Treinamento Colaborativo

Mateus Oliveira e Patrick

Dezembro de 2022

Abstract

O objetivo deste trabalho foi propor um modelo de treinamento distribuído entre máquinas, o qual chamamos de treinamento colaborativo. Para verificação de desempenho do mesmo, foi utilizado um modelo GPT-2 pretreinado, no qual é realizado um ajuste fino em múltiplas GPUs simultaneamente, de forma a agilizar o tempo de treinamento e permitir a utlização de batchs maiores por agregar a memória disponível em cada uma das máquinas. A técnica consiste em sincronizar os pesos dos modelos distribuídos geograficamente a cada N interações. Repositório do projeto.

1 Introdução

Tarefas de processamento de linguagem natural tem como vantagem a escalabilidade, quanto maior for o *dataset*, modelo e poder de processamento, melhor será a assertividade do modelo. Modelos treinados em grande escala como *GPT-3* com 175 bilhões de parâmetros e *Palm* com 540 bilhões de parâmetros, requer um hardware de ponta, que infelizmente é indisponível para muitos pesquisadores. Existem técnicas que permitem o uso deses modelos que são: descarregamento de RAMs ou APIs hospedadas. Acontece que essas técnicas tem limitações como o descarregamento lento para inferência iterativa e o uso de APIs é insuficiente para pesquisas. Por conta disso, treinaremos o modelo *GPT-2* com *dataset* divido em duas máquinas separadas geograficamente, e avaliaremos o tempo de treinamento para convergência e o *trade-off* entre troca de mensagens a cada N *iterações*. Ademais, o treinamento é realizado de maneira paralela entre as diferentes máquinas, o que permite a minimização do tempo ocioso em comparação a técnicas seriais[4].

2 Base de dados

O IMDB um conjunto de dados para classificação de sentimento binário contendo um conjunto de 25.000 críticas de filmes altamente polarizadas para treinamento e 25.000 para teste [5]. Nesse caso, Dividiremos o treinamento em duas partes 20.000 amostras para treino e 5.000 amostras para validação e

25.000 amostras para teste. A tabela 1 exemplifica os textos disponíveis para treinamento. Como treinaremos um modelo gerador do tipo GPT-2, pegará apenas a primeira coluna que é o texto.

Text IMDB	setiment
One of the other reviewers has mentioned that after...	positive
A wonderful little production. The filming technique...	positive
Basically there's a family where a little boy (Jake)...	negative

Table 1: A tabela mostra exemplos de textos presentes no dataset IMDB

3 Métricas

Foi adotada a métrica perplexidade como avaliação da qualidade da geração textual. O tempo de treinamento foi utilizado para avaliar a velocidade de treinamento em cada caso analisado na seção de experimentos.

4 Metodologia

4.1 Estrutura de Rede

Semelhante ao treinamento proposto por [3], o treinamento colaborativo entre máquinas conectadas via internet, uma das frentes mais críticas do projeto, é a estrutura de rede, bem como o paradigma de sincronização utilizado. Latência e banda do framework proposto são métricas cruciais para a eficácia de treinamento distribuído e nesta seção descrevemos cada uma das decisões de projeto de forma detalhada.

