentar a precisão no CIFAR-10 em 30% se estivermos dispostos a compartilhador 5% dos dados.

Para os experimentos IID, as curvas de convergência de FedAvg com um tamanho de lote B se sobrepõem principalmente às curvas SGD (Gradiente Decendente Estocástico Centralizado) com B $\times$ 10 para todos os três conjuntos de dados analisados pelos autores. Isso acontece porque o modelo global produzido pelo FedAVG representa a média dos 10 clientes avaliados (cada cliente foi treinado com 1 classe). Dessa maneira, o FedAVG para

dados IID deve ser comparável com SGD com um tamanho de lote 10 vezes maior. Assim, o FedAvg, nos experimentos de [Zhao et al. 2018], atinge precisão consistente com os resultados obtidos por [McMahan et al. 2017]. Uma redução significativa na precisão do teste é observada para FedAvg em dados não-IID em comparação com SGD, em tamanhos de lote correspondentes. Além disso, os modelos pré-treinados pelo SGD podem não convergir com o treinamento FedAvg em dados não-IID extremos. Para CIFAR-10, a precisão cai quando o FedAvg treina CNNs pré-treinadas em dados não-IID. Assim, foi demonstrado uma redução na precisão do teste FedAvg para dados não-IID. A precisão demonstrada pelo modelo centralizado utilizado não é o estado da arte, mas as CNNs treinadas foram suficientes para avaliar o aprendizado federado em dados não-IID.

Para [Hsieh et al. 2019], existem algumas maneiras de lidar com dados não-IID. Uma deles seria o *batch normalization* para dados não-IID. O *batch normalization* (*BatchNorm*) é um dos mecanismos de DL mais populares (mais de 20.000 citações em agosto de 2020). *BatchNorm* visa estabilizar uma DNN (Deep Neural Network) normalizando a distribuição de entrada para média zero e variância unitária. Como a média e a variância globais são inatingíveis com o treinamento estocástico, o *BatchNorm* usa a média e a variância de *minibatch* para estimar a média e a variância globais. Especificamente, para cada *minibatch* β, o *BatchNorm* calcula a média do *minibatch* μ_β e a variação θ_β, e então usa μ_β e θ_β para normalizar cada entrada em β. *BatchNorm* permite um treinamento mais rápido e estável porque permite maiores taxas de aprendizado.

Ainda para [Hsieh et al. 2019], como o problema de *BatchNorm* na configuração não-IID é devido à sua dependência de *minibatch*, a solução natural é substituir *BatchNorm* por mecanismos de normalização alternativos que não dependem de *minibatch*. Infelizmente, a maioria dos mecanismos de normalização alternativos existentes, Normalização de Camada e Renormalização em Lote, têm suas desvantagens. Em vez disso, um mecanismo particular pode ser o *Group Normalization* (*GroupNorm*). *GroupNorm* é um mecanismo de normalização alternativo que visa superar a normalização de *BatchNorm* e de camada (*LayerNorm*). *GroupNorm* divide canais adjacentes em grupos de um tamanho pré-especificado g_{size} e calcula a média e variância por grupo para cada amostra de entrada. Consequentemente, *GroupNorm* não depende de *minibatch* para normalização (a deficiência de *BatchNorm*), e *GroupNorm* não assume que todos os canais fazem contribuições iguais (a deficiência de *LayerNorm*). Eles avaliaram *GroupNorm* com BN-LeNet sobre CIFAR-10. Eles selecionaram $g_{size=2}$, que funciona melhor com esta DNN. Primeiro, o *GroupNorm* recupera com sucesso a perda de precisão do *BatchNorm* com BSP na configuração Não-IID. *GroupNorm* com BSP atinge 79,2% de precisão de validação na configuração Não-IID, que é tão boa quanto a precisão da configuração IID. Isso mostra que o *GroupNorm* pode ser usado como uma alternativa ao *BatchNorm* para superar o desafio de dados não-IID para o BSP. Em segundo lugar, o *GroupNorm* ajuda drasticamente os algoritmos de aprendizado descentralizado também na configuração não-IID. Com *GroupNorm*, houve 14,4%, 8,9% e 8,7% de perda de precisão para Gaia, *FederatedAveraging* e *DeepGradientCompression*, respectivamente. Embora as perdas de precisão ainda sejam significativas, elas são melhores do que suas contrapartes *BatchNorm* por um aditivo de 10,7%, 19,8% e 60,2%, respectivamente.

Como uma tentativa de contornar a situação, [Jeong et al. 2018] propôs o *Federated Augmentation* (FAug) para lidar com a sobrecarga do balanceamento de dados. Cada

dispositivo recebe rótulos de dados de outros dados e gera localmente amostras de dados ausentes com base nos rótulos, usando um modelo generativo, e o modelo generativo é treinado no servidor com alto poder de computação e conexão à Internet. Cada dispositivo reconhece os rótulos ausentes nas amostras de dados, conhecidos como rótulos de destino, e carrega algumas sementes de amostras de dados desses rótulos de destino para o servidor. Em seguida, o servidor cria novas amostras das sementes de amostras de dados carregadas usando a pesquisa de imagens do Google para os dados visuais a serem treinados usando uma *Generative Adversarial Network* (GAN). Em seguida, baixa o gerador da GAN treinado que permite que cada dispositivo reabasteça os rótulos de destino até atingir um conjunto de dados de treinamento IID, o que reduz significativamente a sobrecarga de comunicação em comparação com trocas diretas de amostra de dados, além de não copiar diretamente dados dos clientes, quebrando assim, uma prermissa básica do FL.

