IT4653 HỌC SÂU VÀ ỨNG DỤNG **Bài đọc số : 5**

Phần cứng và phần mềm dành cho Học Sâu (DL)

Người viết : Trịnh Anh Phúc *Người kiểm tra : x*

1 Giới thiệu

Lĩnh vực học sâu có bước tiến dài trong thập kỷ qua do nhiều nguyên nhân, trong ba trụ cột để nói về sự phát triển của nó không thể không nhắc đến các sự phát triển vượt bậc của kỹ thuật công nghệ phần cứng hay các thư viện phần mềm, miễn phí hoặc trả phí cho một cộng đồng lớn sử dụng cùng các cải tiến liên tục về mô hình. Ứng dụng của học sâu, tạo nên một thị trường khổng lỗ về nhu cầu phần cứng cũng như phần mềm dành cho lĩnh vực này. Các công ty phần cứng, đại diện Nvidia đã trở thành công ty đại chúng nghìn tỷ chỉ sau vài năm. Các khoản đầu tư mạo hiểm luôn được "rót" vào các công ty khởi nghiệp liên quan chặc chẽ đến lĩnh vực học sâu hướng đến ích dụng AI rất gần gũi và thuận tiện như ChatGPT, Dall-E, Sora, Midjourney, DeepFake

(a) (b) (c)

Bảng 1: Bộ ba sản phẩm đến từ OpenAI. Theo thứ tự từ trái sang phải (a) ChatGPT ứng dụng văn bản chatbot (b) Dall-E sinh ảnh từ mô tả còn (c) Sora sinh đoạn video từ văn bản.

Bài đọc thêm cũng sẽ được chia thành hai phần chính, phần cứng và phần mềm. Dù sao, riêng về phần cứng trên thế giới vẫn chỉ có một số công ty lớn đa quốc gia chiếm giữ các bí mật công nghệ không thể chia sẽ như TSMC, ASML, ARM, Nvidia, Qualcomm đang thống trị thị trường phần cứng trong lĩnh vực. Về phần mềm, các công ty phần mềm có thể có các ứng dụng dựa trên nền tảng mã nguồn mở như OpenAI cung cấp các chứng chỉ cho phép sử dụng mục độ hạn chế như Google, Facebook, Microsoft,....

Nội dung bài đọc sẽ được chia tuần tự thành hai phần chính

1. Phần cứng - Hardware
2. Phần mềm - Software

2 Phần cứng - Hardware

Phần cứng máy tính của học sâu cần song song hóa cao bởi các phép nhân tích chập (pairwise-element product) được sử dụng thường xuyên, dẫn đến các GPU trở thành bộ xử lý được sử dụng nhiều trong lĩnh

vực này. Công ty chuyên sản xuất cạc đồ họa trở thành công ty đại chúng chuyên sản xuất các loại thiết bị phần cứng dùng trogn lĩnh vực tính toán học sâu, liên quan nhiều đến máy chủ. Ngoài ra các các công ty khác cũng tham gia vào học sâu, ví dụ như các công ty tạo thiết bị kết nối vạn vật IoT, thiết bị di động cũng tham gia vào lĩnh vực này.

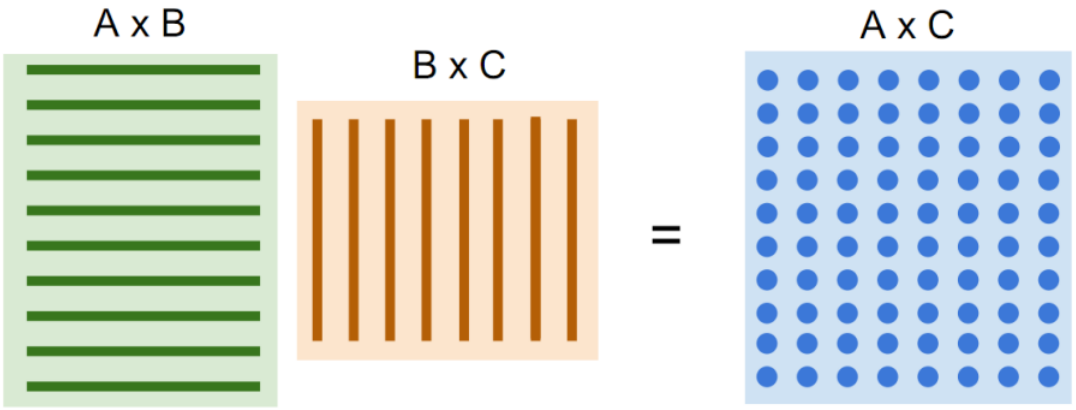
2.1 CPU vs GPU

Đối với bộ vi xử lý trung tâm CPU, thường được thiết kế cho tính toán tuần tự, số lượng các nhân thường ít, tuy nhiên khác với các ứng dụng lập trình tuần tự. Mạng nơ ron học sau chủ yếu dùng các phép toán nhân ma trận hoặc nhân tích chập cần song song hóa tối đa để giảm thời gian trong huấn luyện cũng như dự đoán. Trong bảng 2 tóm tắt các thông số của hai loại CPU vs GPU

