УДК 004.855

Ушаков Ю.А., Полежаев П.Н., Шухман А.Е., Порохненко Ю.С., Чернова Е.В., Очередько О.О.

Оренбургский государственный университет, г. Оренбург, Россия

ЭФФЕКТИВНОЕ ПРИМЕНЕНИЕ ОБЛАЧНЫХ РЕСУРСОВ ПРИ ОБУЧЕНИИ И ИСПОЛЬЗОВАНИИ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ*

Аннотация

Нейронные сети имеют все большее значение в науке и технике. Построением и обучением нейронных сетей занимаются многие научные коллективы и инженеры. Для обеспечения эффективного вычисления нейронных сетей многие ученые используют облачные ресурсы с почасовой оплатой, выделенные GPU ускорители. Данная работа посвящена исследованию эффективности обучения нейронных сетей на GPU ускорителях и обычных серверных процессорах. Проведен обзор всех популярных фреймворков для создания и выполнения расчетов на нейронных сетях, рассмотрены их преимущества и недостатки, а также особенности работы с GPU и в кластерных режимах. Рассмотрены преимущества использования контейнеризации при GPU расчетах на облачных системах с почасовой оплатой, возможность существенно упростить первоначальную настройку и свести к минимуму рассогласование версий драйверов и контейнеров. Также рассмотрено влияния контейнеризации на производительность обучения. Проведено планирование эксперимента, описаны способы запуска вычислений и сбора статистики как внутри контейнеров, так и на хосте. Рассмотрены различные стоимостные показатели прогона нейронной сети, показано ускорение при использовании нескольких GPU, совместное использование GPU и CPU, исследована различная потребность в оперативной памяти для CPU и GPU на одних и тех же моделях и данных. Показана возможность эффективного обучения моделей только на процессорах и завышенные стоимостные характеристики использования GPU ускорителей.

Ключевые слова

Глубокие нейронные сети; облачные вычисления; контейнеры; виртуальные машины; параллельные вычисления.

Ushakov Yu.A.[1], Polezhaev P.N., Shukhman A.E., Porokhnenko Yu.S., Chernova E.V., Ocheredko O.O.

Orenburg State University, Orenburg, Russia

EFFICIENT APPLICATION OF CLOUD RESOURCES FOR TRAINING AND USING DEEP NEURAL NETWORKS

Abstract

Neural networks have great importance in science and technology. Many research teams and engineers are involved in the design and training of neural networks. Many scientists use cloud resources and dedicated GPU accelerators with hourly payment to make the efficient calculations of neural networks. This paper is devoted to research the efficiency of training neural networks on GPU accelerators and server CPUs. We make review of popular frameworks for creation and calculations execution on neural networks. The advantages and disadvantages of frameworks, the features of their work with GPUs and work in cluster mode are considered in this paper. In addition, we consider the advantages of containerization usage with GPU computations on hourly paid cloud systems. One of them is the opportunity to greatly simplify the initial configuration and minimize the discrepancy between the versions of drivers and containers. Also, we consider the impact of containerization on the performance of neural network training. The experiment was planned, and the methods for

* **Труды II Международной научной конференции «Конвергентные когнитивно-информационные технологии» (Convergent'2017), Москва, 24-26 ноября, 2017**

Proceedings of the II International scientific conference "Convergent cognitive information technologies" (Convergent'2017), Moscow, Russia, November 24-26, 2017

starting computations and collecting statistics both inside the containers and on the host are described. Different cost parameters of the neural network execution are considered. The experiment shows the acceleration of neural network training when several GPUs and/or CPUs are used. The different need for RAM for CPUs and GPUs is investigated on the same models and data. The opportunity of the efficient model training only on CPUs and overestimated cost characteristics of the use of GPU accelerators are shown.

Keywords

Deep neural networks; cloud computing; containers; virtual machines; parallel computing.

Введение

Распространение нейронных сетей в науке и технике уже давно вышло за рамки чисто академического интереса. Они используются практически во всех областях информационных технологий, начали заменять юристов и техническую поддержку, позволили создать беспилотный транспорт и автоматизировать множество различных процессов. Но несколько препятствий на пути дальнейшего развития и распространения нейронных сетей не дают применять и изучать их повсеместно – это, во-первых, высокие требования к вычислительным ресурсам, необходимым для активно используемых сейчас глубоких нейронных сетей, во-вторых – высокий порог вхождения в технологии.

Фреймворки для вычислений над графами представляют нейронные сети в виде ориентированных графов, в которых листовые вершины представляют входы/выходы или параметры сети, остальные вершины – векторные операции, такие как матричное сложение/умножение или свертка. Самыми популярными символьными фреймворками для реализации нейронных сетей в настоящий момент являются Google TensorFlow, Theano, Microsoft CNTK, MXNET, Torch, VELES, Neon [1]. Порог вхождения при использовании довольно высок – необходимо уметь писать на языке программирования для фреймворка, устанавливать драйверы и библиотеки в Linux, совмещать навыки системного администратора и разработчика (devops). Microsoft предлагает более адаптированные к широкому использованию продукты, но также требует применения собственного инструментария. Для понижения порога вхождения были разработаны фреймворки для высокоуровневой разработки программного обеспечения для нейронных сетей, которое использует API и готовые библиотеки, реализующие большинство функций обработки входных данных, развертывания и обучения сети – Keras, Lasagne, Caffe.