Para [Duan et al. 2019], os dados de treinamento de cada cliente devem ser reequilibrados para resolver o problema de degradação da precisão. Um método é redistribuir os dados do cliente local até que a distribuição seja uniforme. No entanto, o compartilhamento de dados levanta um problema de privacidade e causa alta sobrecarga de comunicação. Outra forma de reequilibrar o treinamento é atualizar o modelo global de forma assíncrona. Cada cliente calcula as atualizações com base no modelo global mais recente e aplica suas atualizações ao modelo global sequencialmente. Essas atualizações implicam em uma sobrecarga de comunicação e o consumo de tempo do método maiores do que FL convencional. Os autores propuseram um framework chamado Astraea, que combina essas duas ideias, apresentando mediadores entre o servidor FL e os clientes para reequilibrar o treinamento. O fluxo de trabalho do Astraea inclui inicialização, reequilíbrio, treinamento e agregação. Na fase de inicialização, o servidor FL primeiro espera que os dispositivos móveis ingressem na tarefa de treinamento do modelo FL. Os dispositivos participam do treinamento enviando suas informações de distribuição de dados locais para o servidor. Depois de determinar os dispositivos (clientes) envolvidos no treinamento, o servidor FL conta a distribuição global de dados e inicializa os pesos e o otimizador do modelo de aprendizagem. Na fase de reequilíbrio, o servidor primeiro calcula o número de aumentos para cada classe com base na distribuição global. Em seguida, todos os clientes realizam aumento de dados em paralelo de acordo com os resultados do cálculo. A função de aumento de amostra pega uma amostra e gera aumentos, incluindo deslocamento aleatório, rotação aleatória, cisalhamento aleatório e zoom aleatório, para a amostra. O objetivo do aumento de dados é mitigar o desequilíbrio global, em vez de eliminá-lo. Ao mesmo tempo, um aumento significativo no aumento de dados irá gerar muitas amostras semelhantes, tornando o treinamento do modelo mais sujeito a *overfitting*.

Apesar de termos apresentado algumas propostas para endereçar o problema de dados não-IID, essa é uma área que ainda tem muito a ser explorada no contexto de FL [Kairouz et al. 2021].

5.7.2. Busca Automática de Modelos

Algoritmos de DL tem avançado o estado-da-arte em diversas áreas, como visão computacional e processamento de linguagem natural. Dessa forma, naturalmente, estes tornaram-se a primeira opção para as mais diversas atividades. Porém, o design de uma rede neural

profunda que atinja bons resultados é um processo tedioso, complexo e que exige grande esforço humano, em um longo exercício de tentativa e erro, visto que diversos aspectos das redes devem ser considerados, como a topologia, hiper-parâmetros, algoritmos de aprendizado (e seus hiper-parâmetros), etc. Isso levou ao surgimento da Busca de Arquitetura Neural (*Neural Architecture Search* – NAS), ou seja, sistemas semi-automáticos que buscam o melhor modelo de rede neural para a resolução de um problema, com mínima intervenção humana [Zhu et al. 2021, Elsken et al. 2019]. NAS é parte do Aprendizado de Máquina Automático (*Automated Machine Learning* – AutoML).

Para uma certa atividade, há possibilidades infinitas de arquiteturas e modelos que podem ser explorados. Consequentemente, há muitas formas de buscar através de tais possibilidades. De acordo com [Elsken et al. 2019], o procedimento para a construção de métodos de NAS devem considerar os três seguintes aspectos: (i) **Espaço de busca:** visto que redes neurais podem ser compostas dos mais variados elementos, como convolução, *pooling*, funções de ativação, etc., definir o espaço de busca consiste em determinar quais os possíveis componentes que podem ser descobertos durante o processo de busca. Inevitavelmente, essas determinações inserem um viés humano no processo, o que pode impedir o método de encontrar arquiteturas eficientes que vão além do conhecimento humano. Além disso, um espaço de busca muito grande pode dificultar a otimização pela alta dimensionalidade. Dessa forma, é necessário encontrar um balanço entre essas duas características, o que é um desafio que ainda tem muito a ser explorado. (ii) **Estratégia de busca:** o algoritmo que guia a exploração do espaço de busca, isto é, o que determina a próxima arquitetura que será explorada. É possível utilizar um algoritmo aleatório, o que pode não ser uma boa escolha, devido as enormes possibilidades de arquiteturas dentro do espaço de busca. Dessa forma, tipicamente se usa algoritmos que consideram o desempenho de arquiteturas já descobertas, como Otimização Bayesiana, Algoritmos Evolucionários, Aprendizado por Reforço e Métodos baseados no Gradiente. Note que esse processo consome tempo e poder computacional. Dessa forma, é essencial buscar alternativas que diminuam o custo dessa atividade. (iii) **Estratégias para estimativa de desempenho:** determina como medir o desempenho da arquitetura candidata em dados não vistos, para guiar a estratégia de busca a encontrar as melhores arquiteturas. A forma mais simples é treinar as arquiteturas encontradas e medir sua acurácia nas bases de dados de teste; mas isso pode ser muito custoso, podendo levar muitos dias para finalizar. Assim, outras estratégias estão sendo exploradas para aceleração da estimação de desempenho, como *Weight Sharing*, *One-shot Models*, entre outros. Essa aceleração é uma questão importante nesse contexto e que permanece como um desafio em aberto.

Pode-se notar que NAS é um grande campo de pesquisa, o qual ainda é pouco explorado, mas que apresenta grande potencial, especialmente em ambientes federados. [Zhu et al. 2021] apresenta uma introdução ao NAS federado, com os diversos estudos feitos na área recentemente.

A grande vantagem apresentada por essa abordagem é a eliminação do esforço humano da busca da arquitetura, o que pode ajudar a superar diversas desvantagens, como a computação desnecessária, acelerando o processo de treino e inferência das redes neurais. Isso é especialmente interessante no contexto de IoT federado, onde os aparelhos geralmente possuem pouca capacidade de processamento. Dessa forma, é possível utilizar métodos de NAS para encontrar arquiteturas acuradas e pequenas o suficiente

para serem implementadas nesses dispositivos da borda das redes. Além disso, NAS pode ajudar na busca de arquiteturas mais adequadas para dados com distribuição não-IID [Kairouz et al. 2021].