	Số lõi	Đồng hồ xung nhịp	Bộ nhớ	Chi phí	Tốc độ tính toán
CPU (Intel Core-i7 7700K)	4	4.2Ghz	RAM	$385	540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6Ghz	11 GB GDDR6	$1199	13.4 TFLOPs FP32

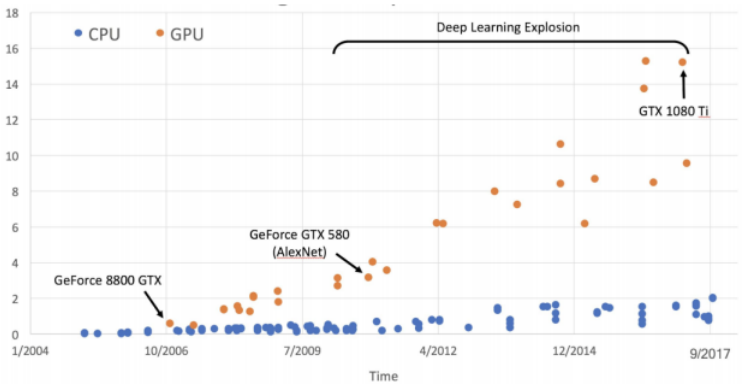
Bảng 2: Tóm tắt thông số hai loại CPU vs GPU

Theo đó lợi thế lớn của GPU là có nhiều lõi vật lý giúp khả năng song song hóa cao, thay vì dùng bộ nhớ RAM của hệ thống, các cạc đồ họa có bộ nhớ đồ họa GDDR riêng cho phép tối ưu hóa về lưu trữ phép tính toán ma trận. Chẳng hạn để dễ hình dung, ta thực hiện nhân vô hướng hai ma trận $\mathbb{R}^{A\times B}$ và $\mathbb{R}^{B\times C}$ để được $\mathbb{R}^{A\times C}$ minh họa theo Hình 3 dưới đây. Nếu với CPU ta sẽ tính lần lượt các giá trị trong ma trận kết quả từ trên xuống dưới, từ trái qua phải . Tuy nhiên theo cơ chế của GPU, tất cả các giá trị của ma trận vế phải sẽ được tính đồng thời bởi các lõi khác nhau của nó mà không cần chờ đợi tuần tự như CPU.



Bảng 3: Phép nhân ma trận khi dùng với GPU

Hình ảnh minh họa tiếp theo 4 cho chúng ta biết lợi thế của GPU trong cuộc đua tốc độ với CPU dựa theo kỹ thuật tính toán song song. Sở dĩ GPU có tận dụng được tốc độ này do tính sogn song hóa cao của các thao tác huấn luyện và dự đoán của Học Sâu đều dựa trên ma trận trong khi các ứng dụng lập trình đều thực hiện dạng tuần tự. Theo trục thời gian, càng về sau tốc độ tính toán đổi với $1 càng cao càng chứng minh tính hiệu quả tài chính khi ta sử dụng GPU trong Học Sâu.



Bảng 4: Chia hai cột cuối của Bảng 2

2.2 Thiết bị biên - Edge device

Thiết bị biên là các thiết bị Học Sâu hướng IoT kết nối vạn vật, khác với các GPU dùng cho các máy chủ cấu hình cao. Các thiết bị biên thường có cấu hình vừa đủ về cả tính toán lẫn bộ nhớ khi ứng dụng Học Sâu vào thực tế. Thông thường ta sẽ không huấn luyện trên chúng mã chỉ nạp các tham số đã huấn luyện rồi cho chúng chạy pha dự đoán. Thiết bị biên có thể coi là "công tắc thông minh" trong nhiều trường hợp mà con người ít có khả năng nhanh tiếp cận vị trí nguy hiểm để đưa ra các quyết định tức thì như động đất, núi lửa, sóng thần hay trong tình huống khẩn cấp.

