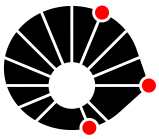
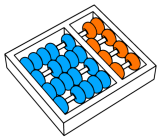




Entropia como otimizador no aprendizado de máquina federado: Um estudo de caso

R. Lellis L.F. Bittencourt J.C.S Anjos

Relatório Técnico - IC-PFG-22-04
Projeto Final de Graduação
2022 - Junho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Entropia como otimizador no aprendizado de máquina federado

Renata Mancilha Lellis[1] Luiz Fernando Bittencourt[†]

Julio Cesar Santos dos Anjos[‡]

Abstract

Machine learning has an increasing importance within society. However, for it to continue to evolve, more data from IoT devices and computing resources are needed, which has become a bottleneck for the scalability of this technology.

In order to seek alternatives to solve the scarcity of resources, the option of migrating the original machine learning configuration in which there is a central server that receives and stores all data from edge devices and builds a model has been studied. A federated architecture, known as distributed machine learning, performs the distribution of decision making, in a way that does not require the sharing of user data, instead, it shares its learning model and receives trained local models and aggregates them.

However, this proposal still finds some relevant points that should be considered limiting. Thus, this project aims to show a case study of a form of optimization in the selection of users who will participate in the model training, in order to avoid unnecessary processing expenses and to verify if this proposed way is capable of bringing differences that are significant for applications of federated learning on IoT devices.

[1] Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP, Inc

[†]Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP, Inc

[‡]Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP, Inc

Resumo

O aprendizado de máquina tem sua importância cada vez mais clara dentro da sociedade, contribuindo para diversas aplicações e impactando diretamente no dia a dia da população mundial. Contudo, para que ele continue evoluindo, são necessários cada vez mais dados dos dispositivos IoT e recursos computacionais, o que, aos poucos, tem se tornado um gargalo para a escalabilidade dessa tecnologia. Esses recursos são necessários pois para aplicar essa tecnologia em cada vez mais contextos, são necessários mais dados e assim, mais tempo de execução e consumo de energia, características que são encontradas de forma limitada e escassa em dispositivos de borda.

Com o intuito de buscar alternativas para a solução da escassez de recursos, tem se estudado a opção de migrar a configuração original do machine learning em que existe um servidor central que recebe e armazena todos os dados de dispositivos de borda e constrói um modelo, para uma arquitetura federada, conhecida como aprendizado de máquina distribuído, este realiza a distribuição das tomadas de decisão, de forma a não exigir o compartilhamento dos dados dos usuários, ao invés disso, compartilha o seu modelo de aprendizagem com os dispositivos para que estes possam realizar o treinamento de um modelo local com seus próprios dados e enviar o resultado para o servidor, que assim, ficará responsável por administrar os hiperparâmetros do modelo global e enviar para os dispositivos, receber modelos locais treinados e encontrar uma forma de agregá-los.

Contudo, essa proposta ainda encontra alguns pontos relevantes que devem ser considerados limitadores, dependendo da forma como que os parâmetros globais forem definidos, alguns resultados de computação dos dispositivos de borda não serão relevantes no cálculo de agregação dos modelos locais, resultando em processamento computacional realizados desnecessariamente, sendo que quando lida-se com dispositivos IoT, sabe-se que qualquer uso de banda larga ou energia é significativo. Dessa forma, este projeto tem como intuito realizar um estudo de caso sobre uma forma de otimização na seleção dos usuários que participarão do treinamento do modelo, a fim de evitar gastos de processamento desnecessário e verificar se essa maneira proposta é capaz de trazer diferenças que sejam realmente significativas para as aplicações de aprendizado federado em dispositivos IoT.

Lista de Figuras

Lista de Tabelas

Índice

1 Introdução

No contexto atual, é possível verificar como o aprendizado de máquina foi capaz de influenciar e modificar a realidade de grande parte da humanidade. Desde melhorar a performance em aplicações móveis, trazer uma experiência personalizada em um aplicativo, atendimentos mais eficientes a partir de uma conversa automatizada e entre outros. No entanto, quando este tema é profundo e pode-se observar, que apesar de sua influência positiva na realidade humana, os métodos tradicionais de aprendizado de máquina utilizados tem despertado algumas preocupações a respeito de privacidade de dados e altos custos com redes de comunicação e servidores para armazenamento de dados pessoais [1, 2].

A partir dessas preocupações, surgiu uma alternativa para otimizar as demandas por recursos computacionais e de comunicação e, ainda, para melhorar a governança dos dados, a fim de evitar que qualquer dado tenha sua privacidade desrespeitada. Esta alternativa ficou conhecida como aprendizado de máquina federado, ou do inglês *Federated Learning*, e suas diferenças em relação ao aprendizado de máquina centralizado podem ser resumidas em dois pontos principais: heterogeneidade de dados e alto grau de sistematização [24].

O Aprendizado Federado tem como característica a descentralização de informação, oferecendo a opção dos usuários evitarem a exposição dos dados para que possam contribuir para o aperfeiçoamento de aplicações que necessitem de alimentação de dados. Isto se dá através da oportunidade de realizar os treinamentos do processo de aprendizado de máquina nos próprios dispositivos de borda do sistema, sem compartilhar os dados brutos de cada dispositivo com os demais atores,o que traz também a oportunidade de diminuir a demanda por recursos computacionais para armazenamento de dados e comunicação entre dispositivos e servidores.

Ainda que o treinamento ocorra na borda, a conclusão do processo de aprendizado, o modelo de *machine learning* criado será consolidado e configurado em um servidor central, este será responsável por definir regras de como ocorrerá o processo de aprendizado, selecionando quais serão os dispositivos que participarão, definindo alguns hiperparâmetros globais necessários e, ainda, implementando o cálculo de média federada, ou Federated Avarage [24] que será detalhado na seção 2.2, no momento de consolidação do modelo. Este é utilizado para definir os parâmetros do modelo global de aprendizado de máquina através de um algoritmo de agregação que será responsável por calcular a média dos parâmetros locais dos dispositivos de borda que serão definidos durante o processo de aprendizado.

Contudo, esta alternativa, recém criada, ainda enfrenta dificuldades para ser aplicada na prática, uma vez que, ao se tornar responsabilidade dos dispositivos de borda o treinamento dos modelos de aprendizagem [1], estes poderão ter seu desempenho sobrecarregado para poder treinar equações complexas relacionadas aos modelos de *machine learning* e lidar com grandes volumes de dados próprios. Além disso, ainda que a

aplicação da média federada neste novo modelo tenha demonstrado certo sucesso empírico [24], também existem problemas relacionados a heterogeneidade dos dados, já que estes estão descentralizados e, assim, podem surgir de diferentes realidades e enviesar o modelo de acordo com a quantidade e qualidade destes dados. Ou seja, encontrou-se o problema de lidar com uma grande diversidade de dados, que serão utilizados para um processo de aprendizado complexo que necessita de uma alta demanda de processamento, sendo que parte dessa computação poderá ser descartada por não trazer relevância para a definição dos parâmetros globais. Além de, fazer com que os dispositivos de borda armazenem grandes quantidades e gastem largura de banda e energia sem que obtenham qualquer retorno significativo.

A proposta deste trabalho, está relacionada à hipótese de que os modelos atuais não avaliam da melhor forma a quantidade de informação associada aos dados, dessa forma, é apresentada uma alternativa que busque certificar a maneira de avaliação da qualidade da informação que os dispositivos de borda possuem, com a intenção de evitar cálculos desnecessários.

Dessa forma, este projeto tem como objetivo realizar um estudo de caso no processamento de dados como tomada de decisão para seleção de quais dados e dispositivos de borda contribuirão para a construção de um modelo de aprendizado de máquina. Isto é, será apresentado o cálculo de entropia como maneira de medir a capacidade de contribuição dos dados para com o modelo, a partir da heterogeneidade do conjunto de dados. Assim, aqueles que obtiverem certa qualidade, poderão contribuir para o treinamento, sem necessidade de demandar que muitos dispositivos participem da construção do modelo sem que efetivamente impactam no resultado final, evitando qualquer processo computacional desnecessário por parte dos dispositivos de borda que possuem limitações de processamento, armazenamento e comunicação.

De forma efetiva, este projeto apresentará os resultados da implementação de um código de aprendizado de máquina federado em um ambiente simulado comparando a assertividade de modelos de aprendizado de máquina em que foi utilizado o cálculo de entropia para a seleção na participação dos dispositivos de borda durante o treinamento do modelo versus a assertividade de modelos de aprendizado de máquina definidos sem a utilização do cálculo de entropia, ainda, este projeto discutir a respeito da performance de ambos modelos para entender se o cálculo de entropia é capaz de melhorar a performance do processo de aprendizado sem prejudicar drasticamente a assertividade do modelo.

Ainda, este projeto não se aprofundará a respeito do tópico de protocolos comunicação entre dispositivos e servidor, pois será realizado em um ambiente de simulação e nem se aprofundará em otimizações relacionadas a assertividade do modelo por si só, ou seja, não serão realizadas simulações com diferentes modelos de aprendizado de máquina existentes na literatura para entender qual consegue trazer melhores resultados de predição, uma vez que este projeto tem como foco entender a respeito da performance de processamento do modelo quando são inseridos mais ou menos dados durante o treinamento do modelo.

Além desta seção introdutória, este relatório apresentará uma seção de conceitos e trabalhos relacionados (seção 2), bem como uma seção responsável por apresentar o modelo arquitetural proposto para o estudo de caso (seção 3), outra seção em que descreve a metodologia utilizada para a implementação do código (seção 4) e, por fim,

respectivamente, nas seções 5 e 6 serão apresentados os resultados do estudo e a conclusão obtida.

2 Conceitos e Trabalhos Relacionados

Nesta seção serão introduzidos os principais conceitos e fundamentos teóricos relacionados ao projeto. Dessa forma, serão apresentados os conceitos de aprendizado de máquina, aprendizado de máquina federado, gradiente descendente, média federada, função de perda, acurácia e aprendizado profundo.

2.1 Aprendizado de Máquina

> *"Um Agente está aprendendo se ele melhora sua performance depois de fazer observações sobre o mundo. O aprendizado pode variar do trivial, como anotar uma lista de compras, ao profundo, como quando Albert Einstein inferiu uma nova teoria do universo. Quando o agente é um computador, chamamos de aprendizado de máquina: um computador observa alguns dados, constrói um modelo com base nos dados, e usa o modelo como uma hipótese sobre o mundo e um pedaço de software que pode resolver problemas" . (Russel, 2021, p. 651).*

De acordo com [25], o aprendizado de máquina é uma maneira de analisar uma grande quantidade de dados e buscar a construção de um modelo automatizado capaz de fazer inferências com base em cálculos estatísticos, sem que o computador tenha sido explicitamente programado para isto. Ou seja, a partir de um conjunto de dados, é construída uma fórmula matemática com parâmetros que buscam explicar como é a correlação entre estes dados e é capaz de tomar decisões ou fazer predições.

Para que seja possível chegar nesta função matemática responsável por fazer predições e que representará um modelo computado, são realizados alguns passos importantes de acordo com [4] e com a Figura 1:

i) processamento dos dados, desde a formatação dos dados da maneira mais adequada para ser utilizada no modelo a ser criado, até a separação dos dados entre treinamento, validação e teste. Cada grupo de dados será utilizado em um momento específico da consolidação do modelo

ii) definição do modelo a ser treinado, podendo ser desde uma função de regressão linear até uma rede neural com múltiplas camadas complexas

iii) treinamento do modelo a partir dos dados processados, sendo necessário, frequentemente, grandes conjuntos de dados e capacidade computacional

iv) fase de inferência na qual o modelo treinado, os parâmetros definidos durante a fase anterior e novos dados inseridos realizarão uma predição.

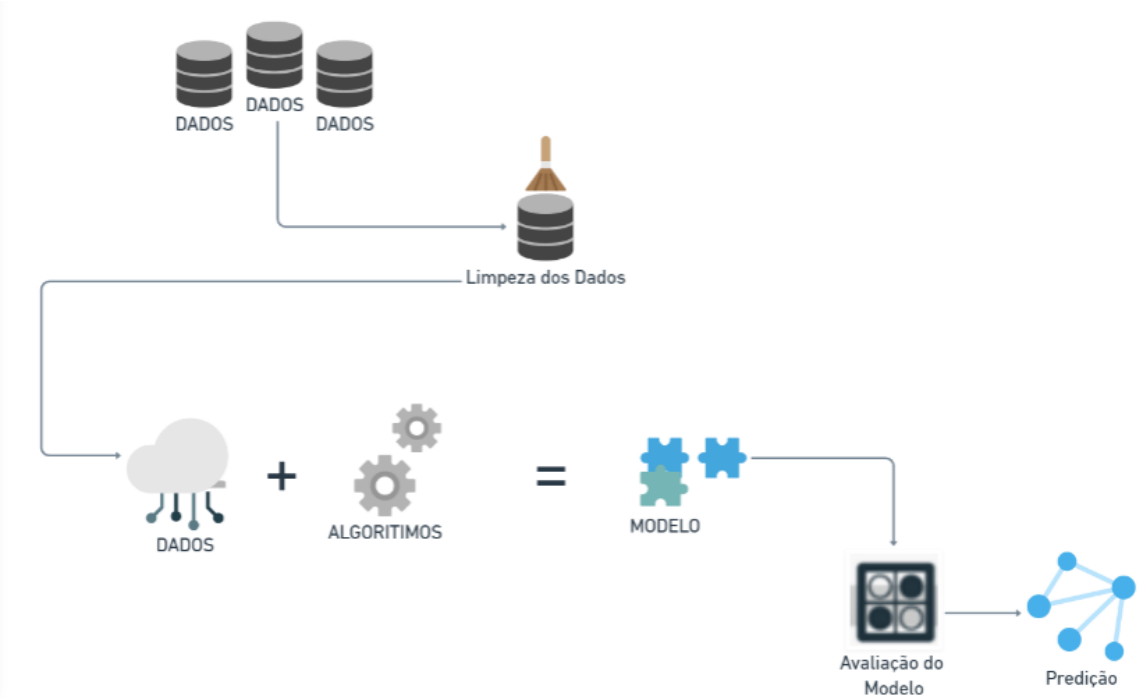


Figura 1 — Processo de aprendizagem de máquina

Fonte: Adaptado de Pandian [26].

Uma das características mais relevantes do aprendizado de máquina e a responsável por classificar o tipo de algoritmo de aprendizado de máquina, é o feedback. Existindo as seguintes classificações:

- ***Aprendizado supervisionado:*** o conjunto de dados de treinamento possuirá dados de entrada e os valores desejados de saída correspondentes. A ideia do aprendizado supervisionado é determinar uma função que mapeia o dado de entrada ao dado de saída com a menor estimativa de erro. Dessa forma, a função pode receber novas entradas de dados e fazer a predição do valor de saída [3,4].
- ***Aprendizado não supervisionado:*** neste caso, o conjunto de dados de treinamento consistirá apenas de dados de entrada, dessa forma, o modelo deverá encontrar uma função capaz de descrever a estrutura dos dados buscando agrupar os dados por similaridade [3,4].
- **Aprendizado semi-supervisionado:** esta forma de aprendizado utiliza em seu treinamento uma pequena quantidade de dados que possuem os valores de saída correspondentes para contribuir no agrupamento da maioria restante dos dados de entrada que não possuem uma saída mapeada. Assim, o modelo será anteriormente treinado pelos dados rotulados e, em seguida, utilizará a outra parte do conjunto de dados para otimizar a função previamente definida [3,4].