Adicionalmente, os estudos atuais de NAS são basicamente focados em aprendizado federado horizontal. Como visto, o aprendizado federado vertical é totalmente diferente, o que impede o uso das mesmas técnicas de NAS em ambos os cenários, tornando o aprendizado federado vertical praticamente inexplorado [Zhu et al. 2021].

Há também outros desafios de aprendizado federado que podem ser transportados para o uso de NAS federado, como a necessidade de técnicas *lightweight* de criptografia e tornar o modelo robusto a ataques através de sistemas adversários [Zhu et al. 2021].

5.7.3. Preservação de Privacidade

Apesar do aprendizado federado promover por definição uma proposta com maior enfoque na privacidade dos dados removendo a importância e necessidade de um servidor central, esta vem apresentando uma série de novos desafios e vulnerabilidades. Por exemplo, em uma abordagem federada cliente-servidor, um adversário mal-intencionado ao controlar um servidor pode invalidar os métodos de segurança de um sistema ao comprometer muito mais participantes em uma rodada do que o esperado. Logo, um problema natural de privacidade trata-se de limitar a capacidade do servidor em reconstruir os dados de um cliente com base em suas entradas. Isso envolve definir muito bem as respostas para as seguintes questões: (i) Quais informações dos clientes o servidor tem acesso como resultado de uma rodada no ambiente federado? (ii) Qual o nível de vazamento de privacidade obtido quando o servidor tem acesso a visualização desses dados?

Embora já existam diversas soluções (principalmente envolvendo criptografia), um dos problemas promissores no modelo federado ainda consiste em desenvolver técnicas que garantam a privacidade dos clientes durante o processo de agregação dos resultados individuais pelo servidor [Kairouz et al. 2021].

Devido ao surgimento recente da proposta federada, uma série de questões se encontram em aberto e podem oferecer futuras vias de pesquisa [Mothukuri et al. 2021], como:

Ataques adversários de dia zero - A grande maioria dos métodos federados foram projetados visando garantir a proteção contra ataques pré-definidos. Como ainda não existem muitos modelos federados em produção (quando comparamos com modelos centralizados), tal abordagem torna o modelo frágil a ataques ainda não documentados. Propostas com aprendizado profundo vêm mostrando resultados promissores quanto a essa vulnerabilidade [Saharkhizan et al. 2020, Karimipour et al. 2019].

Processos bem definidos - Ainda não existe uma padronização em relação aos processos usados em FL. Logo, precisam ser feitas pesquisas adicionais com o objetivo de realizar uma padronização das técnicas e abordagens de privacidade existentes.

Trade-off entre privacidade e eficiência - Ao garantir uma maior proteção dos dados, modelos federados atuais apresentam uma perda da sua precisão. Logo, necessitamos de estudos voltados à compreensão do nível adequado de criptografia aplicado aos dados para garantir a segurança dos clientes e manter a eficiência do modelo.

Criação de frameworks de privacidade em FL - Poucas bibliotecas ou frameworks atualmente permitem a implantação de técnicas de proteção à privacidade. Além de facilitar o estudo e desenvolvimento de novas propostas, a implementação de novos pacotes podem facilitar também a adoção de métodos federados na indústria.

Seleção de clientes - Embora existam trabalhos apresentando estratégias seguras de seleção de clientes para as rodadas de treinamento do modelo [Nishio and Yonetani 2019], a proposta de abordagens padronizadas ainda é um tema de extrema importância no aprendizado federado.

5.8. Aplicações

Como já foi explicado nas seções anteriores, FL é um paradigma de modelagem inovador que permite construir modelos compartilhados e personalizados utilizando dados espalhados em vários dispositivos e organizações, sem comprometer a privacidade e a segurança dos usuários. O FL também possibilita uma forma mais eficiente de se lidar com os dados heterogêneos em relação a abordagem tradicional de ML. Essa característica se amplifica com a utilização do FL com a técnica de Transfer Learning que permite utilizar modelos já pré-treinados.

Essas características possibilitam que FL seja aplicado em diversas áreas como Saúde, Cidades Inteligentes (*Smart Cities*), Financeira [Yang et al. 2019a] e em serviços para *smartphones* [Li et al. 2020c]. Nesta seção, serão exemplificadas algumas dessas aplicações por meio da apresentação de trabalhos relacionados.

5.8.1. Cidades Inteligentes (*Smart Cities*)

Internet of Vehicles (IoV) compreende uma rede de veículos que, além de possuírem inúmeros sensores e softwares para obtenção, armazenamento e análise de dados, estão no processo de se tornarem autônomos e interconectados, trocando informações entre si a fim de otimizar a sua função de transportar passageiros com o máximo de segurança, conforto, e mínimo impacto ambiental [Gerla et al. 2014]. Naturalmente, há vários desafios a serem solucionados durante esse processo de evolução, dentre eles a necessidade de comunicação com baixa latência.

Com isso e mente, [Samarakoon et al. 2018] propõem a utilização de princípios de FL para possibilitar uma comunicação entre veículos que seja extremamente confiável e de baixa latência. Isso é possível porque, ao contrário do caso de uma abordagem centralizada, os *vehicular users (VUEs)* não precisam compartilhar informações com *roadside units (RSUs)* e outros VUEs para que os modelos de ML sejam treinados, mas o VUE constroi e envia seu próprio modelo para a RSU, que então agrega todos os modelos, realiza o ajuste dos pesos e envia o modelo atualizado de volta aos VUEs. Adicionalmente, a aplicação de FL também oferece a vantagem de não depender da sincronização entre VUE. Ou seja, mesmo no caso de perda de conexão entre VUEs e RSUs, VUEs ainda conseguem construir seus próprios modelos e continuar navegando.