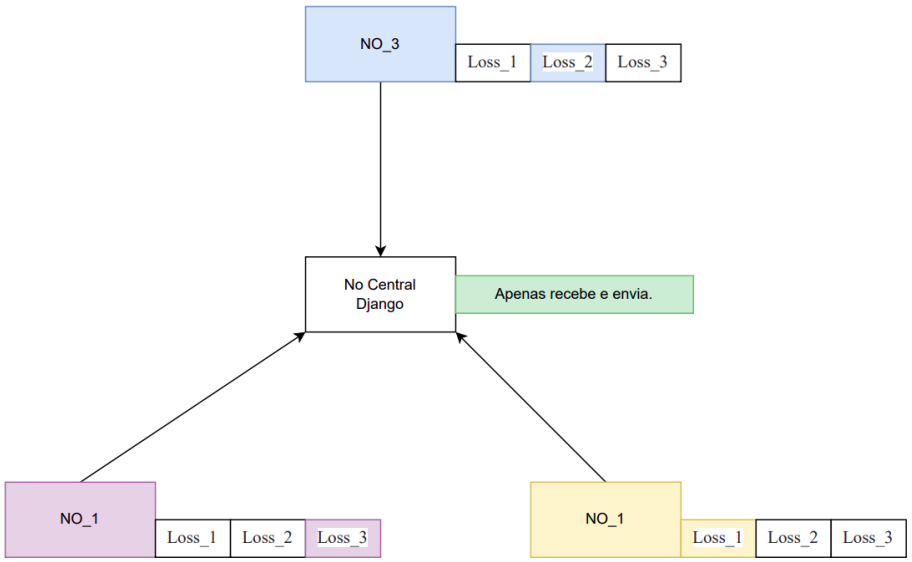


Figure 1: Esquema inicial de compartilhamento de losses via Web.

O primeiro esquema de treinamento proposto pela equipe é indicado na figura 1, no qual existiria um servidor central dedicado exclusivamente a promover o intermédio entre as diferentes máquinas sendo utilizadas no treinamento, as quais compartilhariam e receberiam a informação de loss entre si e retropropagariam-na pelos pesos de seus modelos, reutilizando o grafo computacional já disponível.

```
Out[1]: tensor(4.3464, device='cuda:0', grad_fn=<NllLoss2DBackward0>)
In [2]: loss.set_(torch.randn_like(loss).detach().to(device))
Out[2]: tensor(0.2042, device='cuda:0', grad_fn=<NotImplemented>)
In [3]: loss.data = torch.randn_like(loss)
In [4]: loss
Out[4]: tensor(1.8257, device='cuda:0', grad_fn=<NotImplemented>)

In [5]:
```

Figure 2: Perda de referência no grafo ao tentar modificar manualmente os resultados da loss no Pytorch.

Contudo, verificou-se que atualmente o framework Pytorch não disponibiliza o acesso direto à informação dos tensores sem a alteração de seu grafo (figura 2), impedindo que substituíssemos apenas o valor final de erro e realizássemos

o processo de backpropagation.

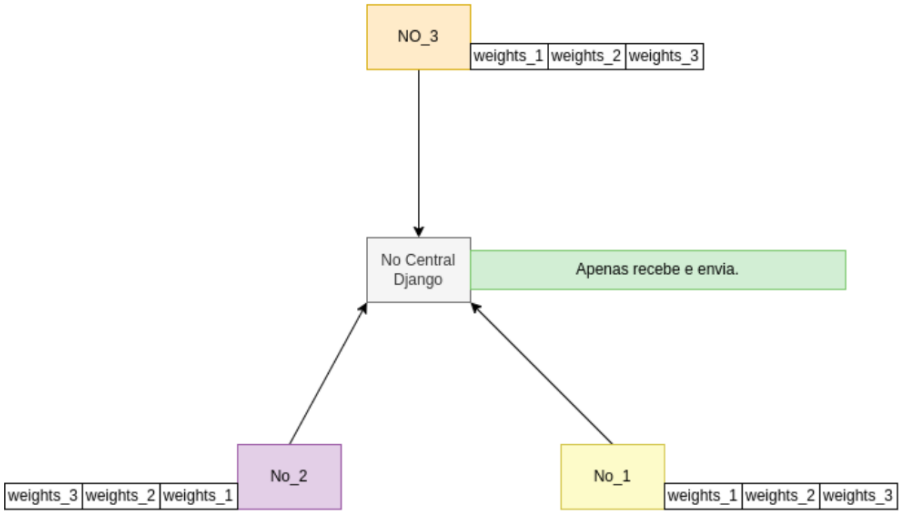


Figure 3: Esquema final de compartilhamento de pesos do modelo entre as iferentes máquinas do treinamento e o servidor central.

