Communication-Efficient Learning of Deep Networks from Decentralized Data

Um estudo do artigo e replicação dos resultados.

Pedro Tokar e Vitor Nascimento

https://github.com/vitor-n/federated-averaging

Contents

O artigo

Communication-Efficient Learning of Deep Networks from Decentralized Data

H. Brendan McMahan **Eider Moore** **Daniel Ramage** **Seth Hampson** **Blaise Agüera y Arcas**
Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

O *federated learning* é uma estratégia para treinar modelos de machine learning em cenários onde os dados de treinamento estão distribuídos em vários dispositivos e são potencialmente sensíveis. O artigo explora tais cenários e propõe uma forma de treinar um modelo de modo colaborativo sem a necessidade de centralizar os dados.

Motivação e background

Motivação

Dispositivos móveis possuem grande poder computacional e são altamente difundidos pelo mundo.[1]

- Em 2016, aproximadamente 3,7 bilhões smartphones estavam em uso;
- Projeções para 2025 apontam para 7,4 bilhões de smartphones ativos.

Estes dispositivos geram e tem acesso a uma grande variedade de dados, dos quais muitos podem ser utilizados para treinar modelos.

- Dados de sensores, como GPS, acelerômetros, giroscópios, etc.;
- Dados como fotos, áudios, textos, etc.

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

Além dos smartphones, que são os principais representantes destes tipos de dispositivos, diversos outros existem e são amplamente adotados:

- Smartwatches;
- Sensores IoT inteligentes;
- Veículos;
- Robôs atuadores em fábricas;
- Módulos de casa inteligente;
- E uma infinidade de outros tipos.

Motivação

Uma série de modelos poderiam se beneficiar destes dados:

- Classificadores de imagens poderiam ser treinados com fotos de galerias;
- Modelos de texto poderiam ser treinados com dados de digitação de usuários;
- Modelos para prever preferências de usuários se beneficiam de estatísticas de uso;
- Aplicações de monitoramento de saúde podem ser treinadas com dados de movimentação;
- etc.

Privacidade dos dados

Entretanto, a maioria destes dados são sensíveis:

- Dados como fotos da câmera, áudio de ligações e textos digitados, são pessoais e não devem ser compartilhados sem consentimento.
- Além de dados gerados, uma quantidade massiva de dados são coletados passivamente de usuários.
- Exemplos: dados de digitação, dados de navegação na internet, estatísticas de uso de aplicativos, etc.

Por mais que estes dados possam ser anonimizados, agentes maliciosos ainda podem usá-los para rastrear pessoas *(digital footprints)*.

Privacidade dos dados

Centralizar essa quantidade massiva de dados gerados também não é uma boa opção.

- Dados podem ser interceptados durante o upload para algum datacenter.
- Manter cópias dos dados em um datacenter aumenta o risco de vazamentos e acessos indevidos.
- Datasets, mesmo que anonimizados, podem ser usados para identificar pessoas[2]
- Manter todos esses dados em um só lugar requer uma grande quantidade de armazenamento.
- E muitas outras preocupações de privacidade...

[2] Latanya Sweeney. Simple demographics often identify people uniquely. 2000.

O cenário

Em resumo:

- Dados são gerados massivamente em bilhões de smartphones e dispositivos equipados com sensores.
- Tais dados são valiosos para o treinamento de diversos tipos de modelos.
- A maioria destes dados são sensíveis em relação a privacidade.
- Mesmo quando os dados não são sensíveis, pode não ser viável centralizá-los.

Solução proposta

O artigo introduz a abordagem do *Federated Learning* para lidar com esse cenário.

> Ideia principal: cada cliente auxilia treinando o modelo localmente, sem precisar fazer upload dos dados. Uma central então agrega os modelos treinados.

Essa abordagem segue o princípio de *focused collection*.[3]

[3] https://obamawhitehouse.archives.gov/sites/default/files/privacy-final.pdf

Contribuições do artigo

1. Identificar o treino em dados descentralizados de dispositivos móveis como um problema a ser resolvido;
2. Propor um algoritmo simples e direto para lidar com este cenário;
3. Avaliar o funcionamento do algoritmo com testes empíricos em datasets e aplicações variadas.

