JAX:

1. Tại Sao Dùng JAX?

* Vì nó nhanh nhất, khi được chạy trên GPU hoặc TPU, CPU cũng chạy được nhưng không nhanh
* Không cần phải viết Code cho từng trường hợp CPU, TPU, GPU, JAX tự động điều chỉnh cho thích hợp

1. JAX Và Numpy?

* Mảng Numpy nằm trên RAM, mảng JAX nằm trên CPU, GPU hoặc TPU
* Dù mảng JAX không có kiểu dữ liệu là mảng Numpy, nhưng mảng của JAX và mảng Numpy có thể tính toán với nhau như bình thường, khi đó mảng Numpy luôn tự động đổi sang mảng JAX, nên tăng thêm thời gian thực thi Code
* Mảng JAX hoạt động y chang mảng Numpy, mọi thuộc tính hay hàm đều như của Numpy

1. Cơ Chế Bất Đồng Bộ Trong JAX?

* Khi tạo hoặc tính toán với mảng JAX, thì nó sẽ không chờ tạo ra hoặc tính toán hết nguyên cái biểu thức rồi mới chạy dòng tiếp theo mà thực hiện luôn mấy câu lệnh khác song song, chỉ khi nào có câu lệnh in giá trị của biểu thức thì mới chờ mấy cái tính toán thực thi hết rồi mới in
* Khi kết quả chưa được tính toán xong ta vẫn có thể truy cập 1 số thuộc tính của kết quả như Shape

1. Cách Để Chờ Tính Toán Xong Thì Mới Nhảy Sang Câu Lệnh Tiếp Theo?

* <Hàm Tính Toán>.block\_until\_ready()
* Ví dụ
* Cho đoạn Code bất đồng bộ
* A = jnp.dot(x, x)
* B = jnp.dot(x, x)
* C = A + B
* print(“foo”)
* So với lúc bắt đầu chạy
* A có kết quả lúc 0.01 giây
* B có kết quả lúc 0.01 giây
* C có kết quả lúc 0.02 giây
* Chữ foo xuất hiện lúc 0 giây
* Cho đoạn Code vừa có đồng bộ vừa có bất đồng bộ
* A = jnp.dot(x, x).block\_until\_ready()
* B = jnp.dot(x, x).block\_until\_ready()
* C = A + B
* print(“foo”)
* So với lúc bắt đầu chạy
* A có kết quả lúc 0.01 giây
* B có kết quả lúc 0.02 giây
* C có kết quả lúc 0.03 giây
* Chữ foo xuất hiện lúc 0.02 giây

1. Nếu 1 Mảng Numpy Đóng Vai Trò Là Nhiều Tham Số Của 1 Hàm JAX Thì Liệu Nó Chỉ Đổi 1 Lần Mảng Numpy Sang Mảng JAX Không?

* Không, nó sẽ đổi nhiều lần 1 cách không cần thiết
* Ví dụ
* A = np.array([1, 2, 3])
* B = jnp.dot(A, A.T)
* Thời gian để B có kết quả là 0.02 giây, vì nó phải chuyển đổi cả A và A.T từ mảng Numpy sang mảng JAX
* Khắc phục = cách sử dụng hàm device\_put
* A = device\_put(np.array([1, 2, 3]))
* B = jnp.dot(A, A.T)
* Thời gian để B có kết quả chỉ còn 0.01 giây, vì A đã trở thành mảng JAX ngay từ đầu
* Hàm device\_put có tác dụng chuyển mảng Numpy sang mảng JAX

Random – Ngẫu Nhiên:

1. Lợi Thế Của Random Trong JAX So Với Numpy?

* Giả sử khi thiết lập Seed trong Numpy, ta Random lần thứ 1, ra được 1.5, Random lần thứ 2, ra được 3.1, … tại lần Random lần thứ 10, ta muốn quay lại lần thứ 1, muốn Random lần thứ 10 ra 1.5, Random lần thứ 11 ra 3.1, … thì dễ thấy rằng trong Numpy không có cách nào làm được điều này, nhưng JAX thì có, bằng việc hàm Random nào trong JAX cũng bắt ta truyền vào cho nó 1 cái Key, thì hàm Random đó sẽ Random lại từ đầu chuỗi ngẫu nhiên gắn với cái Key đó
* Ví dụ
* Cho đoạn Code Numpy
* import numpy as np
* np.random.seed(0)
* a = np.random.uniform()
* b = np.random.uniform()
* c = np.random.uniform()
* d = np.random.uniform()
* Ta được a = 1.5, b = 3.1, c = 3.5, d = 4.6, rõ ràng không có cách nào quay lại số đầu tiên
* Cho đoạn Code JAX
* from jax import random
* key1 = random.PRNGKey(0)
* key2 = random.PRNGKey(1)
* a = random.uniform(key1)
* b = random.uniform(key1)
* c = random.uniform(key2)
* d = random.uniform(key2)
* Ta được a = 1.5, b = 1.5, c = 4.9, d = 4.9, rõ ràng ta có thể quay lại số đầu tiên bất cứ lúc nào ta muốn chỉ bằng việc truyền cho cùng 1 cái Key

1. Tạo 1 Mảng Có Giá Trị Ngẫu Nhiên Tuân Theo 1 Phân Phối Chuẩn Tắc?

* from jax import random
* key = random.PRNGKey(<Seed>)
* samples = random.normal(key, <Shape>)
* <Seed> là 1 số nguyên
* <Seed> khác nhau sẽ cho ra chuỗi ngẫu nhiên khác nhau
* key là 1 mảng JAX 1 chiều chứa 2 phần tử là 0 và <Seed>
* samples là 1 mảng JAX có Shape = <Shape>, các phần tử có giá trị ngẫu nhiên được tạo ra từ 1 phân phối chuẩn có Mean = 0 và Std = 1
* Ví dụ
* from jax import random
* key = random.PRNGKey(2)
* samples = random.normal(key, (2, 3))
* Ta thu được
* samples = [
  + [– 0.15508525, 1.5718881, 0.5495363],
  + [0.17252125, – 0.60508585, 1.6838225]

]