Анализ существующих фреймворков для нейронных сетей

В настоящее время имеется ряд разработок, представляющих собой высокоуровневые инструментальные комплексы проектирования прикладных систем, использующих нейронные сети. Математические пакеты Wolfram и Matlab обрабатывают данные с учетом моделирования и прогнозирования, они могут работать только с небольшими нейронными сетями. Neural Network Package от Wolfram поддерживает сети прямого распространения (Feed Forward Neural Networks, FFNN), сети радиально-базисных функций (Radial Basis Function network, RBFN) и сети Хопфилда (Hopfield Network, NH). Пакет расширения MATLAB Neural Network Toolbox также работает с рекуррентными нейронными сетями (Recurrent Neural Network, RNN) [2], кроме того он поддерживает параллельные вычисления с помощью Parallel Computing Toolbox.

Подробнее рассмотрим и сравним современные библиотеки и фреймворки, позволяющие работать с нейронными сетями и отвечающие ряду условий, включая наличие открытой лицензии и хорошо продуманную и доступную документацию.

Важным критерием для сравнения является поддержка нескольких типов нейронных сетей. Особый интерес представляют свёрточная (англ. Convolutional Neural Network, CNN), рекуррентная, рекуррентная свёрточная (англ. Recurrent Convolutional Neural Network, RCNN) нейронные сети, сеть прямого распространения, глубокая сеть доверия (Deep Belief Network, DBN), ограниченная машина Больцмана (Restricted Boltzmann Machine, RBM), а также разновидность рекуррентной нейронной сети – долгая краткосрочная память (Long Short-Term Memory, LSTM).

Основным ограничением развития нейронных сетей часто признают высокие вычислительные затраты на их реализацию. Один из методов решения данной проблемы – использование параллельных и распределенных вычислений на специализированном аппаратном обеспечении. Наиболее ярким примером является программно-аппаратная вычислительная архитектура параллельных вычислений CUDA компании NVIDIA. CUDA помогает реализовывать алгоритмы, исполняемые на графических процессорах видеоускорителей. OpenCL – еще один стандарт для разработки приложений для гетерогенных систем, он также используется для параллельных вычислений на центральных и графических процессорах, в том числе и под управлением CUDA [3].

Наличие предварительно обученных моделей является большим плюсом библиотеки, поскольку их можно применять в своих приложениях, экономя время и ресурсы. Чтобы лучше адаптировать модель к новым данным, можно провести дополнительную настройку – обучить нейронную сеть с помощью собственного набора данных.

В сравнительной таблице 1 также указаны платформы, на которых библиотеки способны работать, и их программные интерфейсы (API).

Таблица 1 – Сравнение библиотек для нейронных сетей

Название ПО	Платформа	API	Типы нейросетей	Поддержка GPU	Параллельность	Обученные модели
TensorFlow	Linux, Mac OS, Windows	C++, Python	CNN, RNN, DBN, RBM	CUDA	CPU, GPU	TensorFlow Models
Theano	Linux (Windows, Mac)	Python	CNN ,RNN, DBN, RBM	CUDA, OpenCL	GPU	С помощью Model Zoo в Lasagne
Lasagne	Linux, Mac (Windows)	Python	CNN, RNN, LSTM	CUDA, cuDNN	GPU	Использует модели Caffe
CNTK	Windows, Linux	Python, C++, C#, .NET, Java	CNN, RNN, LSTM, FFNN	CUDA, OpenMP	CPU, GPU	Model Gallery
Keras	Linux, Mac OS, Windows	Python	CNN, RNN, DBN, RBM	CUDA, OpenCL	CPU, GPU (с TensorFlow, Theano, CNTK)	Keras Application
MXNET	Linux, Mac for R and Python, Windows only for R	R, Python, Julia, MATLAB, Scala и Javascript	CNN, RCNN, LSTM, DBN, RBM	CUDA	CPU, GPU	Инструмент, преобразующий модели Caffe в формат MXNET
Deeplearning4j	Linux, Mac OS, Windows, Android	Java, Scala, Clojure, Python (Keras)	CNN, RNN LSTM, DBN, RBM	CUDA, OpenMP	GPU	Импорт моделей из Theano, Caffe, Torch, TensorFlow, используя Keras
Caffe	Linux, Mac OS, Windows	Python, MATLAB	CNN, RCNN, LSTM	CUDA, OpenMP	GPU	Model Zoo
Torch	Linux, Mac OS, iOS Windows, Android,	Lua, C, C++	CNN, RNN, DBN, RBM	CUDA, OpenCL	CPU, GPU	Импорт моделей с помощью LoadCaffe

Самыми быстрыми среди рассмотренных фреймворков являются Theano и Torch, использующий малоизвестный язык Lua. Однако Theano – низкоуровневый фреймворк и больше рассматривается как исследовательская платформа, а не библиотека глубокого обучения [4]. Theano часто используется вместе с библиотеками, имеющими более высокий уровень абстракции, такими как Keras и Lasagne.

Caffe хорошо известен из-за своего набора предобученных моделей нейронных сетей Model Zoo, который могут использовать другие фреймворки и библиотеки с помощью дополнительных инструментов [5]. Однако сам фреймворк слишком громоздкий для больших сетей, он не так хорош для работы с рекуррентными сетями и медленно развивается.