Para contonar a impossibilidade de se compartilhar apenas o valor de loss, foi necessário que o grupo decidisse entre compartilhar os valores detalhados do gradiente de cada modelo ou os próprios pesos de seus parâmetros. Compartilhando os gradientes, ainda seria necessário compartilhar (sincronizar) periodicamente os pesos dos modelos entre si, para evitar que eventuais imprecisões numéricas no processo de comunicação impedissem os modelos de convergir para o mesmo ponto. Dessa forma, o grupo escolheu a segunda abordagem (compartilhamento de pesos dos modelos), representada na figura 3.

A troca de pesos entre modelos acarreta um overheading considerável na conexão dos mesmos, pois, ao invés de enviar e receber apenas 1 ponto flutuante relativo à loss de cada um, agora devemos compartilhar cada um dos pesos dos modelos, os quais possuem mais de 500MB em parâmetros, o que requer uma velocidade de conexão razoavelmente alta para não impactar o desempenho do treinamento. Entretanto, compartilhar os gradientes não seria mais rápido, visto que cada parâmetro do modelo possui 1 vetor gradiente associado, gerando um custo computacional equivalente.

Para suprir o papel de servidor central, utlizamos um framework Web para a linguagem Python conhecido como Django[2] (trecho do código construído na figura 4). O uso deste tipo de ferramenta permite manipular com muito mais facilidade requests HTTP direcionados ao seu endereço IP. Por pertencer ao protocolo TCP/IP, este tipo de request conta com correção de erro automática, muito conveniente para tarefas como a de nosso treinamento, em que não pode haver corrupção dos dados no momento da troca.

```python
def receive_weight(request):
    print("Recebendo weight: ", request.headers["model-name"])
    global weight_dict, max_weight_dict_size, share_weight_dict

    weight_dict[request.headers["model-name"]] = request.body

    # Colocamos uma copia do dicionario para cada cliente, se ele ja estiver completo
    if len(weight_dict.keys()) >= max_weight_dict_size:
        for key in weight_dict.keys():
            share_weight_dict[key] = weight_dict

        weight_dict = {}

    return HttpResponse()


def sinc_weight(request):
    print("Compartilhando weight: ", request.headers["model-name"])

    while request.headers["model-name"] not in share_weight_dict.keys():
        time.sleep(0.1)

```

Figure 4: Rotas de comunicação com o servidor para troca de informação acerca dos pesos dos modelos.

Todavia, mesmo fazendo uso de um bom framework web, ainda existe o problema de que as máquinas de treinamento desconhecem, a princípio a localização do servidor na Web, ou seja, seu endereço IP. Existem diversas formas de lidar com este tipo de situação: DDNS, tunelamento de porta, IP estático, etc. A forma mais barata e imediata encontrada de se resolver o problema foi a utilização de um serviço de redirecionamento de porta de rede chamado Localtunnel[1]. Desta forma, é possível obter um endereço DNS com um nome mnemônico associado (figura 5) que redireciona através do servidor do próprio serviço para a API associada (no caso, o nosso servidor Django). Porém, quando elevamos a quantidade de dados sendo enviada, o Localtunnel passou a apresentar erros como perdas de pacotes e alguns travamentos, inviabilizando sua utilização para nossa aplicação.

```
lt --port 8000 --subdomain patrickctrf
```

Figure 5: Exemplo de encaminhamento de porta de rede utilizando o pacote Localtunnel.

A segunda opção foi a utilização de uma máquina de IP fixo alugada sob demanda na Web, da qual conhece-se o endereço de antemão, sendo possível informar aos nós de treino do framework e garantir que tal IP não mudará, o

que permite a comunicação direta via Web entre nós de treino e o nó central do servidor.

Os nós de treinamento (nós com GPU, também foram máquinas alugadas sob demanda e com configuração suficientemente robusta para treinar o modelo em questão (GPT-2), conforme figura **??**.