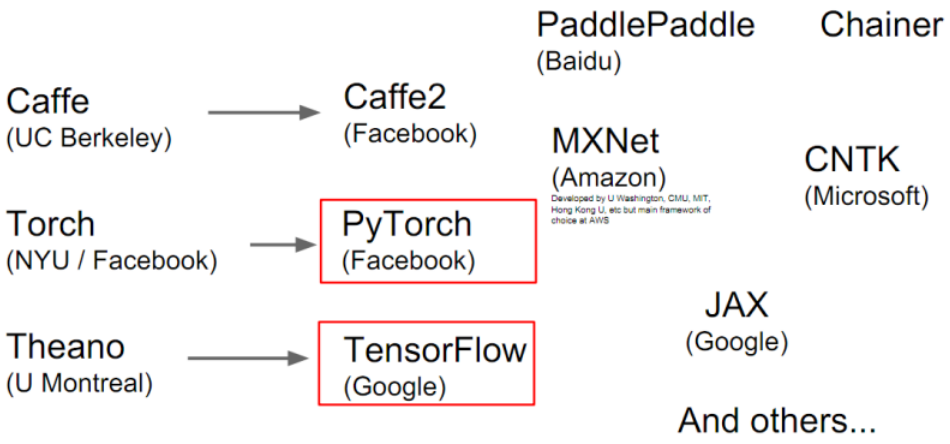
Bảng 5: Thiết bị biên NVidia

Hình 5 minh họa thiết bị biên của NVidia thường có giá thành hợp lý, nguồn điện pin di động, có khả năng kết nối, kích thước nhỏ. Các phần mềm Học Sâu cũng được thiết kế riêng cho thiết bị dạng này thường dung lượng nhỏ. Thông thường thiết bị biên có thể thiết kế riêng cho một ứng dụng Học Sâu cụ thể nào đó như

- Phát hiện chuyển động - Vibration Detection
- Phát hiện từ khóa - Keyword Detection
- Phát hiện bất thường - Anormal Detection
- Phát hiện đối tượng - Object Detection
- Phát hiện cử chỉ - Gesture Detection
- ...

3 Phần mềm - Software

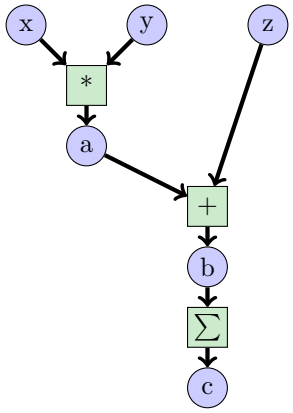
Trong các nên tằng lập trình dùng cho Học Sâu, có hai nên tảng được sử dung phổ biến là Tensorflow và PyTorch. Cả hai nên tảng đều được hỗ trở bởi hai công ty công nghệ khổng lồ lần lượt là Google và Facebook. Tuy nhiên, hãng phần mềm AI sử dụng mô hình học sâu nổi tiếng lại là OpenAI đều có nguồn gốc mã nguồn mở, phi lợi nhuận. Dù có cách nhìn khác nhau về cách thức tạo ra mô hình Học Sâu, hai nền tảng đều cung cấp cho người lập trình đầy đủ các lớp thư viện để có thể tạo, lưu trữ, quản lý các mạng nơ ron Học Sâu cũng như triển khai các dịch vụ AI kèm theo nêu người dùng muốn phát triển thành ứng dụng tương ứng.



Bảng 6: Các nền tảng dùng trong lĩnh vực Học Sâu.

3.1 Đồ thị tính toán

Đa phần các mạng nơ ron có thể coi là một đồ thị có hướng không có chu trình (acyclic directed graph) theo đó các nút tròn dùng để biểu diễn dữ liệu trong khi các nút thao tác được biểu diễn bằng nút vuông. Theo đúng định lý, các đồ thị có hướng không có chu trình luôn có nút bắt đầu - chỉ có mũi tên ra - còn các nút kết thúc - chỉ có mũi tên vào. Trong các mạng nơ ron có thể có nhiều đầu vào cũng như có nhiều đầu ra, các nút trung gian chữa kết quả của các phép toán.



