JAX versus NumPy: A Comparative Analysis of Modern Scientific Computing Frameworks

by

Group 2Deep2Learn: Jazmine Brown, Varit Kobutra, Kimberly Navarrete, Phillip Torres

Houston Community College

Deep Learning in Artificial Intelligence (CRN: 19519)

Professor Patricia McManus

January 2025

Table of Contents

JAX

Origin, Background & Development

JAX (Just After eXecution) is a machine learning and deep learning library developed by Google DeepMind in 2018. Created as a specialized JIT (Just-In-Time) compiler, JAX converts Python and NumPy-based machine learning code into high-performance programs optimized for hardware accelerators such as GPUs and TPUs.

Built on the same tracing library as Autograd, JAX treats its operations as primitives, allowing for self-contained functionality. It also incorporates NumPy's numerical functions as primitives. The design of JAX combines the simplicity of Python with the power of modern hardware, making it a powerful tool for performing complex mathematical tasks.

JAX has gained widespread popularity within the research community due to its efficiency and integration with machine learning frameworks like TensorFlow and PyTorch. Its primary purpose is to enable high-performance numerical computing and machine learning research by leveraging Python’s flexibility alongside the advanced capabilities of GPUs and TPUs.

XLA (Accelerated Linear Algebra) plays a key role in optimizing computations for TensorFlow and enables JAX to run NumPy code efficiently on GPUs and TPUs. Additionally, JAX provides a straightforward API that allows Python functions to be compiled just-in-time into XLA-optimized kernels, further enhancing performance.

Ultimately, JAX serves as a versatile tool for anyone in need of fast, flexible, and scalable computations, particularly for machine learning and scientific computing applications.

NumPy: The Foundation of Scientific Computing in Python

Origins and Development

NumPy (Numerical Python) emerged from earlier array-processing packages in Python. Initially created by Jim Hugunin at MIT in 1995 as "Numeric," it underwent significant evolution when Travis Oliphant combined features from Numeric and Numarray to create NumPy in 2005 (Harris et al., 2020). The project began with limited funding and was primarily developed by graduate students, yet it transformed into a cornerstone of scientific computing (van der Walt et al., 2011).

```
import numpy as np

new_array = np.array([1,2,3])

print(new_array)

# Output

[1 2 3]
```

Figure 1: Creating an array with NumPy. (Soares, 2021)

Purpose and Core Features

NumPy's primary purpose is to provide efficient array processing and numerical computing capabilities in Python. Its key features include:

Array Processing

- Powerful n-dimensional array object (`ndarray`) for efficient data storage and manipulation (Harris et al., 2020)
- Memory-efficient storage through homogeneous data types
- Advanced array operations including reshaping, slicing, and indexing (Oliphant, 2015)

Performance Advantages

- Up to 50x faster than traditional Python lists (van der Walt et al., 2011)

- Optimized memory usage through contiguous storage
- C/C++ backend for high-performance computations (Harris et al., 2020)

Mathematical Capabilities

- Comprehensive linear algebra operations
- Advanced statistical functions
- Fourier transform capabilities (Virtanen et al., 2020)

```
import numpy as np
import matplotlib.pyplot as plt

# sine function
x = np.linspace(0,2*np.pi, 100)
f = np.sin(x)
plt.figure(figsize=(15,7))
plt.subplot(1,3,1)
plt.plot(f, color="green")
plt.title("np.sin(x)")

# cosine function
f = np.cos(x)
plt.subplot(1,3,2)
plt.plot(f, color="blue")
plt.title("np.cos(x)")

# tangent function
f = np.tan(x)
plt.subplot(1,3,3)
plt.plot(f, color="red")
plt.title("np.tan(x)")
plt.show()

# Output
```

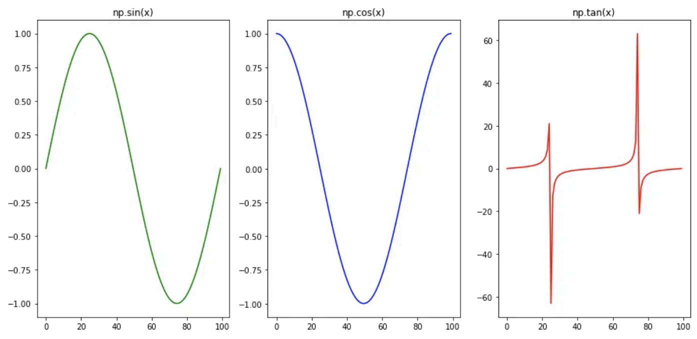


Figure 2: Example of NumPy math functions. (Soares, 2021)