5.8.2. Financeiro

Como já foi mencionado anteriormente, um dos grandes benefícios da aplicação de FL é a confidencialidade e segurança de dados privados durante o treinamento dos modelos de

ML, e um cenário em que esses fatores são de extrema importância é no setor financeiro. Não apenas pessoas não devem ter acesso aos dados privados de outras pessoas, mas os bancos também não podem trocar entre si dados referentes aos seus usuários.

[Yang et al. 2019c] apontam que há muito mais registros de transações legítimas do que fraudulentas nos *databases* dos bancos, o que torna difícil treinar modelos de ML para deteção de fraudes de cartão de crédito quando um único banco não tem dados suficientes, e ele não pode utilizar dados de outros bancos. Por isso, [Yang et al. 2019c] apresentam a criação do *framework* FFD (*Federated learning Fraud Detection*), com o objetivo de possibilitar que os bancos possam todos se beneficiarem de um treino coletivo do modelo de detecção de fraudes de cartão de crédito, sem o comprometimento da segurança dos dados.

5.8.3. Saúde

Com os avanços da tecnologia, mais hospitais estão adotando os sistemas de dados médicos eletrônicos, o que aumenta o armazenamento de dados históricos dos pacientes e de laudos médicos. Em paralelo a isso, os *smartwatches* conseguem coletar informações pessoais de saúde como as medições dos batimentos cardíacos e do nível de oxigenação do sangue. Utilizando esses dados, é possível alimentar algoritmos de aprendizado de máquina que podem ajudar no diagnóstico precoce de algumas doenças. No entanto, esses dados estão segmentados em diversos dispositivos e hospitais além de serem de natureza sensível por estarem acompanhados de uma forte política de proteção à privacidade do usuário [Xu et al. 2021].

O FL seria uma alternativa para lidar com esses dados segmentados e sensíveis uma vez que é possível treinar um modelo global compartilhado mantendo os dados localmente [Xu et al. 2021] de modo a facilitar a aplicação de modelos de aprendizado de máquina. No trabalho [Lee et al. 2018], por exemplo, o FL é utilizado para encontrar similaridade entre pacientes de hospitais diferentes mantendo as informações pessoais dos pacientes restritas à sua instituição de origem. Outro exemplo de trabalho é o [Brisimi et al. 2018] que constrói um sistema de FL para prever futuras hospitalizações de pacientes com doenças cardíacas.

5.8.4. *Smartphones*

Os dados fornecidos por usuários de *Smartphones* podem ser utilizados para treinar modelos que visam melhorar alguns serviços importantes e cotidianos ofertados em aplicativos celulares como predição de próximas palavras, detecção de rosto e detecção de voz. No entanto, existe um empecilho na utilização desses dados, pois muitos usuários não querem compartilhar os seus dados por questões de privacidade, e para evitar gastos de bateria e de largura de banda. Tendo em vista esse problema, o FL poderia ser uma ótima solução para que esses dados possam ser utilizados sem comprometer a privacidade do cliente e evitando problemas como o gasto de bateria e de banda larga [Li et al. 2020c].

Um dos exemplos mais conhecidos da utilização de FL em aplicações celulares é o trabalho [Yang et al. 2018] que destaca o FL como solução para melhorar os serviços ofertados pelo *Google Keyboard*(GBoard) que é um teclado para dispositivos móveis que possui algumas automatizações como auto-correção de textos, predição das próximas

palavras, completar palavras e expressões que podem representar um *emoji*. Nesse trabalho é destacado que o FL permite que os modelos de *machine learning* sejam treinados sem que seja preciso coletar dados sensíveis de usuários ao mesmo tempo que diminui a latência de resposta.

5.9. Estudo de caso - Hands on

Para colocar em prática os conhecimentos adquiridos ao longo dos capítulos anteriores, finalizaremos com uma aplicação em reconhecimento de atividades humanas (Human Activity Recognition). Usaremos dados obtidos por meio de sensores presentes em dispositivos móveis que encontram disponíveis no Github[10]. Embora o objetivo inicial do projeto que originou o conjunto de dados tenha sido apenas a avaliação de atividades de classificação e reconhecimento de atividades com o uso de redes neurais, queremos aqui utilizar esse background para mostrar a implementação de importantes conceitos de FL em um sistema.

5.9.1. Dataset e Modelo de Aprendizado

O conjunto de dados apresenta cerca de 20.000 leituras de sensores, obtidos após a coleta realizada com 6 participantes realizando 5 ações diferentes, onde cada leitura consiste em: (i) Medições de pose (roll, pitch, yaw), (ii) Dados provenientes do acelerômetro (medições de aceleração linear), (iii) Dados provenientes do giroscópio (medições da velocidade de rotação).

Cada feature será então representada por um vetor 3D apontando na direção da leitura em um determinado passo de tempo para os quatro diferentes sensores (cinto, braço, antebraço, haltere). Assim, normalizamos cada leitura em relação às amostras na mesma categoria no conjunto de dados e concatenadas em um único intervalo de tempo, formando um vetor de features de dimensão 40×1.

Ao longo do tutorial optamos pelo uso do framework de aprendizado federado Flower em conjunto com a biblioteca de deep learning Pytorch. Todo o código fonte e dados utilizados se encontram disponíveis em nosso repositório presente no Github[11]. Para começar a nossa análise, o primeiro passo consiste em importar todas as bibliotecas necessárias para a execução:

```
1 import torch
2 import torch.nn as nn
3 import flwr as fl
4 from torchvision import datasets, transforms
5 from torch.utils.data import DataLoader, Dataset
6 from torchvision import datasets, transforms
```

Logo após, precisaremos de uma função auxiliar para carregar os dados e definir os conjuntos de treinamento e teste. Os dados serão particionados horizontalmente, assim os subconjuntos de treinamento e teste irão ser divididos em mini-batches (pequenos lotes) com base no número total de clientes. Nessa função, precisaremos dos seguintes parâmetros: (i) **data_root (str)**: Diretório onde os datasets finais serão armazenados. (ii) **train_batch_size (int)**: Tamanho do mini-batch usado nos dados de treinamento.