Microsoft Cognitive Toolkit, также известный как CNTK, не годится для коммерческого использования, так как имеет разрешительную лицензию, однако он быстрее, чем TensorFlow [6].

MXNET – отличное решение для любителей языка R, единственная платформа, которая поддерживает все его функции и, кроме того, обладает высокой производительностью и эффективным использованием памяти [7].

Deeplearning4j создан для использования в бизнес-среде, адаптирован для микро-сервисной архитектуры, расширяется за счет Hadoop [8]. Может использовать обратное распространение по времени (Backpropagation Through Time, BPTT) – основанную на градиентах технику тренировки определенных типов рекуррентных нейронных сетей.

В настоящий момент самой популярной считается библиотека TensorFlow, хотя она медленнее, чем Torch и Theano и не имеет такого большого набора моделей, как Caffe. В отличие от любой другой архитектуры, TensorFlow имеет возможность делать частичные вычисления подграфа, то есть получение выборки от общей нейронной сети, а затем ее обучение отдельно от остальной части. Это так называемое Model Parallelization, которое используется для распределенного обучения.

Таким образом, несмотря на преимущества прочих библиотек, для исследований в рамках данной работы был выбран фреймворк TensorFlow, как надежный и развивающийся проект, способный работать с разными типами нейронных сетей и поддерживающий параллельные вычисления с помощью графических процессоров, а также Keras в качестве высокоуровневого API для TensorFlow.

Одним из наиболее динамично развивающихся фреймворков является Keras. Он может работать с различными платформами низкого уровня, включая Tensorflow, CNTK, Theano, имеет поддержку ОС Linux, Mac OS и Windows. Keras автоматизирует входную обработку изображений, подключение к внешним хранилищам, обработку выходных данных. В освоении для новичков и инженеров, которые раньше не занимались программированием, Keras является одним из самых простых, с огромным количеством примеров и готовых структур.

В зависимости от используемого базового нейросетевого фреймворка Keras позволяет работать с Nvidia CUDA, AMD FireStream, другими ускорителями через OpenCL. Большинство облачных провайдеров предлагают ускорители на основе технологии CUDA. В силу вышеупомянутых причин фреймворк Keras в сочетании с TensorFlow был выбран для настоящего исследования.

Развертывание и настройка Keras на системах Windows предполагает ручную установку большого количества стороннего ПО [9], внесение правок в конфигурационные файлы, располагаемые в различных местах файловой системы. В Linux сама установка выполняется проще за счет пакетных менеджеров, однако требуется установка проприетарных драйверов, требует знания некоторых особенностей систем Linux [10].

Цель исследования

Необходимо разработать такие подходы и методы установки необходимых программных средств, их запуска и автоматической настройки, которые позволили бы специалисту с базовыми сведениями о работе в системе Linux начать работать с нейросетевыми фреймворками наиболее эффективным способом с точки зрения затраченных финансов и за приемлемое время, затрачиваемое на развертывание и настройку фреймворка.

Основной проблемой является высокая ресурсоемкость при глубоком обучении даже небольших нейросетей с множеством слоев. В [11] показано, что реальный объем памяти, требуемый для сети среднего размера ResNet-50, имеющей 26 миллионов весовых параметров и вычисляющей около 16 миллионов действий при прямом проходе, составляет 7.5 Гб оперативной памяти, при том, что сами весовые коэффициенты занимают менее 170 Мб. Это происходит из-за множества факторов: выравнивания данных по границе слова, использования векторных наборов для GPU шириной 1024 бита, преобразования данных в матричную форму для эффективного умножения графическими процессорами (GEMs).

Использование облачных ресурсов с оплатой за потребление или за время работы позволяет обучать и использовать глубокие нейронные сети наиболее простым образом – покупка необходимых вычислительных ресурсов на требуемое время. Однако при таком подходе существует проблема выбора наиболее эффективных инструментов решения задач по подготовке к запуску фреймворков и собственно их выполнения. Установка библиотек, драйверов, сервисов может занимать гораздо больше времени, чем сам расчет. Кроме того, ученые, которые хотят использовать готовые модели нейронных сетей, не всегда разбираются в тонкостях настройки фреймворков под GPU, версиях модулей ядра и пр. Упрощение решения обозначенных задач может быть достигнуто за счет использования готовых преднастроенных контейнеров Docker. Кроме того, в большинстве случаев, контейнер Docker гораздо быстрее готов к запуску (зависит от скорости канал Интернет), чем происходит установка требуемого программного обеспечения на вычислительный узел.

Для минимизации человеческого фактора требуется разработать автоматизированный способ установки требуемого программного обеспечения, загрузки и выгрузки данных, выбора модели, запуска обучения. На первом этапе разработки нужно оценить, насколько накладные расходы Docker влияют на производительность расчетов, и в конечном итоге на общее время. Также необходимо оценить экономическую целесообразность использования ускорителей GPU на тех же наборах данных.