4.2 Troca de mensagens entre as GPUs

O treinamento colaborativo é realizado de maneira majoritariamente idêntica ao treinamento simples original, com a diferença que, após um determinado número de steps, os modelos sincronizam seus pesos atuais computados para os modelos, conforme a figura 6.

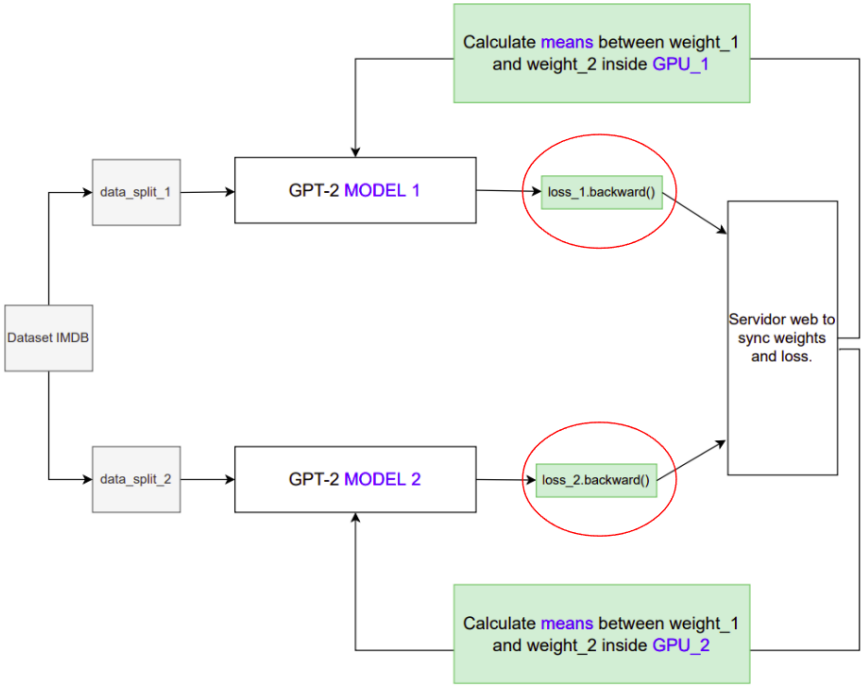


Figure 6: Fluxograma de treinamento colaborativo.

Uma interface para utilização nos códigos de treinamento de cada modelo, de forma ocultar instruções de rede do código de treinamento principal e aumentar a legibilidade do mesmo (figura 7).

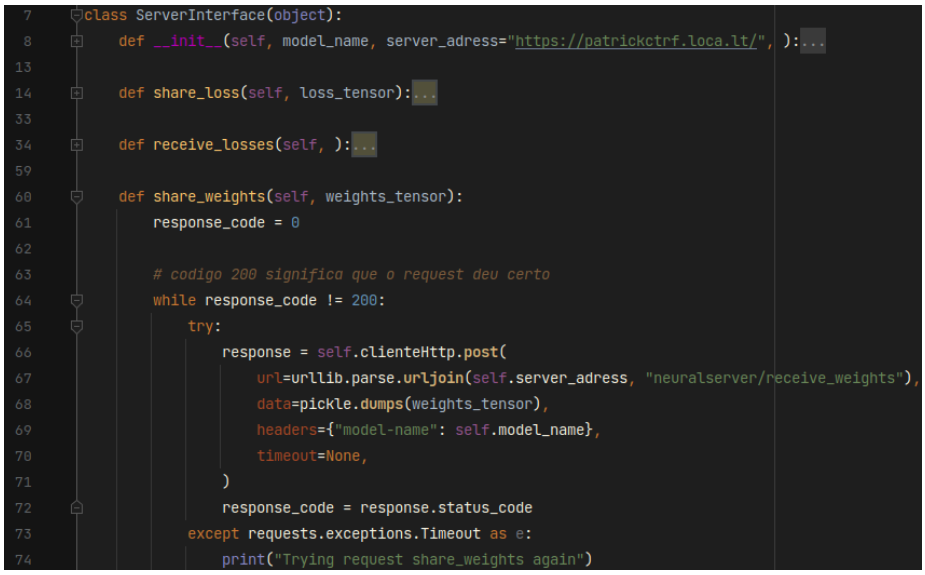


Figure 7: Classe de interface de comunicação com o servidor, abstraindo todos os comandos http do código python.