[10] https://github.com/jchiang2/Human-Activity-Recognition
[11] https://github.com/EduardaChagas/Aprendizado-Federado-aplicado-IOT

(iii) **test_batch_size (int)**: Tamanho do mini-batch usado nos dados de teste. (iv) **id (int)**: Client ID usado para selecionar uma partição específica. (v) **nb_clients (int)**: Número total de clientes usados no treinamento.

No fim, ela retornará uma tupla contendo DataLoaders para os conjuntos de treinamento e teste. Usamos como padrão a taxa de divisão de 0.2 para dados de treinamento e validação (80% de treinamento, 20% de validação).

```
def load_data(
    data_root: str,
    train_batch_size: int,
    test_batch_size: int,
    cid: int,
    nb_clients: int,
    ) -> Tuple[DataLoader, DataLoader]:
    train_dataset = Dataset([e for e in range(19622)], data_root)
    test_dataset = Dataset([e for e in range(19622)], data_root)
    # Particionando os dados de treinamento com base no número de clientes
    train_loader = dataset_partitioner(
        dataset = train_dataset,
        batch_size = train_batch_size,
        client_id = cid,
        number_of_clients = nb_clients
    )
    # Particionando os dados de teste com base no número de clientes
    test_loader = dataset_partitioner(
        dataset = test_dataset,
        batch_size = test_batch_size,
        client_id = cid,
        number_of_clients = nb_clients,
    )
    return (train_loader, test_loader)
```

Atualmente o modelo de classificação mais adequado e vantajoso para a modelagem de um ambiente federado são as redes neurais. Desse modo, em nosso exemplo o modelo de aprendizado utilizado será uma CNN 1D com três camadas convolucionais e duas camadas totalmente conectadas, onde cada camada é seguida por uma função de ativação ReLU e uma camada de dropout. A taxa de aprendizado aplicada aqui será de 0.003 e uma taxa de decaimento de 0.95 por época. Definimos essa configuração de arquitetura por meio da criação de uma classe em Pytorch denominada **HARmodel**:

```
class HARmodel(nn.Module):
    """Modelo para reconhecimento de atividades humanas."""
    def __init__(self, input_size, num_classes):
        super().__init__()
        # Camada 1D convolucional
        self.features = nn.Sequential(
            nn.Conv1d(input_size, 64, 1),
            nn.ReLU(),
            nn.Dropout(),
            nn.Conv1d(64, 64, 1),
            nn.ReLU(),
            nn.Dropout(),
            nn.Conv1d(64, 64, 1),
            nn.ReLU(),
            nn.Flatten(),
            )
        # Camadas totalmente conectadas
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(128, num_classes),
            )
    def forward(self, x):
        x = self.features(x)
        out = self.classifier(x)
        return out
```

Até o momento mostramos como criar uma aplicação padrão de Deep Learning, nas próximas subseções iremos abordar como realizar a construção de um ambiente federado ao implementar o algoritmo de média federada. Para isso, precisaremos nos preocupar com dois scripts que funcionaram paralelamente: um responsável pela interação do

cliente com o sistema federado e outro que definirá as regras do servidor central.

5.9.2. Cliente Flower

O próximo passo é definir a alocação dos dispositivos no ambiente federado. Quando o servidor seleciona um dispositivo específico do ambiente federado para realizar um treinamento, ele envia as instruções pela rede, por meio de uma interface chamada Client. Assim, o cliente recebe as instruções do servidor e chama um dos métodos desta classe para executar seu código (ou seja, para treinar a sua rede neural local). O framework Flower fornece uma classe chamada NumPyClient, que torna mais fácil implementar a interface do cliente quando utilizamos PyTorch. Quando implementamos um NumPyClient devemos definir os seguintes métodos: (i) **get_parameters**: retorna o peso do modelo como uma lista de ndarrays (ii) **set_parameters (opcional)**: atualiza os pesos do modelo local com os parâmetros recebidos do servidor (iii) **fit**: define os pesos do modelo local, treina o modelo localmente e recebe o update dos pesos locais (iv) **evaluate**: define como o modelo local será testado. Abaixo mostramos como a classe Client foi implementada para o exemplo apresentado:

```
class HARClient(fl.client.Client):
    def __init__(
        self,
        cid: int,
        train_loader: datasets,
        test_loader: datasets,
        epochs: int,
        device: torch.device = torch.device("cpu"),
    ) -> None:
        self.model = HARmodel(40, 5).to(device)
        self.cid = cid
        self.train_loader = train_loader
        self.test_loader = test_loader
        self.device = device
        self.epochs = epochs

    #Obtendo os pesos do modelo como uma lista de ndarrays NumPy
    def get_weights(self) -> fl.common.Weights:
        return [val.cpu().numpy() for _, val in self.model.state_dict().items()]

    #Configurando os pesos do modelo como uma lista de ndarrays NumPy
    def set_weights(self, weights: fl.common.Weights) -> None:
        state_dict = OrderedDict(
            {
                k: torch.Tensor(v)
                for k, v in zip(self.model.state_dict().keys(), weights)
            }
        )
        self.model.load_state_dict(state_dict, strict=True)

    #Encapsulando os pesos em parâmetros flowers
    def get_parameters(self) -> fl.common.ParametersRes:
        weights: fl.common.Weights = self.get_weights()
        parameters = fl.common.weights_to_parameters(weights)
        return fl.common.ParametersRes(parameters=parameters)

    #Treinando o modelo com o dataset local
    def fit(self, ins: fl.common.FitIns) -> fl.common.FitRes:

        # Definindo a semente para que possamos gerar os mesmos índices de batches para todos os clientes
        np.random.seed(123)
        weights: fl.common.Weights = fl.common.parameters_to_weights(ins.parameters)
        fit_begin = timeit.default_timer()

        # Configurando os pesos do modelo
        self.set_weights(weights)

        # Treinando o modelo
        num_examples_train: int = train(
            self.model, self.train_loader, epochs=self.epochs, device=self.device, cid=self.cid
        )

        # Retornando os pesos e o número de exemplos usados para treinamento
        weights_prime: fl.common.Weights = self.get_weights()
        params_prime = fl.common.weights_to_parameters(weights_prime)
        fit_duration = timeit.default_timer() - fit_begin
        return fl.common.FitRes(
            parameters = params_prime,
            num_examples = num_examples_train,
```

```
            num_examples_ceil = num_examples_train,
            fit_duration = fit_duration,
        )

    #Função de avaliação do modelo
    def evaluate(self, ins: fl.common.EvaluateIns) -> fl.common.EvaluateRes:

        weights = fl.common.parameters_to_weights(ins.parameters)

        # Usando os pesos fornecidos para atualizar o modelo local
        self.set_weights(weights)
        (
            num_examples_test,
            test_loss,
            accuracy,
        ) = test(self.model, self.test_loader, device=self.device)

        # Retornando o número de exemplos de avaliação e o resultado da avaliação (loss)
        return fl.common.EvaluateRes(
            num_examples=num_examples_test,
            loss = float(test_loss),
            accuracy = float(accuracy),
        )
```

