DOI: 10.36335/VNJHM.2020(709).63-70

NÂNG CAO HIỆU NĂNG CỦA DEEP LEARNING TRONG HỆ THỐNG TÍNH TOÁN HIỆU NĂNG CAO CRAY-XC40

Ngô Văn Mạnh [1], Nguyễn Thị Hiền[2], Nguyễn Xuân Hoài[3], Đặng Văn Nam[4], Nguyễn Việt Huy[5]

Tóm tắt: *Deep Learning (DL) đang trở thành một công cụ quan trọng cho nghiên cứu và được ứng dụng vào nhiều lĩnh vực khác nhau trong cuộc sống. Ứng dụng DL trong bài báo dự báo, cảnh báo liên quan đến khí tượng thủy văn đang là một hướng nghiên cứu tiềm năng và có nhiều thách thức. Với lượng dữ liệu đầu vào lớn và yêu cầu dự đoán nhanh tức thời, tính chính xác cao là những điểm khiến cho mạng nơ ron trong DL trở nên phức tạp và bị hạn chế trong hiệu suất tính toán, thời gian tính toán bị kéo dài so với yêu cầu nghiệp vụ dự báo, cảnh báo thực tế. Tính toán hiệu năng cao (High Performance Computing - HPC) với số lượng nút tính toán lớn được sử dụng để giải quyết các vấn đề hạn chế của DL trong bài toán dữ liệu lớn. Hãng Cray đã cung cấp một module cắm (Cray Programming Environments DL Plugin – Cray PE DL Plugin) cho phép lập trình DL trên môi trường song song cho tính toán hiệu năng cao. Trong bài báo này, nghiên cứu trình bày phương pháp thiết lập cấu hình mạng nơ ron trong DL sử dụng Tensorflow trên nền tảng Cray-XC40.*

Từ khóa: *Công cụ Cray PE DL, học sâu.*

Ban Biên tập nhận bài: 12/12/2019 Ngày phản biện xong: 05/1/2020 Ngày đăng bài: 25/01/2020

1. Đặt vấn đề

Trí tuệ nhân tạo (AI) đã thay đổi cách thức mà các viện nghiên cứu và các ngành công nghiệp giải quyết một loạt các vấn đề phức tạp. Đặc biệt, Deep Learning (DL) với mạng nơ ron là một công cụ mạnh để trích xuất thông tin từ bộ dữ liệu lớn thông qua hoạt động phân loại, dự đoán và hồi quy. DL cũng có tiềm năng trong các hoạt động phân tích chủ quan, giúp trả về kết quả tính toán lại chính xác hơn. Các mạng nơ ron sâu đòi hỏi lượng tính toán rất lớn, có thể mất vài tuần nếu chỉ được thực hiện trên một nút CPU hoặc GPU. Đây là một trong những rào cản chính trong việc áp dụng DL vào thực tế.

Kỹ thuật giảm dần ngẫu nhiên (*Stochastic gradient descent - SGD*) là kỹ thuật tối ưu hóa thường được sử dụng nhất để đào tạo các mạng nơ ron sâu. Quá trình đào tạo đòi hỏi một tập dữ liệu lớn, các thông tin của mỗi mẫu đều được gán nhãn. Một bước của SGD sử dụng một tập con ngẫu nhiên của bộ dữ liệu, được gọi là một lô, để tính toán các đạo hàm riêng cho mỗi tham số điều chỉnh được trong mạng. Các đạo hàm riêng này (hoặc các biến thiên riêng), đo sự khác biệt giữa đầu ra của mạng nơ-ron và các giá trị quan sát được (nhãn). Mỗi mẫu trong tập hợp con ngẫu nhiên lại tạo ra các biến thiên đạo hàm riêng. Tất cả các đạo hàm riêng của mỗi mẫu sẽ được tính trung bình và giá trị trung bình này được sử dụng để cập nhật các tham số mạng cho bước SGD tiếp theo. SGD thường thay đổi khi sử dụng các công cụ tối ưu hóa mới (là các phương thức dùng để cập nhật mô hình để tính toán các giá trị đạo hàm riêng).

SGD có thể được song song hóa bằng cách chia đều một số lượng đủ lớn các lô nhỏ cho một tập các tiến trình xử lý. Mỗi tiến trình sẽ tính toán đạo hàm cục bộ và sau đó gửi kết quả để tính toán đạo hàm trung bình toàn cục. Các tham số mạng nơ-ron sau đó được cập nhật với các giá trị đạo hàm tính được này. Kỹ thuật này được gọi là SGD song song dữ liệu đồng bộ (*Synchronous data parallel SGD - SSGD*).