A tabela 2 mostra especificações entre o treinamento distribuído e o treinamento simples de um só nó.

		unico nó	Distribuido
Tamanho do batch	Treino	256	127
	Validação	127	127
Tamanho do Dataset	Treino	20000	10000
	Validação	5000	2500
Tamanho contexto		20	20
Otimizador		Adam	Adam
Taxa de aprendizado		0.0005	0.0005
Epocas		4	4

Table 2: A tabela mostra as especificações experimentais adotadas para o treinamento do modelo GPT-2 paralelo.

É possível observar que no campo treinamento tem-se um tamanho de lote igual 255 para um único nó. Todavia, quando vamos treinar de forma distribuída em dois nós diferentes, o lote é divido em 2. Isso é feito porque estamos usando a ideia de acumulo de gradiente [6]. As máquinas utilizadas para treinamento foram as seguintes:

		Distribuido	
	serial	máquina 1	máquina 2
nome	Tesla V100	Tesla V100	Tesla V100
Memoria (GRAM)	16GB	16GB	16GB

Table 3: Especificação de máquinas utilizadas para o treinamento do modelo serial e paralelo.

5 Parte Experimental

5.1 Analise da função de perda no treinamento serial

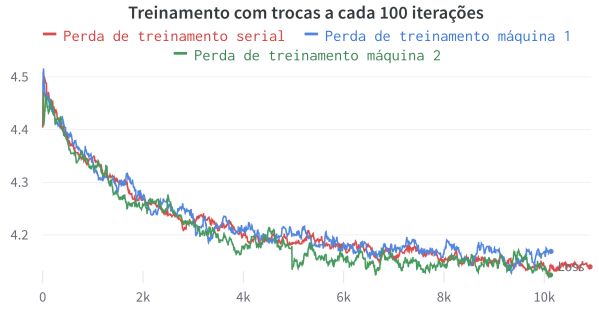
O trabalho apresentará quatro experimentos, o primeira será um treinamento serial sem troca de amostras consequentemente sem paralelismo (um treinamento padrão, apenas para base de comparação). O outros três experimentos foram executados fazendo sincronizações de pesos entre os 2 nós de treinamento implementados a cada 100, 1000, e 2000 iterações durante o treino, respectivamente.

A abordagem serial teve a sua perplexidade de treino e validação descendo suavemente a cada iteração de 256 amostras de treino. Conforme é ilustrada na figura 8.

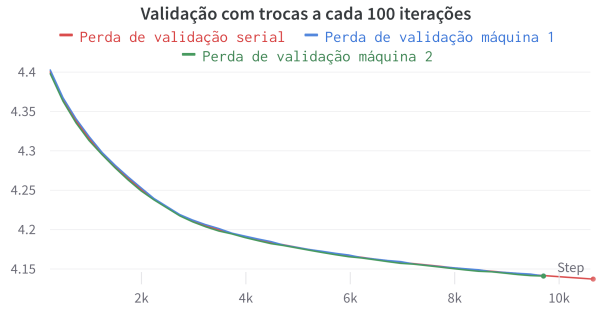
Figure 8: Treinamento e validação da rede GPT-2 serialmente (treinamento individual).

A suavidade justifica-se, pois, estamos usando uma arquitetura pre-treinada, por conta disso, esperasse que os pesos aprendidos da *GPT-2*, tenha informações relevantes entre palavras e sentenças.