Federated Optimization

Federated Optimization

O problema de otimização, neste contexto, possui algumas características especiais as quais se é necessário atenção:

- **Dados não i.i.d**
- **Dados desbalanceados**
- **Comunicação limitada**[4]

[4] A velocidade de upload normalmente é muito menor que a de download: https://www.speedtest.net/global-index/united-states

Rounds de Comunicação

Chamamos de ***clientes*** os dispositivos que contém os datasets locais e de ***servidor*** a máquina central que armazena o modelo e faz requisições aos *clients*.

Assume-se um esquema de comunicação síncrono, feito com rounds:

- Há um conjunto de K clientes, cada um com seu dataset próprio.
- A cada round, escolhe-se uma fração C de clientes (aleatoriamente).
- Os clientes selecionados recebem os parâmetros do modelo global.
- Cada cliente otimiza o modelo localmente e envia os parâmetros para o servidor.
- O servidor atualiza o modelo global com base nos parâmetros recebidos de cada cliente.

Rounds de Comunicação

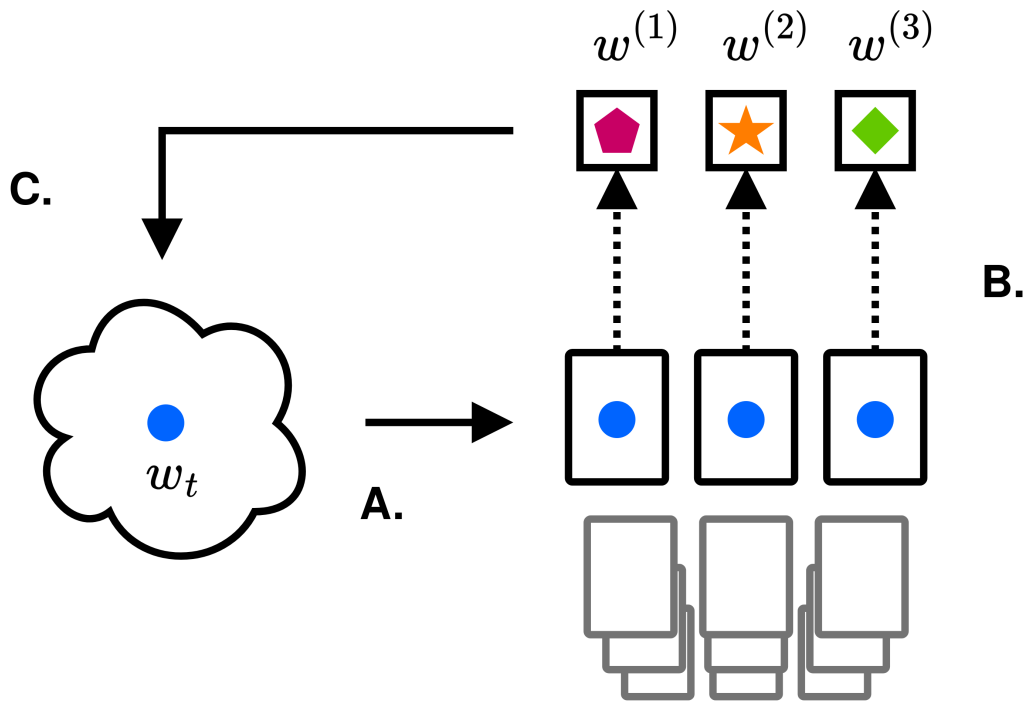


Figura 1: O servidor envia o estado do modelo global para os clientes selecionados (A). Os clientes otimizam o modelo com base no seu dataset local (B). Os parâmetros são agregados para atualizar o modelo global (C), depois disso o processo se repete.

Rounds de Comunicação

O treinamento descentralizado tem diferenças de custo do treinamento centralizado.

- **Comunicação mais custosa.**
- **Processamento menos custoso.**

Algoritmo

Relembrando

Em um problema geral de treinamento de uma rede neural, temos o seguinte problema de otimização:

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{onde} \quad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

Onde $f_i(w)$ é usualmente a perda da predição no dado (x_i, y_i).

