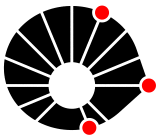

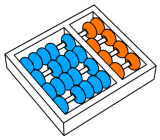

Classificação de Texto baseado em LLM e em Aprendizado Federado

M. Previti

Relatório Técnico - IC-PFG-23-25
Projeto Final de Graduação
2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Classificação de Texto baseado em LLM e em Aprendizado Federado

Marcelo Salles Previti

Resumo

Este trabalho busca implementar e comparar diferentes tipos de modelos de neurais e técnicas de treinamento. Para tal, comparou-se a performance de dois modelos, um baseado na arquitetura perceptron e outro baseado no LLM GPT2, ambos treinados de maneira centralizada e federada. Para fazer a avaliação, utilizou-se da métrica de precisão sobre o dataset "agnews", que contém notícias de diferentes segmentos, como tecnologia e esportes.

1 Introdução

O uso de inteligência artificial vem crescendo de maneira exponencial nos últimos nos mais diversos campos, como na áreas de visão computacional, previsão de demanda e processamento de linguagem natural.

Um dos problemas clássicos dessa área seria na classificação de texto. Por exemplo, um determinado empreendimento possui uma série de avaliações escritas por clientes e o mesmo gostaria de saber se a maioria possui caráter positivo ou negativo.

Técnicas mais "tradicionais" de computação possuem grande dificuldade em realizar esta tarefa, já que existe uma grande falta de padrão nos textos: cada avaliação possui uma quantidade indeterminada de diferentes combinações de palavras, sem contar em problemas de erro de ortografia e contexto (uma palavra possui diferente semântica dependendo do contexto).

Nesse cenário, algoritmos de aprendizado de máquina tem se destacado para a tarefa[3]. Ao serem treinados em um conjunto de dados específicos, se faz possível inferir o que aquele conjunto de palavras significa naquele contexto.

Um problema inerente a esse tipo de tarefa é em relação a privacidade dos dados. Muitos contextos de empresas que precisam realizar esse tipo de tarefa possuem dados sigilosos, que não podem ser exportados para a nuvem. Ainda assim, mesmo que o treinamento fosse feito de maneira "on-premise", muitas vezes isso não garante privacidade o suficiente: clientes com interesses diferentes podem ter os dados concentrados em um mesmo local, o que gera uma brecha de segurança importante

Nesse contexto, o aprendizado federado [4] surge como uma alternativa interessante. Ao invés da abordagem tradicional onde os dados são acumulados e assim utilizados para treinar um modelo, um modelo inicial é exportado para determinada quantidade nós. Cada nó continuará seu treinamento localmente e eventualmente ocorrerá uma agregação dos

modelos de cada nó (mais detalhes serão abordados no capítulo 2). A grande vantagem dessa abordagem é que o conjunto de treino será mantido localmente nas máquinas em questão, entretanto, um maior uso de recursos de rede é necessário.

Para a realização deste trabalho então, será feito uma análise compartiva entre modelos treinados de maneiras centralizada e federada, para assim possuir uma dimensão do tamanho da perda de eficiência do modelo. Para tal, criou-se um modelo baseado na arquitetura perceptron [6], no qual ele foi treinado em ambas as técnias. O mesmo processo se repetiu para o fine-tuning de uma variação do LLM GPT2 [7]. Por fim, também comparou-se o uso de rede para os modelos treinados de forma federada.

2 Conceitos Relacionados

Antes de abordar o experimento em si, segue uma seção onde apresentaremos conceitos que serão utilizados mais a frente

2.1 Redes Neurais

Redes Neurais são modelos computacionais gerados a partir de técnicas de aprendizado de máquina. Sua origem vem da década de xx, no qual a ideia era simular o funcionamento do cérebro humano.

O cérebro é composto por células especializadas chamadas de neurônio. Esses neurônios são ligados entre si por estruturas celulares chamadas de dendritos e axônios. O funcionamente geral seria de que um sinal químico, proveniente de um estímulo, é passado para um neurônio através de seu dendrito. Em seguida, o sinal é transmitido para o corpo celular, onde ocorreram processos químicos. O resultado desses processos é passado para o próximo neurônio através dos axônios, que se ligam nos dentritos de outro axônio e assim sucessivamente.