- ***Aprendizado por reforço:*** esta última maneira de aprendizado define uma função a partir da observação e de feedbacks a respeito das informações obtidas no estado atual do modelo, que em um primeiro momento é definido a partir de tentativas e erros, dessa forma, é essencial definir quais ações resultarão em uma recompensa positiva a respeito do modelo [3,4].

Uma informação importante que deve ser ressaltada antes de avançar para outras características do aprendizado de máquina, é de que os dados coletados para realizar os treinamentos e as avaliações podem vir de diferentes fontes, sendo diferentes modelos de dispositivos móveis, ou mesmo diferentes equipamentos capazes de coletar dados. Por exemplo, hoje temos grande número de dispositivos de IoT disponíveis no mundo que geram grandes quantidades de dados e por conta dessa grande quantidade, estes se tornam muito interessantes para a esta área e então a forma de obter acesso a eles dependerá da característica do modelo de aprendizado ser centralizado ou distribuído a ser implementado, a arquitetura dos dois formatos pode ser visualizada na Figura 2.

De acordo com a Figura 2, no primeiro formato, um modelo "M" é treinado após ocorrer a agregação, centralização e processamento dos conjuntos de dados de todos os dispositivos que coletaram os dados a serem utilizados. Outro formato é o distribuído, no qual temos um servidor "S" e um conjunto "C" de clientes, em que cada cliente possui um dataset privado e cada cliente utiliza seu próprio conjunto de dados para treinar o modelo local e então envia para o servidor "S" os parâmetros locais obtidos para a atualização do modelo global. Assim, o servidor "S" recebe todos os parâmetros de seus clientes e, por uma regra de agregação, define os novos parâmetros do modelo global.

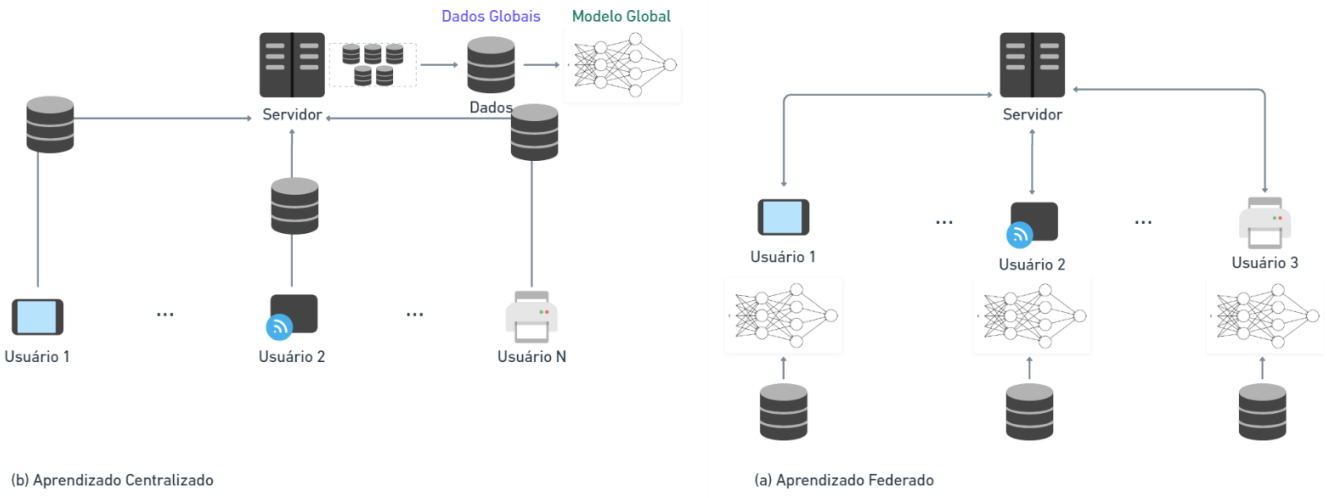


Figura 2 —Aprendizado centralizado versus federado

Fonte: Adaptado de Bo Liu [9].

Como neste projeto será apresentado um estudo de caso realizado em um modelo de aprendizado de máquina federado, no próximo subtópico iremos nos aprofundar a respeito do funcionamento deste formato.

2.1.1 Gradiente Descendente

A respeito do modelo de aprendizado de máquina, deve-se encontrar uma função que consiga tomar decisões a respeito de dados de entrada, dessa forma busca-se atentar à forma como que o modelo decide quais serão os parâmetros mais adequados para a tomada de decisão. Para a decisão desses itens existem diversos métodos que auxiliam o modelo a escolher os melhores gradientes (pesos) para a função, e um dos principais é o Gradiente Descendente (GD).

Este método consiste em encontrar, de forma iterativa, os valores dos parâmetros que minimizam determinada função de interesse, buscando-se uma convergência da função e obtendo-se a garantia de pelo menos uma convergência para um mínimo local, independente da complexidade do modelo [2].

Devido a utilização de toda a amostra de treino em cada iteração do método, são encontrados alguns pontos de atenção: i) O método pode se tornar computacionalmente muito custoso, pois para cada atualização do conjunto de treinamento, precisamos ensinar o algoritmo desde o início; ii) Em muitos casos, o conjunto completo de treinamento pode apresentar exemplos que pouco irão contribuir para a atualização dos parâmetros.

Uma das variações desse método é o Gradiente Descendente Estocástico (SGD), sua diferença está na quantidade de amostras (batches - um hiperparâmetro) que ele utilizará em cada iteração em busca da convergência. Neste caso, ele utiliza apenas uma amostra do conjunto de treinamento, esta modificação é feita buscando otimizar o custo computacional de cada iteração realizada e, consequentemente, do treinamento do modelo como um todo.

O SGD funciona como um padrão numérico de aproximação para estimar o parâmetro ótimo e ele será definido enquanto passar pelo conjunto de dados de forma iterativa (sendo que as amostras de dados serão modificadas a cada época) [7].

Algorithm 1: SGD Algorithm

$\vec{w} \leftarrow$ any point in the parameter space
while *not converged* **do**
  for *each* w_i **do**
    $\vec{w}_i \leftarrow \vec{w}_{i-1} - \alpha \frac{\partial}{\partial w_i} L(\vec{w})$

Figura 3 — Algoritmo SGD

Fonte: Retirado de J.C.S Anjos [4].

Outro método derivado do GD é o Gradiente Descendente Mini-batch, neste caso, a cada iteração é utilizada uma amostra do conjunto, sendo que o tamanho da amostra será mais um hiperparâmetro a ser estudado, uma vez que o tamanho pode variar de 2 a 100. A vantagem deste método sobre o primeiro apresentado está na velocidade de atualização dos parâmetros, já sobre o segundo está na implementação vetorizada (paralelização dos cálculos) [7].

Quando olhamos para o aprendizado de máquina federado existem algumas preocupações a respeito de como ocorrerá a atualização do modelo global, sendo que o processo de atualização dos dispositivos de borda para o servidor central pode ocorrer utilizando métodos do gradiente descendente estocástico síncrono (SSGD) ou assíncrono (ASGD).

No SSGD, os usuários locais calculam seu próprio gradiente sobre os dados locais e, na sequência, enviam o resultado para o servidor central, este será responsável por calcular o algoritmo de agregação e enviar um gradiente global atualizado para os usuários locais que irão calcular um novo gradiente local na próxima rodada. O problema deste método está na forma de lidar com usuários atrasados, pois todos os usuários devem aguardar o usuário mais lento para executar suas sincronizações [4].

Quando olhamos para o ASGD, cada cliente local envia seu gradiente calculado para ser agregado ao modelo global, porém, a atualização pode chegar com atraso para um retardatário (ele não terá participado daquela atualização do modelo), por outro lado, o cliente agrega seu gradiente antes de um retardatário [4]. Os métodos assíncronos funcionam da seguinte forma: todos os processadores têm permissão para acessar uma memória compartilhada de forma simultânea e podem atualizar o vetor do parâmetro armazenado no nó superior. Cada trabalhador acessa o parâmetro do vetor central independentemente dos outros.

O problema deste método está no fato de que um trabalhador pode ter atualizado o parâmetro no nó central enquanto um trabalhador distinto ainda está calculando seu gradiente. Dessa forma pode ser que o valor computado foi baseado em parâmetros já obsoletos, além de ocasionar ruídos e baixa performance em larga escala [6].

2.2 Mecanismos de otimização do Aprendizado de Máquina

2.2.1 Épocas

Relembrando o conceito de *batches*, sabemos que o modelo deve ser atualizado quando um número específico de amostras são processados.

Existe um outro hiperparâmetro muito importante que deve ser definido para o treinamento do modelo que é a quantidade de épocas, este representa número de

repetições que ocorrem o processo de aprendizado dentro do aprendizado de máquina, isto é, o número de vezes que os dados serão exibidos para o modelo, esperando que o resultado apresente um conjunto de pesos bem ajustados com as características que desejadas. Ou seja, quando uma época for concluída, terá passado por todas as fases de treinamento do modelo e já terá como validar o valor da tomada de decisão do modelo [10].

Ambos hiperparâmetros não são encontrados dentro do processo de aprendizado, pois eles devem ser especificados antes do treinamento ser iniciado, e não existem valores fixos, dependendo do algoritmo, pode necessitar o teste de vários valores inteiros antes de encontrar o mais adequado para o processo.

Vale ressaltar que os dispositivos distribuídos devem ter mais épocas de interação para obter precisão semelhante nos conjuntos de dados não-iid do que nos conjuntos de dados iid [4].

2.2.2 Função de Perda

Para entender a respeito da função de perda, é importante compreender a definição de 'processo de aprendizado':

> *"Uma forma de mensurar quando o algoritmo está fazendo um bom trabalho - Isso é necessário para determinar a distância entre a saída atual do algoritmo e a saída esperada. A medida é utilizada como um feedback para ajustar a forma como o algoritmo está funcionado. A etapa de ajuste é o que chamamos de aprendizado". (CHOLLET, 2017, p. 6).*

Entendendo o que é o aprendizado, tem-se que a função de perda é a função responsável por determinar a distância entre a saída esperada e a atual do algoritmo, ou seja, é uma forma de avaliar como o algoritmo está modelando os dados.

Existem duas categorias de função de perdas, a primeira é para realizar classificações, desta forma, lidará com valores discretos. Já a segunda é para lidar com regressões e, portanto, abrangerá valores contínuos.

Atualmente existem diversas formas de se calcular a função de perda, o desafio está em entender qual delas se adequa ao modelo que iremos treinar e, para isso, devemos avaliar o tipo dos dados utilizados e o tipo de tomada de decisão que o modelo deverá tomar, sendo que a saída oferecida pode ser a classificação de uma imagem na qual precisamos de um resultado preciso, ou a probabilidade de uma frase se encaixar em uma certa intenção.

2.2.2.1 Função de Perda: Entropia Cruzada

O objetivo dessa função é medir a diferença entre duas médias do número de bits de distribuição da informação, ou seja, ela calcula a diferença entre duas funções de distribuição de probabilidade.

Para um conjunto de dados $\{\phi_n, t_n\}$, onde $t_n \in \{0, 1\}$ e $\phi_n = \phi(x_n)$, com n = 1, . . ., N a função de probabilidade pode ser escrita como:

$$\text{Equação 1: } p(t|w) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n} \quad (1)$$

Em que $t = (t_1, \ldots, t_N)^T$ e $y_n = p(C_1|\phi_n)$. A partir desta definição mostrada em [14] , pode-se então obter a função de erro tomando o logaritmo negativo da probabilidade, que é representada em [14] por:

$$\text{Equação 2: } E(w) = -\ln p(t|w) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \quad (2)$$

Onde $y_n = \sigma(a_n)$ *e* $a_n = w^T \phi_n$. Com esta função é possível calcular o gradiente da função de erro que será utilizada durante o processo de definição dos parâmetros do modelo com a intenção. O gradiente da função de erro é representado em [14] por:

$$\text{Equação 3: } \nabla E(w) = \sum_{n=1}^{N} (y_n - t_n)\phi_n \quad (3)$$

Assim, quando esta função é inserida em um modelo, busca-se minimizar a perda de entropia cruzada calculada a cada época, para que ocorra esta minimização devemos constantemente ajustar os pesos, assim modelo continuará sendo treinado e a perda continuará sendo minimizada durante todo o processo de aprendizado.

A cada iteração do modelo, a perda será calculada e irá penalizar a probabilidade baseada na distância do valor atual do valor esperado.

Esta função, atualmente, é uma das mais utilizadas dentro do aprendizado de máquina devido a uma melhor generalização de modelos e um treinamento mais rápido, além de poder ser utilizada tanto em problemas de classificação binária ou de multiclasses.

2.2.3 Acurácia

Existem diversas formas de avaliar os resultados de um modelo após seu treinamento, a definição de como será feita esta avaliação acaba sendo uma etapa que requer muita atenção durante o projeto, pois enquanto que um método de avaliação pode fornecer uma análise positiva do modelo, outra metodologia pode ir no caminho contrário. Dessa forma é necessário encontrar uma forma de avaliação que seja apropriada para os tipos de dados que estão sendo utilizados.

A acurácia é uma métrica que descreve como o modelo performa ao longo das classificações possíveis para um certo dado. Esta é útil quando as classes estão bem distribuídas e então irá calcular a porcentagem dos valores de saída calculados que são

iguais aos valores esperados. Assim, se o valor esperado é igual ao valor de saída calculado, o modelo é considerado acurado.

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

Figura 4 — Cálculo de Acurácia

Fonte: Adaptado de Singh [27].

Este cálculo tornou-se uma referência para verificar a qualidade de um modelo, entretanto, devido a forma como é calculada, esta se restringe a avaliação de um tipo muito específico de modelo no qual a saída pode ser verdadeiro ou falso.

Assim, foram criadas outras formas de acurácia com o intuito de adaptar para diferentes casos de usos, por exemplo, a acurácia binária que é capaz de calcular a porcentagem de valores de saída preditos que são iguais aos valores originais para rótulos de saída binários.

Da mesma forma, foi criada a acurácia categórica esparsa que é capaz de calcular a porcentagem de saídas preditas para os rótulos (inteiros) que estão nas principais “K” previsões [11].

Um ponto relevante que vale ser comentado, é o fato de que quanto mais um modelo visualiza o mesmo dado, maior a chance de ele se enviesar para sempre mostrando a mesma saída, consequentemente aumentando a acurácia de treino dando a falsa sensação de que o modelo está se desempenhando muito bem, até que são utilizados os dados de testes e verifica-se uma baixa acurácia, por exemplo na Figura 6.