Có thể giảm thời gian đào tạo DL sử dụng

[1]*Trung tâm Thông tin và Dữ liệu khí tượng thủy văn*
[2]*Học viện Kỹ thuật quân sự*
[3]*Viện AI Việt nam*
[4]*Đại học Mỏ-Địa Chất*
Email: nguyenthihienqn@gmail.com

SSGD bằng cách tăng kích thước lô toàn cục (tổng trên tất cả các tiến trình) và tăng kích thước bước SGD, còn được gọi là tốc độ học (*learning rate*). Các lỗi trên các đạo hàm trung bình toàn cục sẽ giảm khi đào tạo bằng nhiều mẫu hơn. Lỗi giảm cho phép cập nhật nhiều hơn cho mô hình tại mỗi bước, từ đó dẫn đến sự hội tụ nhanh hơn. Việc tăng kích thước các lô chỉ đến một giới hạn nhất định khi đó sự hội tụ sẽ chậm đi hoặc không hội tụ nữa [1-5].

Module cắm Cray PE DL Plugin giải quyết vấn đề học song song thông qua một kết hợp các cải tiến thuật toán và tối ưu hóa cao hoạt động giao tiếp dựa trên giao diện truyền thông điệp (*Message Passing Interface – MPI*). So với các khung DL tính toán các đạo hàm trung bình toàn cục chỉ dựa trên một tính toán MPI chung Allreduce thì Cray PE DL Plugin vượt trội hơn hẳn. Nghiên cứu này mô tả các giải pháp được sử dụng trong Cray PE DL Plugin và làm thế nào các giải pháp này tạo ra hiệu suất tối ưu trên nền tảng Cray. Nghiên cứu thảo luận về việc áp dụng Cray PE DL Plugin vào TensorFlow , một khung DL phổ biến và đánh giá các cải tiến hiệu suất trên nền tảng Cray-XC40 với Bộ xử lý Intel KNL. Kết quả cho thấy hiệu suất thời gian tính toán giảm khoảng 10 (khi sử dụng 8 nút tính toán) cho bài toán dự báo tốc độ gió sử dụng DL dựa trên hồi quy (*Long Short Term Memory - LSTM*).

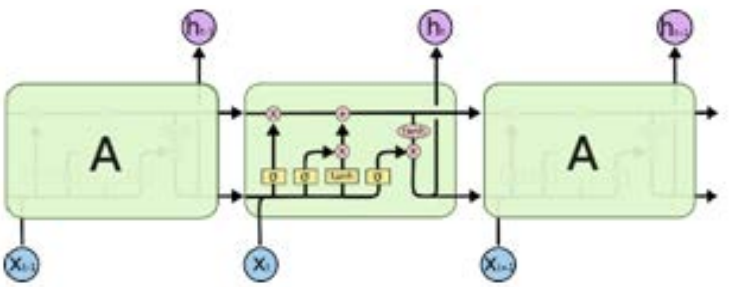
2. Phương pháp triển khai

2.1. Long Short Term Memory - LSTM

Mạng bộ nhớ dài-ngắn (*Long Short Term Memory networks*), thường được gọi là LSTM - là một dạng đặc biệt của RNN, nó có khả năng học được các phụ thuộc xa. LSTM được giới thiệu bởi Hochreiter & Schmidhuber (1997) [6], và sau đó đã được cải tiến và phổ biến bởi rất nhiều người trong ngành. LSTM là một trong những mạng thần kinh nhân tạo được sử dụng phổ biến trong phân tích dữ liệu chuỗi thời gian (*time-series*).

LSTM được thiết kế để tránh được vấn đề phụ thuộc xa (*long-term dependency*). Việc nhớ thông tin trong suốt thời gian dài là đặc tính mặc định của chúng, chứ không cần phải đào tạo nó để có thể nhớ được. Tức là ngay nội tại của nó đã có thể ghi nhớ được mà không cần bất kì can thiệp nào.

Mọi mạng hồi quy đều có dạng là một chuỗi các mô-đun lặp đi lặp lại của mạng nơ-ron. Với mạng RNN chuẩn, các mô-dun này có cấu trúc rất đơn giản, thường là một tầng tanh. LSTM cũng có kiến trúc dạng chuỗi như vậy, nhưng các mô-đun trong nó có cấu trúc khác với mạng RNN chuẩn. Thay vì chỉ có một tầng mạng nơ-ron, chúng có tới 4 tầng tương tác với nhau một cách đặc biệt.