Esse comportamento é também simulado na redes neurais. A fase inicial de estímulo é gerada através do input de dados do modelo. Na computação, o chamado neurônio na verdade mais se assemelha a estrutura do corpo celular, no qual uma vez que recebe a informação, um determinado processamneto é feito naquele sinal. Esse processamento é chamado de função de ativação, como SeLU e ReLU. Para transferir os pesos de um neurônio para o outro, existe um fator multiplicativo para a informação gerada no neurônio comutacional. Treinar um modelo de rede neural é encontrar o conjunto de pesos para as ligações do neurônio que otimizem a tarefa em questão.

Embora tenha sido descrito o comportamento de um único neurônio, redes neurais são compostas por uma série de aglomerações de neurônios. Podemos chamar essas aglomerações de camadas que, por sua vez, estão conectadas a uma série de outras camadas, ou as vezes, ela mesma.

Pode-se destacar 3 principais tipos de camadas: camada de entrada, camadas escondidas, camada de saída. A camada de entrada refere-se ao input do modelo que, na nossa analogia, recebe o estímulo. As camadas escondidas ou internas são o conjunto de camadas ligadas, onde ocorrerá o processamento da tarefa. E por fim, a camada de saída é onde se encontrará o resultado gerado pelo modelo.

2.2 Modelo Perceptron

Pode-se definir como a arquitetura de um modelo de rede neural como a maneira pelo qual é construído. Ou seja, a quantidade de neurônios e a forma pelas quais são ligados definem sua arquitetura.

Neste contexto, uma das arquiteturas mais famosas de redes neurais é referente ao modelo perceptron. Ela é caracteriza pela ligação entre os neurônios. Cada neurônio de uma camada é ligado a todos os outros neurônios da camada seguinte. Este foi o primeiro modelo de rede neural concebida, inspirada no modelo biológico de neurônios e foi utilizada como modelo base da aplicação

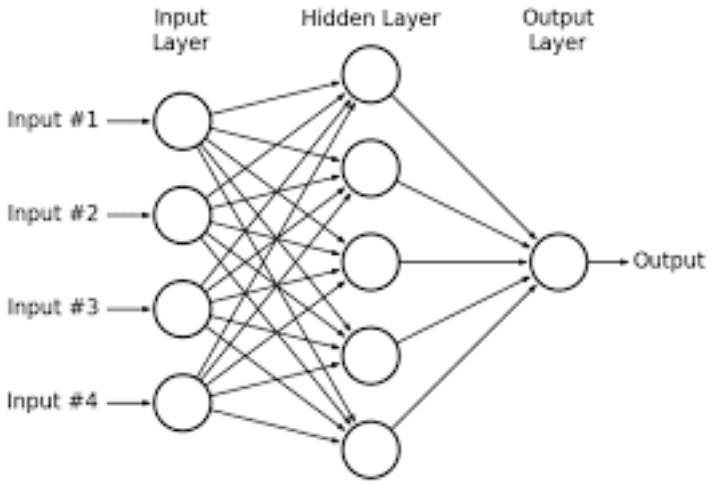

Figura 1: Exemplo genérico de perceptron

2.3 Otimizadores

Como dito anteriormente, treinar uma rede neural é o processo de encontrar os pesos das arestas que ligam os neurônios, com o intuito de maximizara eficiência para o problema em questão - como por exemplo, aumentar a acurácia de um modelo que faz identificação de imagem.

Para a realização desse processo, define-se uma função de perda (loss), no qual ela expressará a discrepância entre os valores preditos pelo modelo e os valores reais. Assim, o treinamento consistirá em encontrar o arranjo de parâmetros que minimizem a perda.

Neste contexto, surgem os otimizadores de redes neurais. Otimizadores são os algoritmos pelos quais o modelo encontrará esses pesos. Para realizar esse processo, uma quantidade de dados de entrada se faz necessário, no qual o modelo tentará generalizar para casos antes não vistos. O otimizador mais básico é a descida do gradiente [1].

Primeiramente ocorre a inicialização dos pesos do modelo. Embora existam várias estratégias, uma das mais utilizadas é da inicialização aleatória. Conforme uma determinada quantidade de samples de treino alimentam o modelo, será calculada as derivadas parciais

da perda (também denominada função de custo). Dessa forma, conseguimos descobrir como mudar os pesos do modelo de tal forma que a função é minimizada.

O tamanho da atualização será determinada por uma variável chamada learning rate. Quanto maior, mais rápido são feitos as atualizações, entretanto, mais instável fica o processo. Quanto menor, mais estável porém mais lento.