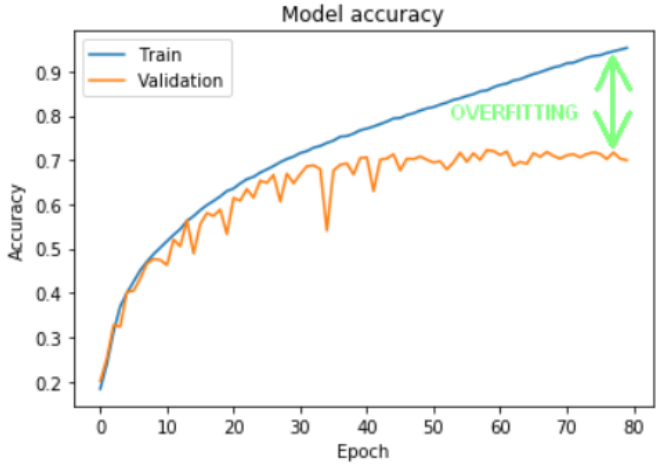


Figura 5— Diferença entre as acurácias de treino e validação/teste quando ocorre o enviesamento dos dados

Fonte: Retirado de Ferdjouni [28].

Assim, se a quantidade de dados que é utilizada ao longo do modelo é menor e é mantida a mesma quantidade de épocas que aqueles dados serão visualizados durante o treinamento, pode ocorrer do modelo se tornar tendencioso.

2.3 Aprendizado de Máquina Federado

No aprendizado de máquina Federado, proposto pelo Google em 2017, é obtido um modelo de aprendizado de máquina através do processo de computar um gradiente descendente, que de forma recursiva é compartilhado em cada rodada com os clientes, sendo que este gradiente é composto de hiperparâmetros, que definem um valor para a função de perda, sendo esta a diferença entre o valor computado e o desejado para a saída do modelo supervisionado. Em seguida, os hiperparâmetros de cada cliente serão computados a partir de um mesmo modelo recebido pelo servidor central e, de forma local, com seus dados individuais, enviarão para o servidor o resultado destes parâmetros locais que são responsáveis por ajustar seus parâmetros de um modelo conhecido como global armazenado no servidor central, isto será feito a partir de algum algoritmo de agregação. ao fim da rodada o servidor deve enviar o modelo global atualizado para os clientes participantes, que, novamente, irão computar os hiperparâmetros locais com o intuito de reduzir a função de perda e computar o gradiente. Este procedimento ocorre recursivamente até que seja possível alcançar uma acurácia desejada pelo modelo e a função de perda seja mínima

. O diferencial desta estrutura está nos seguintes aspectos: aqui podemos obter um modelo capaz de tomar decisões sem que os dados dos usuários sejam acessados pelo servidor, sendo necessário somente que cada cliente calcule uma atualização para o modelo compartilhado a partir de variáveis globais mantidas por uma máquina central.

Detalhando um pouco mais, o treinamento distribuído ocorre da seguinte forma: o modelo realizará rodadas que contemplarão dentre as 3 etapas abaixo e encontradas na Figura 3, alguns protocolos de comunicação como seleção, configuração, relatório, treinamento e agregação, até que seja alcançada uma acurácia desejada ou até a função do modelo convergir.

1. Inicialização da tarefa e distribuição de modelos: no round 0 o servidor (S) determina qual será a tarefa de treinamento, define os parâmetros da função (inicialização), além de definir outras características do modelo, tais como o tamanho do batch e da época, distribui o modelo global inicial e as configurações da tarefa para os clientes participantes.
2. Treinamento e atualização do modelo local: baseado no modelo global (Mg), os clientes atualizam os parâmetros do modelo local utilizando os dados locais, de forma a obter parâmetros ótimos que minimizem a função de perda. E então o cliente carrega no servidor "S" os parâmetros locais atualizados.
3. Agregação e atualização do modelo global: "S" agrega todos os modelos locais recebidos de forma a minimizar a função de perda global, então "S" irá distribuir o modelo global Mg+1 para os clientes para o treinamento do próximo round, isso ocorre até que se alcance a convergência da função de perda ou uma acurácia desejada.

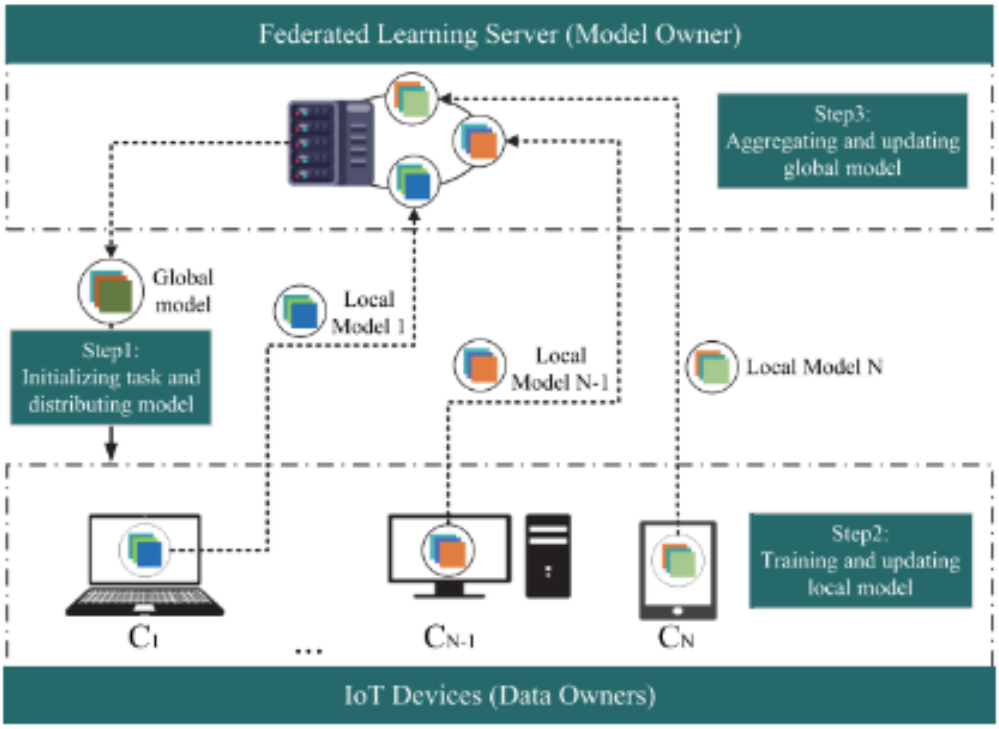


Figura 6 — Uma visão global do processo de treinamento do aprendizado federado

Fonte: Xu et al [2].

É importante ressaltar que os dados distribuídos dentre os usuários podem encontrar problemas diferentes de um modelo centralizado em que os dados são independentes e identicamente distribuídos (iid), ou seja, todas as ocorrências têm a mesma probabilidade de acontecer em qualquer dispositivo para todos os eventos. No caso do modelo federado os dados são distribuídos de forma não independente Ae idêntica (não-iid), ou seja, são dados que possuem a característica de serem desequilibrados, massivamente distribuídos e/ou apresentarem comunicação limitada com o canal do cliente.[4]

Dessa forma, deve-se sempre levar em conta que a tomada de decisão do modelo nem sempre será uma reflexão do contexto global, além de que, dependendo da forma como os clientes forem selecionados durante as rodadas de treinamento, pode-se ter resultados diferentes, ou seja, se os mesmos dispositivos forem escolhidos para executar muitas rodadas de atualização local os resultados poderão ser enviesados para o que eles contém de dados.

2.3.1 Média Federada

Quando discute-se a respeito de média federada (Federated Averaging) relacionado ao aprendizado federado estamos relacionando diretamente a um dos algoritmos criados que pode ser utilizado na etapa de agregação dos rounds do aprendizado de máquina federado, consistindo na distribuição dos parâmetros do modelo para cada modelo local.

Quando este algoritmo é escolhido para ser utilizado na etapa de agregação o que ocorre é que no round 0 todos os gradientes dos clientes são inicializados igualmente e a cada round que o modelo local é treinado com o conjunto de dados próprio e tem seus gradientes e parâmetros atualizados, os gradientes serão agregados pelo servidor a cada round a partir do cálculo da média dos gradientes locais recebidos.

2.4 Deep Learning

Após a criação do aprendizado de máquina, o qual oferece a capacidade da máquina tomar decisões sem que tenha sido explicitamente programado para aquela função, verificou-se que, apesar dessa capacidade, o computador ainda pensa e age como uma máquina. Dessa forma, a habilidade desse executar tarefas mais complexas como retirar dados de uma imagem ou vídeo ainda estava longe de ser algo semelhante ao que humanos são capazes de fazer.

Assim, o aprendizado profundo (DP) [16] propõe uma forma de aprendizado de máquina através da experiência e do entendimento do mundo em um viés hierárquico de conceitos. Devido ao formato como o computador pode absorver informações, a

interferência humana se torna desnecessária, no aprendizado e na absorção de informações, pois o computador adquire a capacidade de saber quais conhecimentos são necessários para sua evolução.

A intenção do deep learning é conseguir se aproximar de uma função f * capaz de mapear uma entrada x para uma categoria y, isto é, $y = f$ * (x). Para que esse mapeamento ocorra, é necessário definir uma função f capaz de resultar em $y = f(x; \theta)$, nesse caso, o deep learning é responsável por criar uma rede que faça este mapeamento e seja capaz de encontrar os parâmetros θ que mais consigam aproximar f de f * [16].

Um ponto importante a respeito do "DP" é o fato de que normalmente essa função f é uma composição de diferentes funções que quando unidas conseguem se aproximar do mapeamento esperado, criando uma rede. Por exemplo, podemos ter três funções conectadas em uma cadeia que formam $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$, essa estrutura é conhecida como rede neural. Sendo que $f^{(1)}$ será a primeira camada desta rede, $f^{(2)}$ a segunda camada e assim por diante. De forma geral, uma rede neural profunda possui camadas de entrada, camadas intermediárias (ocultas) e uma camada de saída (responsável por trazer o resultado no formato esperado).

O tamanho total da cadeia de funções construída será responsável por determinar a profundidade do modelo. Durante o treinamento da rede, os dados de treinamento serão exemplos aproximados de f *, com alguns dados que podem causar interferência no modelo, estes exemplos são responsáveis por especificar o que a camada de saída deve mostrar para cada ponto de x, ou seja, deve produzir um valor próximo de y [16].

Ainda, o algoritmo de aprendizado profundo deve decidir como utilizar as camadas ocultas para influenciar na determinação do valor de saída desejado, como os dados de treinamento não apresentam o valor esperado para essas camadas, elas recebem o nome de ocultas e a quantidade de camadas ocultas que determinará a largura do modelo.

Um ponto importante a respeito do "DP" é que, para a construção de um modelo capaz de tomar decisões próprias, são necessários grandes volumes de dados para alimentar e construir o sistema como um todo. Outra característica do DP, que se difere do aprendizado de máquina, é o tempo de configuração do modelo. Enquanto que no primeiro há uma demanda maior de tempo para a preparação do modelo, o segundo fornece modelos que podem ser configurados rapidamente. Ainda, é importante ressaltar que o "DP" requer sistemas computacionais com recursos e hardwares mais potentes do que o outro.

Para construir um modelo utilizando aprendizado profundo existem alguns modelos de rede neurais que podem ser utilizados. Por exemplo, no contexto de classificação de imagens, o tipo de rede mais recomendado é a rede convolucional [16], a qual contém este nome devido às camadas de convolução que ela possui juntamente com outros tipos de camadas. Outro tipo de camada comumente utilizado no reconhecimento de imagens é a camada densa [16], esta é caracterizada por implementar uma rede neural totalmente conectada.

Dessa forma, uma etapa muito importante na construção de um modelo de DP é a decisão de quantas e quais camadas de neurônios o modelo contemplará em sua arquitetura. A rede recebe o nome de neural pelo fato de serem inspiradas na neurociência e como, tipicamente, a rede é vetorizada, cada elemento do vetor pode ser interpretado como sendo um neurônio [16].

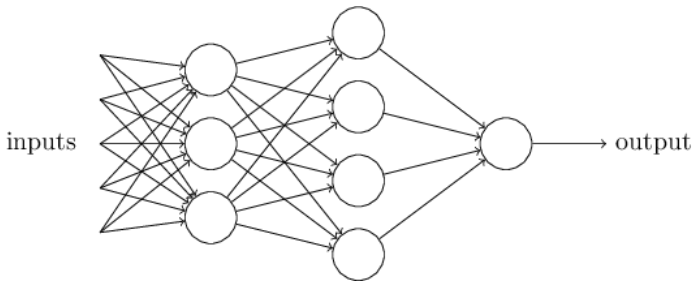


Figura 7 — Estrutura de uma rede neural

Fonte: Retirado de Gad

No DP também precisamos definir quais serão os pesos da função não linear responsável por tomar decisões, esses parâmetros serão definidos utilizando também uma das variações do gradiente descendente. Contudo, quando tratamos de redes neurais, o gradiente descendente estará acoplado em um outro algoritmo chamado retropropagação (*backpropagation*), este é responsável por calcular o gradiente (derivada parcial) da função de perda em relação aos pesos da rede neural. Em seguida, diferentes pesos serão atribuídos a cada nó ajustando em cada etapa para minimizar essa função de perda e buscando otimizar os parâmetros da rede.

O algoritmo *backpropagation* consiste em duas principais etapas [12, 20] que ocorrem de forma recursiva sob o conjunto de treinamento:

1. Fase de propagação (forward pass): esta etapa tem como função passar os dados de entrada através da rede e obter previsões de saída, ou seja, a entrada x provê a informação inicial que será propagada para as unidades ocultas e então produzir um y .
2. O passo para trás (backward pass): é responsável por calcular o gradiente da função de perda na camada de saída da rede e aplicar o resultado obtido recursivamente para aplicar a regra da cadeia para atualizar os pesos da rede.

Quando tratamos de aprendizado distribuído, o método de aprendizado ainda deve abranger mais uma etapa:

3. Agregação: esta tem a função de atualizar os parâmetros do modelo dos usuários distribuídos, que podem adotar uma atualização síncrona ou assíncrona.

O ritmo que marca como os pesos são atualizados, que pode ser alterado de modo adaptativo ou ser fixado, é conhecido como taxa de aprendizado.

2.4.1 Função de Ativação

Quando estamos construindo o nosso modelo, estamos criando uma estrutura complexa na qual buscamos encontrar uma função capaz de receber entradas e tomar uma decisão em cima destes dados de entrada, ou seja, esperamos que o modelo seja capaz de trazer um resultado que esteja o mais próximo possível do correto. Para isso, como já foi discutido anteriormente,a configuração de hiperparâmetros e a escolha de componentes corretamente que irão influenciar o modelo para próximo do valor de saída esperado, entretanto, ainda que, com diversas opções encontradas para a configuração do modelo, deseja-se realizar pequenas alterações, quando necessário, em um peso do modelo e, consequentemente, espera-se que estas modificações sejam capazes de provocar uma pequena alteração no resultado final.

Com a possibilidade de realizar pequenas alterações que impactem em apenas suaves modificações no resultado, tem-se a chance de modificar os pesos do modelo para que ele se comporte mais perto da maneira desejada. Por exemplo, suponha que a rede classificasse equivocadamente uma imagem como “8” quando deveria ser um “9”. É necessário descobrir como modificar minimamente os pesos para que a rede se aproxime da classificação da imagem como “9”. E então, esta ação poderá ser repetida, mudando os pesos quantas vezes for necessário para produzir resultados cada vez mais próximos do esperado, ou seja, a rede estará aprendendo [12].