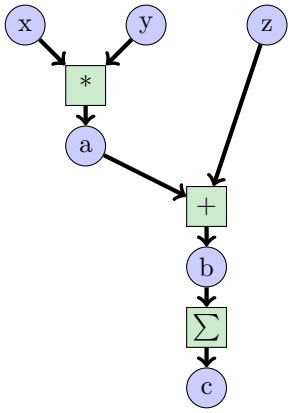
Hình 1: Minh hoạ một đồ thị tính toán

Đồ thị tính toán minh hoạ tại Hình 1 có thể nói có ba đầu vào tham số x, y và z trong khi c là đầu ra. Các phép toán nhân (*) cộng (+) hay tổng ($\sum$) sẽ tạo ra các kết quả trung gian a, b. Để lập trình xây dựng đồ thị tính toán trên ta có thể dùng thư viện số của Numpy tuy nhiên các phép tính lan truyền ngược sẽ phải lập trình và tính riêng biệt cho từng đồ thị. Quá trình tổng hợp, suy diễn của đồ thị sẽ phải được lập trình và hiểu được bởi người lập trình. Thêm nữa, quan trọng là ta không dùng GPU để tính toán song song được. Lúc này ta cần các nền tảng lập trình dành riêng cho Học Sâu để hiện thực hoá và sử dụng đồ thị tính toán này.

3.2 PyTorch

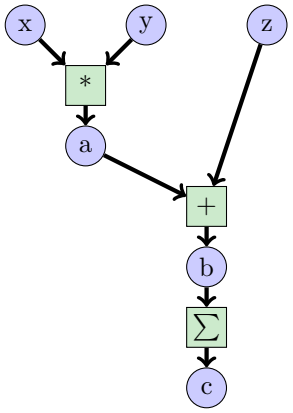
Thư viện lập trình PyTorch vốn nằm trong dự án Torch của đại học NYU sau đó được tạo lớp các thư viện ngôn ngữ Python cho phép người lập trình có thể xây dựng đồ thị tính toán động (Dynamical graph). Do cách nhìn về kiến trúc mạng nơ ron khác so với Tensorflow nên việc lập trình trên PyTorch được đánh giá là khá "dễ hiểu" hơn so với Tensorflow hay Keras.

```
import torch

N, D = 3,4
x = torch.rand(N,D)
y = torch.rand(N,D)
z = torch.rand(N,D)

a = x*y
b = a+z
c = torch.sum(b)
```



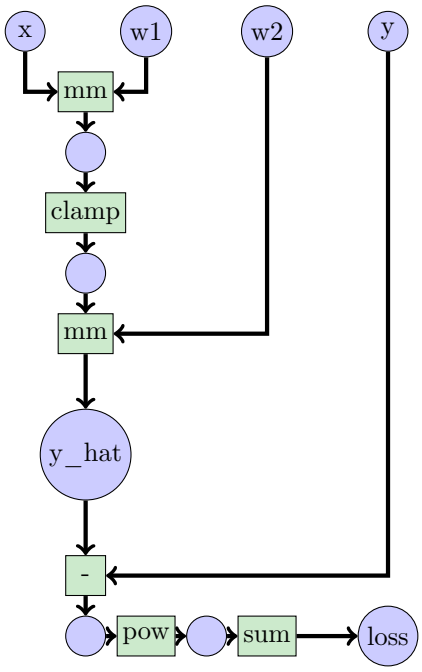
Để tính toán đạo hàm lan truyền ngược, ta yêu cầu PyTorch tự động tính như sau

```
import torch

N, D = 3,4
x = torch.rand(N,D, requires_grad=True)
y = torch.rand(N,D)
z = torch.rand(N,D)

a = x*y
b = a+z
c = torch.sum(b)

c.backward()
```



Để sử dụng GPU, PyTorch cung cấp đơn vị tính toán là Tensor tương đương ma trận array trong Numpy. Lớp Tensor sẽ thực hiện lưu trữ, tính toán đồ thị trong môi trường GPU. Trong ví dụ dưới đây, ta cấu hình mạng nơ ron truyền thẳng có kích thước đầu vào $D_{in} = 1000$, lớp ẩn có kích thước $H = 100$ và lớp ra $D_{out} = 10$ còn kích thước bó là $N = 64$.

```
import torch

device = torch.device('gpu')
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.rand(N, D_in, device=device)
y = torch.rand(N, D_out, device=device)
w1 = torch.rand(D_in, H, device=device)
w2 = torch.rand(H, D_out, device=device)

learning_rate = 1e-6
epochs = 500

for t in range(epochs):
 h = x.mm(w1)
 h_relu = h.clamp(min=0)
 y_hat = h_relu.mm(w2)
 loss = (y_hat - y).pow(2).sum()

 grad_y_hat = 2.0*(y_hat - y)
 grad_w2 = h_relu.t().mm(grad_y_hat)
 grad_h_relu = grad_y_hat.mm(w2.t())
 grad_h = grad_h_relu.clone()
 grad_h[h<0] = 0
 grand_w1 = x.t().mm(grad_h)

 w2 -= learning_rate*grad_w2
 w1 -= learning_rate*grad_w1
```