Com o desenvolvimento das redes neurais, o volume de dados e a complexidade aumentaram muito, o que tornou a utilização da descida do gradiente lenta. Assim, surgiram variações dela [2], como RMSProp, Adam e Nadam. Seus mecanismos base são de fazer updates mais eficientes na hora de atualizar os parâmetros do modelo, tentando direcioná-lo o quanto antes para o mínimo global.

2.4 Aprendizado por transferência

Treinar um modelo de rede neural novo normalmente é um desafio. Dentre os desafios, podemos destacar: encontrar uma base de dados suficientemente grande, garantir que ela possua qualidade e o alto custo do treinamento (como o gasto de energia elétrica e de gpus, no qual embora possa ser feito o sem, aceleram muito).

Dessa forma, o aprendizado por transferência[8] aparece como uma ótima alternativa para reaproveitar modelos pré-treinados, o que reduz não só o tempo gasto para o treinamento. mas também a quantidade de dados necessária.

O conceito base dessa técnica é de que as camadas internas das redes neurais fazem processamentos que extraem informações dos dados inputados. Camadas mais próximas da camada de input extraem features mais gerais dos dados, enquanto camadas mais próximas às camadas de saída farão processamentos mais voltados para a tarefa específica.

Dessa forma, supondo que eu queira treinar um modelo e possuo outro que faça uma tarefa similar. Por exemplo, quero fazer um classificador de imagem de raças de cachorro e possuo um classificador geral de imagens. O processos iniciais de extração de features das imagens do modelo já treinado provavelmente serão úteis na minha aplicação também.

Assim, o que pode ser feito é refazer o treino, imitando a arquitetura do modelo antigo e inicializando os parâmetros da mesma forma. Entretanto, a ideia aqui é não atualizar as camadas mais próximas a camada de input, congelando-as. Assim, a necessidade de dados e de tempo de treino reduz bastante.

2.5 Aprendizado Federado

Até então, toda a discussão feita sobre redes neurais partia de um pressuposto da centralização do poder computacional e dos dados disponíveis. Entretanto, muitas vezes no mundo real esses pressupostos acabam sendo um grande empecilho.

Um grande problema da centralização dos dados está relacionado à privacidade. Em contextos sigilosos enviar dados para um servidor central pode acarretar em riscos de segurança.

Mesmo que ocorra medidas de segurança para o envio dos dados através da nuvem, a própria concentração de dados em um servidor central pode acarretar em riscos. Suponha uma instituição financeira. Naturalmente, dados sigilosos de empresas rivais acabam sobre

a custódia de uma instituição desse ramo. Centralizar os dados em um único local poderia gerar falhas de segurança, como vazamento de informações.

Outra dificuldade é em relação à infraestrutura legada. Muitas vezes, gerar uma base central com alto poder computacional demanda investimentos que muitas instituições não possuem, Entretanto, a mesma pode possuir uma infraestrutura formada por uma série de máquinas de menor porte. Seria interessante poder reaproveitar essa infraestrutura.

A dependência de um servidor central também é um risco por si só. Caso ocorra algum problema com ele, o fluxo inteiro de trabalho é interrompido e, no pior dos casos, o trabalho inteiro é perdido.

Nesse cenário, o aprendizado federado se mostra uma alternativa capaz de atacar vários desses problemas. O aprendizado federado consiste em um servidor centralizado no qual exportará o modelo para uma série de nós. Esses nós utilizarão esse modelo para realizar o treino em um conjunto de dados local.

Em seguida, a ideia é que ocorra uma agregação entre os modelos locais. Essa agregação normalmente é feita exportando parâmetros para o servidor, que por sua vez fará a agregação dos mesmos. A maneira como será feita e quais parâmetros serão exportados varia conforme o contexto. O processo se repete de maneira iterativa até a convergência.

Com essa abordagem, os dados são mantidos de forma local, sendo exportados apenas parâmetros do modelo (o que garante maior segurança). Além disso, o processamento é feito de maneira distribuída (o que reaproveita a infraestrutura pré-estabelecida) e ocorre replicabilidade do treino: caso o servidor central caia, modelos estarão disponíveis em uma série de outras máquinas.