Esse tipo de modificação é possível através da inserção de funções de ativação no modelo que permitem pequenas mudanças nos pesos que impactem em suaves alterações no resultado de saída.

Quando tem-se um modelo complexo em que são utilizadas redes neurais artificiais, as funções de ativação são responsáveis por definir quando que o neurônio está recebendo uma informação que é verdadeiramente relevante para a sua decisão.

É preciso entender que qualquer tipo de unidade da rede neural que pode ser utilizada como um valor de saída também pode ser utilizado dentro das camadas ocultas, dessa forma pode-se supor que a rede oferece um conjunto de recursos $h = f(x; \theta)$, o papel das camadas da rede é prover transformações adicionais a partir dos recursos recebidos para alcançar a tarefa que a rede deve realizar [16].

A função de ativação, h, é responsável por transformar, não linearmente, um sinal de entrada em um sinal de saída que será utilizado na próxima camada de neurônios como entrada ou ainda, que será a saída final do modelo. Sendo que quando não é utilizada, os pesos conseguem apenas transformar linearmente um sinal de entrada. A importância da transformação não linear está na capacidade limitada que uma função linear é capaz de resolver um problema.

2.4.1.1 Função de Etapa Binária

A função de etapa binária é responsável por determinar quando um neurônio deve ou não ser ativado, ou seja, ela deve definir se quando o valor f(x) estiver acima de um limite determinado se o neurônio deve ficar desativado ou ser ativado.

Devido ao formato de funcionamento desta função, ela pode ser utilizada para criar um classificador binário, ou seja, quando devemos dizer sim ou não para uma única classe essa se encaixaria para ativar o neurônio ou deixá-lo zerado.

Uma consideração importante a ser feita a respeito, é que o gradiente da função de etapa é zero. Assim, como explicado anteriormente, em uma rede neural a utilização do algoritmo de *backpropagation* se torna essencial para uma melhor assertividade do modelo, por isso faz com que a função de etapa não seja tão útil durante o *backpropagation* quando os gradientes das funções de ativação são enviados para cálculos de erro para melhorar e otimizar os resultados. Isso ocorre pois o gradiente da função de etapa reduz tudo para zero e a melhoria dos modelos realmente deixa de acontecer.

2.4.1.2 Função Linear

Outra função existente é a função linear, ela pode ser definida como:

$$\text{Equação 4: } f(x) = ax \qquad (4)$$

Contudo, essa função também não conseguirá agregar tanto para o algoritmo de *backpropagation*, uma vez que sua derivada será sempre constante, ou seja, não dependerá do valor de x, implicando que o gradiente será sempre o mesmo. Assim, ela não estará melhorando o erro.

Outro ponto importante é que se essa função for utilizada para realizar uma tarefa complicada na qual sejam necessárias diversas camadas na rede neural e em cada camada for utilizada uma transformação linear, mesmo que existam mais que uma camada, a saída será sempre o resultado de uma transformação linear da entrada.

2.4.1.3 Função Sigmóide

A função sigmóide é uma função continuamente diferenciável e suave, sua diferença em relação às duas anteriores é que ela não é linear, isto é, quando existem diversos neurônios tendo essa função como sua função de ativação, a saída não será linear, sendo que ela poderá variar sua de 0 a 1 tendo um formato “S”.

$$\text{Equação 5: } f(x) = \frac{1}{1+e^{-x}} \qquad (5)$$

Esta é caracterizada por tentar empurrar os valores de f(x) para os extremos, o que é muito importante quanto tenta-se classificar os valores para uma classe específica.

Deve-se atentar apenas a alguns pontos relacionados a sua utilização: i) seus valores apenas variam de 0 a 1, ou seja, ela não é simétrica em torno da origem e seus valores são todos positivos. Isto torna-se um problema quando não é desejado que os valores enviados ao próximo neurônio sejam todos do mesmo sinal; ii) Quando os valores dos gradientes se tornam muito pequenos significa que o gradiente está se aproximando de

zero e a rede não estará realmente aprendendo e pelo fato de essa função deixar os valores somente entre 0 e 1, podem existir muitos casos em que eles se aproximem do zero.

2.4.1.4 Função Tanh

A função tanh é responsável por resolver o problema de simetria encontrado na função sigmóide, pois ela escalona a função anterior permitindo que o resultado varie entre -1 e 1, permitindo também que os valores tenham sinais diferentes.

Equação 6: $tanh(x) = \frac{2}{1+e^{-2x}} - 1$ (6)

No tocante às outras propriedades, estas serão as mesmas da função sigmóide: i) é contínua e diferenciável em todos os pontos; ii) Não é uma função linear, sendo possível utilizar o algoritmo de *backpropagation* para melhorar o erro.

2.4.1.5 Função ReLU

Esta função é conhecida por ser a mais amplamente utilizada para a construção de redes neurais, atualmente. A função ReLu tem como característica a não linearidade, o que permite a sua utilização para a otimização de erros através do *backpropagation*, além de poder ter diversas camadas de neurônios ativados por ela.

Equação 7: $f(x) = max(0, x)$ (7)

O diferencial da implementação dessa função de ativação em comparação com as outras é que a ReLu não ativa todos os neurônios ao mesmo tempo, isto é, se a entrada for negativa, ela será convertida em zero e não ocorrerá a ativação do neurônio. Essa característica implica na criação de uma rede esparsa e eficiente.

Por fim, vale atentar-se ao mesmo ponto que a função sigmóide: valores que se deslocam em direção a zero podem significar que a rede não está mais aprendendo.

Equação 8: $f(x) = ax,\ x < 0$ (8)

$$f(x) = x,\ x >= 0$$

Desta forma, ao invés de obter uma linha horizontal para casos negativos, obtém-se uma linha não horizontal, removendo a possibilidade de gradientes iguais a zero e, consequentemente, o não aprendizado da rede.

2.4.1.6 Função SOFTMAX

Uma função de ativação muito importante é a função softmax, esta é utilizada para realizar uma multi classificação dos dados, ou seja, o softmax transforma as saídas para cada classe para valores 0 e 1 e também divide pela soma das saídas, mostrando a probabilidade da entrada estar em uma determinada classe.

Esta é utilizada na camada de saída do classificador, pois ela será responsável por gerar as probabilidades para definir a classe de cada entrada, por exemplo, se temos as saídas [1.2, 0.9, 0.75], após a aplicação da softmax, seria obtido [0.42, 0.31, 0.27], ou seja, um vetor de probabilidades de classificação da entrada em relação à classe de saída, sendo que sempre somará 1.

$$\text{Equação 9: } \sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \; para\; j = 1,..., K \qquad (9)$$

Assim, por exemplo, se for construído um modelo DP em que são utilizadas camadas densas, será necessário que o modelo contenha uma outra camada densa com a dimensão igual a quantidade de classes a serem classificadas com a função de ativação softmax para apresentar o resultado no formato adequado [12,16].

Tabela 1 — Comparativo função de ativação

Função de Ativação	***Transformação***	***Momento de utilização***	***Observação***
Função de Etapa binária	linear	Camadas de saída	Não funciona no algoritmo de *backpropagation*, adequada para classificações binárias
Função linear	linear	Camadas ocultas ou de saída	Não funciona no algoritmo de *backpropagation*, tem capacidade limitada de compreender relações mais complexas
Sigmóide	não-linear	Camadas de saída	Valores dos gradientes tendem a se aproximar do zero e assim parar o aprendizado da rede
Tanh	não-linear	Camadas ocultas ou de saída	Valores dos gradientes tendem a se aproximar do zero e assim parar o aprendizado da rede, mas é uma função simétrica e que permite diferentes sinais
ReLu	não-linear	Camadas ocultas	A função de ativação mais utilizada em camadas ocultas, entretanto pode também ter o problema de aproximar os

			gradientes de zero
Softmax	não-linear	Camadas de saída	Utilizada na camada de saída de redes de multi-classificação

Fonte: O autor

3 Modelo

A partir dos conceitos de aprendizado de máquina distribuídos vistos, é possível entender que esse formato tem como intuito melhorar a maneira como os dados de usuários são armazenados, além de economizar armazenamento de grandes quantidades de dados em servidores centrais. Contudo, quando busca-se aplicar este modelo na prática, são encontrados alguns problemas que ainda não permitem uma utilização em massa do modelo: i) grande demanda de processamento nos dispositivos de borda, que na maioria das vezes, são dispositivos limitados em capacidade e armazenamento; ii) Transmissão de informações limitada pelas características de dispositivos IoT.

Neste projeto, será apresentada uma alternativa de solução para o problema de processamento nos dispositivos de borda. A proposta tem como base a utilização do cálculo de entropia dos dados armazenados nos dispositivos, com o intuito de obter para o treinamento do modelo apenas os dados mais significativos, pois, tendo como base [65], acredita-se que os dados geram valor ao conter um grau de novidade sobre uma variável aleatória observada. Dessa forma, se conseguirmos determinar a quantidade de informação que um conjunto de dados leva consigo antes de processá-lo, será possível evitar o uso desnecessário de processamento e recursos, além de ter a possibilidade de reduzir o tempo de computação do modelo e o número de épocas por rodadas sem interferir drasticamente em sua precisão.

A entropia se encaixa na proposta pelo fato de representar a quantidade média de informação necessária para especificar o estado de uma variável, isto é, a baixa entropia significa alto conteúdo de informação e, consequentemente, a baixa probabilidade de ocorrência de eventos. Assim, se a informação contida nos dados estiver espalhada em muitos valores, ela terá uma alta entropia.

Dessa forma, tem-se que o objeto de estudo de otimização do modelo de aprendizado centralizado neste trabalho será o cálculo de entropia detalhado em [4], que define o cálculo da entropia para um estado x_i para uma variável X sendo:

Equação 10: $$H[p(X = x_i)] = -\sum_i p(x_i)\, ln\, p(x_i) \qquad (10)$$

Além da aplicação do cálculo de entropia para auxiliar na escolha dos dados que serão utilizados para o treinamento do modelo localmente, o projeto pretende implementar toda a arquitetura do modelo de aprendizado distribuído, anteriormente explicada, ou seja,

será simulado um servidor central responsável por definir o modelo de treinamento, seus hiperparâmetros, a quantidade de rodadas que ocorrerão e por calcular a média federada dos parâmetros recebidos dos dispositivos de borda.

Também serão simulados dispositivos de borda que armazenam dados específicos relacionados ao que o modelo de aprendizado de máquina deseja treinar. Estes serão responsáveis por receber os parâmetros enviados pelo servidor, calcular a entropia de seus dados e decidir quais terão relevância na definição dos parâmetros locais, definir os parâmetros, a partir do cálculo do gradiente, e enviar para o servidor.

A diferença do modelo tradicional de aprendizado federado, será a inclusão do cálculo de entropia no processo, para isso será necessário que o cliente tenha mais de uma troca de informação com o servidor durante a rodada. Isso é necessário para definir quais clientes farão parte da rodada de definição dos parâmetros. Cada rodada executa o algortimo descrito a seguir:

1. O servidor definirá randomicamente quais clientes farão parte daquela rodada, e informará isso para os clientes, como indicado na Figura 8.
2. Os clientes calculam a entropia dos seus dados, Figura 8.
3. Os clientes enviam sua entropia para o servidor, Figura 9.
4. O servidor calcula a média da entropia de todos os clientes disponíveis para a rodada, Figura 9.
5. O servidor envia o modelo global somente para os clientes que possuírem a entropia abaixo da média calculada anteriormente, Figura 10.
6. Os clientes que receberam o modelo global realizam o cálculo dos parâmetros locais, Figura 10.
7. Os clientes enviam os parâmetros locais para o servidor, Figura 11.
8. O servidor calcula a média federada dos parâmetros recebidos, Figura 11.
9. Servidor define o novo modelo global a partir dos parâmetros calculados, Figura 11.
10. Volta para o passo 1.

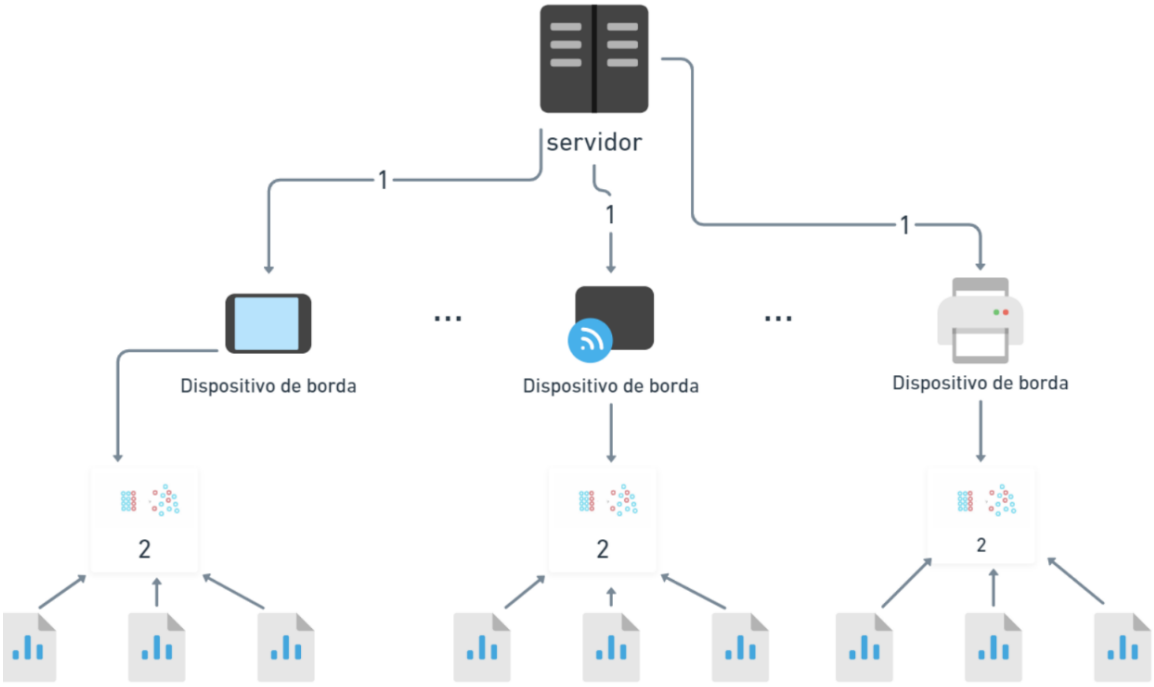


Figura 8 — Servidor define os clientes da rodada e os clientes calculam a entropia de seus dados