Rõ ràng, việc sử dụng đoạn mã nguồn trên PyTorch của lớp Tensor giống hệ như NumPy tuy nhiên về cơ bản PyTorch cung cấp các chức năng hiệu quả hơn để thực hiện việc huấn luyện và suy dẫn. Chẳng hạn như ta có thể dùng yêu cầu tính đạo hàm tự động với các Tensor tham số như sau

```
import torch


N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.rand(N, D_in)
y = torch.rand(N, D_out)
w1 = torch.rand(D_in, H, requires_grad=True)
w2 = torch.rand(H, D_out, requires_grad=True)

learning_rate = 1e-6
epochs = 500

for t in range(epochs):
 y_hat = x.mm(w1).clamp(min=0).mm(w2)
 loss = (y_hat - y).pow(2).sum()

 loss.backward()

 with torch.no_grad():
  w2 -= learning_rate*grad_w2
  w1 -= learning_rate*grad_w1
  w2.grad.zero_()
  w1.grad.zero_()
```

Ta cũng có thể dùng lớp nn (neural networks) để giúp Tensor dễ dàng nhận diện lớp mạng nơ ron tuần tự để cấu hình chúng, các bạn để ý một chút là trong bộ dữ liệu trước ta không có tham số lệch b.

```
import torch
```

```
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.rand(N, D_in)
y = torch.rand(N, D_out)

model = torch.nn.Sequential(
                torch.nn.Linear(D_in, H),
                torch.nn.ReLU(),
                torch.nn.Linear(H, D_out))

learning_rate = 1e-6
epochs = 500

for t in range(epochs):
 y_pred = model(x)
 loss = torch.nn.functional.mse_loss(y_pred, y)

 model.backward()

 with torch.no_grad():
  for param in model.parameters():
   param -= learning_rate*param.grad
 model.zero_grad()
```

Tất nhiên, PyTorch cũng cung cấp cho người dùng các thư viện tối ưu optim, các biến thể của giải thuật tụt dốc ngược hướng vec tơ tiếp tuyến -Gradient Descent.

```
import torch


N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.rand(N, D_in)
y = torch.rand(N, D_out)

model = torch.nn.Sequential(
                torch.nn.Linear(D_in, H),
                torch.nn.ReLU(),
                torch.nn.Linear(H, D_out))

learning_rate = 1e-6
epochs = 500

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(epochs):
 y_pred = model(x)
 loss = torch.nn.functional.mse_loss(y_pred, y)

 model.backward()

 optimizer.step()
 optimizer.zero_grad()
```

Ngoài ra, PyTorch cũng cung cấp các mạng nơ ron đã huấn luyện sẵn như AlexNet, VGG-16 hay ResNet101 trong torchvision. `https://github.com/torchvision`.

PyTorch nguyên thủy cung cấp cách nhìn tạo *đồ thị tính toán động* cho phép người dùng từng bước tạo các nút và kết nối có định hướng giữa chúng để cuối cùng tạo nên một đồ thị tính toán hoàn chỉnh, gồm cả quá trình huấn luyện và dự đoán. Ví dụ tương tự như đồ thị tính toán trên ta có thể từng bước tạo các nút và các tính toán từng phần của đồ thị mà không cần tạo dự án hoàn chỉnh từ đầu mới thực hiện quá trình tính

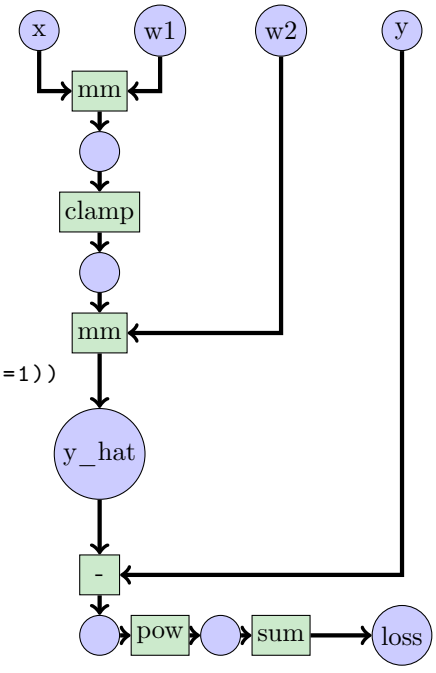
toán (on-fly-mode). Tuy nhiên, hướng tiếp cận linh hoạt này cũng có vấn đề do ta cần **xây dựng lại đồ thị động** này tại mỗi bước lặp khi huấn luyện ảnh hưởng đến tốc độ khi huấn luyện.

