Treinamento Paralelo de Redes Neurais usando o Modelo de Atores

Andre Rauber Du Bois, Ariam Moresco Bartsch

[1]Laboratory of Ubiquotous and Parallel Systems
Universidade Federal de Pelotas (UFPel)
R. Gomes Carneiro, 01 – 96.010-6100 – Pelotas – RS – Brazil

{dubois,ambartsch}@inf.ufpel.edu.br

Resumo. *O Modelo de Atores usa concorrência baseada em processos que não compartilham memória e que se comunicam por troca de mensagens, e tem sido usado no treinamento federado de redes neurais. Esse artigo é um trabalho em andamento que investiga o treinamento paralelo de redes neurais em dispositivos multiprocessados, utilizando o Modelo de Atores para distribuir e processar batches de treinamento, com objetivo de melhorar algoritmos de treinamento.*

1. Introdução

Algoritmos de aprendizado de máquina, em especial de redes neurais, muitas vezes precisam ser treinados em dispositivos que possuem hardware subótimo ou ineficiente, devido a fatores como distribuição de dados em dispositivos geograficamente distantes, limitação de largura de banda para transferência de dados, ou privacidade dos dados [Yuan et al. 2022]. Além desses dispositivos não terem hardware dedicado para execução de algoritmos de treinamento, os poucos recursos que eles tem podem ser mal aproveitados devido a heterogeneidade dos dispositivos ou a não exploração da arquitetura multiprocessada que dispositivos embarcados podem apresentar.

O Modelo de Atores, definido por [Hewitt et al. 1973], é um modelo conceitual de computação concorrente. Nele, "atores" são unidades primitivas de computação, que não compartilham memória entre si, e que se comunicam por meio de troca de mensagens assíncronas. O Modelo de Atores permite explorar paralelismo em diferentes arquiteturas, e hoje está presente em diversas linguagens como Erlang [erl 2023], Elixir [eli 2023], Scala [Sca 2023] e Java [Srirama et al. 2021]. Recentemente, devido ao seu modelo distribuído, o Modelo de Atores tem sido usado para o treinamento federado de redes neurais, como por exemplo em [Srirama and Vemuri 2023, Xu et al. 2022].

O objetivo desse trabalho em andamento é investigar o treinamento paralelo de redes neurais em dispositivos com vários núcleos usando o Modelo de Atores para distribuir e processar os *batches* em paralelo. Espera-se que o trabalho proposto possa ser aproveitado dentro do contexto de arquiteturas que usam o modelo de Atores para o treinamento distribuído de redes neurais, como por exemplo [Srirama and Vemuri 2023, Xu et al. 2022].

2. Conceitos

2.1. Modelo de Atores

O Modelo de Atores é um modelo de concorrência, onde os processos não compartilham memória e se comunicam apenas por troca de mensagens. Cada ator possui um endereço

e uma caixa de correspondência para a comunicação assíncrona. Esse endereço pode ser um endereço IP com porta, o que permite que dois agentes em dispositivos geograficamente distantes se comuniquem [Srirama and Vemuri 2023]. Dado o grau de isolamento dos atores, cada um deles possui um estado próprio, que não pode ser alterado por outros atores. O modelo pode ser usado em diferentes arquiteturas, não sendo restrito ao processamento distribuído, permintindo a exploração de recursos de paralelismo locais aos dispositivos.

2.2. Elixir

Elixir é uma linguagem de programação funcional que executa sobre a máquina virtual do Erlang [Ruokolainen et al. 2017]. De memória imutável e organizada em módulos, o código Elixir é compilado para *bytecode* antes de ser interpretado pela máquina virtual BEAM (Björn/Bogdan's Erlang Abstract Machine). Com foco em concorrência e escalabilidade, Elixir adotou o Modelo de Atores e o implementou nativamente em sua linguagem de forma eficiente, se beneficiando da natureza de escalabilidade do Erlang.