Uma vez definidos os métodos e as classes, precisaremos apenas instanciar o cliente e inicializá-lo. O flower nos fornece a possibilidade de rodar o servidor e o cliente na mesma máquina, configurando o endereço do servidor como "[::]: 8080". Porém, se quisermos implementar uma aplicação realmente federada com o servidor e clientes em execução em diferentes máquinas, precisaremos apenas alterar o server_address para o respectivo endereço da máquina do cliente.

```
#id do atual cliente
cid = 0

#número de épocas de treinamento
epochs = 14

#Definindo a alocação de dispositivos no pytorch
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

#Instanciando o cliente
client = HARClient(
        cid = cid,
        train_loader = train_loader,
        test_loader = test_loader,
        epochs = epochs,
        device = device,
)

# Endereço do servidor que iremos nos conectar
# "[::]: 8080" - define que podemos rodar o servidor e o cliente na mesma máquina usamos
server_address = "[::]:8080"

# Inicializando o cliente
fl.client.start_client(server_address, client)
```

5.9.3. Servidor Flower

Flower permite que o usuário personalize o processo de aprendizagem de acordo com a sua aplicação através da abstração chamada **Strategy**. Existem três maneiras de personalizar a maneira como Flower orquestra o processo de aprendizagem no servidor: (i) Usando uma estratégia já existente, como por exemplo o FedAvg; (ii) Personalizando uma estratégia já existente; (iii) Implementando uma nova estratégia do zero. O Flower contém uma série de estratégias de aprendizado federado já integradas, logo optamos pelo uso de uma delas, o algoritmo de média federada, como orquestrador de atualização do modelo. Uma vez escolhido como será realizado o processo de orquestramento de aprendizado, também podemos definir os seguintes parâmetros: (i) Percentual de clientes que deverão estar disponíveis para o treinamento e avaliação do modelo, assim como também o número mínimo de clientes necessários. (ii) Podemos definir uma função própria de avaliação do modelo (iii) Realizar a configuração dos parâmetros iniciais do modelo global (iv) Definir o número total mínimo de clientes que deverão estar no sistema (v) Definir

manualmente como deverá ocorrer o processo de treinamento e avaliação

Aqui apenas definiremos manualmente a função de avaliação do nosso modelo. Uma vez que estamos seguindo uma proposta cliente-servidor, é de extrema importância saber como se comporta a acurácia do servidor central e não apenas as métricas individuais dos clientes. Como o framework flower não possui por padrão a implementação de métricas importantes para entender a convergência do servidor, implementamos manualmente.

```
# Implementação da rotina de teste do servidor
def test(
    model: torch.nn.Module,
    test_loader: torch.utils.data.DataLoader,
    device: torch.device = torch.device("cpu"),
    ) -> Tuple[int, float, float]:

    model.eval()
    test_loss: float = 0
    correct: int = 0
    num_test_samples: int = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            num_test_samples += len(data)
            output = model(data.unsqueeze(1).permute(0, 2, 1))

                # Realizando a soma dos loss
                # Definindo a entropia cruzada como função de loss
            test_loss += torch.nn.CrossEntropyLoss()(
                output, target).item()

            # Pegando o índex com máxima log-probabilidade
            pred = output.argmax(
                dim = 1, keepdim = True
            )
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= num_test_samples
    return (test_loss, correct / num_test_samples, num_test_samples)

def eval(w):
    train_loader, test_loader = load_data(
        data_root = DATA_ROOT,
        train_batch_size = 64,
        test_batch_size = 4,
        cid = 5,
        nb_clients = 6,
    )

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    server = HARClient(
        cid = 999,
        train_loader = train_loader,
        test_loader = test_loader,
        epochs = 1,
        device = device
    )

    server.set_weights(w)
    return test(server.model, train_loader, device)
```

Definida a função de avaliação, agora precisaremos apenas configurá-la na estratégia de aprendizado:

```
strategy = fl.server.strategy.FedAvg(
            eval_fn = eval)

fl.server.start_server(config={"num_rounds": 10}, strategy = strategy)
```

Com os resultados obtidos pelo modelo federado proposto observamos que em pouquíssimas rodadas conseguimos alcançar uma acurácia similar ao modelo centralizado (neste experimento, apenas em 10 épocas). No entanto, com o uso do modelo federado conseguimos construir de um sistema mais seguro e que forneça uma garantia maior quanto a privacidade do usuário.