Hình 1. Mạng LSTM

2.2. Nền tảng triển khai DL trên Cray

a) Nền tảng hỗ trợ DL

Một số nền tảng được xây dựng sẵn hỗ trợ cho DL gồm:

TensorFlow

TensorFlow là một lớp phần mềm mã nguồn mở cung cấp một bộ các quy tắc tính toán số hiệu suất cao. TensorFlow có thể dễ dàng triển khai trên nhiều nền tảng phần cứng, từ các CPU thông thường và các bộ xử lý đồ họa GPU cho đến các bộ xử lý Tensor thiết kế chuyên biệt (*Tensor Processing Units - TPU*). Ngoài ra, TensorFlow cũng có thể được triển khai trên các cụm máy tính (*cluster*) bao gồm các máy chủ tính toán hiệu suất cao và các kết nối nội bộ. Tuy nhiên, cơ sở hạ tầng phần cứng và giao tiếp đóng một vai

trò quan trọng trong việc mở rộng TensorFlow trên một số lượng lớn các nút tính toán.

gRPC

gRPC là một lớp lớp gọi thủ tục từ xa mã nguồn mở (*Remote Procedure Call layer - RPC*) ban đầu được phát triển bởi Google. gRPC cung cấp một lớp trừu tượng để phát triển các dịch vụ và các ứng dụng phân tán và được xếp lớp trên giao thức HTTP/2. gRPC cũng cung cấp một loạt các tính năng cho phép giao tiếp, đồng bộ hóa và kiểm soát luồng giữa máy khách và máy chủ trong một ứng dụng phân tán. gRPC là một trong những giao thức giao tiếp được sử dụng trong khung TensorFlow của Google.

Horovod

Horovod là một khung DL phân tán, mã nguồn mở cho TensorFlow của Uber. Horovod sử dụng giao diện truyền thông điệp MPI để thiết lập một hạ tầng phân tán cho TensorFlow. Trong nội bộ, hoạt động giảm thiểu toàn cục được thực hiện bằng một công cụ Allreduce dạng vòng để sử dụng băng thông truyền tin được cung cấp bởi một kết nối cụm hiệu suất cao điển hình. Các phiên bản gần đây của Horovod sử dụng các lớp truyền tin NCCL và NCCL2 của NVIDIA để tối ưu hóa hiệu suất truyền tin trên các hệ thống hiện đại với nhiều GPU trên mỗi nút.

b) Giải pháp thiết kế

Nhiều khung song song hóa cho DL, chẳng hạn như gRPC trong TensorFlow, gồm hai lớp xử lý. Các hoạt động xử lý máy chủ tham số (*Parameter Server – PS*) thu thập biến thiên từ các hoạt động xử lý máy trạm, tính toán biến thiên trung bình toàn thể, cập nhật các tham số mạng và gửi các giá trị tham số mới tới các máy trạm. Thông thường người dùng có thể chọn số lượng hoạt động xử lý PS. Chỉ có một hoặc một số giới hạn các hoạt động xử lý PS trên một số lượng lớn máy trạm sẽ gặp phải các vấn đề về hiệu suất và hạn chế quy mô. Cấu hình kiểu này thiết lập một mẫu giao tiếp nhiều-đến-ít, gây tắc nghẽn hầu hết các mạng. Một số lượng hạn chế các hoạt động xử lý PS cũng sẽ gặp khó khăn trong việc gửi các giá trị tham số cập nhật đủ nhanh để theo kịp nhu cầu của máy trạm. Tăng số lượng các hoạt động xử lý PS có thể làm giảm các nút thắt cổ chai trong truyền tin và cập nhật tham số. Tuy nhiên, sử dụng quá nhiều các hoạt động xử lý PS lại dẫn đến các mẫu giao tiếp nhiều-nhiều, sẽ không đáp ứng số lượng lớn các nút. Xác định số lượng tối ưu các hoạt động xử lý PS sẽ rất mất công sức của người dùng. Dùng gRPC trong TensorFlow, người dùng còn phải cung cấp tên nút và số cổng, như vậy lại nảy sinh các vấn đề về khả năng sử dụng.

