JULIA E FLUX: MODERNIZANDO O APRENDIZADO DE MÁQUINA

A EVOLUÇÃO DAS TÉCNICAS DE APRENDIZADO DE MÁQUINA (AM) IMPULSIONA O AVANÇO DE FERRAMENTAS PARA FACILITAR O DESENVOLVIMENTO DE APLICAÇÕES DE ÚLTIMA GERAÇÃO. ESSAS FERRAMENTAS POSSIBILITAM QUE APLICAÇÕES UTILIZEM TECNOLOGIAS DE PONTA ADAPTANDO ÀS SUAS NECESSIDADES.

por **Dhairya Gandhi, Mike Innes, Elliot Saba, Keno Fischer e Viral Shah**
(Traduzido pelos Editores)

Computação contemporânea evoluiu de tal forma que nos incita a escrever programas de alto nível, de fácil compreensão e que sejam executados rapidamente em diferentes tipos de hardware. A solução comum para esse problema é utilizar várias linguagens de programação, conciliando a alta eficiência das de baixo nível com a facilidade das de alto nível. Contudo, utilizar várias linguagens diminui a produtividade do programador.

A linguagem de programação Julia[1] é uma solução elegante para esse problema por conciliar alta eficiência de execução (similar a C) com abstrações de programação de alto nível, permitindo o aumento da produtividade de programadores nas últimas décadas. O que torna Julia interessante é que, diferente de C, ela é totalmente derivável e desenvolvida na própria linguagem Julia, assim como a maioria dos seus pacotes. Esse cenário é adequado para o Aprendizado de Máquina (AM) porque, por exemplo, motores de física, equações diferenciais ordinárias (EDO) ou traçados de raios (ray tracers) podem tornar-se facilmente camadas de uma rede artificial neural e o processo de aprendizagem acontece de modo similar às camadas tradicionais. Essa flexibilidade permite que Julia satisfaça diferentes aplicações de AM. Flux aproveita muito bem essa característica para expandir esses benefícios à comunidade de AM.

Acreditamos que o futuro do AM está voltado para a linguagem de programação e compilação, especificamente, estendendo linguagens novas ou já existentes para atender a necessidades de pesquisa da área.

Flux é um arcabouço (framework) para AM onde modelos são desacoplados de detalhes específicos do arcabouço, permitindo que programadores foquem nos detalhes dos modelos, ou seja, nas especificidades do problema a ser resolvido utilizando o AM. Flux oferece simultaneamente simplicidade de uso e facilidade de customização do arcabouço (hackability). Apesar de essas características serem muito úteis, é muito difícil disponibilizar tal solução a desenvolvedores de aplicações de AM.

Historicamente, os modelos de ML estão cada vez menos estáticos e estruturados em sequências predefinidas de camadas. Esse cenário faz com que a programação diferenciável [2] torne-se um padrão em ML. Nesse contexto, Flux permite que técnicas de ML possam ser empregadas em várias aplicações, ampliando a capacidade de se resolver em diferentes tipos de problemas de forma inovadora e eficiente. Por exemplo, é possível modelar um sis-

tema caótico, aplicar um algoritmo de retropropagação para o aprendizado dos parâmetros deste sistema com base em uma função de custo para resolver um problema específico.

Com Flux, definimos regras de diferenciação para várias operações disponíveis em Julia. Flux monitora essas regras nos dados de entrada e nos parâmetros automaticamente [3]. Esse monitoramento permite aplicarmos técnicas de Diferenciação Automática (DA) que criam os gradientes das regras de diferenciação para, finalmente, derivá-las.

O Flux atraiu a atenção da comunidade de aprendizado de máquina devido ao suporte à DA através do Zygote. Zygote permite descrevermos DAs, possibilitando o uso de todas as funcionalidades da linguagem Julia. Assim, o código diferenciável pode ser passado para compiladores tradicionais LLVM (Low-level Virtual Machine), resultando em um programa derivado altamente eficiente.

O fato de Flux ser escrito em Julia permite que ele seja compatível com hardware de alto desempenho como GPUs (CuArrays.jl) e TPUs (XLA.jl), reutilizando códigos nativos da linguagem Julia. Flux usa funções nativas extensivamente, evitando a utilização de outras linguagens de programação, tornando-o um arcabouço completo e simples, escrito em menos de 6.000 linhas de código.

Flux é um marco na programação de aprendizado de máquina. Por exemplo, DifferentialEquations.jl com Flux permitiu implementar puramente em Julia EDOs neurais. Já o DiffEqFlux.jl disponibiliza uma biblioteca completa que integra equações diferenciais e redes neurais harmoniosamente. O DiffEqFlux.jl representa uma generalização da EDO neurais, sendo a primeira ferramenta desse tipo.

Acreditamos que o futuro do AM está voltado para a linguagem de programação e compilação, especificamente, estendendo linguagens novas ou já existentes para atender a necessidades de pesquisa da área. Isso é bom não apenas para a comunidade de AM, mas também para a programação numérica de uma forma geral. Linguagens que oferecem suporte à implementação de derivadas, à vetorização e a diferentes tipos de hardware conseguirão nortear muitos avanços na ciência. ●

Endereço com a versão original do artigo em inglês: https://bit.ly/2OTGhBk

Referências

1. Mike Innes, Stefan Karpinski, Viral Shah, David Barber, Pontus Stene- torp, Tim Besard, James Bradbury, Valentin Churavy, Simon Danisch, Alan Edelman, et al. On Machine Learning and Programming Languages, 2018. URL https://julialang.org/blog/2017/12/ml&pl
2. Mike Innes. What is Differentiable Programming?, 2019. URL https://fluxml.ai/2019/02/07/what-is-differentiable-programming.html
3. Mike Innes, James Bradbury, Keno Fischer, Dhairya Gandhi, Neethu Mariya Joy, Tejan Karmali, Matt Kelly, Avik Pal, Marco Rudilosso, Saba Elliot, Viral Shah, and Deniz Yuret. Building a Language and Compiler for Machine Learning, 2018. URL https://julialang.org/blog/2018/12/ml-language-compiler.

DHAIRYA GANDHI | É bacharel em Engenharia Elétrica e Eletrônica pelo Instituto Birla de Tecnologia e Ciência, Pilani (2018) e atualmente é cientista de dados na Julia Computing.

MIKE INNES | É bacharel em Física pela Universidade de Oxford e criador do Flux.jl e Zygote.jl. Hoje, é desenvolvedor de software na Julia Computing.

ELLIOT SABA | É bacharel em Ciências em Engenharia Elétrica (2011) e mestre pela Universidade de Washington (2014). É Ph.D. pelo Laboratório de Computação Ubíqua da Universidade de Washington (2018). Atualmente, é engenheiro sênior de pesquisa na Julia Computing.

KENO FISCHER | É bacharel em Artes/ciências e mestre, em 2016, pela Universidade de Harvard. É cofundador e CTO (Tools) da Julia Computing.

VIRAL SHAH | É Ph.D em Ciências Computacionais da Universidade da Califórnia, Santa Bárbara. Também é um cocriador da linguagem Julia e cofundador e CEO da Julia Computing.