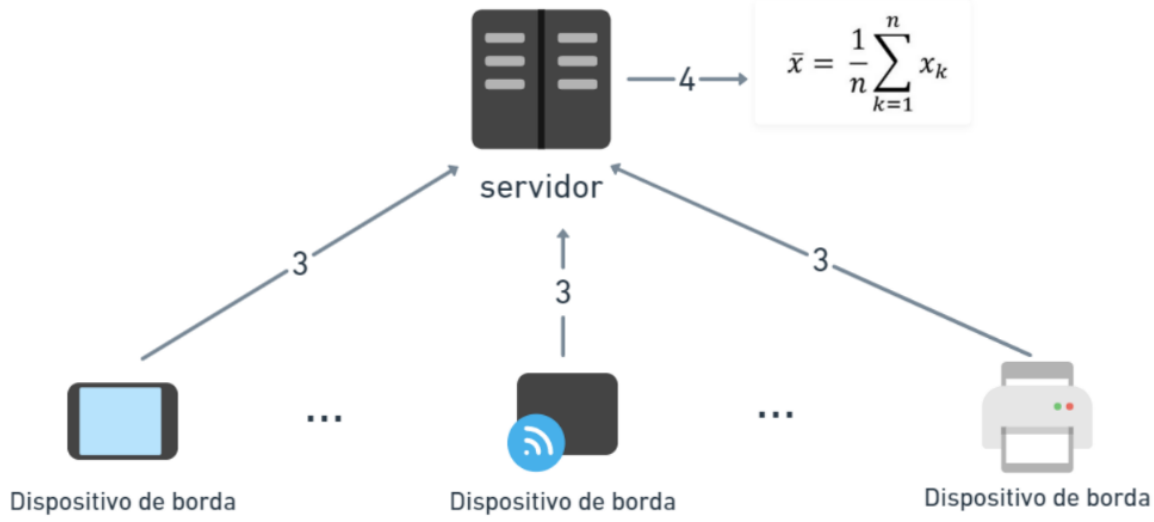


Figura 9 — Servidor recebe entropias e calcula a média

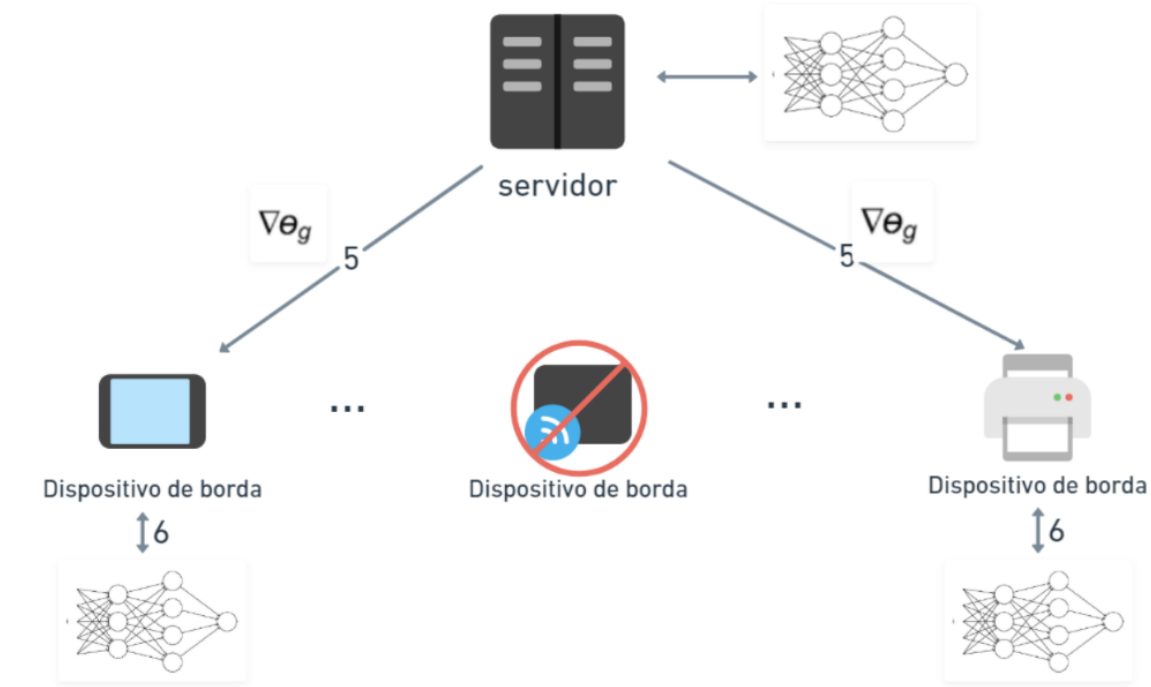


Figura 10 — Clientes selecionados recebem o modelo global e calculam parâmetros locais

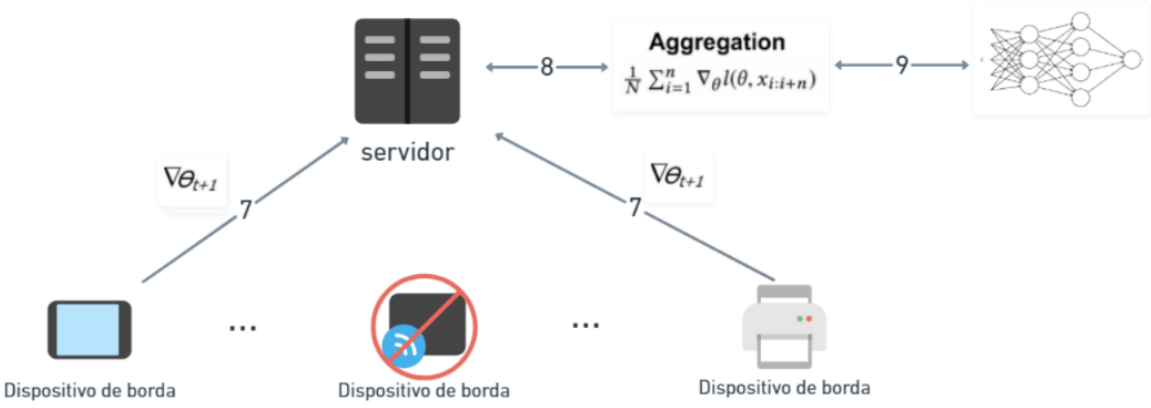


Figura 11 — Clientes enviam novos parâmetros para servidor que calcula a média federada e define novo modelo global

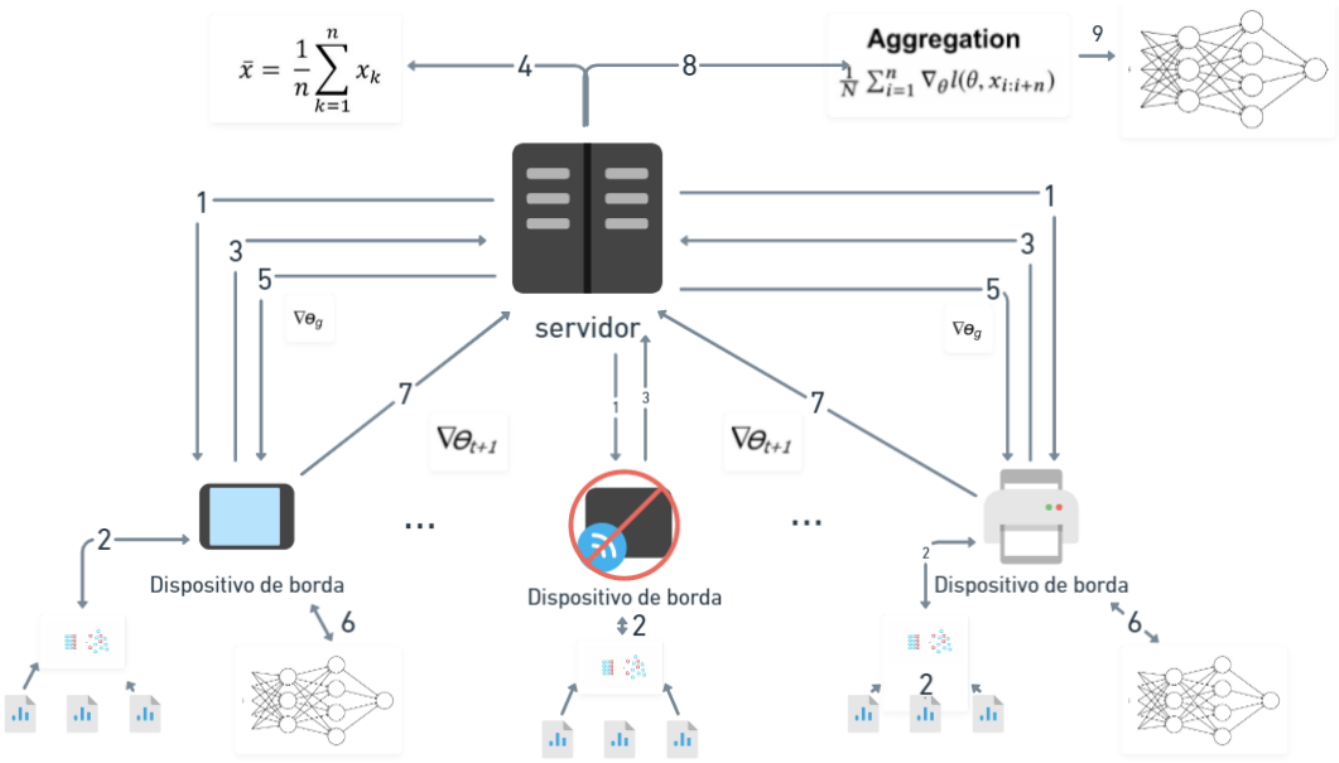


Figura 12 — Aprendizado de máquina federado com processo de média federada e cálculo de entropia

A hipótese que deseja-se provar é de que o tempo utilizado para calcular a entropia de cada cliente (etapa 1), a transmissão destes dados para o servidor (etapa 2), o cálculo da média de entropia pelo servidor (etapa 3) e a definição de quais clientes estão abaixo da média e poderão participar da rodada (etapa 4), somado ao restante das etapas, Figura 12, terá um tempo de processamento menor do que treinarmos o modelo com os dados de todos os clientes, Figura 12 e 13. Caso esta hipótese esteja correta, consequentemente, pode-se inferir que este formato demandará menos esforço dos dispositivos de borda, tornando a aplicação do aprendizado de máquina federado viável para ser utilizado.

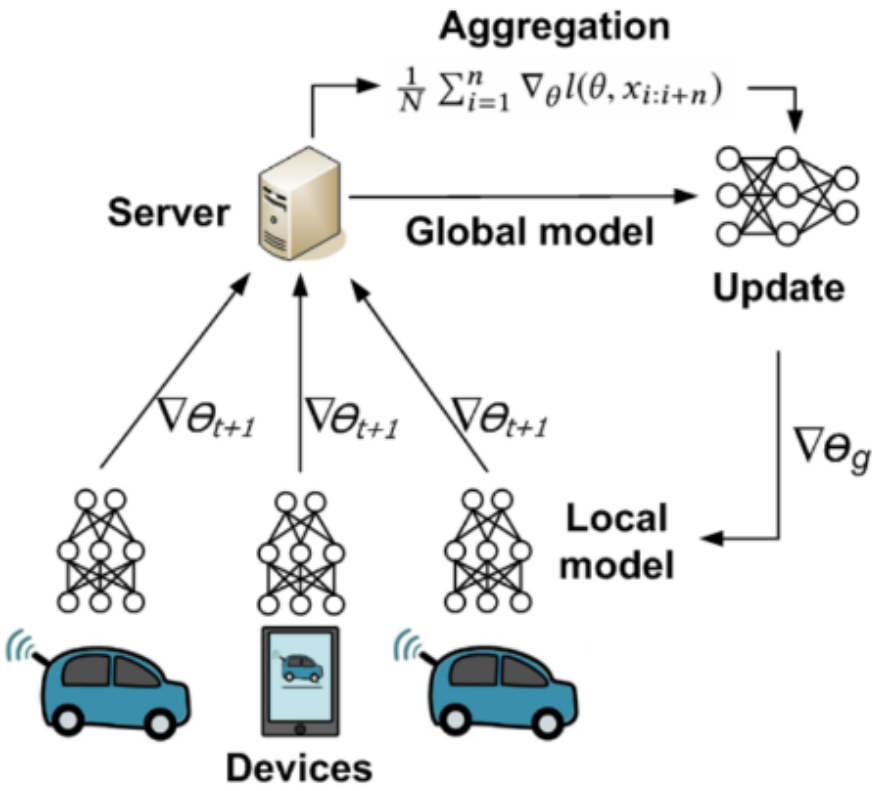


Figura 13 — Aprendizado de máquina federado com processo de média federada

Fonte: Retirado de J.C.S. Anjos [4]

Em relação ao modelo de treinamento que será utilizado, optou-se por utilizar uma rede neural artificial ao invés de um simples modelo de aprendizado de máquina. Essa opção nasceu do contexto em que são analisados dados de dispositivos de borda e estes podem estar relacionados a diversas situações, ou seja, uma rede neural tem maior chance de conseguir lidar com os tipos de dados e relações que surgirão do que um modelo de aprendizado de máquina que pode ser limitado. Como a otimização pretende ser utilizada para outros contextos além do que será apresentado, é válido realizar testes de performance, eficiência e processamento em situações mais próximas das que serão encontradas no dia a dia.

Dessa forma, ao longo do experimento serão realizados testes com diferentes quantidades de batches e épocas com a intenção de avaliar se a quantidade destes precisam ser modificadas a partir do momento que há maior seletividade nos dados que serão utilizados para treino. Ainda, com o intuito de evitar o enviesamento dos dados, serão propostos testes em que a cada rodada serão escolhidos novos clientes, para que seja possível abranger a maior diversidade de dados possível, o que pode ser conflitante para

conseguir avaliar a acurácia de treino do modelo, pois cada modelo gerado terá seus parâmetros definidos por dados diferentes.

Serão também variadas as quantidades de rodadas nos treinamentos dos modelos, buscando entender quantas rodadas serão necessárias para que o modelo consiga ser consistente o suficiente para avaliar os dados de testes.

Em relação ao modelo utilizado, será um modelo supervisionado pelo fato de que este projeto tem como intuito comparar a quantidade de processamento computacional com e sem o cálculo de entropia dos dados de entrada e também a acurácia dos modelos, a fim de entender se será possível poupar processamento mantendo um certo nível de acurácia, dessa forma, dados supervisionados irão contribuir para alcançar uma boa acurácia do modelo, neste momento.

Por fim, para que seja possível validar a qualidade do modelo, serão utilizados funções de perdas, acurácia, enquanto que para validar sua eficiência serão calculados os tempos de processamento do modelo com o cálculo de entropia e comparados ao modelo sem a otimização.

4 Materiais e Métodos

4.1 Ferramental

O aprendizado de máquina federado, por ser um tema muito recente, traz a possibilidade de realizar estudos de casos com diferentes ênfases, por exemplo estudos a respeito da eficiência da comunicação síncrona em relação a comunicação assíncrona entre servidor e cliente, bem como estudos a respeito da definição do melhor tipo de comunicação de dados (RPC, REST e outros), ou até mesmo a respeito da maneira de calcular os parâmetros globais, trazendo cálculos diferentes do que a média federada.

Contudo, optou-se neste estudo de caso ter como objeto de estudo a implementação de uma ferramenta de otimização na decisão dos conjuntos de dados dos clientes que farão parte das rodadas de treinamento do modelo. A otimização tem como objetivo diminuir o tempo de processamento do treinamento do modelo, a fim de diminuir a exigência computacional dos dispositivos de borda.