Real-World Applications

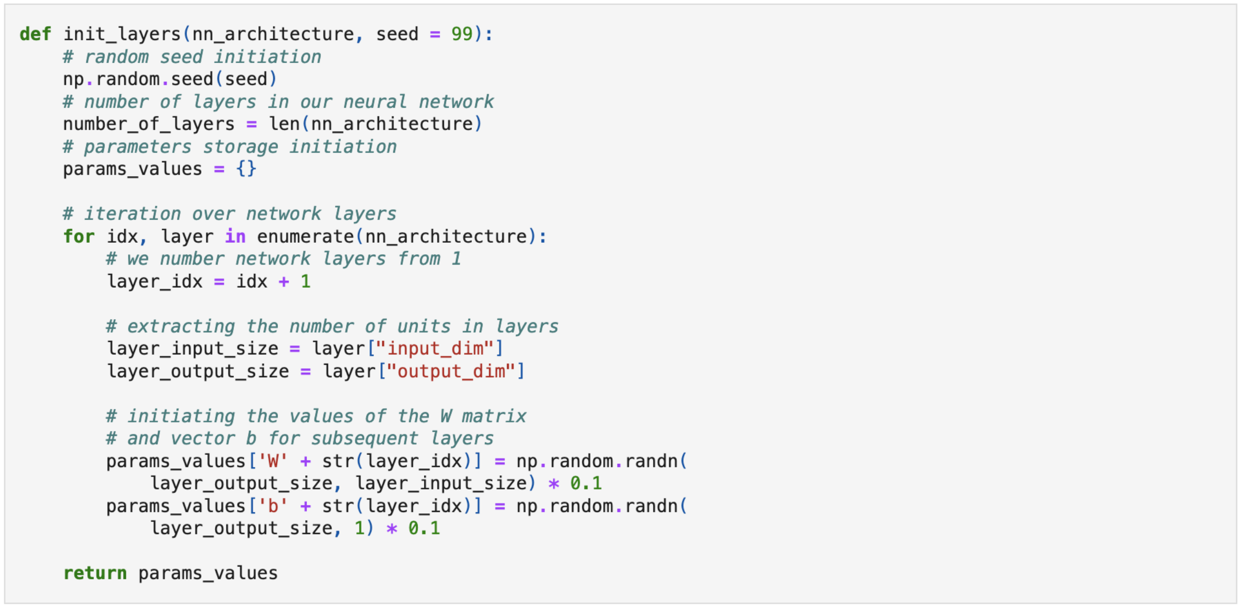
NumPy has demonstrated its effectiveness across various domains:

Scientific Research

- Used in gravitational wave analysis (Abbott et al., 2016)
- Instrumental in the M87 black hole imaging project (Event Horizon Telescope Collaboration, 2019)
- Applied in astronomical data processing and climate science (Harris et al., 2020)

Industry Applications

- Financial analysis and modeling
- Image and signal processing
- Machine learning and AI applications (Virtanen et al., 2020)



```
def init_layers(nn_architecture, seed = 99):
    # random seed initiation
    np.random.seed(seed)
    # number of layers in our neural network
    number_of_layers = len(nn_architecture)
    # parameters storage initiation
    params_values = {}

    # iteration over network layers
    for idx, layer in enumerate(nn_architecture):
        # we number network layers from 1
        layer_idx = idx + 1

        # extracting the number of units in layers
        layer_input_size = layer["input_dim"]
        layer_output_size = layer["output_dim"]

        # initiating the values of the W matrix
        # and vector b for subsequent layers
        params_values['W' + str(layer_idx)] = np.random.randn(
            layer_output_size, layer_input_size) * 0.1
        params_values['b' + str(layer_idx)] = np.random.randn(
            layer_output_size, 1) * 0.1

    return params_values
```

Figure 3: Initiation of layers for a neural network in deep learning using plain NumPy. (Skalski, 2019)

Comparative Analysis: NumPy vs JAX

Performance Characteristics

- NumPy operations are CPU-bound, while JAX can leverage GPU/TPU acceleration (Harris et al., 2020)
- NumPy provides immediate execution, whereas JAX uses Just-In-Time compilation for optimization (Bradbury et al., 2022)
- NumPy performs better for small-scale operations due to JAX's compilation overhead
- JAX excels in performance, particularly in machine learning tasks. It can run on CPUs, GPUs, and TPUs, thanks to its integration with XLA (Accelerated Linear Algebra), providing faster execution times for large-scale operations.
- JAX allows users to compile Python functions just-in-time into optimized code, enabling highly efficient execution, especially for array-level operations. Its ability to run on multiple hardware platforms offers a significant performance boost compared to NumPy.
- JAX can achieve up to 100x speedup over NumPy for large matrix operations when using hardware acceleration (Frostig et al., 2018)

Feature Comparison

- NumPy offers mutable arrays, while JAX enforces functional programming with immutable arrays (Bradbury et al., 2022)
- JAX provides automatic differentiation capabilities not available in NumPy
- NumPy has a more extensive ecosystem of established scientific computing libraries
- JAX requires pure functions, whereas NumPy allows side effects and in-place operations (Harris et al., 2020)

Development and Usage

- NumPy has a more gradual learning curve and extensive documentation
- JAX requires understanding of functional programming concepts
- NumPy's API is more stable and well-established
- JAX offers more advanced features for machine learning research and optimization

- JAX is built on top of NumPy and offers a similar API, making it relatively easy for those familiar with NumPy to get started.
- JAX's advanced features like Just-In-Time (JIT) compilation and automatic differentiation (autograd) require some learning, particularly for non-experts.
- JAX has more complex functionality for machine learning applications, requiring additional annotation (such as the jit_ps decorator) to enable advanced features, which might increase the learning curve.