Treinamento

Já em Federated Optimization, temos:

- Conjunto de dados do cliente k: $\mathcal{P}_k$
- Uma função de perda geral: $f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w)$, onde $n_k = |\mathcal{P}_k|$
- Uma função de perda para o cliente k: $F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w)$.
- Parâmetros do modelo global na rodada t: w_t

O treinamento do modelo nos clientes é feito utilizando SGD. O learning rate η é o compartilhado entre os clientes.

Idealmente teríamos $\mathbb{E}_{\mathcal{P}_k}[F_k(w)] = f(w)$. Mas note que F_k nem sempre será uma aproximação ótima de f, como no caso não i.i.d.

`FederatedSGD`: versão em que cada cliente computa o gradiente uma única vez por round de comunicação.

Cada cliente k computa $g_k = \nabla F_k(w_t)$ e o servidor agrega os gradientes para update:

$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$

Equivalentemente pode-se fazer:

$\forall k, \quad w_{t+1}^{(k)} \leftarrow w_t - \eta g_k$

E então, no servidor, ocorre a agregação $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^{(k)}$.

Treinamento

A primeira abordagem requer muitas rodadas de comunicação, já que cada passo de descida de gradiente requer a resposta dos clientes.

Para diminuir os rounds de comunicação e aproveitar melhor a capacidade computacional dos clientes, é possível fazer os clientes otimizarem os pesos diversas vezes antes de enviar para o servidor o estado.

$w^{(k)} \leftarrow w^{(k)} - \eta \nabla F_k\left(w^{(k)}\right)$.

Este método é chamado de `FederatedAveraging`.

Servidor

Servidor executa:
inicializa w_0
para cada round $t = 1, 2, \ldots$ **faça:**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (conjunto aleatório de m clientes)
    para cada cliente $k \in S_t$ **faça:**
        $w_{t+1}^k \leftarrow \mathbf{ClientUpdate}(k, w_t)$
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$

Cliente

$\mathbf{ClientUpdate}(k, w)$:
    $\mathcal{B} \leftarrow$ (divide $\mathcal{P}_k$ em lotes de tamanho B)
    para cada local epoch i **de** 1 até E **do:**
        para cada batch $b \in \mathcal{B}$ **faça:**
            $w \leftarrow w - \eta \nabla \ell(w; b)$
    retorna w ao servidor

Três parâmetros controlam a quantidade de computação feita em cada passo:

- C: Fração de quantos clientes são selecionados por round;
- E: Número de iterações que cada cliente faz por round;
- B: Tamanho do minibatch usado no cliente.

Nota-se que para $B = \infty$ e $E = 1$, o algoritmo é equivalente ao `FederatedSGD`.

Convergência

Resultados empíricos apontam que, para duas partições disjuntas de um mesmo dataset:

- Treinar dois modelos com **pesos** iniciais **diferentes** gera uma **má agregação**.
- Treinar dois modelos com **pesos** iniciais **iguais** gera uma **melhor agregação**.

O `FederatedAveraging` aproveita do segundo caso, uma vez que os clientes recebem o estado do modelo global antes de continuar o treino.