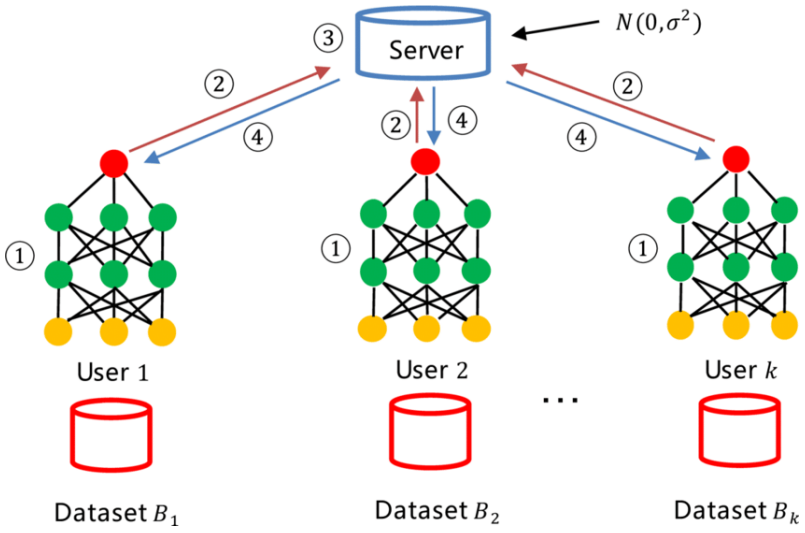

Figura 2: Exemplo visual de uma aplicação federada

Entretanto, essa modalidade não apresenta apenas vantagens. O processo demanda uma capacidade de rede para seu funcionamento, algo desnecessário anteriormente.

Somado a esse fator, existe uma preocupação em manter a qualidade dos dados em uma série de nós, o que antes era feita de forma unificada.

Pela natureza descentralizada do treino, esse trabalho visa investigar o impacto dessa forma de treinamento em relação à performance do modelo, tentando observar o impacto que essa modalidade pode causar.

2.6 FedAvg

Nesta seção iremos explicitar o algoritmo de agregação de modelos usado no trabalho: o FedAvg[5].

Após o processamento de treino local em cada um nós, os pesos do modelo são enviados para um servidor central. Esses modelos são unificados através de uma média ponderada. A escolha da ponderação pode variar conforme o contexto. Um critério pode ser, por exemplo, tamanho do dataset pelo qual aquele modelo foi treinado.

Por fim, após a agregação, o servidor central retorna o novo modelo global e assim se inicia uma nova rodada de treinamento, repetindo o processo.

2.7 Large Language Models (LLM)

O termo LLM ganhou notório destaque no ano de 2023, após a popularização do produto disponibilizado pela empresa OPENAI denominado chatGPT. O produto em si é um chatbot de propósito geral (respondo perguntas das mais diversas áreas de conhecimento) baseado em inteligência artificial.O coração por trás da aplicação são os LLMs da família GPT.

Um LLM é uma rede neural de processamento de linguagem natural (normalmente baseada na arquitetura transformers) no qual foi treinada em uma quantidade massiva de dados, com uma arquitetura extremamente complexa. Esses modelos tem ganhado força nos últimos anos pelo aumento da capacidade de processamento e pela grande disponibilidade de dados. Esses modelos tentam prever qual a palavra seguinte correta em uma frase incompleta.

2.8 GPT2

Para este trabalho, utilizou-se o GPT2 como LLM. O GPT2 é um LLM lançado pela OPENAI em 2019. O mesmo é baseado na arquitetura transformers. Para este trabalho, optou-se por utilizar a versão mais simples do modelo.

Essa versão (denominada "small") possui tamanho de vetor de embedding de 768 dimensões. Forma utilizadas 12 camadas internas, cuja função de ativação é a "ReLU".

3 Metodologia

3.1 Conjunto de dados

Para realizar o treinamento dos modelos propostos, utilizou-se o conjunto de dados agnews. Ele foi selecionado por ser um dataset referência para tarefas de classificação de texto. Possui alta qualidade e uma quantidade razoável de samples para fazer o treinamento

O conjunto de dados baseia-se em uma série de notícias em inglês, no qual estão classificadas em um dos quatros estados: notícia de esportes, ciência/tecnologia, mundo e negócios.

Para todos os modelos, o conjunto de treino possuía 120000 samples e para o conjunto de teste, 7600 samples. O mesmo treino e teste foi utilizado para todos os modelos. A distribuição das classes no dataset é igual para todas as classes, ou seja, cada uma possui 30000 samples para o conjunto de treino e 1900 para o conjunto de teste.

O pré-processamento utilizado para os modelos criados (mais simples) seguiu duas etapas. O texto corrido foi passado em um tokenizador da biblioteca pytorch ("basic english"). Em seguida, os tokens são mapeados para índices, criando-se assim um vocabulário do dataset, utilizando-se também da função padrão da biblioteca. Esses índices alimentarão o modelo.