Cray PE DL Plugin xử lý các vấn đề về khả năngsử dụng và hiệu suất mở rộng trong TensorFlow và các khung DL tương tự. Không có hoạt động xử lý PS khi sử dụng Cray PE DL Plugin. Mỗi xử lý là một máy trạm, và một hoạt động giảm thiểu toàn thể tùy chỉnh thay thế cho hoạt động tính toán biến thiên trung bình của toàn bộ các hoạt động xử lý PS. Mỗi máy trạm sau đó có thể dễ dàng tính toán cập nhật tham số mạng, việc này thường chỉ tốn một phần nhỏ của thời gian thực hiện. Thuật toán trong Hình 1 mô tả sơ bộ cách học song song dữ liệu sử dụng Cray PE DL Plugin. Giảm thiểu tùy chỉnh được tối ưu hóa cụ thể cho hoạt động DL và có thể thấy hiệu suất cao hơn 35% so với MPI Iallreduce() mặc định có sẵn trong Cray MPICH khi kích thước thông điệp và vị trí xử lý là tương đương. Ngoài ra để cải thiện hiệu suất truyền tin ở một quy mô lớn hơn, thiết lập giảm thiểu tùy chỉnh cũng cung cấp khả năng tuyệt vời chồng lớp truyền tin/tính toán. Khả năng tạm ẩn truyền tin trong pha tính toán biến thiên trung bình đóng vai trò chính trong việc cải thiện thời gian cho đào tạo phân tán.

Require: N = total number of epochs
Require: n = total number of training samples
Require: k = number of MPI ranks
Require: b = number of training samples in a batch per process
1: **for** epoch = $1 \cdots N$ **do**
2: **for** step = $1 \cdots n/(bk)$ **do**
3: $g_{step} \leftarrow$compute_gradients(local_batch$_{step}$)
4: $G_{step} \leftarrow$mc.gradients(g_{step})
5: $loss_{step} \leftarrow$apply_gradients(G_{step})

Hình 2. Code giả lập cho thuật toán đào tạosong song dữ liệu. Cray PE DL Plugin được trình bày bằng mc, và hàm trung bình đạo hàm là hàm mc.gradients().

Trong đó N là tổng số chu kỳ; n là số lượng mẫu đào tạo; k là số cấp MPI; b là số mẫu đào tạo trong một lô dữ liệu trong một lần xử lý.

Không cần điều chỉnh TensorFlow để sử dụng Cray PE DL Plugin cho song song hóa. Tính năng TenorFlow Op được sử dụng để thêm các bước truyền tin cần thiết vào đồ thị thực hiện một cách tối ưu (tài liệu có tại https://www.tensorflow.org/extend/adding_an_op). Người dùng có thể bắt đầu với một TensorFlow nối tiếp hoặc một tập lệnh client khung khác rồi gọi thêm các thành phần cần thiết để khởi tạo, truyền tin và kết thúc. Đối với các tình huống yêu cầu nhiều giảm thiểu biến thiên cùng lúc, các nhóm luồng (thread) giảm thiểu được sử dụng để tăng tốc độc lập với một vài hoạt động gọi đơn giản. Giao diện C / C ++ và Python 2/3 có sẵn trong Cray PE DL Plugin.

Cray PE DL Plugin đã có sẵn trong gói Cray Developer Toolkit - CDT được cài cho các hệ thống Cray XC. Bản hiện tại CDT 19.09 hỗ trợ Keras, TensorFlow 1.3.1 và kiến trúc dựa trên Intel®CPU và GPU NVIDIA.

c) Bộ tập lệnh trong Cray PE DL plugin

Các bước cần có trong Cray PE DL Plugin bao gồm:

• Khởi tạo Cray PE DL Plugin.

• Khởi tạo các giá trị tham số mô hình ban đầu: Chỉ định số lượng nhóm, luồng, kích thước mô hình.

• Sử dụng Cray PE DL Plugin để truyền các biến thiên sau khi tính toán các biến thiên và thực thi mô hình.

• Kết thúc Cray PE DL Plugin.

Trong phần này sẽ trình bày chi tết cho việc áp dụng cho một tập lệnh bằng Python cho phép sử dụng Keras thực thi các mô hình học máy DL.