3. Trabalhos Relacionados

Em [Srirama and Vemuri 2023], faz-se o uso de aprendizagem federada em um ambiente de *fog computing* para predição de incêndios florestais, usando o Modelo de Atores disponibilizado pelo ferramental *Akka* – em Java – para coordenação da distribuição dos atores. No trabalho, usa-se dispositivos Raspberry Pi para construção da *fog*, e é usado de paralelismo de dados para fazer o treinamento federado de uma rede neural. O paralelismo interno de cada nodo não é explorado.

No contexto da linguagem Elixir, esta é usada por [Xu et al. 2022] para a realização do treinamento federado de redes neurais e, diferente do trabalho anterior, o treinamento nesse trabalho foi realizado numa máquina com grande capacidade de hardware dedicado. Nesse trabalho, o paralelismo local de cada nodo também não é explorado.

Em [Monteiro et al. 2018], observa a performance de treinamento de uma rede neural em uma Raspberry Pi, e nota-se que existe possibilidade de exploração de paralelismo, mesmo em dispositivos mais limitados como a Raspberry Pi.

4. Desenvolvimento

Para esse trabalho, foi desenvolvido código Elixir voltado ao treinamento de redes neurais. Ele conta com funções para criação de camadas, definição de funções de ativação e cálculos de precisão e perda. As operações de multiplicação matricial são feitas utilizando a biblioteca Matrex, que usa código C otimizado para multiplicação de matrizes em baixo nível [mat 2023]. O código não faz uso de hardware dedicado (GPUs).

Durante a execução do código, o *dataset* é dividido em *batches*, que são entregues a um ator coordenador que serve como uma fila de trabalho, delegando os *batches* como tarefas a cada ator processador – usando a técnica de paralelismo de dados, também explorada por [Srirama and Vemuri 2023]. O resultado de erro dos atores é então acumulado no ator coordenador para atualização da rede neural. No código, cada ator é um processo sem paralelismo interno, que recebe o seu *batch* e retorna sua taxa de erro calculada ao coordenador por meio de troca de mensagens. Portanto, a execução com apenas um ator e um único *batch* é análoga a sequencial.

5. Resultados Preliminares

O conjunto de dados utilizado na execução dessas funções foi o MNIST, um *dataset* de imagens de dígitos usado para treinamento de sistemas de reconhecimento de imagens [mni 2023]. A rede neural construída para a execução possui uma camada de entrada com 784 neurônios, um para cada pixel da imagem, seguido por uma camada densa com 512 neurônios e função de ativação ReLU, outra camada densa com 256 neurônios e função de ativação ReLU, e uma camada de saída com 10 neurônios (um para cada classe — ou dígito — de classificação). O método de cálculo de erro é *Mean Squared Error*.

Foram utilizadas 20 épocas de treinamento, e 30 testes para cada configuração de número de atores. O treinamento ocorreu numa máquina com 4 processadores AMD Opteron 6367, 128 Gigabytes de memória RAM, e sistema operacional Ubuntu 12.04.5 LTS Precise Pangolin.