Quando analisamos o gráfico 9 que treina em paralelo, **dividindo o mini-batch** entre todas as máquinas envolvidas no treinamento colaborativo, que troca a cada 100 iterações, obtêm-se o seguinte resultado:



(a)

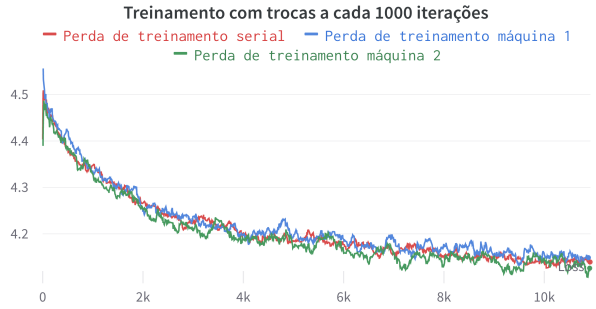


(b)

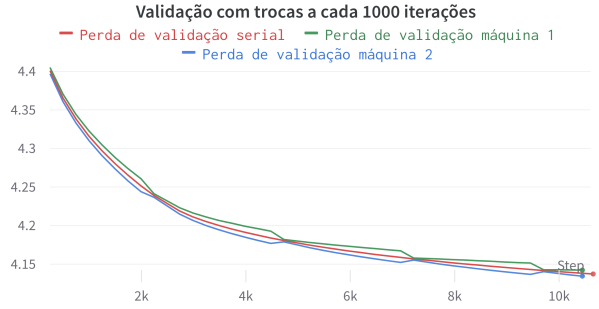
Figure 9: Treinamento colaborativo paralelo e validação da rede GPT-2 com trocas a cada 100 iterações.

Observa-se que o modelo treinado distribuidamente em máquinas geograficamente separadas, obtendo um erro baixíssimo entre essas maquinas em paralelo e uma única máquina em serial. Quase não se nota a diferença de validação entre a linha vermelha que representa a validação serial, e as linhas azul e verde representam a curva de erro obtida isoladamente em cada uma das máquinas envolvidas no treinamento distribuído via Web.

Todavia, quando aumentamos a troca para 1000 iterações, fica nítido o momento de troca de pesos entre *GPU-1* e a *GPU-2* do treinamento distribuído. Como pode-se ver no gráfico 10 (b).



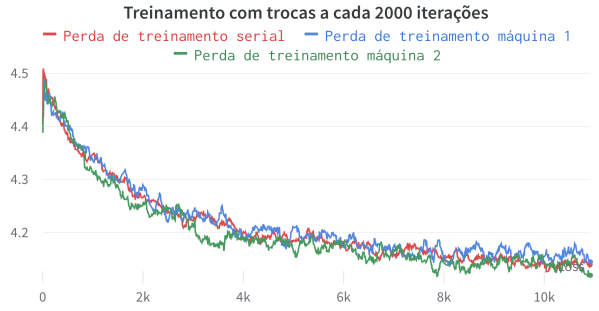
(a)



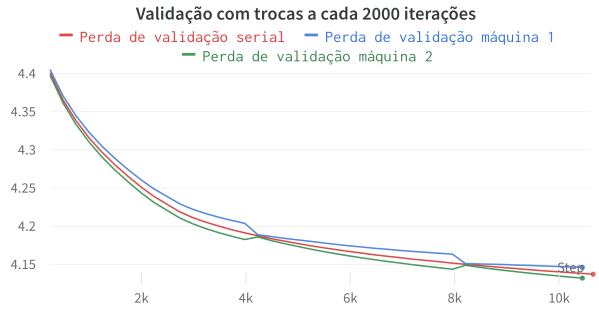
(b)

Figure 10: Treinamento colaborativo e validação da rede GPT-2 com sincronização e pesos a cada 1000 iterações. Nota-se que a cada troca de pesos, tem-se uma aproximação do que seria rede no modo única GPU.

Além disso, pode-se ver também que no momento em que os gráficos se tocam, é o momento de sincronização dos pesos da rede neural. O experimento fica da vez mais explícito quando aumentamos a troca para cada 2000 iterações, o resultado é ilustrado na figura 11.



(a)



(b)

Figure 11: Treinamento e validação da rede GPT-2. Nota-se que a cada troca de pesos, tem-se uma aproximação do que seria rede no modo única GPU.

