Rose: Extensible Autodiff on the Web

Raven Rothkopf
rgr2124@barnard.edu
Barnard College, Columbia University

Sam Estep
estep@cmu.edu
Carnegie Mellon University

Joshua Sunshine
sunshine@cs.cmu.edu
Carnegie Mellon University

ABSTRACT

Automatic differentiation (autodiff) has become the backbone for a new wave of optimization-driven domains such as computer graphics and machine learning over the past decade. However, existing autodiff systems face limitations, either lacking support for in-browser development or failing to harness more recent, compiler-based approaches to achieve both expressiveness and size-preserving differentiation. This work introduces Rose, a portable, extensible autodiff language that runs on the web. Through Rose, we aim to increase accessibility to autodiff algorithms and empower end-user programming in optimization-driven domains. We plan to evaluate Rose by replacing the autodiff engines of real-world, client-side optimization systems and assess the improvements on the computation power, expressiveness, and efficiency of such systems.

1 PROBLEM AND MOTIVATION

Automatic differentiation is a computational technique used to efficiently compute the derivatives of functions. The computed derivatives can be used to analyze how small perturbations in a program's inputs can effect its outputs. These derivatives are particularly useful for determining which direction to move to optimize an objective function, a process known as gradient-based optimization.

Significant advancements in autodiff algorithms have resulted in development of systems like PyTorch [5] and TensorFlow [1], and more experimental autodiff projects like Google's JAX [2, 3]. These frameworks provide high-level APIs for computing gradients of functions, making it easier for users to leverage autodiff in their work. However, most autodiff compilers do not run on the web.

With the increasing capabilities of web clients, there is great opportunity to leverage autodiff techniques in web-based applications. Web-based languages require no installation and provide convenient and familiar platforms for users to interact with optimization-driven domains from any device with a web browser. Additionally, harnessing autodiff for web-based applications unlocks opportunities for optimization that centralize user interaction, but do not necessarily need the raw native speed of machine learning training frameworks.

TensorFlow.js [7] is a widely used framework for autodiff on the web, demonstrating the impact and demand for such capabilities. However, TensorFlow.js has several limitations that impact the expressiveness and efficiency of the system. To automatically differentiate a program, Tensorflow.js constructs a computation graph of all functions and operations that subsequently undergoes differentiation. The downside of this method is that the computation graph corresponding to a TensorFlow.js program is asymptotically larger than the size of the input program, making the differentiation process computationally expensive. Additionally, TensorFlow.js uses a layer of abstraction to differentiate programs, forcing all operations to be at the array-level and computed in bulk. While this technique is powerful, it limits the expressiveness of the system to only programs that can be differentiated using TensorFlow.js's built in array operations. Furthermore, TensorFlow.js does not support taking higher-order derivatives for all functions, which are necessary for certain optimization methods.

We present Rose, a language equipped to automatically differentiate web-based programs without sacrificing expressive power and computational efficiency. Our goal is to address the limitations of existing autodiff systems and extend the reach of autodiff to domains that can make good use of optimization like diagramming and interactive simulation, which focus on end-user programming.

By leveraging the power of WebAssembly [4], Rose achieves native-like performance in the browser while maintaining compatibility across platforms and devices. By introducing the concept of differentiable functions into a program's computation graph, Rose can take the derivative of complicated programs while still asymptotically preserving the program, minimizing runtime and memory cost. Additionally, using recent, compiler-based techniques [6], Rose can differentiate all operations within and between arrays and can take higher-order derivatives of both built-in and custom primitives. Additionally, Rose is designed to be extensible. Extensibility allows users to construct custom autodiff primitives and computations that may not be expressible in a constrained autodiff system.

In the following sections, we demonstrate an example Rose program and walk through some elements of our system design which address and mitigate the aforementioned limitations of existing autodiff systems.

2 ROSE BY EXAMPLE

To demonstrate the utility of Rose, we provide a small JavaScript program, comprised of two `.js` files. A distinguishing feature of Rose is that it closed under automatic differentiation, meaning that anything that can be written in Rose can be automatically differentiated as many times as needed. This program first extends the Rose language with the custom differentiable primitives, `pow` and `log`. Using the newly defined primitives, the program then applies a set of compiled differentiable functions to an input vector corresponding to the `x` and `y` coordinates of a point in 2D space.

Extensibility in Rose is achieved through the `custom` function, enabling users to construct their own primitives for differentiation. In Listing 1, `custom.js` illustrates an example of extending Rose by adding a new `pow` function that takes its first argument and raises it to the power of its second argument. This file also defines a custom helper function, `log`, that is utilized when the `pow` function is differentiated.

To define a new differentiable primitive in Rose using the `custom` function, one must provide both the implementation of the primitive itself and an implementation of its derivative. Note that in Rose, only the definition of one derivative is needed in order to derive the

```
import { Real, Vec, add, custom, div, mul } from "rose";
const log = custom([Real], Real,
  (x) => Math.log(x),
  (x, dx) => [log(x), div(dx, x)]
);
const pow = custom([Vec(Real, 2)], Real,
  (v) => Math.pow(v[0], v[1]),
  (v, dv) => {
    const z = pow(v);
    const [x, y] = v;
    const [dx, dy] = dv;
    return [
        z,
        mul(z, add(mul(div(y, x), dx), mul(log(x), dy)))
    ];
  });
```

Listing 1: Defining custom primitives in Rose.

vector-Jacobian product (VJP), a different kind of derivative that is more efficient for computing the gradient in numerical optimization, but is often much more challenging to specify [6]. This technique enables users to apply a breadth of differentiation techniques to their custom primitives without needing to separately implement them.