Como o aprendizado de máquina distribuído possui algumas especificidades a respeito de comunicação entre servidor e cliente, o não compartilhamento dos dados do usuário com o servidor e o cálculo do modelo no dispositivo de borda, as ferramentas utilizadas para construção de modelos de aprendizado centralizado acabam sendo limitadas neste contexto. Dessa forma, buscou-se entender a respeito de ferramentas desenvolvidas com o foco em aprendizado distribuído.

Uma ferramenta que está em experimento é o NVFlare [17], construído pela Nvídia, esta utiliza um sistema operacional na nuvem, Openstack [18] e permite a criação de um ambiente de aprendizado distribuído. Outra ferramenta disponível que apresenta certa facilidade na construção e utilização do aprendizado distribuído é o Flower [19], entretanto é importante ressaltar que os métodos utilizados dentro do algoritmo para a troca de informações entre servidor e cliente são uma caixa preta, o que pode trazer certa dificuldade caso o objetivo seja testar diferentes tipos de comunicação.

Como o foco deste projeto é validar o cálculo de entropia aplicado a seleção dos dados de treinamento, buscou-se utilizar ferramentas que oferecessem clareza nos métodos de comunicação entre cliente e servidor, de forma que fosse possível modificar sua ordem ou acrescentar condições sem interferir no funcionamento da ferramenta, isto é, arquiteturas modulares. Também procurou-se algo que fosse capaz de fazer a construção e utilização do modelo sem muitas complicações a respeito de como configurar o ambiente de teste. Por fim, havia a intenção de utilizar uma ferramenta com robustez para lidar com o processamento de grandes quantidades de dados sem que fosse comprometido o desempenho do modelo.

Dessa forma, optou-se por utilizar o Google Colab [20] como plataforma para inserir os algoritmos utilizados, bem como para realizar os testes do projeto e gerar gráficos e métricas, uma vez que esta plataforma utiliza de hardware online para o processamento computacional e oferece o uso de GPUs.

Tabela 2 — Ferramentas e ambientes de desenvolvimento

Ferramenta	***Descrição***	***Disponível em***
Google Colab	Plataforma para desenvolvimento de inteligência artificial	https://colab.research.google.com/?utm_source=scs-index
Flare	Ambiente de execução e aplicação de aprendizado distribuído	https://developer.nvidia.com/flare
FedAvg (média federada)	Exemplos de algoritmos de média federada	https://github.com/vaseline555/Federated-Averaging-PyTorch https://github.com/AshwinRJ/Federated-Learning-PyTorch
Implementação do aprendizado distribuído	FL code example Flower FL com comunicação REST	https://www.tensorflow.org/federated/federated learning https://flower.dev/ https://github.com/FederatedAI/FATE

Fonte: Adaptado de J.C.S. Anjos [4]

4.2 Bibliotecas

A partir da decisão da ferramenta, foi tomada a decisão de quais bibliotecas utilizar para a escrita dos algoritmos, optou-se por utilizar as bibliotecas Tensorflow Federated (TFF) para a construção do modelo distribuído, a qual possui métodos personalizáveis para a construção da rodada do modelo, bem como quais dados serão transmitidos entre cliente e servidor, além de outras opções de otimização.

É importante ressaltar que essa ferramenta foi criada para conduzir pesquisas a respeito de aprendizado federado, simulando cálculos federados em conjuntos de dados, dessa forma o ambiente utilizado para este projeto é um ambiente simulado em que um driver externo simula a lógica de controle de um sistema federado de produção, selecionando clientes simulados de um conjunto de dados e executando cálculos federados como a média federada.

Para a construção do modelo de deep learning, foi utilizada a biblioteca Tensorflow, que oferece de maneira muito intuitiva a possibilidade de criar modelos de aprendizado personalizados, podendo escolher a quantidade de camadas da rede, funções de ativação, tipos de camadas, função de perda e tipo de acurácia que são responsáveis por analisar o modelo, além de dar a permissão de escolher como será feita a descida do gradiente e quais serão os hiperparâmetros do modelo.

Na seleção do conjunto de dados utilizou-se a própria biblioteca Tensorflow Federated que possui um módulo de conjunto de dados próprios chamada Datasets, este módulo é responsável por armazenar diferentes conjuntos de dados que são comumente utilizados para experimentos de aprendizado de máquina federado e que permitem algumas configurações do formato como o conjunto virá, além de oferecer a oportunidade de optar por um conjunto de dados que já possua alguns tratamentos para facilitar no momento de usar os dados.

Ainda, para auxiliar nos cálculos de entropia e na hora de manusear os dados foram utilizadas bibliotecas auxiliares como Numpy, Matplotlib e Scipy. Por fim, para o cálculo de performance dos modelos, foram utilizados os módulos Functolls, Tracemalloc e Time do python.

Tabela 3 — Bibliotecas do *python* utilizadas

Biblioteca/módulo	***Descrição***	***Disponível em***
Tensorflow Federated	Biblioteca que fornece métodos de construção de um modelo distribuído de aprendizado de máquina e conjunto de dados apropriados	https://www.tensorflow.org/federated
Tensorflow	Biblioteca de código aberto para desenvolver e criar modelos de ML	https://www.tensorflow.org/

Numpy	Biblioteca para auxiliar no processamento de grandes, multi-dimensionais arranjos e matrizes	https://numpy.org/
Matplotlib	Biblioteca de software para criação de gráficos e visualizações de dados em geral	https://matplotlib.org/
Scipy	Biblioteca que contém algoritmos de otimização, interpolação, equações diferenciais, estatísticas e outros	https://scipy.org/
Functools	Para funções de ordem superior: funções que atuam ou retornam outras funções	https://docs.python.org/pt-br/3/library/functools.html
Tracemalloc	Uma ferramenta de debug para rastrear blocos de memória alocados	https://docs.python.org/3/library/tracemalloc.html
Time	Fornece funções relacionadas ao tempo	https://docs.python.org/3/library/time.html

Fonte: O Autor

4.3 Conjunto de dados

Na definição de qual conjunto de dados ser utilizado para os experimentos, foi explorado um conjunto de dados reais chamado MNIST que vem de um banco de dados que armazena dígitos manuscritos com o intuito de ser utilizado para testes e treinos de modelos de aprendizado de máquina. Então, optou-se pela utilização do conjunto de dados conhecido como EMNIST, um conjunto de dados pré-processado retirado do MNIST, adaptado em [21] e publicado em [22]. Suas diferenças em relação ao conjunto MNIST são o formato como as cores são representadas, enquanto que o EMNIST representa a cor de fundo como 1.0 e a cor do dígito como 0.0, o MNIST representa o primeiro como 0 e o segundo como 255. Além de já agrupar os dados e definir usuários que contenham esses grupos de dados.

Dessa forma, o conjunto de dados tem o tamanho de 1,7GB e contém 341.873 exemplos para treino, 40.832 exemplos de teste, sendo que estes dados estão dispostos em 3.383 usuários. Em relação a estrutura dos dados, os dados de entrada encontram-se em uma matriz de 28x28 pixels e classificados entre 10 rótulos de saída.

4.4 Entropia

Na construção do projeto, foram feitas análises de quatro algoritmos responsáveis por calcular a entropia dos conjuntos de dados dos clientes buscando o que obtivesse o melhor tempo de processamento, sempre com o intuito de otimizar o tempo de computação do modelo como um todo. Para a comparação, foi calculada a média de tempo para obter a entropia do conjunto de dados de um cliente, dessa forma, cada algoritmo de entropia calculou a entropia de 5 clientes e assim foi tirada a média do tempo utilizado para cada cliente. E o algoritmo E4 foi o que obteve menor tempo, realizando o cálculo de entropia de um usuário em aproximadamente 0,26 milissegundos.

Tabela 4 — Tempo de cálculo de entropia por diferentes algoritmos

Cálculo de entropia	***Descrição***	***Tempo médio***
E1	Utilizado o método *entropy* da biblioteca scipy	0,38944 milissegundos
E2	Calculado a probabilidade de cada rótulo presente no conjunto de dados e cálculo da entropia a partir da função de log do python	0,3181 milissegundos
E3	Calculado a probabilidade de cada rótulo presente no conjunto de dados a partir da biblioteca Pandas e calculada a entropia a partir das funções de log do Numpy	4,79368 milissegundos
E4	Calculado a probabilidade de cada rótulo presente no conjunto de dados a partir da biblioteca Numpy e calculada a entropia a partir das funções de log do Numpy	0,25624 milissegundos

Fonte: O Autor

4.5 Rede neural

Dentro da estrutura apresentada na seção 2, será necessário definir um modelo de deep learning que será utilizado na determinação dos parâmetros locais, bem como métodos de otimização que serão utilizados tanto na definição do modelo global armazenado no servidor quanto nos modelos armazenados pelos clientes.

A troca de dados, no formato apresentado na seção anterior, durante as rodadas serão realizadas por funções da biblioteca Tensorflow Federated, sendo que existem um conjunto de funções, definidos como “Partes individuais de códigos”, responsáveis por encapsular a lógica que será executada em um único local, por exemplo nos cálculos realizados nos clientes para a definição do modelo local pela criação do modelo global. Também existe um outro conjunto de métodos responsáveis pela parte de orquestração federada que une as partes individuais, por exemplo o cálculo da média federada dos parâmetros locais recebidos pelo servidor ou a transmissão do modelo global para os clientes.

4.5.1 Tratamento de dados

Para a utilização dos dados no modelo foi utilizada uma função da biblioteca do Tensorflow Federated que retorna instâncias dos usuários trazendo funcionalidades que permitem manusear os dados de cada cliente individualmente, além de criar identificadores para cada cliente, facilitando na hora de chamar um cliente específico caso seja preciso analisar seus dados e na iteração dos clientes pelo identificador, contribuindo no momento do cálculo da entropia de cada dispositivo de borda.

Ainda, com o intuito de otimizar o manuseamento dos dados e torná-los mais apropriados para o treinamento do modelo, foi realizado um pré-processamento dos dados com os seguintes passos:

1. Transformação da matriz de 28x28 pixels em um arranjo de uma única dimensão contendo 784 pixels
2. Normalização dos dados a partir da divisão de cada pixel por 255, sendo este o valor máximo para cada pixel, fazendo com que os dados sejam 0 ou 1.
3. Definição do número de batches do modelo (que serão modificados para cada experimento realizado)
4. Determinação da quantidade de épocas (também serão modificados para cada experimento)
5. Escolha do número de clientes que serão chamados por rodada

4.5.2 Definição da rede neural

Com o intuito de avaliar o desempenho do modelo com a inserção do cálculo de entropia dos dados, optou-se por utilizar uma rede neural não muito complexa presente em diversos artigos criados pelo Tensorflow no Google colab, a fim de compreender ao máximo quais seriam as interferências existentes no modelo que poderiam influenciar na assertividade do modelo de multi classificação.

Dessa forma, foi utilizada uma biblioteca do Tensorflow que possui diferentes tipos de camadas que podem ser inseridas na rede neural, chamada Keras [23] e optou-se por uma estrutura com uma camada de entrada com 784 nós, uma camada densa de 10 nós com a função de ativação Relu e uma camada de saída com a aplicação da função de ativação Softmax para permitir a classificação com 10 rótulos, sendo que estes são 10 categorias de 0 a 9 definidas de acordo com a Figura 14.

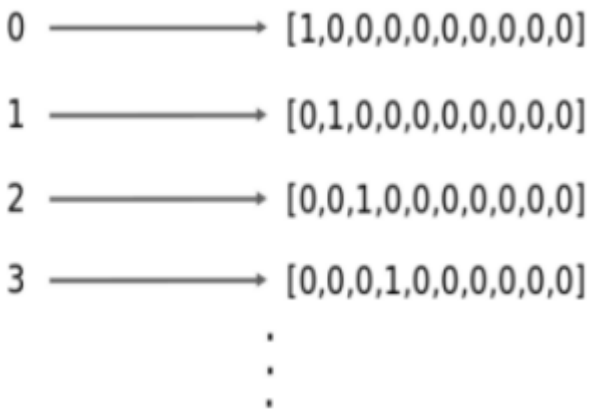


Figura 14 — Categorias possíveis na camada de saída da rede neural

Por conta de utilizar-se uma camada densa têm-se as seguintes conexões (784X10 + 10X10 + 10X10) = 8.040 Conexões.

4.5.3 Mecanismos de otimização

Os mecanismos de otimização utilizados foram: i) Épocas, com o intuito de que a rede consiga absorver mais informações dos dados dos clientes; ii) Entropia cruzada categórica esparsa, sendo este o cálculo de entropia cruzado aplicado para quando existem mais de duas classes de saída [23]; iii) Cálculo da Acurácia categórica esparsa do modelo, isto é, o cálculo da acurácia realizado para modelos com mais de duas classes de rótulos; iv) Na atualização dos parâmetros locais, com o intuito de otimizar, foi inserido o algoritmo de *backpropagation* durante o treinamento local, algo que vem ativado como padrão quando é utilizada a biblioteca do tensorflow federated para construir o modelo de aprendizado de máquina utilizado. Dessa forma, a cada batche foram armazenados os pesos de saída,

retropropagadas as perdas para calcular o gradiente da perda destes e, em seguida, a atualização dos pesos da rede a partir dos gradientes definidos.

O código que foi utilizado para a realização do projeto encontra-se no GitHub (https://github.com/relellis/Projeto-Final-Graduacao).

5 Resultados

Os primeiros experimentos realizados foram com o intuito de compreender o funcionamento da ferramenta TFF de modo geral, dessa forma, o cálculo de entropia e as modificações necessárias para sua implementação não foram implementados no primeiro momento, ou seja, no primeiro, segundo e terceiro experimento não foi utilizado o cálculo de entropia.

5.1 Primeiro experimento

Decidiu-se treinar o modelo com 30 rodadas em que eram executadas 100 épocas, cada uma com 100 dados rotulados por batches por rodada, com uma taxa de aprendizado de 0,02 e o treinamento foi executado com a contribuição de 10 clientes.

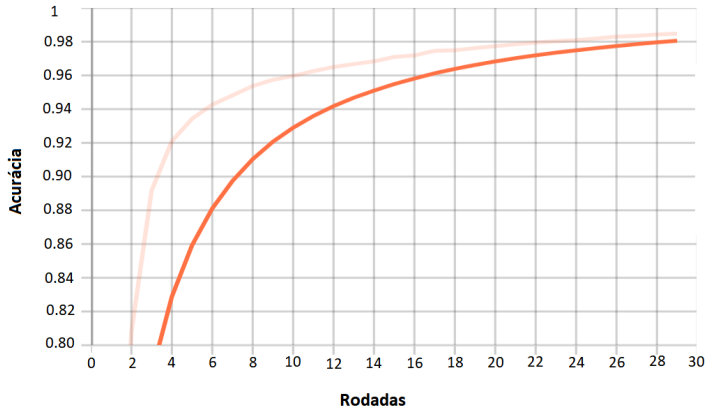


Figura 15 — Evolução da acurácia do modelo durante o treinamento do experimento 1

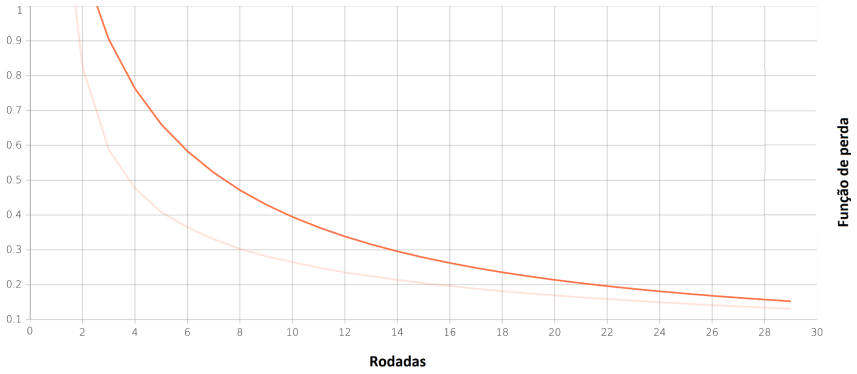


Figura 16 — Evolução da função de perda durante o treinamento do experimento 1

Neste primeiro experimento foi obtida uma acurácia de 0,98 e perda de 0,26. Observa-se que em torno da rodada 28 o modelo já alcança o máximo de acurácia, não sendo necessário 30 rodadas para chegar em parâmetros melhores.