Зададим критерии и граничные условия. Будем использовать фреймворк Keras совместно с TensorFlow, наборы данных MNIST и CIFAR-10, о которых подробнее рассказано в разделе о планировании эксперимента. За максимальное допустимое время обучения возьмем время прогона обучения модели на виртуальном сервере с 1 ядром и 8 Гб оперативной памяти. За эталонное время обучения возьмем время прогона с использованием GPU Nvisia Tesla K80 (с 4992 ядрами CUDA). Остальные прогоны должны выполняться на подобном сервере, но:

1. с большим количеством ядер без графического ускорителя;
2. с одним и несколькими графическими ускорителями.

Пути решения проблем

Минимизация общего времени использования облачных ресурсов с оплатой за время работы виртуальной машины (большинство провайдеров VDS/VPS, предоставляющие доступ к графическим сопроцессорам и/или большому количеству ядер) состоит из нескольких этапов. Общее время, затраченное инженером на получение результата, рассчитывается по формуле (1)

$$t_{sum} = t_{startup} + t_{install} + t_{data\_load} + t_{start} + t_{learn} + t_{verify} + t_{save\_data} + t_{poweroff} \quad (1)$$

где

t_{sum} - суммарное время от момента старта до момента выключения машины, за него будет начислена оплата;

$t_{startup}$ - время запуска ОС;

$t_{install}$ - время, требуемое для установки необходимого программного обеспечения, закачки образов (для Docker), установки драйверов (для NVidia) и выполнения прочих предварительных действий;

t_{data_load} - время для загрузки наборов данных и выполнение подготовки к запуску;

t_{start} – время старта, необходимого для инициализации обучения нейронной сети, запуска фреймворка, старта Docker-контейнера;

t_{learn} - время, затраченное на полное обучение нейронной сети на наборе данных. Очень часто состоит из нескольких этапов, связанных с разбиением массива данных на части, перемешиванием, выполнением нескольких эпох и прочим;

t_{verify} - время, затраченное на верификацию и тестирование обученной нейронной сети;

t_{save_data} – время для выгрузки и сохранения всех коэффициентов и структурных данных нейронной сети;

$t_{poweroff}$ - время выключения ОС;

Поскольку оплата у провайдера производится за время работы виртуальной машины с некоторым округлением (у Google Cloud – 10 минут, у Microsoft Azure – 1 минута, провайдеры РФ – от 10 минут до 1 часа), функция оптимизация времени использования должна иметь возможность учета дискретного шага биллинга.

Автоматизация установки ПО и запуска модели будет происходить скриптом, который будет автоматически загружать все требуемое ПО на виртуальный сервер, вносить изменения в конфигурационные файлы, закачивать набор данных, запускать контейнер (для Docker) или фреймворк (для стандартного запуска) с требуемыми параметрами, отслеживать время работы, выгружать на удаленный сервер результаты и выключать сервер. Инженеру требуется только скопировать строку sh -c “$(curl -sSL http://путь_к_скрипту)”. Сам скрипт включает в себя команды установки фреймворка, скачивания нужных наборов данных и запуска прогона, выгрузку результатов и выключение машины. Скрипт написан на интерпретаторе sh и работает в Debian-подобных системах. Для Redhat-подобных систем нужен другой скрипт с измененными командами.

Часть скрипта установки программного обеспечения для запуска Keras включает в себя обновление репозитариев, обновление Python, установка необходимых библиотек (длительностью $t_{install}$). Аналогичный скрипт для Docker включает в себя установку самой системы Docker, скачивание контейнеров. Скрипт для GPU (длительностью t_{data_load}) самый сложный и включает в себя скачивание и установку драйверов для GPU, установку нужного дополнительного программного обеспечения, для Docker – установка nvidia-docker.

Планирование эксперимента

Эксперимент проводился с помощью услуг публичных провайдеров – 1Cloud, Azure, Google Cloud, у которых заказывались виртуальные сервера нужной конфигурации с поддержкой GPU и без нее. Эксперимент можно разделить на две стадии:

1. Обучение нейронной сети на наборе данных с использованием только ресурсов процессоров виртуальной машины или контейнеров, запущенных на виртуальной машине. Исследование проводилось при увеличении количества используемых Keras вычислительных ядер от 1 до 24.

2. Обучение нейронной сети на наборе данных с использованием ресурсов процессоров и GPU виртуальной машины или контейнеров, запущенных на виртуальной машине.

В обоих сценариях вычисления запускались как на процессоре виртуальной машины, так и внутри контейнера Docker. Во втором сценарии внутри контейнера Keras запускался с использованием nvidia-docker.

Обучение производилось на изображениях. С помощью нейронной сети решалась задача по их распознаванию – определения одного из 10 классов. Для этой цели в Keras была описана модель сверточной нейронной сети для каждого набора данных. Было выбрано два набора данных.