3.3 TensorFlow

Hướng tiếp cận của TensorFlow là đồ thị tính toán tĩnh, theo đó đồ thị được xây dựng một lần (once) và quá trình tính toán được thực hiện lặp lại (for-backward) trên nó mà không cần xây đi xây lại như hướng tiếp cận động. Nó giúp tăng đáng kể tốc độ nhưng giảm độ linh hoạt khi phải xây dựng đồ thị tính toán *hoàn chỉnh* ngay từ đầu. Nên tảng TensorFlow sẽ thực hiện tổng hợp và tính toán mọi thứ chi phí cho một lần xây dựng ban đầu. Hướng tiếp cận đồ thị tĩnh gồm hai bước

1. Xây dựng một đồ thị gồm đầy đủ các nút toán hạng và toán tử
2. Dùng đồ thị này thực hiện tính toán cho mọi bước lặp

```
import numpy as np
import tensorflow as tf


N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_hat = tf.matmul(h,w2)
diff = y_hat - y
loss = tf.reduce_mean(tf.reduce_sum(diff**2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
 values = {x : np.random.randn(N, D),
                 w1 : np.random.randn(D, H),
                w2 : np.random.randn(H, D),
                y : np.random.randn(N, D),}
 out = sees.run([loss, grad_w1, grad_w2],
                              feed_dict=values)
 loss_val, grad_w1_val, grad_w2_val = out
```



Như đã nói, việc tạo một đồ thị tính toán tĩnh hoàn chỉnh ngay từ đầu làm quá trình gỡ rối (debug) đồ thị tính toán của TensorFlow diễn ra khó khăn., vậy nên bắt đầu từ phiên bản **TensorFlow 2.0** cung cấp cơ chế thực thi nóng (eager execution) cho phép thực thi tính toán mà chưa cần xây dựng đồ thị hoàn chỉnh.

3.3.1 Keras

Đa phần với người mới tiếp xúc với TensorFlow để lập trình Học Sâu cảm thấy khá khó khăn với hướng tiếp cận đồ thị tĩnh của nó. Ngoài kiến thức thuần túy về mạng nơ ron, kiến trúc mạng truyền thẳng, lớp tích chập, lớp đầy đủ thì việc vận dụng nó với TensorFlow trở nên khá khó khăn. Keras giúp đóng gói các khá niệm này mức cao hơn, giảm thiểu số lượng mã nguồn phải lập trình, dễ dùng và gần với kiến trúc mạng nơ ron cơ bản hơn. Người dùng không cần lo lắng học thêm về kiến thức đồ thị tính toán ở trên nữa. Hiện nay, về cơ bản số người dùng Keras là lớn nhất dù nó chỉ đóng gói các lớp của TensorFlow.

3.4 So sánh giữa PyTorch vs TensorFlow

Bảng tổng kết so sánh giữa hai nền tảng học sâu

Đặc trưng	PyTorch	TensorFlow
Đồ thị tính toán	Động	Tĩnh
Phạm vi	Dự án cỡ nhỏ vừa, nghiên cứu	Dữ án lớn dễ triển khai sang ứng dụng
Linh hoạt	Hơn	Kém
Dễ học	Hơn	Kém
Cộng đồng	Nhỏ	Lớn
Hỗ trợ GPU	Hơn	Kém
Gỡ rối	Hơn	Kém
Tương thích ngôn ngữ	Chủ yếu Python và C++	C++, JavaScript, Python, C#, Ruby và Swift
Tính toán	Tập chung	Phân tán
Ứng dụng nổi bật	Testla copilot, ChatGPT	Google search, Uber services

4 Tổng kết

Hiện tại, các cải tiến về phần cứng và phần mềm trong lĩnh vực Ứng dụng Học Sâu đang diễn ra hàng ngày, hàng giờ. Việc thêm các tiện ích liên quan đến các nên tẳng phần mềm cũng như cải tiến chất lượng phần cứng cần cập nhật liên tục. Nó giúp cho ứng dụng Học Sâu khi ra đến thành dịch vụ ngày càng sớm và nhanh hơn.