Integration Capabilities

- NumPy serves as the foundation for traditional scientific computing libraries (Pandas, Matplotlib, SciPy)
- JAX integrates well with modern ML frameworks and Google Cloud TPUs
- Both libraries support interoperability with each other, allowing gradual migration (Bradbury et al., 2022)
- NumPy has broader support in legacy scientific computing applications
- JAX is designed for scalability and is a powerful tool for large-scale machine learning and scientific computing tasks. Thanks to its integration with XLA, JAX can scale across multiple CPUs, GPUs, and TPUs, making it ideal for high-performance computations in machine learning.
- JAX is built with distributed programming in mind, supporting operations like map_reduce and parallel computation through features like pmap (parallel mapping), which allow for efficient scaling in large models and datasets.
-

References

Abbott, B. P., Abbott, R., Abbott, T. D., Abernathy, M. R., Acernese, F., Ackley, K., ... & Zweizig, J. (2016). *Observation of gravitational waves from a binary black hole merger*. Physical Review Letters, 116(6), 061102.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., & Wanderman-Milne, S. (2022). *JAX: composable transformations of Python+NumPy programs*. http://github.com/google/jax

Event Horizon Telescope Collaboration. (2019). *First M87 event horizon telescope results*. I. The shadow of the supermassive black hole. The Astrophysical Journal Letters, 875(1), L1.

Frostig, R., Johnson, M. J., & Leary, C. (2018). *Compiling machine learning programs via high-level tracing*. Systems for Machine Learning, 1(1), 1-12.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). *Array programming with NumPy*. Nature, 585(7825), 357-362.

Oliphant, T. E. (2015). *Guide to NumPy (2nd ed.)*. CreateSpace Independent Publishing Platform.

Skalski, P. (2019). *ILearnDeepLearning.py [Computer software]*. GitHub. https://github.com/SkalskiP/ILearnDeepLearning.py

Soares, L. (2021, December 12). *A compressed summary of NumPy in 18 Python snippets*. Towards Data Science. https://towardsdatascience.com/a-compressed-summary-of-numpy-in-18-python-snippets-1a8a63fc7b4f

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). *The NumPy array: A structure for efficient numerical computation*. Computing in Science & Engineering, 13(2), 22-30.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & SciPy 1.0 Contributors. (2020). *SciPy 1.0: Fundamental algorithms for scientific computing in Python*. Nature Methods, 17(3), 261-272.

The Upwork Team. " Google Jax: What it is and How It Got Started." Up Work. 24 May 2024 https://www.upwork.com/resources/google-jax

Johnson, Matthew. “ JAX: Accelerating Machine-Learning Research with Composable Function Transformations in Python.” Nvidia. March 2020 https://www.nvidia.com/en-us/on-demand/session/gtcsj20-s21989/

Ai Jobs. “JAX Explained.” Ai Jobs 30 October 2024 https://aijobs.net/insights/jax-explained/#:~:text=and%20data%20science.-,Origins%20and%20History%20of%20JAX,frameworks%20like%20TensorFlow%20and%20PyTorch.

Bhagyashree R. “Google researchers introduce JAX” PackT. 11 December 2018

Roy Fostig “ Compiling machine learning programs via high-level tracing” Ml SYS February 2018 https://mlsys.org/Conferences/doc/2018/146.pdf

Claudio Lemos. “ JAX vs NumPy” TensorOps 7 July 2022. https://www.tensorops.ai/post/should-i-switch-from-numpy-to-jax-1

Venkata Harsha Vardhan Gangala “JAX vs. NumPy: Key Differences and Benefits” 17 August 2024 https://medium.com/@harshavardhangv/jax-vs-numpy-key-differences-and-benefits-72e442bbf67f

Wesley Kambale “ NumPy vs JAX: Optimizing Performance and Functionality” 8 May 2023 https://kambale.dev/numpy-vs-jax

Azzedine Aitsaid “Breaking Up with NumPy: Why JAX is Your New Favorite Tool” Medium 28 September 2023 https://python.plainenglish.io/breaking-up-with-numpy-why-jax-is-your-new-favorite-tool-1f91faf4cb15