3.2 Avaliação

Para a avaliação dos resultados, utilizou-se duas métricas principais: a acurácia e o custo de comunicação.

3.2.1 Acurácia

A acurácia é a medida estatística no qual mede-se a precisão das classificações geradas pelo modelo, ou seja, trata-se da capacidade do modelo em prever as classificações pelo qual foi treinado. Ela é obtida através da razão do número de acertos feito pelo modelo pelo número total de previsões.

3.2.2 Custo de comunicação

O custo de comunicação é a grandeza no qual se metrifica a quantidade de recursos de rede necessários para a realização da aplicação (no caso, do aprendizado federado). Quanto maior seu valor, maior a necessidade de sistemas mais robustos de rede. Ela é obtida através da multiplicação do número total de bytes necessários para representar um modelo (proveniente dos pesos apreendidos) pela quantidade de vezes que esses dados trafegam pela rede.

3.3 Modelos

O modelo base para a análise foi um perceptron também gerado a partir da biblioteca pytorch. Ele é composto por uma camada inicial de embedding, conectado com uma camada interna completamente ligada de 12 neurônios, utilizando-se da função de ativação "Log-Softmax". Por fim, uma camada de saída de quatro neurônios é utilizada, onde cada um corresponde a uma classe e a previsão do modelo é daquele neurônio de maior saída -ou seja, de maior confiança do modelo. Para regularização do modelo, aplicou-se dropout.

Para o treino destes modelos, utilizou-se o otimizador "Adam"com learning rate de 0.0005. Adotou-se treino de 20 épocas com tamanho de batch de 64.

Após a criação do modelo, o mesmo foi replicado para o contexto de aprendizado federado. Para tal, utilizou-se a biblioteca flower. Essa biblioteca permite o treino de diferentes instâncias do modelo em ambientes isolados, a partir de modelos pytorch. Após o treino de

determinada quantidade de épocas, exportam-se os parâmetros e eles são unificados baseados no algoritmo "FedAvg". Utilizou-se uma quantidade de 10 clientes para fazer o treino, separando os dados de treino de maneira igual em cada um dos nós (3000 samples). As divisão foi feita de forma aleatória.

Em seguida, utilizou-se o modelo do LLM GPT2 mais simples (ou seja, com menor quantidade de parâmetros, que no caso foi de 124 milhões), proveniente da biblioteca transformers disponibilizada pela comunidade do Hugging Face.

Originalmente, o GPT2 é um modelo que, a partir de um input de texto, tenta prever qual a próxima palavra. Entretanto, para a classificação de texto, utilizou-se uma variante do mesmo no qual ao invés de possuir um único token de saída, gera-se uma quantidade de n neurônios (representando as classes a serem preditas).

Uma vez com o modelo em mãos, realizou-se o procedimento de transfer learning, no qual a ideia era reaproveitar todo o processamento e treino em texto que havia feito antes porém direcioná-los para o nosso conjunto específico de dados. Mudando apenas as camadas finais do modelo.

O treinamento foi realizado durante quatro épocas. Utilizou-se também uma learning rate linear variável, utilizando-se do otimizador "Adamw". Também adotou-se a técnica de "gradient clipping"para evitar o problema de exploding gradients.

Para o pré-processamento, repetiu-se o passo anterior de tokenização, porém utilizando-se do tokenizador padrão do GPT2, disponibilizado também pela comunidade do Hugging Face.

Utilizou-se também a biblioteca flower para realizar o treinamento federado do LLM em questão. Para tal, o modelo foi adaptado para uma classe de pytorch para poder ser utilizado pela biblioteca. Os parâmetros do modelo foram mantidos em relação ao centralizado e os parâmetros do treinamento federado também se mantiveram iguais em relação ao modelo menor.

4 Resultados

Após a execução do programa, obtivemos os seguintes resultados, que estão resumidos na tabela 1:

Perceptron	Perceptron Federado	GPT	GPT Federado
0.9	0.87	0.94	0.92

Tabela 1: Resultado final do conjunto teste da acurácia

O menor modelo treinado de maneira centralizada obteve acurácia de 0.9 no conjunto teste. O modelo equivalente, porém treinado de forma federada teve acurácia de 0.85. O GPT2 treinado de forma centralizada possuiu a maior acurácia, com 0.94. Por fim, o equivalente treinado de maneira federada teve 0.92.