No gráfico 11 (b), novamente os gráficos se tocam quando as trocas são feitas. Nesse momento, os pesos são sincronizados, e o modelo se aproxima da rede que foi treinada serialmente.

Consideramos que o motivo das redes divergirem sutilmente ao longo do treinamento individual seja devido a pequenos desbalanços no dataset que, embora esteja embaralhado, ainda está sujeito e vieses, os quais podem aproximar ou afastar o conjunto de dados de treinamento dos dados de validação, afetando assim a performance dos modelos sujeitos a cada um desses splits de treinamnto.

5.2 Resultados alcançados

A perplexidade alcançada durante o a validação no treinamento foi de aproximadamente 62 para todos os experimentos, conforme é explicitado na tabela 4. Esse resultado é bem importante porque mostra que os experimentos do treinamento distribuído não está focado somente na menor perplexidade, mas sim no tempo de treinamento quando trocamos mensagens entre N iterações, obtendo performance textual próxima aos modelos treinados de maneira usual (em máquina única).

N Trocas	Epocas	Batch	Perplexidade 1	Perplexidade 2	Media Perplexidade	Time
0	4	255	-	-	62.23	1h 8min
100	4	127	62.87	62.86	62.86	1h 3min
1000	4	127	62.91	62.34	**62.625**	37min
2000	4	127	63.33	62.12	62.725	**36.46 min**

Table 4: A tabela mostra os resultados dos tempos de treinamento a cada N trocas de pesos entre os modelos.

Nota-se que na tabela 4, o melhor tempo de treinamento foi de troca de pesos a cada 2000 iterações. Todavia, pode-se afirmar que o melhor resultado foi de 1000 iterações, uma vez que, quanto mais o modelo tem intervalos para troca, mais ele aproximará do modelo serial que não faz troca alguma, e isto fica explícito no fato de sua perplexidade final atingida ser a menor entre as abordagens colaborativas utilizadas.

6 Conclusão

A premissa de um treinamento de redes neurais colaborativo entre máquinas é a viabilização e/ou agilização deste processo em relação ao seu equivalente em máquinas isoladas. O que se espera é que a performance obtida ao final do processo seja semelhante àquela encontrada em métodos de treinamento usuais, porém em menor tempo, e podendo utilizar máquinas (GPUs) de menor capacidade, já que no modelo de treinamento colaborativo paralelo (o caso apresentado neste trabalho) a quantidade de amostras utilizadas por mini-batch é menor, exigindo menor memória em tempo de treinamento.

Foram utilizadas 2 máquinas nos experimentos aqui realizados, devido aos custos envolvidos para realização dos experimentos (até o framework se tornar funcional demanda-se muitas tentativas, e todas elas geram custos, não apenas as que deram certo). Entretanto, o framework aceita N máquinas, a critério de ser utilizado, podendo obter um grande incremento na velocidade de treinamento, com a utilização de GPUs muito mais modestas em relação a máquinas aptas para treinamento individual, alcançando equivalente ao treinamento clássico, conforme mostram nossos resultados.

$$\Theta_j = \Theta_i - \alpha \frac{\partial J(\Theta)}{\partial x} \tag{1}$$

$$\Theta_j = \Theta_i - mean(\alpha * \frac{\partial J(\Theta)}{\partial x}) \tag{2}$$

$$\Theta_j = mean(\Theta_i - \alpha * \frac{\partial J(\Theta)}{\partial x}) \tag{3}$$

References

[1] Localtunnel ∼ Expose yourself to the world.

[2] The web framework for perfectionists with deadlines — Django.

[3] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models, 2022.

[4] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko Yandex, Pavel Samygin, and Colin Raffel. Petals: Collaborative Inference and Fine-tuning of Large Models. sep 2022.

[5] Klaus Dodds. Popular geopolitics and audience dispositions: James bond and the internet movie database (imdb). *Transactions of the Institute of British Geographers*, 31(2):116–130, 2006.

[6] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.