- Khởi tạo: Bước đầu tiên là khởi tạo Cray PE DL Plugin. Điều này được thực hiện bằng cách trước tiên import module rồi thiết lập môi trường ban đầu như hình 3:

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
# BO SUNG CHO CRAY
import math
import tensorflow as tf
import dl_comm.keras as cdl

config = tf.ConfigProto()
#config.gpu_options.allow_growth = True
#config.gpu_options.visible_device_list = str(cdl.local_rank())
#config.gpu_options.per_process_gpu_memory_fraction = 0.8
K.set_session(tf.Session(config=config))
# KET THUC BO SUNG CHO CRAY
```

Hình 3. Khởi tạo Cray PE DL Plugin

Cray PE DL Plugin sử dụng cấu hình cho cả CPU và GPU, để thực hiện sử dụng GPU cho tính toán cần thiết lập tham số cấu hình ban đầu như hình 4:

```
config = tf.ConfigProto()
#config.gpu_options.allow_growth = True
#config.gpu_options.visible_device_list = str(cdl.local_rank())
#config.gpu_options.per_process_gpu_memory_fraction = 0.8
```

Hình 4. Thiết lập tham số cấu hình ban đầu

- Khởi tạo các tham số mô hình ban đầu: Với Keras, cần thiết lập các thông số ban đầu của mô hình sẽ sử dụng như hình 5:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Hình 5. Thiết lập các thông số ban đầu

- Tổng hợp biến thiên: Hoạt động truyền tin và tính toán chuyên sâu này được tối ưu hóa cao trong Plugin Cray PE DL Plugin. Hoạt động này được đặt giữa hoạt động tính toán biến thiên và cập nhật mô hình, được cấu hình như hình 6.

```
# BO SUNG CHO CRAY
opt = keras.optimizers.Adadelta(1.0 * cdl.get_nranks())

# Wrap the optimizer to use the Plugin
opt = cdl.DistributedOptimizer(opt)
# KET THUC BO SUNG CHO CRAY
```

Hình 6. Sử dụng tối ưu hóa trong Cray

Một tập lệnh học nối tiếp thường sử dụng một phương thức tối thiểu hóa minimize() của một đối tượng tối ưu hóa optimizer. Phương thức này tính toán biến thiên và cập nhật mô hình với các biến thiên này. Việc cập nhật kết quả tính toán của các bước song song được thực hiện bởi hàm callback như hình 7:

```
# BO SUNG CHO CRAY
# Add callback to broadcast initial variables
callbacks = [cdl.BroadcastVariablesCallback(0, K)]
```

Hình 7. Cập nhật kết quả tính toán song song

Thực hiện học máy của mô hình và đánh giá kết quả của mô hình được thực hiện như hình 8:

```
model.fit(x_train, y_train,
          batch_size=batch_size,
# BO SUNG CHO CRAY
          callbacks=callbacks,
# KET THUC BO SUNG CHO CRAY
          epochs=epochs,
          verbose=verbose_level,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Hình 8. Học máy và đánh giá kết quả mô hình

- Kết thúc: Bước bắt buộc cuối cùng để chuyển đổi một tập lệnh đào tạo nối tiếp là kết thúc Cray PE DL Plugin, tương tự như việc kết thúc MPI. Khi rank của nút trả về là 0 khi đó quá trình tính toán song song kết thúc như hình 9.

```
# BO SUNG CHO CRAY
if cdl.get_rank() == 0:
  print('Test loss:', score[0])
  print('Test accuracy:', score[1])
# KET THUC BO SUNG CHO CRAY
```

Hình 9. Kết thúc Cray PE DL Plugin

d) Triển khai, cài đặt Cray PE DL Plugin trên Cray-XC40

Hiệu suất tốt nhất đạt được với một cấp MPI trên mỗi nút. Cray PE DL Plugin phải được cấu hình để sử dụng 2-4 luồng (thread) truyền tin. Trong vài trường hợp với các nút GPU, hiệu suất có thể được cải thiện bằng cách sử dụng lên đến 8 luồng. Đối với đào tạo MKL và MKL-DNN, quan trọng là không đặt OMP_NUM_THREADS quá cao, nếu không các core có thể bị đăng ký vượt mức. Ví dụ: nếu có 36 core vật lý trên một nút, hiệu suất tối ưu đạt được với OMP_NUM_THREADS=34, nên để lại 2 core/luồng để liên lạc với Cray PE DL Plugin. Ngoài ra, với TensorFlow và ví dụ tf_cnn_benchmarks, num_intra_threads nên được đặt để phù hợp với giá trị OMP_NUM_THREADS, và num_inter_threads thường được đặt từ 1-3 luồng tùy thuộc vào số lượng HyperThreads có trên mỗi core. Đối với các KNL CPU, tốt nhất để lại một HyperThread rảnh trên mỗi core. Biến môi trường KMP_BLOCKTIME có thể giúp cải thiện hiệu suất đôi chút nếu được đặt là 0 hoặc 30.