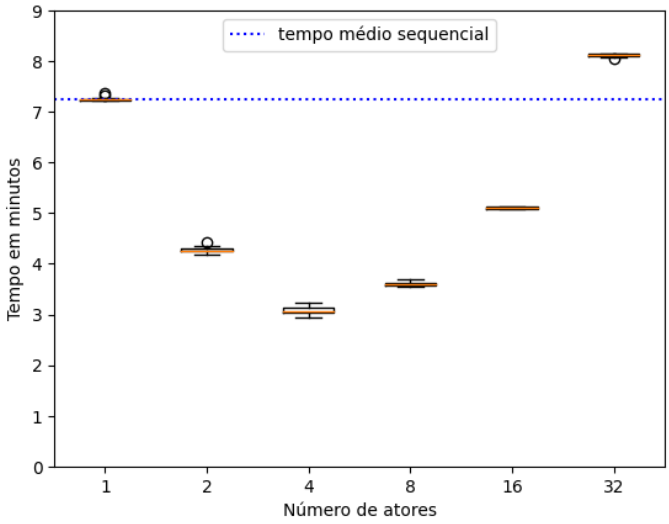

Figura 1. Comparação entre diferentes números de atores

A figura 1 mostra o tempo, em minutos, de cada configuração de atores no formato de diagrama de caixa. A execução com 1 ator é a execução sequencial, com sua média destacada na linha pontilhada para comparação com outros resultados. A execução com 4 atores foi a que apresentou melhor redução de tempo com esse *dataset*, constatando um *speedup* médio de 2,35 em relação ao sequencial. Também foram realizados testes com 2 atores (1,69 de *speedup*), 8 atores (2,01 de *speedup*), 16 atores (1,42 de *speedup*) e 32 atores (0,82 de *speedup*). Também foi observado que as redes neurais geradas com e sem paralelismo apresentaram precisão semelhante.

6. Conclusões

O Modelo de Atores, bastante utilizado em arquiteturas distribuídas, também pode ser aplicado fora desse contexto. Com uso de técnicas oriundas de algoritmos de aprendizado federado distribuído – em especial, a função agregação de modelos e o paralelismo de dados – é possível fazer uma rede federada local, baseada no Modelo de Atores, sem uma infraestrutura de rede, onde todos os atores estão localizados no mesmo dispositivo.

Para que isso seja possível, é necessário que o *dataset* de treinamento seja particionado de modo a ser exclusivo a cada ator (assim como em uma rede federada distribuída onde os dados são privados a cada dispositivo). A definição de uma quantidade ótima de atores num mesmo dispositivo é um desafio que pode reduzir o tempo em que agentes ociosos esperam resultados para a realização da agregação dos submodelos.

Dessa forma, esse trabalho explora a utilização o Modelo de Atores na paralelização de redes neurais. É necessário a realização de testes com outros *datasets*, de forma a observar o impacto do tamanho dos *batches* nas taxas de *speedup*. Também espera-se que o modelo concorrente baseado em Atores desenvolvido neste trabalho, possa ser usado para o processamento paralelo de redes neurais em outros trabalhos que já utilizam esse modelo no treinamento federado de redes neurais, como [Srirama and Vemuri 2023, Xu et al. 2022].

Referências

(2023). Matrex. WWW page, https://hex.pm/packages/matrex.

(2023). The Elixir Language. WWW page, https://elixir-lang.org/.

(2023). The Erlang Language. WWW page, https://www.erlang.org/.

(2023). The MNIST Database. WWW page, http://yann.lecun.com/exdb/mnist/.

(2023). The Scala Language. WWW page, https://www.scala-lang.org/.

Hewitt, C., Bishop, P., and Steiger, R. (1973). Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*, volume 3, page 235. Stanford Research Institute Menlo Park, CA.

Monteiro, A., De Oliveira, M., De Oliveira, R., and Da Silva, T. (2018). Embedded application of convolutional neural networks on raspberry pi for shm. *Electronics Letters*, 54(11):680–682.

Ruokolainen, T. et al. (2017). Development of a distributed web server utilizing elixir.

Srirama, S. N., Dick, F. M. S., and Adhikari, M. (2021). Akka framework based on the actor model for executing distributed fog computing applications. *Future Generation Computer Systems*, 117:439–452.

Srirama, S. N. and Vemuri, D. (2023). Canto: An actor model-based distributed fog framework supporting neural networks training in iot applications. *Computer Communications*, 199:1–9.

Xu, R., Michala, A. L., and Trinder, P. (2022). Caefl: composable and environment aware federated learning models. In *Proceedings of the 21st ACM SIGPLAN International Workshop on Erlang*, pages 9–20.

Yuan, B., Wolfe, C. R., Dun, C., Tang, Y., Kyrillidis, A., and Jermaine, C. (2022). Distributed learning of fully connected neural networks using independent subnet training. *Proc. VLDB Endow.*, 15(8):1581–1590.