Convergência

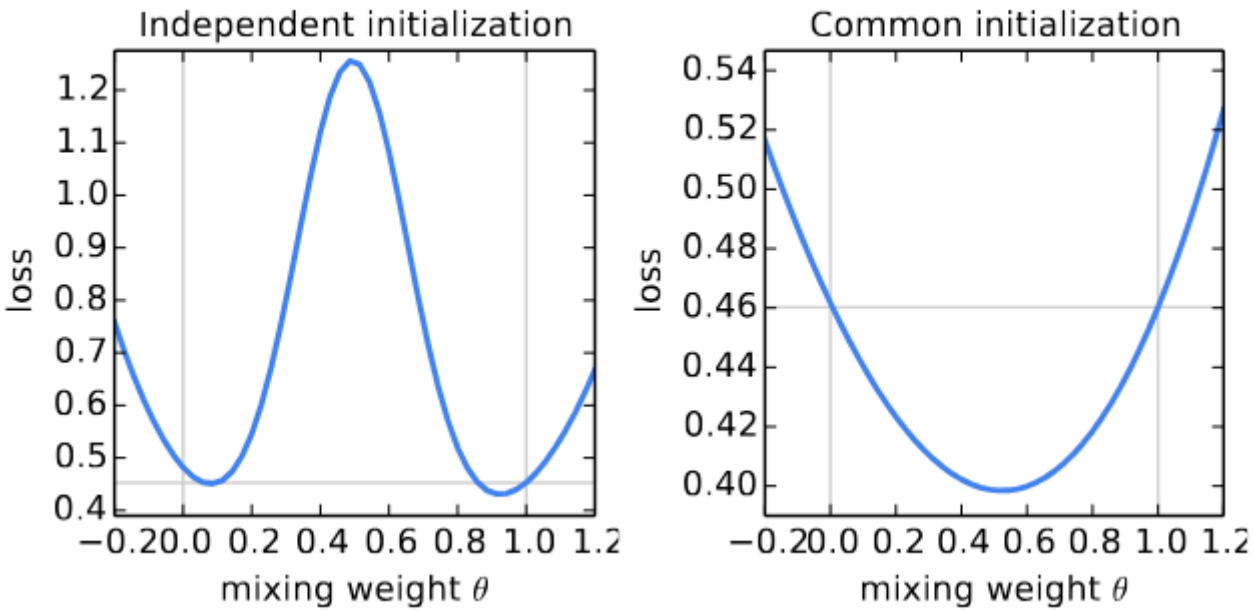


Figura 2: Testes realizados com o MNIST comparando duas estratégias: combinar modelos otimizados individualmente com e sem pesos iniciais iguais.[5]

[5] A mistura é feita da seguinte forma: $\theta w + (1 - \theta)w'$

Resultados experimentais

Setup dos experimentos

Para por em prova o funcionamento algoritmo, foram feitos testes usando o dataset MNIST e duas redes: um MLP com 2 camadas ocultas e uma CNN com dois layers convolucionais 5x5.

Duas configurações de distribuição de dados existiram:

- Uma distribuição i.i.d entre os clientes.
- Uma distribuição não i.i.d em que os clientes recebiam apenas 1 ou 2 dígitos em seus datasets.

Foram usados 100 clientes para este experimento.

Os modelos foram treinados com alguns valores de learning rate diferentes, e os resultados exibidos correspondem aos que apresentaram melhores treinos.

Aumentando o paralelismo

2NN	IID		Non-IID	
C	$B = \infty$	$B = 10$	$B = \infty$	$B = 10$
0.0	1455	316	4278	3275
0.1	1474 (1.0×)	87 (3.6×)	1796 (2.4×)	664 (4.9×)
0.2	1658 (0.9×)	77 (4.1×)	1528 (2.8×)	619 (5.3×)
0.5	— (—)	75 (4.2×)	— (—)	443 (7.4×)
1.0	— (—)	70 (4.5×)	— (—)	380 (8.6×)
CNN, $E = 5$				
0.0	387	50	1181	956
0.1	339 (1.1×)	18 (2.8×)	1100 (1.1×)	206 (4.6×)
0.2	337 (1.1×)	18 (2.8×)	978 (1.2×)	200 (4.8×)
0.5	164 (2.4×)	18 (2.8×)	1067 (1.1×)	261 (3.7×)
1.0	246 (1.6×)	16 (3.1×)	— (—)	97 (9.9×)

Tabela 1: Rounds necessários para alcançar uma acurácia de 97% para o MLP e 99% para a CNN de acurácia no MNIST.