5.2 Segundo experimento

No segundo experimento, optou-se por realizar 10 rodadas com 100 épocas e com 100 dados rotulados por batches a cada época com uma taxa de aprendizado de 0,02, também foi mantido o treinamento para 10 clientes, contudo, a seleção clientes para cada rodada foi feita de forma aleatória, evitando a repetição dos mesmos conjuntos de dados durante as rodadas.

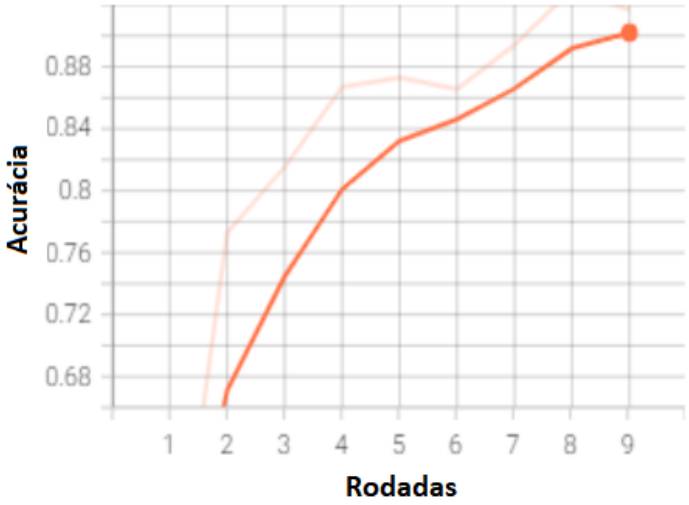


Figura 17 — Evolução da acurácia do modelo durante o treinamento do experimento 2

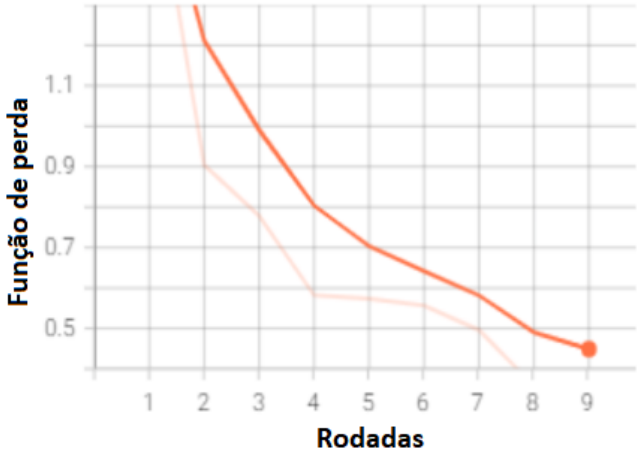


Figura 18 — Evolução da função de perda durante o treinamento do experimento 2

Foi encontrada uma acurácia de 0,90 e uma perda de 0,44. Dessa forma, infere-se que uma maior oferta de dados diferentes para o modelo faz com que sejam necessárias menos rodadas para se alcançar uma acurácia próxima a do experimento anterior, contudo foi necessário o processamento computacional de mais clientes e maior demanda por armazenamento do servidor para comunicar-se com diferentes usuários.

5.3 Terceiro experimento

Com os experimentos anteriores realizados, foi possível compreender o funcionamento das ferramentas e então foram iniciadas as medições a respeito de tempo de processamento juntamente com a acurácia e função de perda. Neste experimento ainda não foi calculada a entropia dos dados, pois desejou-se adotar um referencial de tempo de processamento sem este cálculo.

Para este experimento, foram mantidas as quantidades de rodadas, número de clientes (não escolhidos randomicamente) do experimento anterior. Somente foi alterado o número de batches para 5 dados rotulados por intervalo, a fim de entender como este pode influenciar no comportamento do modelo.

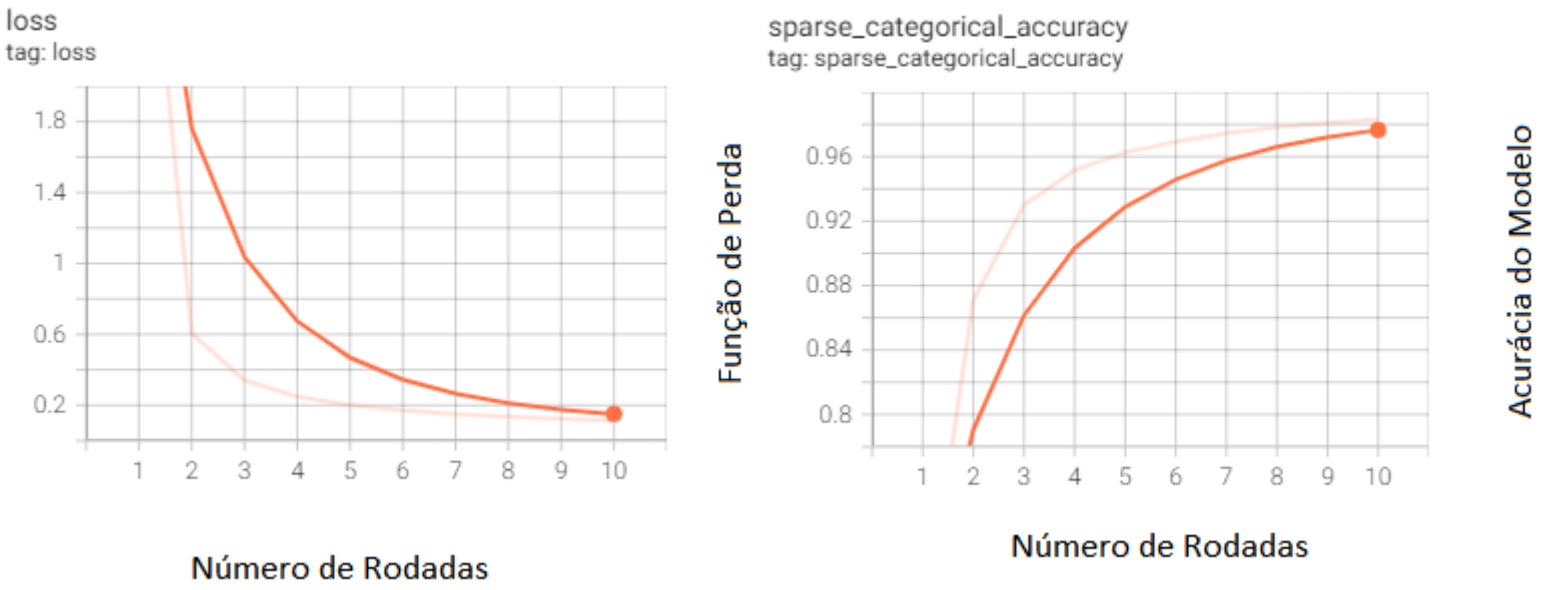


Figura 19 — Acurácia e função de perda do treinamento do Experimento 3

Tabela 5 — Resultados obtidos no experimento 3

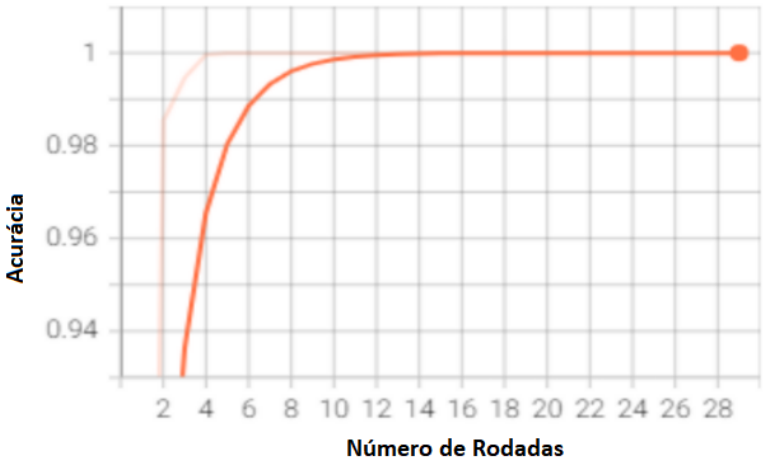
Parâmetros	***Resultados***
Acurácia do Treinamento	0,9084
Função de perda do treinamento	0,2971
Quantidade de batches no treinamento	19440
Número de exemplos no treinamento	97200
Acurácia do teste	0,8534
Função de perda do teste	0,508
Quantidade de batches no teste	2320
Número de exemplos no teste	11600

Fonte: O Autor

A partir dos resultados obtidos observou-se que não é necessário configurar um tamanho grande de amostragem de números por batche para o modelo, uma vez que foi alcançada uma acurácia muito próxima dos experimentos anteriores com um tempo total de execução de 169.128 milissegundos, sendo que 547 milissegundos foram para o pré-processamento dos dados e 168.581 milissegundos para a execução do modelo.

5.4 Quarto experimento

A partir deste experimento foi inserido o cálculo de entropia no modelo. Para ser possível realizar a comparação da acurácia e da função de perda para modelos com hiper-parâmetros semelhantes, foram definidos os seguintes hiperparâmetros: 10 clientes por rodada, 100 épocas, sendo que cada uma terá 100 dados rotulados por batches e 30 rodadas.



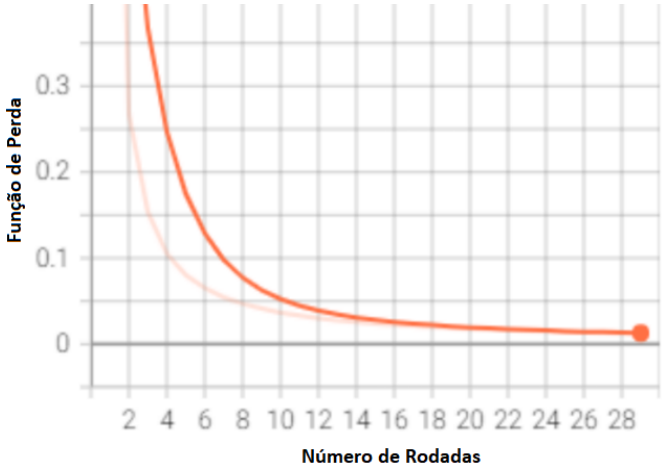


Figura 20 — Acurácia e função de perda do treinamento do Experimento 4

O que foi observado neste experimento foi o enviesamento do modelo, uma vez que alcançou-se 100% de assertividade, ou seja, o modelo adaptou-se totalmente aos dados mostrados, sem considerar qualquer variação dos dados. Acredita-se que isto ocorreu devido ao fato de em todas as rodadas de treinamento foram utilizados os conjuntos de dados dos mesmos 10 clientes, dessa forma, o modelo se adaptou apenas aos dados mantidos por eles, caso fosse aplicado um teste nesse momento, poderia ser verificado uma menor assertividade por parte do modelo, devido ao fato de que os dados de teste serem diferentes dos utilizados durante o treinamento, dessa forma, apesar de a acurácia ser uma ferramenta muito importante para se medir a qualidade do modelo, esta deve ser utilizada com atenção para não trazer a falsa sensação de que com uma acurácia alta têm-se um modelo otimizado para a utilização na vida real.

5.5 Quinto experimento

Com o intuito de evitar o enviesamento dos dados encontrou-se a estratégia de escolher aleatoriamente os clientes de cada rodada de treinamento, a fim de ofertar dados diversos e evitar que o modelo ficasse restrito a um pequeno formato de dados. Dessa forma, foram utilizadas as seguintes configurações: 30 rodadas, cada uma com 100 épocas, sendo que cada uma terá 100 dados rotulados por batches e 100 clientes diferentes por rodadas.

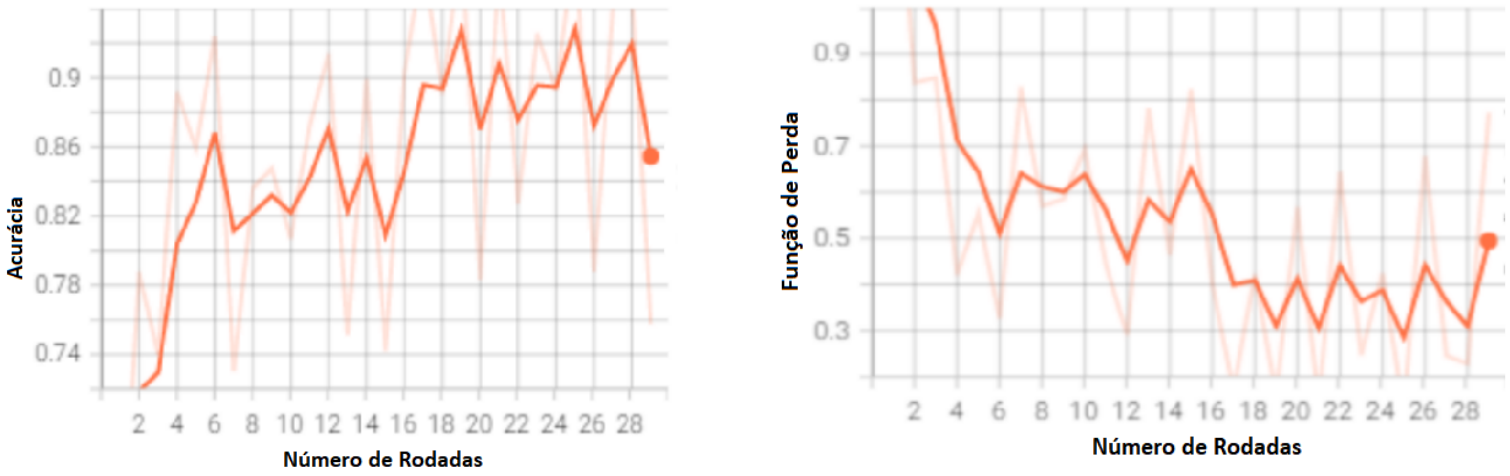


Figura 21 — Acurácia e função de perda do treinamento do Experimento 5

Neste experimento foi obtida uma acurácia de 0,865 e uma função de perda de 0,5, sendo possível validar a diminuição no enviesamento do modelo, assim, validando a inserção de clientes randômicos por rodadas, mas ainda foi verificado que o número de rodadas pode ser menor, uma vez que há uma maior quantidade de dados diferentes sendo mostrados ao longo das rodadas, o modelo consegue chegar em sua maior acurácia já entre a 18ª e 20ª rodada, não sendo necessário comprometer o processamento computacional dos dispositivos de borda por tanto tempo.