Para uma melhor visualização da evolução do treino conforme as as rodadas de treinamento federado, foi plotado um gráfico presente na figura 3 com a acurácia dos modelos em cada uma dessas etapas:

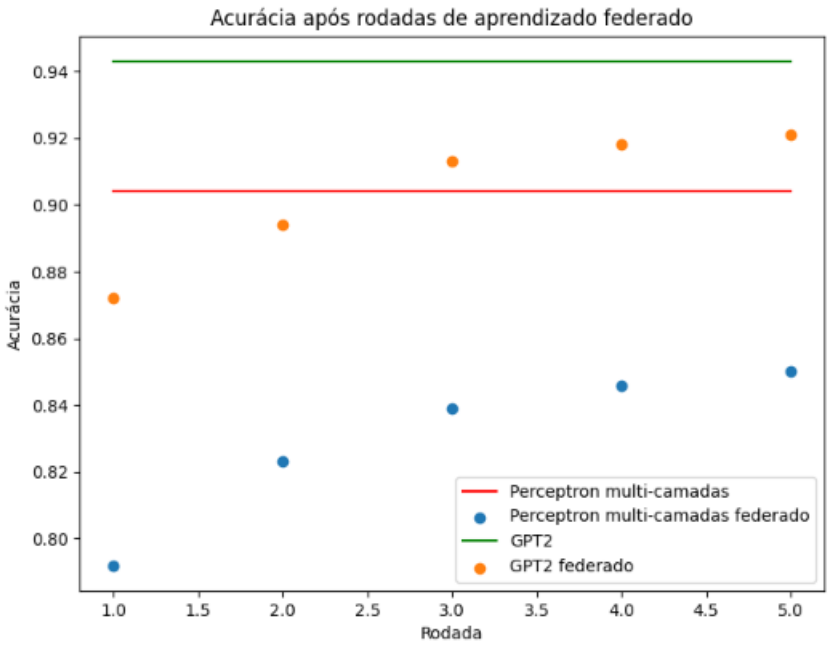

Figura 3: Evolução da acurácia dos modelos em relação as rodadas de treino federado

Quanto ao tamanho dos modelos em bytes, segue na tabela 2:

Perceptron	GPT
45997600bytes	497771520bytes

Tabela 2: Resultados referente ao custo de comunicação

Com o tamanho dos modelos, pode-se obter o uso da rede. Como ambos foram treinados em 5 rounds, ou seja, 5 agregações dos modelos no servidor central, basta multiplicar o tamanho em bytes por 10 (visto que o trafego ocorre do servidor até os nós e dos nós até o servidor).

Perceptron	GPT
459976000bytes	4977715200bytes

Tabela 3: Caption

5 Conclusões

Com os resultados gerados, pode-se concluir aquilo já esperado antes da realização do experimento: modelos mais complexos com maior capacidade de entender o contexto das frases performaram melhor do que modelos mais simples.

A diferença de performance para as duas abordagens não foi muito grande, o que pode indiciar que os fatores como tamanho do modelo e técnica de aprendizado devem ser avaliados em cada contexto.

Contudo, podemos notar que a diferença no uso de rede, comparativamente, foi grande. O uso federado do GPT2 utilizou-se de praticamente dez vezes o recurso em comparação ao menor modelo.

Assim, na hora de escolher qual abordagem utilizar, podemos descrever três principais fatores: necessidade de privacidade, disponibilidade de conexão, acurácia necessária, quantidade de nós disponíveis para treino e poder computacional por nó.

Quanto maior for a necessidade da aplicação por performance, deve-se aumentar a prioridade em relação ao treinamento centralizado. Quanto maior a necessidade de privacidade daqueles dados, mais inclinado ao treinamento no estilo federado.

Caso a aplicação possua poucas ou apenas uma unidade de treinamento, a modalidade centralizada também aparece naturalmente, entretanto, se a aplicação possuir vários nós com pouca capacidade computacional - como por exemplo uma empresa - o federado se torna bem interessante.

O último fator a ser considerado é a disponibilidade de rede. Caso a conexão entre os nós não seja suficientemente boa, isso pode dificultar a implementação federada.

Referências

[1] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[2] Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

[3] Mita K Dalal and Mukesh A Zaveri. Automatic text classification: a technical review. *International Journal of Computer Applications*, 28(2):37–40, 2011.

[4] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[5] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[6] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[7] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[8] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.