The following file then imports the custom primitives defined in Listing 1, and demonstrates how they can be used and differentiated using the Rose `vjp` function.

```
import { Real, Vec, fn, jit, vjp } from "rose";
import { pow } from "./custom.js";
const Vec2 = Vec(Real, 2);
const g = fn([Vec2], Vec2, (v) => vjp(pow)(v).vjp(1));
const h = fn([Vec2], Vec(Vec2, 2), (v) => {
  const x = vjp(g)(v);
  return [x.vjp([1, 0]), x.vjp([0, 1])];
});
const l = await Promise.all([jit(pow), jit(g), jit(h)]);
export default (x, y) => l.map((f) => f([x, y]));
```

Listing 2: A Rose program.

This file takes an imported function, `pow`, that calculates the result of raising the first argument to the power of the second argument, and computes its gradient and Hessian using a Rose autodiff function, `vjp`, which transforms the input function. The resulting gradient and Hessian correspond to the first and second order partial derivatives of a function, respectively. They provide information about the rate and direction a function is changing, which is crucial for optimization.

The custom primitive, `pow`, and the two differentiable functions from Listing 2 are compiled and composed into a program that could be used to compute the local approximation of the quadratic form of the Rose `pow` function at a specified point.

A mock-up visualization of the local quadratic approximation of the `pow` function is provided in Fig. 1. Since the derivatives can be calculated in the browser using Rose, the user can interact with the visualization in their browser, dragging the point along the xy plane and observing the corresponding function's changes in 3D space. Since the Rose compiler both runs in and emits WebAssembly, the user could even specify their own mathematical expression for which Rose can take derivatives through the interface.

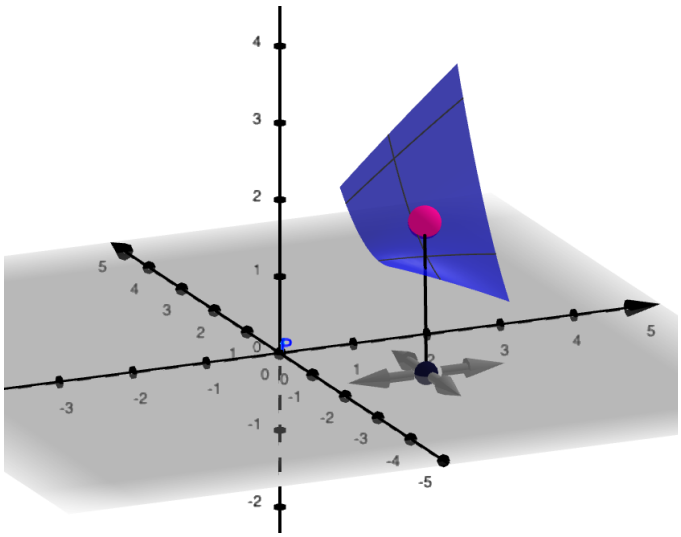

Figure 1: An interactive visualization of the quadratic approximation of the `pow` function. Users can drag the black dot around the 2D plane using the gray arrows, changing the approximation of the `pow` function's derivative in 3D space represented by the blue curved surface. Since Rose runs in the browser, the approximation can be recomputed on the user's device.

3 EVALUATION

In order to evaluate the utility of the Rose language, we provide an example of its application to an existing web-based, optimization-driven domain: mathematical diagramming. Penrose [8] is a set of three domain specific languages and an optimizer for making conceptual diagrams, and it uses a custom-built JavaScript autodiff engine. Penrose runs completely client-side, enabling users to construct mathematical diagrams directly in their browser.

Like TensorFlow.js, Penrose computation graphs grow asymptotically larger than their input program without the concept of differentiable functions introduced by Rose. Additionally, the Penrose autodiff engine has a similarly limited expressiveness, with no support for extensibility and taking higher-order derivatives.

We plan to port the Penrose library of differentiable functions to Rose and assess the resulting impact on the computation power, expressiveness, and efficiency of the Penrose system.

4 CONCLUSION

We have presented Rose, a portable, extensible autodiff system designed to address the limitations of existing web-based autodiff frameworks and target optimization for end-user programming. Prioritizing accessible, expressive, and extensible web-based autodiff will lower the barrier of entry to optimization-driven domains and encourage user interaction through intuitive web interfaces.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg,

Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. 2018. JAX: composable transformations of Python+ NumPy programs. (2018).

[3] Roy Frostig, Matthew James Johnson, and Chris Leary. 2018. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning* 4, 9 (2018).

[4] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the Web up to Speed with WebAssembly. *SIGPLAN Not.* 52, 6 (jun 2017), 185–200. https://doi.org/10.1145/3140587.3062363

[5] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[6] Alexey Radul, Adam Paszke, Roy Frostig, Matthew J. Johnson, and Dougal Maclaurin. 2023. You Only Linearize Once: Tangents Transpose to Gradients. *Proc. ACM Program. Lang.* 7, POPL, Article 43 (jan 2023), 29 pages. https://doi.org/10.1145/3571236

[7] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Charles Nicholson, Nick Kreeger, Ping Yu, Shanqing Cai, Eric Nielsen, David Soegel, Stan Bileschi, et al. 2019. Tensorflow. js: Machine learning for the web and beyond. *Proceedings of Machine Learning and Systems* 1 (2019), 309–321.

[8] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: From Mathematical Notation to Beautiful Diagrams. *ACM Trans. Graph.* 39, 4, Article 144 (aug 2020), 16 pages. https://doi.org/10.1145/3386569.3392375