Đối với các nút GPU, số lượng luồng CUDA được sử dụng để nhớ đệm dữ liệu đến chủ thể tính toán có thể được điều chỉnh thông qua biến môi trường ML_COMM_NUM_CUDA_STREAMS và số lượng bản sao mà mỗi luồng thực hiện có thể được thay đổi với biến môi trường ML_COMM_CPY_PER_CUDA_STREAM. Các cài đặt mặc định 2 và 8, tương ứng, theo thử nghiệm là tốt nhất cho gần như tất cả các tình huống.

3. Thực nghiệm

Trong thực nghiệm của nghiêm cứu sẽ sử dụng mạng DL long short term memory-LSTM cho bài toán dự đoán tốc độ gió trong 18 giờ tiếp theo.

Dữ liệu thực nghiệm là dữ liệu quan trắc tốc độ gió từ 01 tháng 1 năm 2014 đến 31 tháng 12 năm 2019, với tần suất quan trắc 3h một lần, tại 13 trạm quan trắc: Hà Giang, Cao Bằng, Tuyên Quang, Hòa Bình, Nam Định, Hà Đông, Phú Liễn, Lạng Sơn, Bãi Cháy, Tiên Yên, Móng Cái, Bạch Long Vĩ, Hội Xuân.

Số lượng mẫu học là 15,200 mẫu, số lượng mẫu kiểm tra là 500 mẫu. Thực nghiệm sẽ thực thi LSTM với các epoch =500, 800.

Thiết lập mô hình LSTM trên máy chủ thông thường và máy chủ Cray-XC40 sử dụng Cray PE DL Plugin. Thông số máy chủ thường CPU (2 Multi cores (28 thread) Intel Xeon E5-2690 v4s), RAM (64 GB). Đối với máy Cray-XC40, 1 nút có CPU (2 cores (36 thread), Intel Xeon E5-2697 v4 18C 2.3 GHz), RAM (16 GB). Cấu hình thực thi LSTM trong thực nghiệm trên Cray-XC40 sử dụng 8 nút, mỗi nút sử dụng 34 thread.

```
"""##Xây dựng mô hình LSTM"""
#Stacked LSTM (Nhiều lớp LSTM)
model = Sequential()
model.add(LSTM(50, activation='relu',
               return_sequences=True,
               batch_input_shape=(None,8, 13)))
model.add(LSTM(50,
               return_sequences=False,
               activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(13))
model.compile(optimizer='adam',
              loss='mse',
              metrics=['accuracy'])
model.summary()
"""##Huấn luyện mô hình"""
#Fit model Vanilla LSTM
start_time = time.time()
epochTime = 500
model.fit(x_train, y_train, batch_size=3, epochs=epochTime, validation_split=validation_split)
end_time = time.time()
```

Hình 10. Cấu hình LSTM trên máy chủ thường

```
"""##Xây dựng mô hình LSTM"""
validation_split = 0.05 #Tỷ lệ % tách tập Validation trong tập Test
#epochTimes = 12.0
epochTimes = 500
# CRAY ADDED
epochs = int(math.ceil(epochTimes / cdl.get_nranks()))
# END CRAY ADDED
#Stacked LSTM (Nhiều lớp LSTM)
model = Sequential()
model.add(LSTM(50, activation='relu',
               return_sequences=True,
               batch_input_shape=(None,8, 13)))
model.add(LSTM(50,
               return_sequences=False,
               activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(13))
# CRAY ADDED
opt = keras.optimizers.Adadelta(1.0 * cdl.get_nranks())
# Wrap the optimizer to use the Plugin
opt = cdl.DistributedOptimizer(opt)
#END CRAY ADDED
model.compile(optimizer = opt,
              loss='mse',
              metrics=['accuracy'])
model.summary()
# CRAY ADDED
# Add callback to broadcast initial variables
callbacks = [cdl.BroadcastVariablesCallback(0, K)]
```