Первый. набор данных MNIST состоит из 70000 серых изображений 28x28 с рукописными цифрами. Всего 60000 тренировочных изображений и 10000 тестовых изображений. Количество эпох – 12. Для устранения эффекта переобучения после третьего и четвертого слоев применялась Dropout-регуляризация. Набор будет использован для CPU расчетов, так как в GPU имеет время расчета меньшее, чем время загрузки. Структура сети для MNIST:

1. Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape= (img_rows, img_cols, 1)) – сверхточный двумерный слой с 32 фильтрами, ядром свертки 3x3 и функцией активации ReLU;

2. Conv2D(64, (3, 3), activation='relu') – сверхточный двумерный слой с 64 фильтрами, ядром свертки 3x3 и функцией активации ReLu;

3. MaxPooling2D(pool_size=(2, 2)) – слой подвыборки для уменьшения размерности двумерной сети на основе определения максимума по нейронам областей 2 x 2;

4. Dense(128, activation='relu') – слой полносвязной сети с 128 выходами и функцией активации ReLU;

5. Dense(10, activation='softmax') – слой полносвязной сети с 10 выходами и функцией активации Softmax.

Первый слой нейронной сети имеет количество входов, соответствующее размеру изображений обучающей выборки. У последнего пятого слоя 10 выходов, что соответствует 10 классам распознаваемых цифр. Первые два слоя сети осуществляют свертку изображения, составляя карты признаков, которые затем сжимаются на третьем слое и соединяются на четвертом слое в одну полносвязную сеть и распределяются по классам четвертым и пятым слоем. Первые два слоя и четвертый используют функцию активации ReLU, что позволяет адекватно выделять признаки изображений. Пятый слой использует функцию Softmax, которая больше подходит для задач классификации. Обучение сети производилось с использованием перекрестной энтропии в качестве функции потерь, оптимизатор Адама.

Второй набор CIFAR-10 состоит из 60000 цветных изображений 32x32 с 10 классами, 6000 изображений на класс. Всего 50000 тренировочных изображений и 10000 тестовых изображений. Количество эпох – 200. Для устранения эффекта переобучения после третьего и четвертого слоев применялась Dropout-регуляризация. Набор будет использован для CPU и GPU расчетов. Структура сети для CIFAR-10:

1. Сверточный входной слой, с 32 фильтрами, ядром свертки 3x3 и функцией активации ReLU;

2. Сверточный входной слой с 32 фильтрами, ядром свертки 3x3 и функцией активации ReLU;

3. Слой подвыборки для уменьшения размерности двумерной сети на основе определения максимума по нейронам областей 2 x 24;

4. Сверточный входной слой, с 64 фильтрами, ядром свертки 3x3 и функцией активации ReLU;

5. Сверточный входной слой, с 64 фильтрами, ядром свертки 3x3 и функцией активации ReLU;

6. Слой подвыборки для уменьшения размерности двумерной сети на основе определения максимума по нейронам областей 2 x 24;

7. Сглаживающий слой;

8.Полносвязный уровень с 512 элементами и функцией активации ReLU;

9. Полносвязный уровень с 10 элементами и функция активации SoftMax.

Для сбора статистики во время проведения эксперимента использовался инструмент dstat, который запускался после первого обучения нейронной сети со следующими параметрами:

dstat -cmdr —output <output_file_name>.cvs

Данные параметры позволяют обеспечить получение статистики использования процессора, памяти, диска и запросов ввода-вывода и сохранить их в cvs-файл. Каждый эксперимент повторялся 10 раз, полученные результаты усреднялись с использованием Excel.

Полученные результаты

Прогон первой серии экспериментов (рисунок 1) был произведен на сервере Dell R810 с процессором Intel Xeon с частотой 2.8 ГГц, с использованием виртуальной машины на гипервизоре VMware vSphere 6.5. Виртуальной машине было выделено 8 Гб оперативной памяти и от 1 до 24 ядер, диски SSD. Для Docker прогонов использовался docker-ce 17.05.

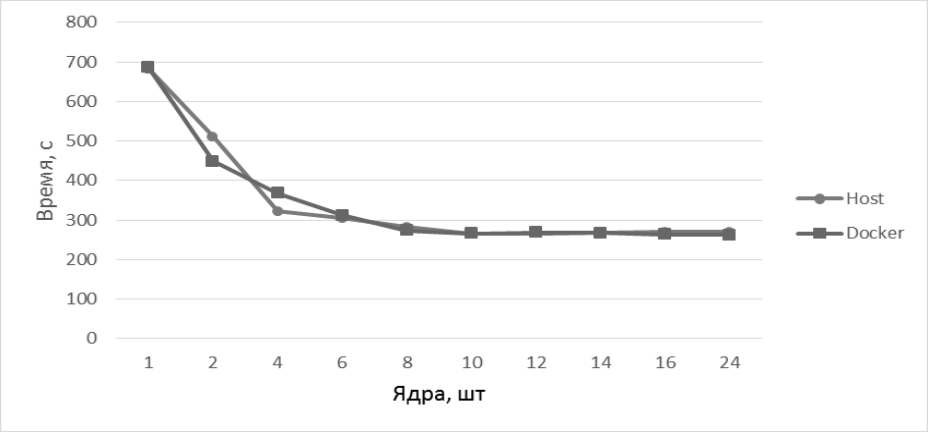
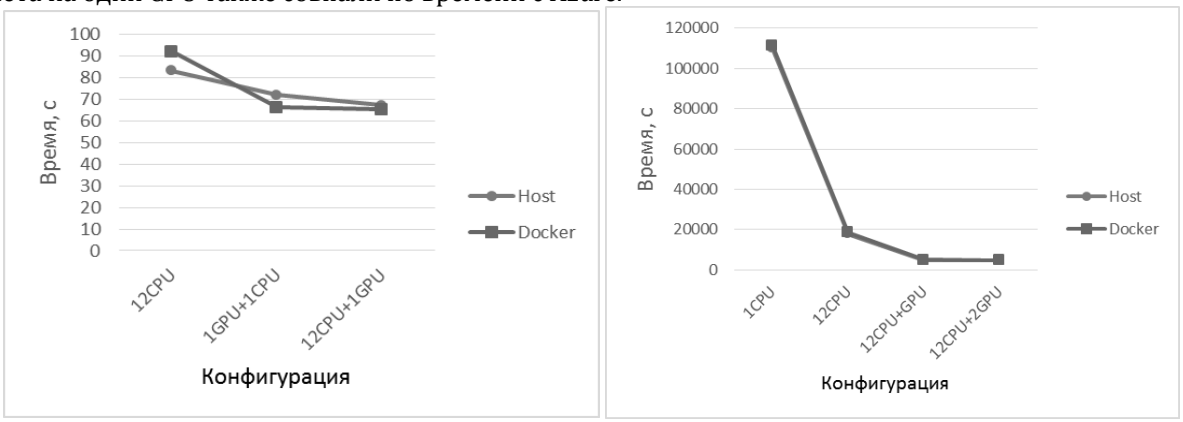