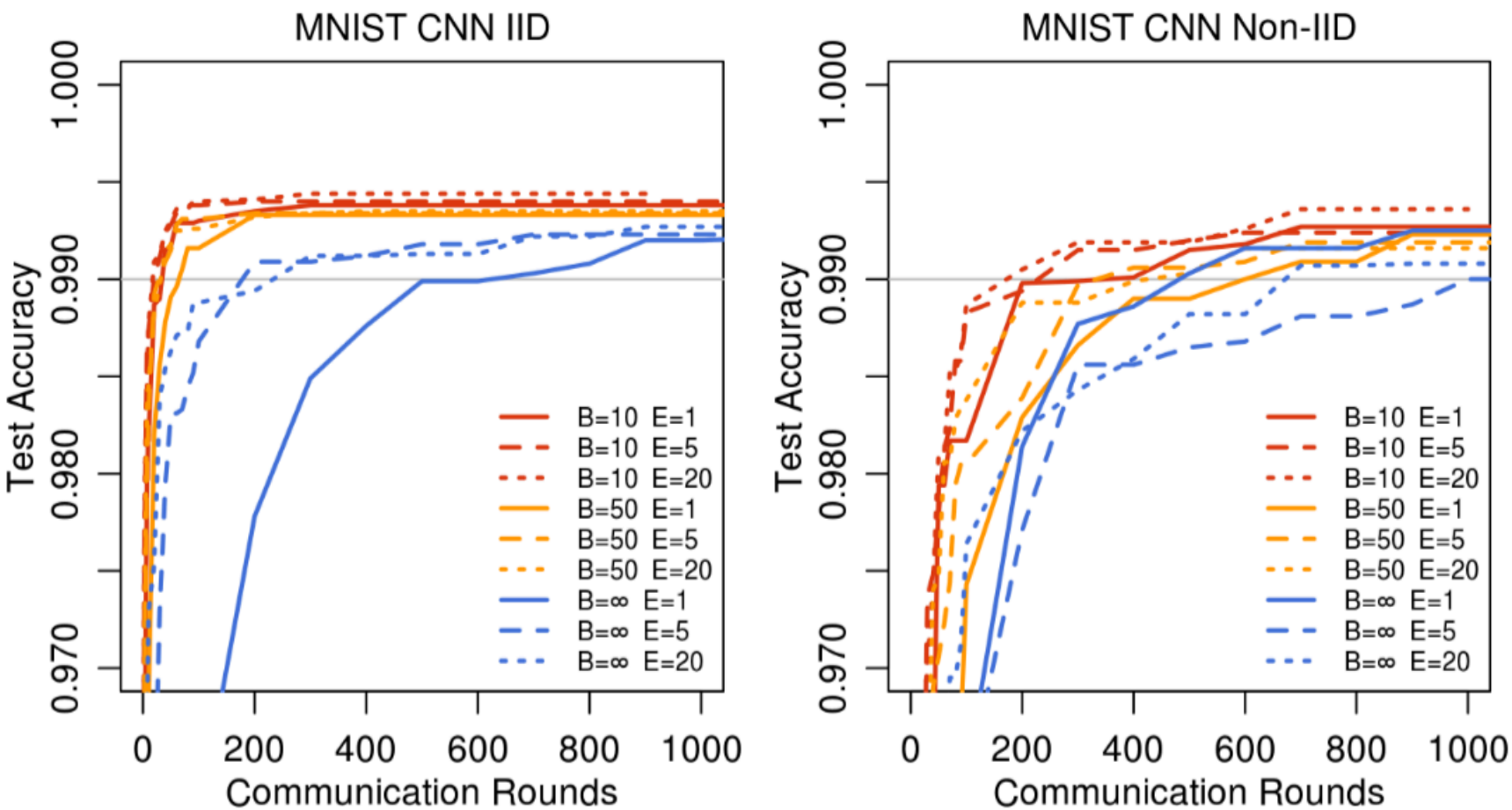
Aumentando a computação

MNIST CNN, 99% ACCURACY					
CNN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	626	483
FEDAVG	5	∞	5	179 (3.5×)	1000 (0.5×)
FEDAVG	1	50	12	65 (9.6×)	600 (0.8×)
FEDAVG	20	∞	20	234 (2.7×)	672 (0.7×)
FEDAVG	1	10	60	34 (18.4×)	350 (1.4×)
FEDAVG	5	50	60	29 (21.6×)	334 (1.4×)
FEDAVG	20	50	240	32 (19.6×)	426 (1.1×)
FEDAVG	5	10	300	20 (31.3×)	229 (2.1×)
FEDAVG	20	10	1200	18 (34.8×)	173 (2.8×)

Tabela 2: Rounds necessários para alcançar 99% de acurácia no MNIST com c = 0.1.[6]

[6] u é a quantidade estimada de descidas do gradiente por rodada.