Hình 11. Cấu hình LSTM trên máy Cray-XC40

```
#!/bin/hash
#PBS -q aiq
#PBS -l nodes=4
#PBS -S /bin/bash
#PBS -N ai
#PBS -l walltime=0:10:0
#PBS -j oe

cd $PBS_O_WORKDIR
module load craype-network-aries
module load cray-mpich
module load craype-dl-plugin-py3

export OMP_NUM_THREADS=34
export MPICH_MAX_THREAD_SAFETY=multiple
export MPICH_NEMESIS_ASYNC_PROGRESS=ML

beg_secs=`date +"%s"`
echo "START JOB AT" `date +"%d%m%y %H:%M"`
date
time aprun -n 4 -N 1 -j 1 -d $OMP_NUM_THREADS -cc none python3.6 /cray_home/cntt/ai/code/VNMHA_AT_FINAL.py
date
echo "END JOB AT" `date +"%d%m%y %H:%M"`
export end_secs=`date +"%s"`
(( wallsecs = end_secs - beg_secs ))
echo "time taken by run in seconds is " $wallsecs
date
exit 0
```

Hình 12. Cấu hình PBS để submit job trên Cray-XC40

4. Phân tích kết quả

Trong phần này, nghiên cứu sẽ phân tích và đánh giá kết quả đạt được khi thực thi thực nghiệm trên máy chủ thường và Cray-XC40. Kết quả dự đoán với tính chính xác của LSTM khi chạy trên máy chủ thường và Cray-XC40 là tương đương nhau với khoảng sai số trung bình giữa máy chủ thường và Cray-XC40 khoảng 0.03, kết quả chi tiết tại bảng 1, bảng 2 và bảng 3 là kết quả sai số trung bình.

Bảng 1. Sai số dự báo của LSTM trên Cray-XC40

STT	Station	MAE (Epoch = 500)	MAE (Epoch = 800)
1	48805 - HA GIANG	0.81523	0.9741
2	48808 - CAO BANG	0.77397	0.65291
3	48812 - TUYEN QUANG	0.55088	0.67463
4	48818 - HOA BINH	0.66411	0.52507
5	48823 - NAM DINH	0.73743	0.72945
6	48825 - HA DONG	0.42744	0.72918
7	48826 - PHU LIEN	0.85284	0.71751
8	48830 - LANG SON	0.58352	0.53875
9	48833 - BAI CHAY	0.74615	0.74463
10	48837 - TIEN YEN	0.81491	0.86019
11	48838 - MONG CAI	0.85648	0.88636
12	48839 - BACH LONG VY	0.89402	0.68479
13	48842 - HOI XUAN	0.82788	0.88395
14	All Station	0.73422417	0.738584

Bảng 2. Sai số dự báo của LSTM trên máy chủ thường

STT	Station	MAE (Epoch = 500)	MAE (Epoch = 800)
1	48805 - HA GIANG	0.81484	0.82883
2	48808 - CAO BANG	0.71858	0.85846
3	48812 - TUYEN QUANG	0.56044	0.66563
4	48818 - HOA BINH	0.58697	0.67589
5	48823 - NAM DINH	0.72503	0.75761
6	48825 - HA DONG	0.46304	0.55307
7	48826 - PHU LIEN	0.83042	0.91817
8	48830 - LANG SON	0.58961	0.53911
9	48833 - BAI CHAY	0.82476	0.79877
10	48837 - TIEN YEN	0.84871	0.86146
11	48838 - MONG CAI	0.80696	0.8723
12	48839 - BACH LONG VY	0.74131	0.73887
13	48842 - HOI XUAN	0.8547	0.97105
14	All Station	0.72041706	0.77225093

Bảng 3. Sai số dự báo của LSTM trung bình cho tất cả các trạm

STT	Machine	MAE (Epoch = 500)	MAE (Epoch = 800)
1	Máy chủ cứng	0.70315556	0.77225093
2	Cray-XC40	0.73422417	0.738584

Đối với hiệu suất về thời gian thực thi mô hình LSTM thì trên Cray-XC40 cho kết quả nhanh trung bình gấp 10 lần so mới máy chủ thông thường (trong trường hợp Cray-XC40 sử dụng 8 nút, mỗi nút sử dụng 34 thread và máy chủ thường có 28 thread). Kết quả chi tiết ở bảng 4.

Bảng 3. Sai số dự báo của LSTM trung bình cho tất cả các trạm

STT	Cấu hình	Thời gian (phút)
2	Cray-XC40, 8 nút, epoch = 500	34.8
3	Cray-XC40, 8 nút, epoch = 800	54.1
5	Máy chủ cứng, epoch = 500	382.68
6	Máy chủ cứng, epoch = 800	577.04

Với kết quả về tính chính xác giống nhau khi sử dụng LSTM trên môi trường máy chủ thường và Cray-XC40 và hiệu năng về thời gian học của Cray-XC40 tốt hơn hẳn máy chủ thường (giảm 10 lần, nếu tăng số lượng nút tiếp thì thời gian tiếp tục giảm). Với thời gian như trong bảng 4, LSTM hoàn toàn có thể được sử dụng được vào trong nghiệp vụ dự báo khí tượng thủy văn. Đây chính là ý nghĩa khi sử dụng Cray PE DL cho các bài toán DL trong nghiệp vụ thực tế.