Рисунок 1 – Время выполнения тестового прогона на наборе MNIST

Прогон второй серии экспериментов (рисунок 2) был произведен на сервере платформы Microsoft Azure на машинах серии NC с процессором Intel Xeon E5-2690v3 с частотой 2.6 ГГц, с использованием виртуальной машины на гипервизоре Microsoft Azure. Виртуальной машине было выделено от 56 (модель NC6) до 112(модель NC12) Гб оперативной памяти, от 1 до 12 ядер (все показатели не изменяются, нельзя понизить объем ресурсов), от 1 до 2 виртуальных GPU Nvidia Tesla K80 (от 1/2 до 1 физической GPU Tesla K80), диски HDD. Для Docker прогонов использовался docker-ce 17.05, для GPU-Docker использовался nvidia-docker 17.05. Драйверы NVIDIA GRID для Tesla K80 версии R375 для CUDA 2.0.

Прогон третьей серии экспериментов было решено провести в Google Cloud с помощью подключения GPU к виртуальной машине с аналогичными Azure показателями, но по результатам GPU тестов все показатели в рамках погрешности измерений совпали для 1 и 2 GPU Tesla K80.

Провайдер Amazon AWS предоставляет только одну фиксированную конфигурацию малого размера с GPU Tesla K80 – 4 ядра, 16 Гб оперативной памяти и 1 Tesla K80, остальные конфигурации начинаются от 8 карт GPU, отдельная оплата за потребленные ресурсы процессора и диска. Для Amazon результаты расчета на один GPU также совпали по времени с Azure.



а) *б)*

Рисунок 2 – Результаты прогона в Azure: а) результаты на наборе MNIST; б) результаты на наборе CIFAR-10.

В результате был составлен сводный график результатов по времени прогона MNIST, поскольку на одинаковом наборе лучше видна тенденция стоимости прогонов (рисунок 3).

Для оценки затрат были использованы данные биллинга по серверам за время проведения тестов, полученные от провайдеров, все цены были представлены провайдерами в рублевом эквиваленте на время прогона тестов (рисунок 4).

Из графика, показанного на рисунке 4, видно, что, хотя GPU и выигрывает в скорости прогона, на итоговую стоимость прогона влияет в большей мере стоимость аренды, которую провайдеры взымают за время использования ускорителя.

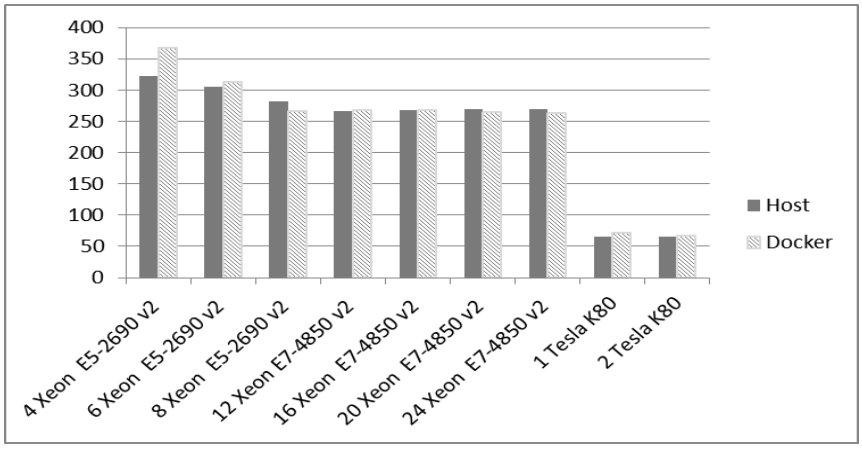
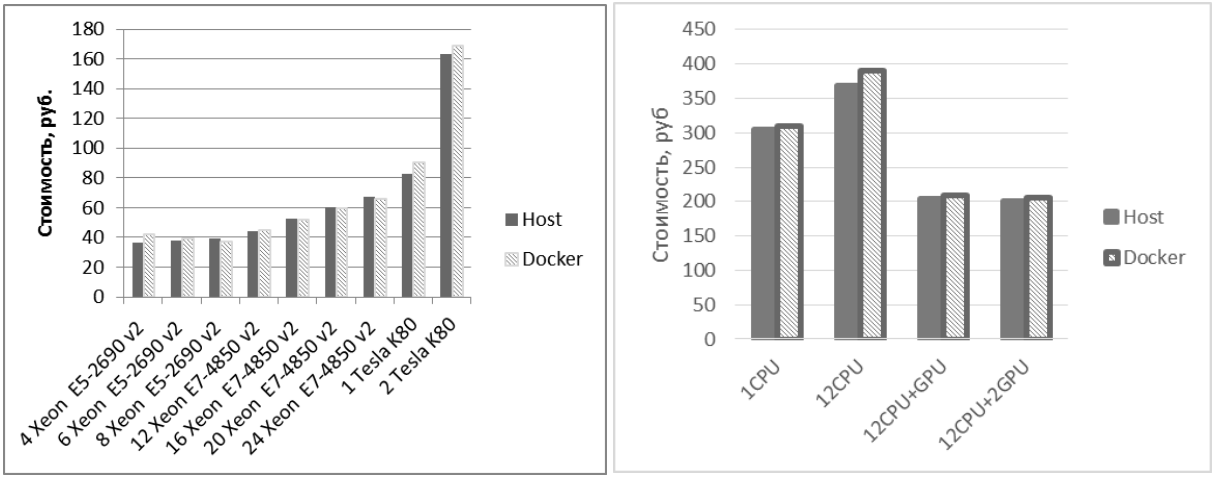