5.10. Considerações Finais

Este minicurso introduziu o conceito de Aprendizado Federado no contexto de Internet das Coisas. Inicialmente apresentamos uma breve introdução as conceitos de aprendizado de máquina e aprendizado de máquina distribuído, no intuito de diferenciar o Aprendizado Federado deste último. Em seguida, apresentamos os conceitos fundamentais de Aprendizado Federado e introduzimos os modelos mais comumente encontrados na literatura. Introduzimos também, as técnicas e conceitos relativos à preservação de privacidade no contexto de aprendizado federado. No contexto de IoT, entendemos que o modelo geral empregado por Aprendizado Federado, não necessariamente atende às restrições de hardware impostas por dispositivos IoT. Dessa maneira, discutimos como podemos comprimir modelos para que eles sejam operacionais em dispositivos com restrições de recursos. Apresentamos também aspectos relacionados à aprendizado por reforço e computação de borda no contexto de aprendizado federado. Finalmente, apresentamos os principais desafios de pesquisa, aplicações habilitadoras da tecnologia e introduzimos um estudo de caso de uma aplicação prática de Aprendizado Federado.

Entendemos que o Aprendizado Federado é um campo amplo e multidisciplinar, que envolve fundamentos dos algoritmos de aprendizado máquina, aprendizado distribuído, criptografia, segurança, miniração de dados com preservação de privacidade, além de disciplinas que não foram discutidas nesse texto, mas que são bastante relevantes para a habilitação da tecnologia, como questões regulatórias e legais e mecanismos de incentivo para criação bem sucedida da federação. Aprendizado Federado é um área nova de pesquisa que apresenta um campo fértil para o desenvolvimento de estudos e para a criação de novas aplicações que irão beneficiar a sociedade.

Referências

[Al-Rubaie and Chang 2016] Al-Rubaie, M. and Chang, J. M. (2016). Reconstruction attacks against mobile-based continuous authentication systems in the cloud. *IEEE Transactions on Information Forensics and Security*, 11(12):2648–2663.

[Aono et al. 2016] Aono, Y., Hayashi, T., Trieu Phong, L., and Wang, L. (2016). Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 142–144.

[Blalock et al. 2020] Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*.

[Brisimi et al. 2018] Brisimi, T. S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I. C., and Shi, W. (2018). Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67.

[Cheng et al. 2017] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.

[Duan et al. 2019] Duan, M., Liu, D., Chen, X., Tan, Y., Ren, J., Qiao, L., and Liang, L. (2019). Astraea: Self-balancing federated learning for improving classification ac-

curacy of mobile deep learning applications. *Proceedings - 2019 IEEE International Conference on Computer Design, ICCD 2019*, (Iccd):246–254.

[Elsken et al. 2019] Elsken, T., Metzen, J. H., Hutter, F., et al. (2019). Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21.

[Foerster et al. 2016] Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[Frankle and Carbin 2019] Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

[Fredrikson et al. 2015] Fredrikson, M., Jha, S., and Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333.

[Gerla et al. 2014] Gerla, M., Lee, E.-K., Pau, G., and Lee, U. (2014). Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE world forum on internet of things (WF-IoT)*, pages 241–246. IEEE.

[Grounds and Kudenko 2005] Grounds, M. and Kudenko, D. (2005). Combining reinforcement learning with symbolic planning. In *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pages 75–86. Springer.

[Guo 2018] Guo, Y. (2018). A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*.

[Han et al. 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hollemans 2017] Hollemans, M. (2017). Google's mobilenets on the iphone. [Online; posted 14-June-2017].

[Howard et al. 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[Hsieh et al. 2019] Hsieh, K., Phanishayee, A., Mutlu, O., and Gibbons, P. B. (2019). The non-iid data quagmire of decentralized machine learning. *Arxiv preprint arXiv:1910.00189*.

[Iandola et al. 2016] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360*.

[Jeong et al. 2018] Jeong, E., Oh, S., Kim, H., Park, J., Bennis, M., and Kim, S.-L. (2018). Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *Arxiv preprint arXiv:1811.11479*.

[Ji et al. 2019] Ji, S., Pan, S., Long, G., Li, X., Jiang, J., and Huang, Z. (2019). Learning private neural language modeling with attentive aggregation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[Jiang et al. 2019] Jiang, Y., Wang, S., Valls, V., Ko, B. J., Lee, W.-H., Leung, K. K., and Tassiulas, L. (2019). Model pruning enables efficient federated learning on edge devices. *arXiv preprint arXiv:1909.12326*.

[Kairouz et al. 2021] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Qi, H., Ramage, D., Raskar, R., Raykova, M., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210.

[Karimipour et al. 2019] Karimipour, H., Dehghantanha, A., Parizi, R. M., Choo, K.-K. R., and Leung, H. (2019). A deep and scalable unsupervised machine learning system for cyber-attack detection in large-scale smart grids. *IEEE Access*, 7:80778–80788.

[Kim et al. 2015] Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. (2015). Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*.

[Konečnỳ et al. 2016] Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

[Kretchmar and Jacobvitz 2002] Kretchmar, M. D. and Jacobvitz, D. B. (2002). Observing mother-child relationships across generations: Boundary patterns, attachment, and the transmission of caregiving. *Family process*, 41(3):351–374.

[Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

[Lee et al. 2018] Lee, J., Sun, J., Wang, F., Wang, S., Jun, C.-H., and Jiang, X. (2018). Privacy-preserving patient similarity learning in a federated environment: development and analysis. *JMIR medical informatics*, 6(2):e20.