Aumentando a computação

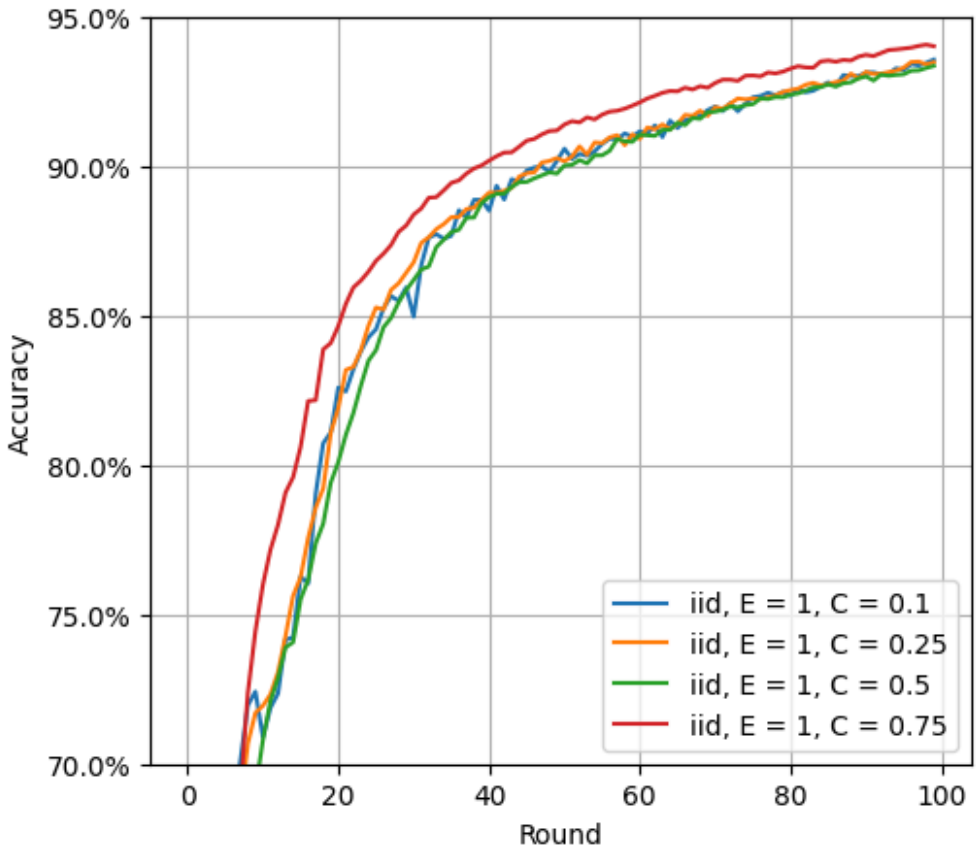


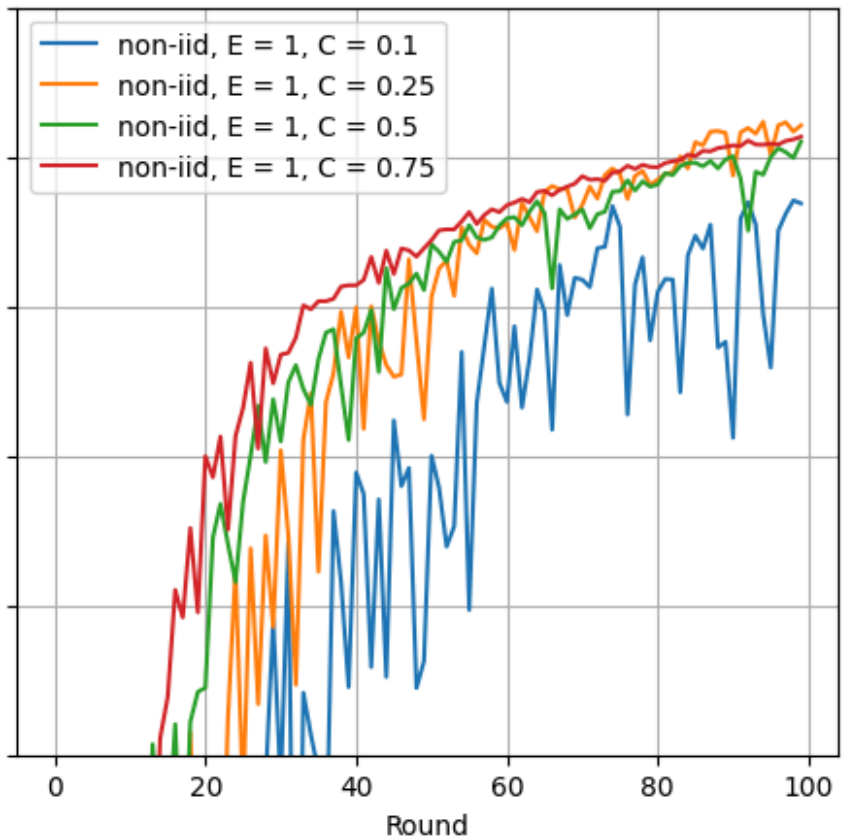
Observações

- Os experimentos com o MNIST confirmaram que a combinação de modelos treinados em dígitos diferentes gera um modelo que converge.
- Apesar de aumentar a fração C também melhorar os resultados, valores mais baixos já trazem resultados vantajosos e não requerem excesso de comunicação.
- Aumentar a quantidade de computações feitas nos clientes é vantajoso, e os valores de B e E devem ser balanceados de acordo com as capacidades dos clientes.

Hipótese levantada: fazer a média dos modelos traz um efeito de regularização similar a um *dropout*.

Experimento próprio





Setup dos experimentos 2

Além de redes classificadoras com o MNIST, também foram conduzidos testes com um modelo de linguagem LSTM treinado na obra completa de Willian Shakespeare.

Duas configurações de partição do dataset foram usadas:

- Uma distribuição i.i.d entre os clientes.
- Uma distribuição na qual cada cliente tinha as falas de uma personagem diferente.

Foram usados 1146 clientes para este experimento.

A segunda configuração, além de não ser i.i.d, também é desbalanceada, já que alguns personagens tem mais falas do que outros.

Experimentação com Shakespeare

SHAKESPEARE LSTM, 54% ACCURACY					
LSTM	E	B	u	IID	NON-IID
FEDSGD	1	∞	1.0	2488	3906
FEDAVG	1	50	1.5	1635 (1.5×)	549 (7.1×)
FEDAVG	5	∞	5.0	613 (4.1×)	597 (6.5×)
FEDAVG	1	10	7.4	460 (5.4×)	164 (23.8×)
FEDAVG	5	50	7.4	401 (6.2×)	152 (25.7×)
FEDAVG	5	10	37.1	192 (13.0×)	41 (95.3×)

Tabela 3: Rounds necessários para alcançar 54% de acurácia em Shakespeare com C = 0.1.

Conclusões e importância

Conclusão