Рисунок 3 – Сводный график времени прогона на наборе MNIST



а) *б)*

Рисунок 4 – Сравнение стоимости прогона модели наборе MNIST: а) результаты на наборе MNIST; б) результаты на наборе CIFAR-10

Если взять цену одного прогона за единицу измерения эффективности, то получится, что оптимально, с точки зрения затрат, использовать такое количество обычных процессоров, которое позволяет максимально эффективно реализовать параллелизм при минимальном времени. В данном случае, это вариант с 8-12 ядрами процессора без GPU.

Заключение

В результате экспериментальных исследований эффективности обучения нейронных сетей в облачных ресурсах было выявлено:

1. Использование контейнеризации Docker при достаточном количестве вычислительных ядер позволяет не ухудшить показатели производительности стоимости, при этом существенно упрощает первоначальный запуск модели.

2. Использование специализированных графических ускорителей не всегда оправданно, если речь идет о научных исследованиях, в которых не критично время расчета. При снижении времени прогона в 2-4 раза (для 12 ядер на наборе MNIST) стоимость повышается в те же 2-4 раза, но при больших ресурсоемких наборах стоимость наоборот, уменьшается до 50%.

3. Экономическая эффективность расчета только на CPU занижена в ряде источников, особенно для небольших наборов и сетей, а производительность GPU, наоборот, завышена (по официальным сведениям производителя – быстрее в 8-10 раз).

В целом, имеется необходимость в дальнейшем более детального изучения и профилирования выполнения фреймворков нейронных сетей на GPU, в контейнерах Docker, особенно при распределенных вычислениях, где есть сильная зависимость от производительности сетевого соединения.

Благодарности

Исследование выполнено при финансовой поддержке Правительства Оренбургской области и РФФИ

(проекты №17-47-560046, №16-07-01004 и №15-07-06071), Президента Российской Федерации в рамках стипендии для молодых ученых и аспирантов (СП-2179.2015.5).

Литература

1. Нейросетевое программное обеспечение [Электронный ресурс] // URL: http://bookflow.ru/nejrosetevoe-programmnoe-obespechenie/ (дата обращения 11.09.2017).
2. Медведев В.С., Потемкин В.Г. Нейронные сети MATLAB 6. – М.: Диалог-МИФИ, 2002. – 496 с.
3. Парубец В.В., Берестнева О Г., Девятых Д.В. Применение технологии CUDA для ускорения вычислений в нейронных сетях //Известия Томского политехнического университета. – 2012. – Т. 320. – №. 5. – С. 121-125.
4. Bastien F. et al. Theano: new features and speed improvements [Электронный ресурс] //arXiv preprint arXiv:1211.5590. – 2012. – URL: https://arxiv.org/abs/1211.5590 (дата обращения 11.09.2017).
5. Jia Y. et al. Caffe: Convolutional architecture for fast feature embedding //Proceedings of the 22nd ACM international conference on Multimedia. – ACM, 2014. – С. 675-678.
6. Reasons to Switch from TensorFlow to CNTK [Электронный ресурс] //URL: https://docs.microsoft.com/en-us/cognitive-toolkit/reasons-to-switch-from-tensorflow-to-cntk (дата обращения: 23.06.2017)
7. Chen T. et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems [Электронный ресурс] // arXiv preprint arXiv:1512.01274. – 2015. – URL: https://arxiv.org/abs/1512.01274 (дата обращения: 22.06.2017)
8. Deep Learning for Java [Электронный ресурс] //URL: https://deeplearning4j.org (дата обращения: 22.06.2017)
9. Huerta I. Installing Keras, Theano and TensorFlow with GPU on Windows 8.1 and 10 in less than 4 hours [Электронный ресурс] // URL: https://sites.google.com/site/ivanhuertacasado/installing-keras-theano-tensorflow-with-gpu-windows (дата обращения 11.09.2017).
10. Crosson A. Installing Nvidia, Cuda, CuDNN, TensorFlow and Keras [Электронный ресурс] // URL: https://medium.com/@acrosson/installing-nvidia-cuda-cudnn-tensorflow-and-keras-69bbf33dce8a (дата обращения 11.09.2017).
11. Jamie Hanlon. Why is so much memory needed for deep neural networks? [Электронный ресурс] // URL: https://www.graphcore.ai/posts/why-is-so-much-memory-needed-for-deep-neural-networks (дата обращения 11.09.2017).