[Li et al. 2020a] Li, A., Sun, J., Wang, B., Duan, L., Li, S., Chen, Y., and Li, H. (2020a). Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371*.

[Li et al. 2020b] Li, L., Fan, Y., Tse, M., and Lin, K. Y. (2020b). A review of applications in federated learning. *Computers and Industrial Engineering*, 149(September).

[Li et al. 2020c] Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020c). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60.

[Lim et al. 2020] Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y. C., Yang, Q., Niyato, D., and Miao, C. (2020). Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(3):2031–2063.

[Mao et al. 2018] Mao, H., Zhang, Z., Xiao, Z., and Gong, Z. (2018). Modelling the dynamic joint policy of teammates with attention multi-agent ddpg. *Arxiv preprint arXiv:1811.07029*.

[McMahan et al. 2017] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, Fort Lauderdale, FL, USA. PMLR.

[McMahan et al. 2016a] McMahan, H. B., Moore, E., Ramage, D., and Arcas, B. A. Y. (2016a). Federated learning of deep networks using model averaging. *ArXiv preprint arXiv:1602.05629*, abs/1602.05629.

[McMahan et al. 2016b] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2016b). Communication-efficient learning of deep networks from decentralized data. *Arxiv preprint arXiv:1602.05629*.

[Melis et al. 2019] Melis, L., Song, C., Cristofaro, E. D., and Shmatikov, V. (2019). Exploiting unintended feature leakage in collaborative learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 691–706.

[Mothukuri et al. 2021] Mothukuri, V., Parizi, R. M., Pouriyeh, S., Huang, Y., Dehghantanha, A., and Srivastava, G. (2021). A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640.

[Nadiger et al. 2019] Nadiger, C., Kumar, A., and Abdelhak, S. (2019). Federated reinforcement learning for fast personalization. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 123–127. IEEE.

[Nishio and Yonetani 2019] Nishio, T. and Yonetani, R. (2019). Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[Niu et al. 2021] Niu, Z., Zhong, G., and Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62.

[Preuveneers et al. 2018] Preuveneers, D., Rimmer, V., Tsingenopoulos, I., Spooren, J., Joosen, W., and Ilie-Zudor, E. (2018). Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12):2663.

[Qiao et al. 2021] Qiao, Z., Yu, X., Zhang, J., and Letaief, K. B. (2021). Communication-efficient federated learning with dual-side low-rank compression. *arXiv preprint arXiv:2104.12416*.

[Rastegari et al. 2016] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). XNOR-Net: ImageNet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer.

[Rivest et al. 1978] Rivest, R. L., Adleman, L., Dertouzos, M. L., et al. (1978). On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180.

[Rummery and Niranjan 1994] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

[Saha and Ahmad 2021] Saha, S. and Ahmad, T. (2021). Federated transfer learning: concept and applications. *Arxiv preprint arXiv:2010.15561*.

[Saharkhizan et al. 2020] Saharkhizan, M., Azmoodeh, A., Dehghantanha, A., Choo, K.-K. R., and Parizi, R. M. (2020). An ensemble of deep recurrent neural networks for detecting iot cyber attacks using network traffic. *IEEE Internet of Things Journal*, 7(9):8852–8859.

[Samarakoon et al. 2018] Samarakoon, S., Bennis, M., Saad, W., and Debbah, M. (2018). Federated learning for ultra-reliable low-latency v2v communications. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE.

[Sandler et al. 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

[Seo et al. 2020] Seo, H., Park, J., Oh, S., Bennis, M., and Kim, S.-L. (2020). Federated knowledge distillation. *arXiv preprint arXiv:2011.02367*.

[Sheller et al. 2020] Sheller, M. J., Edwards, B., Reina, G. A., Martin, J., Pati, S., Kotrotsou, A., Milchenko, M., Xu, W., Marcus, D., Colen, R. R., and Bakas, S. (2020). Federated learning in medicine: Facilitating multi-institutional collaborations without sharing patient data. *Scientific Reports*, 10(12598):1–12.

[Shokri et al. 2017] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. (2017). Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE.

[Sutton et al. 1998] Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

[Tamuly 2018] Tamuly, K. (2018). Compression and acceleration of high-dimensional neural networks. [Online; posted 15-September-2018].

[Watkins and Dayan 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[Xie et al. 2019] Xie, P., Wu, B., and Sun, G. (2019). Bayhenn: combining bayesian deep learning and homomorphic encryption for secure dnn inference. *arXiv preprint arXiv:1906.00639*.

[Xu et al. 2021] Xu, J., Glicksberg, B. S., Su, C., Walker, P., Bian, J., and Wang, F. (2021). Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19.

[Yang et al. 2019a] Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019a). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19.

[Yang et al. 2019b] Yang, Q., Liu, Y., Cheng, Y., Kang, Y., Chen, T., and Yu, H. (2019b). *Federated learning*, volume 13. Morgan & Claypool Publishers.

[Yang et al. 2018] Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F. (2018). Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*.

[Yang et al. 2019c] Yang, W., Zhang, Y., Ye, K., Li, L., and Xu, C.-Z. (2019c). Ffd: a federated learning based method for credit card fraud detection. In *International Conference on Big Data*, pages 18–32. Springer.

[Yao 1982] Yao, A. C. (1982). Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE.

[Zhao et al. 2018] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.

[Zhu et al. 2021] Zhu, H., Zhang, H., and Jin, Y. (2021). From federated learning to federated neural architecture search: a survey. *Complex & Intelligent Systems*, 7(2):639–657.

[Zhuo et al. 2020] Zhuo, H. H., Feng, W., Lin, Y., Xu, Q., and Yang, Q. (2020). Federated deep reinforcement learning. *Arxiv preprint arXiv:1901.08277*.