5.6 Sexto experimento

Neste último experimento, com base nos aprendizados retirados dos anteriores, foram feitas algumas alterações nos hiperparâmetros do modelo a fim de buscar uma melhor acurácia e tempo de processamento otimizado, além de deixar condições semelhantes com o experimento 3 para efeitos de comparação, ou seja, foram configuradas 10 rodadas, sendo que cada com 100 épocas, batches de tamanho 5 e taxa de aprendizado 0,02.

Neste foram realizadas as medições a respeito de tempo de processamento juntamente com a acurácia e função de perda, também foi utilizado o cálculo de entropia dos dados dos clientes, sendo que para cada rodada foram escolhidos 10 clientes randomicamente.

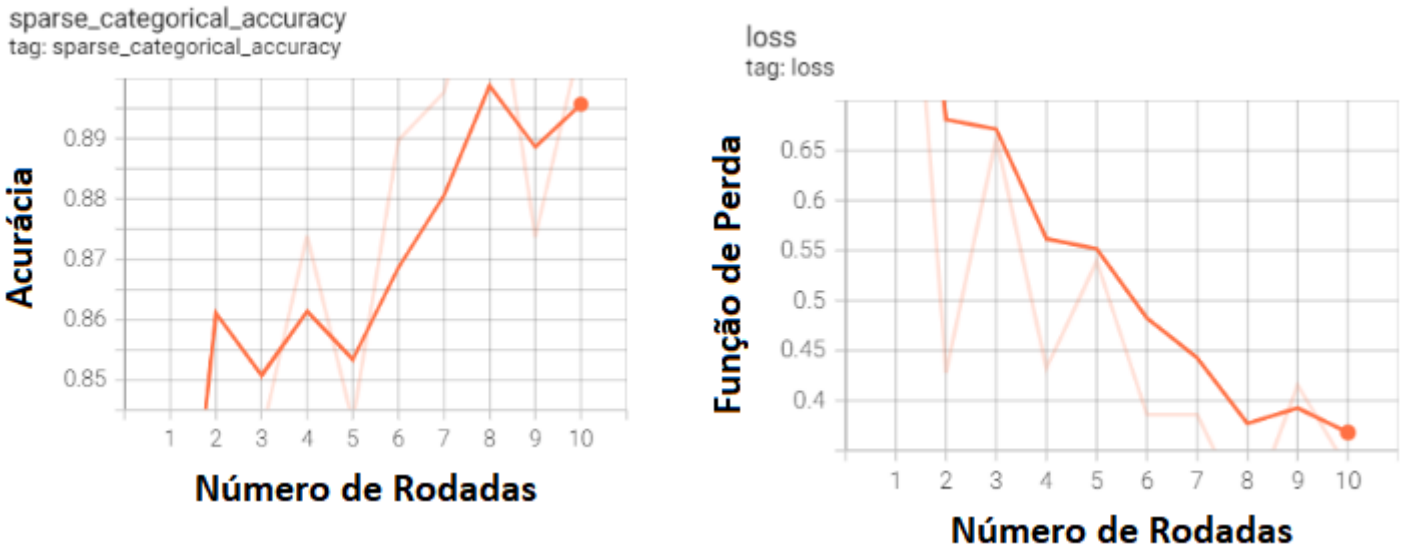


Figura 22 — Acurácia e função de perda do treinamento do Experimento 6

Tabela 6 — Resultados obtidos no experimento 6

Parâmetros	***Resultados***
Acurácia do Treinamento	0,9061
Função de perda do treinamento	0,3317
Quantidade de batches no treinamento	5220
Número de exemplos no treinamento	26100
Acurácia do teste	0,7234
Função de perda do teste	1,2309
Quantidade de batches no teste	7520
Número de exemplos no teste	37600

Fonte: O Autor

A partir dos resultados obtidos observou-se que não é necessário configurar um tamanho grande de amostragem de números por batche para o modelo também quando há cálculo de entropia, uma vez que foi alcançada uma acurácia muito próxima dos experimentos anteriores com um tempo total de execução de 50.575 milissegundos, sendo que para cada cliente foi utilizado um tempo de 0,27 milissegundos para o cálculo de entropia, resultando em um tempo total 27 milissegundos e um tempo de processamento total de 547 milissegundos, dessa forma restando 50.000 milissegundos para a execução do modelo.

5.7 Análise final

Com os experimentos concluídos, é possível dividir a análise final em 2 subtópicos diferentes: valores dos hiperparâmetros e maneira de selecionar os dados e aplicação do cálculo de entropia.

5.7.1 Hiperparâmetros e Seleção dos dados

Durante a realização dos experimentos foi possível validar que, não é necessário, para este caso, manter-se um número alto de amostragem por batche e de rodadas quando os dados são inseridos de forma randômica ao longo das rodadas, isso se dá pelo fato de que com a aleatoriedade dos dados, o modelo consegue ter contato com os diferentes casos de forma mais rápida, não sendo necessário calcular seus parâmetros para muitos dados. O fato de inserir o cálculo de entropia no modelo também contribui para uma convergência do modelo mais rápida, indo ao encontro da hipótese de que o cálculo de entropia auxilia em uma escolha de dados de maior qualidade.

Aqui, foi possível validar também que com essa melhor seleção dos dados, não é preciso que muitos clientes sejam selecionados por rodada para aperfeiçoar a assertividade do modelo criado. Entretanto, é necessário que mais estudos sejam feitos com foco neste tema, pois pelo fato de o modelo lidar com dados heterogêneos, não é possível afirmar que esse padrão de poucos clientes será mantido para outros modelos e temas, uma vez que quando a quantidade de clientes é menor, pode-se estar perdendo dados importantes que estejam armazenados em dispositivos específicos.

Ainda, é importante citar a respeito da importância da escolha de dados de forma randômica ao longo das rodadas, pois, de acordo com o quarto experimento verificou-se que ao manter sempre os mesmos clientes para todo o treino, por se tratar de uma rede neural que possui um alto nível de complexidade e resolubilidade, quando os mesmos dados são percorridos diversas vezes, o modelo rapidamente é capaz de encontrar o padrão dos dados e assim trazer uma informação 100% precisa para aqueles dados. Entretanto, quando esse modelo é utilizado fora do ambiente de treinamento, pode ter sua assertividade prejudicada, pois não teve contato com dados diferentes. Assim, verifica-se

como a escolha da quantidade de clientes e a heterogeneidade destes pode impactar na performance do modelo.

Por fim, acredita-se que ao contemplar todas essas características citadas, o modelo foi capaz de otimizar sua eficiência de modo a reduzir seu tempo de processamento de dados e manter a qualidade da informação resultante.

5.7.2 Cálculo de Entropia

O que é possível verificar é que a execução do modelo com o cálculo de entropia tem um desempenho 70,34% melhor, quando são inseridos os tempos de pré-processamento dos dados e do cálculo de entropia, obtém-se um desempenho de 70,09% melhor. Vale ressaltar que o overhead existente pelo cálculo de entropia equivale a 0,0053% do tempo de execução, ou seja, o tempo utilizado para este cálculo é insignificante comparado ao aumento de desempenho que ele provoca. Contudo, é possível verificar que há uma diminuição de 13% na acurácia dos testes do modelo do experimento 3 para o experimento 6, o que pode ser muito significativo para a utilização do modelo em algumas aplicações que necessitem de resultados mais precisos.

Em relação ao treinamento do experimento 3 e 6, é verificada uma diferença de 8% de acurácia, o que demonstra que o modelo de treinamento tem a chance de ser otimizado a fim de diminuir esta diferença e assim impactar na acurácia do teste. Também é válido ressaltar que, como os dados são inseridos de forma randômica em cada rodada, é possível que em outros treinamentos sejam alcançadas acurácias diferentes. Ainda, quando é visualizado o gráfico de acurácia do experimento 6 na Figura 21, verifica-se que houve um pico de acurácia na rodada 8, ou seja, para projetos futuros podem ser implementadas maneiras de eleger os parâmetros finais como sendo aqueles que alcançaram a maior acurácia dentre todas as rodadas sem impactar no desempenho do modelo ou na demanda computacional dos dispositivos de borda.

Dessa forma, verifica-se que, apesar nestes experimentos, ser obtido uma acurácia menor quando implementado o cálculo de entropia, pode-se entender que com aperfeiçoamentos futuros na implementação, será possível obter uma assertividade semelhante a de modelos que não possuem o cálculo de entropia, mas com uma melhor performance do modelo em relação a processamento computacional. De forma geral, é possível inferir que a metodologia aplicada, neste estudo de caso, é viável quando têm-se a intenção de melhorar o desempenho de um modelo de aprendizado de máquina federado sem interferir bruscamente em sua assertividade.

6 Conclusão

Este projeto teve como objetivo propor um estudo de caso de aprendizado de máquina federado com a inserção do cálculo de entropia dos dados dos dispositivos de borda como forma de otimizar o desempenho do modelo de deep learning. Esta proposta surgiu da necessidade dos sistemas atuais de aprendizado de máquina de melhorar desempenho, gerenciamento de recursos e governança dos dados de usuários, que tem conseguido evoluir conceitualmente com a criação do aprendizado de máquina federado,

mas que tem encontrado dificuldades de implementação quando encontram com as limitações de hardware e de comunicação dos dispositivos de borda.

Dessa forma, o objeto de estudo deste projeto buscou entender a relação do cálculo de entropia dos dados, a otimização dos tempos de execução de treinamento dos modelos de deep learning e a redução da quantidade de dados de entrada necessários para obter uma assertividade aceitável do modelo obtido. Assim, verificou-se que, para este estudo de caso, é verdade que o cálculo de entropia é capaz de otimizar o desempenho do aprendizado de máquina e assim, sendo necessário que mais estudos sejam realizados para validar que esta alternativa seja uma forma de possibilitar a implementação do aprendizado federado sem sobrecarregar os dispositivos dos usuários que serão responsáveis por realizar o treinamento dos modelos.

7 Referências

[1] L. Feng, Y. Zhao, S. Guo, X. Qiu, W. Li and P. Yu, "BAFL: A Blockchain-Based Asynchronous Federated Learning Framework," in IEEE Transactions on Computers, vol. 71, no. 5, pp. 1092-1103, 1 May 2022, doi: 10.1109/TC.2021.3072033.

[2] W. Xu, W. Fang, Y. Ding, M. Zou and N. Xiong, "Accelerating Federated Learning for IoT in Big Data Analytics With Pruning, Quantization and Selective Updating," in IEEE Access, vol. 9, pp. 38457-38466, 2021, doi: 10.1109/ACCESS.2021.3063291.

[3] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. 2020. A Survey on Distributed Machine Learning. ACM Comput. Surv. 53, 2, Article 30 (March 2021), 33 pages. https://doi.org/10.1145/3377454

[4] J.C.S. Anjos, N. Fonseca, L. Bittencourt, “Architecture and model to the edge intelligence”, Post-doc Research Report - IC/UNICAMP, São Paulo, March 2022, 31 pages.

[5] Jiang, Jichu & Kantarci, Burak & Oktug, Sema & Soyata, Tolga. (2020). Federated Learning in Smart City Sensing: Challenges and Opportunities. Sensors. 20. 6230. 10.3390/s20216230.

[6] N. Ferdinand, B. Gharachorloo and S. C. Draper, "Anytime Exploitation of Stragglers in Synchronous Stochastic Gradient Descent," 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), 2017, pp. 141-146, doi: 10.1109/ICMLA.2017.0-166.

[7] FERREIRA, Eduardo. Gradiente Descendente. Machine Learning para Cientista de Dados. Paraná. 1 p. Disponível em: http://cursos.leg.ufpr.br/. Acesso em: 30 mai. 2022

[8] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Asynchronous Stochastic Gradient Descent with delay compensation. In

Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17). JMLR.org, 4120–4129.

[9] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. 2021. When Machine Learning Meets Privacy: A Survey and Outlook. ACM Comput. Surv. 54, 2, Article 31 (March 2022), 36 pages. https://doi.org/10.1145/3436755

[10] Widi Hastomo et al 2021 J. Phys.: Conf. Ser. 1933 012050

[11] Tensorflow, TensorFlow Core v2.9.1, jun 2022. Disponível em: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy

[12] Data Science Academy. Deep Learning Book, 2022. Disponível em: <https://www.deeplearningbook.com.br/>. Acesso em: 15 Junho. 2022.

[13] CHOLLET, Francois. Deep Learning with Python. Manning Publications, f. 192, 2017. 384 p.

[14] Bishop, Christopher: Pattern Recognition and Machine Learning , 8th edition, Springer New York, Cambridge , 738, February 2011

[15] Russel, Stuart, Norving, Peter: Artificial Intelligence, A Modern Approach , 4th edition, Pearson Education Limited, 1066, 2021

[16] Goodfellow, Ian, Bengio, Yashua, Courville, Aaron: Deep Learning , 1st edition, MIT Press, 1–775, 2017

[17] Flare, "NVIDIA FLARE", Jun. 2022. Disponível em: https://developer.nvidia.com/flare

[18] Openstack, "WHAT IS OPENSTACK?", jun. 2022. Disponível em: https://www.openstack.org/software/

[19] Flower, "Flower A Friendly Federated Learning Framework", jun. 2022. Disponível em: https://flower.dev/

[20] Google Colabolatory, "Olá, este é o Colaboratory", jun. 2022. Disponível em: https://colab.research.google.com/?utm_source=scs-index

[21] Caldas, Sebastian, Peter Wu, Tian Li, Jakub Konecný, H. B. McMahan, Virginia Smith and Ameet S. Talwalkar. "LEAF: A Benchmark for Federated Settings." ArXiv abs/1812.01097 (2018): n. pag.

[22] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

[23] Biblioteca Keras, jun. 2022. Disponível em: https://www.tensorflow.org/guide/keras?hl=pt-br

[24] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V., "Federated Optimization in Heterogeneous Networks", 2018.

[25] School of information, UC Berkeley. "What Is Machine Learning (ML)?". 2020. Disponível em: olonline.berkeley.edu/blog/what-is-machine-learning/. Acesso em: 07 jul. 2022.

[26] Pandian, Shanthababu. “Machine Learning Process - Overview”. 2020. Disponível em: https://medium.com/analytics-vidhya/machine-learning-process-overview-1daa05c30150. Acesso em: 09 jul. 2022.

[27] Singh, Himanshu. “Why Precision-Recall? Why not Accuracy?”. 2019. Disponível em: https://medium.com/@himanshuit3036/why-precision-recall-why-not-accuracy-83349aa4c829. Acesso em: 09 jul. 2022.

[28] Ferdjouni, Meriem. “Overfitting vs. Data Leakage in Machine Learning”. 2021. Disponível em: https://medium.com/analytics-vidhya/overfitting-vs-data-leakage-in-machine-learning-ec59baa603e1. Acesso em: 09 jul. 2022.