References

1. Nejrosetevoe programmnoe obespechenie [Электронный ресурс] // URL: http://bookflow.ru/nejrosetevoe-programmnoe-obespechenie/ (дата обращения 11.09.2017).
2. Medvedev V.S., Potemkin V.G. Nejronnye seti MATLAB 6. – M.: Dialog-MIFI, 2002. – 496 p.
3. Parubec V.V., Berestneva O.G., Devjatyh D.V. Primenenie tehnologii CUDA dlja uskorenija vychislenij v nejronnyh setjah //Izvestija Tomskogo politehnicheskogo universiteta. – 2012. – Vol. 320. – No. 5. – PP. 121-125.
4. Bastien F. et al. Theano: new features and speed improvements [Электронный ресурс] //arXiv preprint arXiv:1211.5590. – 2012. – URL: https://arxiv.org/abs/1211.5590 (дата обращения 11.09.2017).
5. Jia Y. et al. Caffe: Convolutional architecture for fast feature embedding //Proceedings of the 22nd ACM international conference on Multimedia. – ACM, 2014. – PP. 675-678.
6. Reasons to Switch from TensorFlow to CNTK [Электронный ресурс] //URL: https://docs.microsoft.com/en-us/cognitive-toolkit/reasons-to-switch-from-tensorflow-to-cntk (дата обращения: 23.06.2017)
7. Chen T. et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems [Электронный ресурс] // arXiv preprint arXiv:1512.01274. – 2015. – URL: https://arxiv.org/abs/1512.01274 (дата обращения: 22.06.2017)
8. Deep Learning for Java [Электронный ресурс] //URL: https://deeplearning4j.org (дата обращения: 22.06.2017)
9. Huerta I. Installing Keras, Theano and TensorFlow with GPU on Windows 8.1 and 10 in less than 4 hours [Электронный ресурс] // URL: https://sites.google.com/site/ivanhuertacasado/installing-keras-theano-tensorflow-with-gpu-windows (дата обращения 11.09.2017).
10. Crosson A. Installing Nvidia, Cuda, CuDNN, TensorFlow and Keras [Электронный ресурс] // URL: https://medium.com/@acrosson/installing-nvidia-cuda-cudnn-tensorflow-and-keras-69bbf33dce8a (дата обращения 11.09.2017).
11. Jamie Hanlon. Why is so much memory needed for deep neural networks? [Электронный ресурс] // URL: https://www.graphcore.ai/posts/why-is-so-much-memory-needed-for-deep-neural-networks (дата обращения 11.09.2017).

Об авторах:

Ушаков Юрий Александрович, кандидат технических наук, доцент кафедры геометрии и компьютерных наук, Оренбургский государственный университет, unpk@mail.ru

Полежаев Петр Николаевич, преподаватель кафедры компьютерной безопасности и математического обеспечения информационных систем, Оренбургский государственный университет, newblackpit@mail.ru

Шухман Александр Евгеньевич, кандидат педагогических наук, доцент, заведующий кафедры геометрии и компьютерных наук, Оренбургский государственный университет, shukhman@gmail.com

Порохненко Юлия Сергеевна, студент специальности «Компьютерная безопасность», факультет математики и информационных технологий, Оренбургский государственный университет, yulkins2@gmail.com

Чернова Екатерина Владимировна, студент специальности «Компьютерная безопасность», факультет математики и информационных технологий, Оренбургский государственный университет, katin_box@mail.ru

Очередько Ольга Олеговна, студент специальности «Компьютерная безопасность», факультет математики и информационных технологий, Оренбургский государственный университет, olik-oo@yandex.ru

Note on the authors:

Ushakov Yury A., Candidate of Engineering Sciences, Associate Professor at the Department of Geometry and Computer Science, Orenburg State University, unpk@mail.ru

Polezhaev Petr N., Lecturer at the Department of Computer Security and Mathematical Maintenance of Information Systems, Orenburg State University, newblackpit@mail.ru

Shukhman Aleksandr E., Candidate of Pedagogic Sciences, Associate Professor, Head of the Department of Geometry and Computer Science, Orenburg State University, shukhman@gmail.com

Porokhnenko Yuliya S., Student of «Computer security» specialty, Faculty of Mathematics and Information Technologies, Orenburg State University, yulkins2@gmail.com

Chernova Ekaterina V., Student of «Computer security» specialty, Faculty of Mathematics and Information Technologies, Orenburg State University, katin_box@mail.ru

Ocheredko Olga O., Student of «Computer security» specialty, Faculty of Mathematics and Information Technologies, Orenburg State University, olik-oo@yandex.ru