Os experimentos empíricos mostram que o Federated Learning pode ser usado em cenários práticos, uma vez que o algoritmo foi capaz de treinar modelos de alta qualidade e precisão.

Também foi mostrado que esse sucesso de treino não se restringe a um único tipo de modelo, já que tanto classificadores quanto modelos de linguagem foram treinados.

Aplicações práticas

- Modelo de predição da próxima palavra do teclado Gboard;
- Sensoreamento de poluição do ar por meio de dispositivos IoT;
- Avaliação de modelos médicos com dados de diferentes clínicas e hospitais;
- Treinamento de robôs para se localizarem corretamente em novos ambientes;
- Treinamento de modelos de detecção de obstáculos para carros autônomos;
- Entre muitas outras.

Variações

- FedDyn: mais robusto à divergência das distribuições dos datasets locais, utiliza uma regularização dinâmica à cada round para facilitar a convergência;
- FCL: combina federated learning com técnicas de compressão de dados e com intermediários entre os clientes e os servidores;
- LFRL: combina técnicas de reinforcement learning com federated learning para garantir modelos que consigam "guardar conhecimento" mesmo após múltiplos treinos.

Obrigado!

Referências

- McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. a. Y. (2016, February 17). Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv.org. https://arxiv.org/abs/1602.05629

Links para as aplicações práticas

- Federated Learning for Mobile Keyboard Prediction
- [...] Prediction in Smart City Sensing Applications
- Federated benchmarking of medical artificial intelligence with MedPerf
- [...] A Learning Architecture for Navigation in Cloud Robotic Systems
- Federated Learning in Vehicular Networks