5. Kết luận

Nghiên cứu đã trình bày về ứng dụng của công cụ Cray PE DL Plugin trong bài toán DL, Cray PE DL cho phép mở rộng quy mô học DL tới một số lượng lớn các nút trong hệ thống qua đó giảm đáng kể thời gian học cho mạng nơ ron và tăng tính hiệu quả của DL khi đưa vào ứng dụng thực tế trong bài toán dự báo liên quan đến Khí tượng thủy văn. Nhất là trong giai đoạn hiện nay khi yêu cầu của xã hội cần các bản tin cảnh báo cực ngắn trong thời gian 30 phút - 1h.

Lời cảm ơn: *Nghiên cứu này được hỗ trợ bởi đề tài "Nghiên cứu cơ sở khoa học và giải pháp ứng dụng trí tuệ nhân tạo để nhận dạng, hỗ trợ dự báo và cảnh báo một số hiện tượng khí tượng thủy văn nguy hiểm trong bối cảnh biến đổi khí hậu tại Việt Nam", mã số BĐKH.34/16-20".*

Tài liệu tham khảo

1. Das, D., Avancha, S., Mudigere, D., Vaidynathan, K., Sridharan, S., Kalamkar, D., Kaul, B., Dubey, P., (2016), *Distributed Deep Learning Using Synchronous Stochastic Gradient Descent. ArXiv e-prints.*

2. Kingma, D.P., Ba, J., (2014), *Adam: A Method for Stochastic Optimization. ArXiv e-prints*.

3. Iandola, F.N., Ashraf, K., Moskewicz, M.W., Keutzer, K., (2015), *Fire-Caffe: near-linear acceleration of deep neural network training on compute clusters.* ArXiv e-prints.

4. Mendygral, P., Hill, N., Kandalla, K., Moise, D., Balma, J., Marcel Schongens, M., (2018), *High Performance Scalable Deep Learning with the Cray Programming Environments Deep Learning Plugin*. CUG 2018.

5. Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z.M., Tie-Yan Liu, T.Y., (2016), *Asynchronous Stochastic Gradient Descent with Delay Compensation*. ArXiv e-prints.

6. Hochreiter, S., Jurgen Schmodhuber, J., (1997), *Long short-term memory. Neural computation, 9(8)*, 1735-1780.

7. Chen, J., Pan, X., Monga, R., Bengio, S., Jozefowicz, R., (2017), *Revisiting Distributed Synchronous SGD. ArXiv e-prints.*

8. You, Y., Gitman, I., Ginsburg, B., (2017), *Scaling SGD batch size to 32k for imagenet training*. ArXiv e-prints.

[9] Peter, A., Whigham, Crapper, P.F., (2001), *Modeling rainfall-runoff using genetic programming. Mathematical and Computer Modelling*, 33 (6-7), 707-721.

IMPROVING THE PERFORMANCE OF DEEP LEARNING IN HIGH-PERFORMANCE COMPUTING SYSTEMCRAY-XC40

Ngo Van Manh[1], Nguyen Thi Hien[2], Nguyen Xuan Hoai[3], Dang Van Nam[4], Nguyen Viet Huy[1]

[1]Center for Hydro-Meteorological Data and Information

[2]Le Quy Don Technical University

[3]AI Academy Vietnam

[4]Hanoi University of Mining and Geology

Abstract: *Deep Learning (DL) is becoming an important tool in research and is applied in many different areas. The application of ML in forecasting and warning bulletins of Meteorology and Hydrology is a potential and challenging research field. Due to large input data and requirement for instantaneous and high accuracy prediction, the neural network in DL becomes complex and limited in computing performance, thus the calculating time is prolonged compared with requirements of actual forecasting and warnings. High Performance Computing (HPC) with a large number of computing nút is used to solve the limitation of DL in big-data subject. Cray has provided a plug-in module (Cray Programming Environments DL Plugin - Cray PE DL Plugin) that enables DL programming in parallization environment for high-performance computing. In this paper, the study presents the method of configuring neuron network in DL using Tensorflow on Cray XC-40 platform.*

Keywords: *Cray PE DL plguin, deep learning.*