ИСПОЛЬЗОВАНИЕ НЕЙРОННЫХ СЕТЕЙ В КОНТЕЙНЕРНЫХ СРЕДАХ

Чернова Е.В., Полежаев П.Н.

Оренбургский государственный университет

На сегодняшний день нейронные сети обрели общую популярность. Появилось множество приложений в самых разных областях, использующих нейронные сети для решения сложных задач. Но все же существуют проблемы, тормозящие развитие этого направления. Главным образом – это высокие требования к вычислительным ресурсам, используемым для вычисления нейронных сетей. Также немало важным фактором считается необходимость хорошо знать саму технологию, в том числе фреймворки и языки. Разработка методов установки необходимых программных средств, их запуска и автоматической настройки помогли бы в решении этих задач. Подобные подходы позволили бы специалисту с базовыми сведениями о работе в системе Linux начать работать с нейросетевыми фреймворками наиболее эффективным способом.

Для реализации подобных идей необходимо провести эксперимент с целью выяснения наиболее эффективных конфигураций и набора ресурсов для запуска моделей нейронной сети. Главным критерием является время, которое потенциальный пользователь потратит на получение результата работы с нейронной сетью. Эксперимент можно разделить на две стадии:

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

2. Обучение нейронной сети на наборе данных с использованием ресурсов центральных и графических процессоров. Запуск происходит на виртуальных машинах или запущенных на виртуальной машине контейнерах с поддержкой ускорения на графических процессорах.

Для работы с разными типами нейронных сетей была выбрана библиотека TensorFlow, как надежный и развивающийся проект, поддерживающий параллельные вычисления с помощью графических процессоров, а также Keras в качестве высокоуровневого API для TensorFlow [1]. В обоих сценариях вычисления будут запускаться как на процессоре виртуальной машины, так и внутри контейнера Docker. Во втором сценарии внутри контейнера запустится Keras с использованием nvidia-docker, который позволяет использовать мощность графических процессоров для параллельных вычислений.

В данной статье будет описана установка необходимых библиотек и фреймворков на операционную систему Linux Ubuntu. Python 2.7 уже должен быть установлен.

Для первой стадии эксперимента установим версию с поддержкой CPU (центральных процессоров). Установка TensorFlow может осуществляется с помощью следующих инструментов: virtualenv, pip, Docker, Anaconda. Будем использовать pip, следующая строка установит его в систему [2]:

sudo apt-get install python-pip python-dev

Теперь используя pip, установим Tensorflow без поддержки GPU:

pip install tensorflow

Для проверки правильности установки запустим скрипт Python:

```
import tensorflow as tf
hello = tf.constant('Hello world!')
sess = tf.Session()
print(sess.run(hello))
```

Если Tensorflow установлен правильно, появится сообщение «Hello world!», иначе – сообщение об ошибке.

Keras требует наличие нескольких пакетов, их также можно установить с помощью pip [3]:

```
pip install numpy scipy
pip install scikit-learn
pip install pillow
pip install h5py
```

После этого можно установить Keras:

pip install keras

Необходимо убедиться, что конфигурационный файл keras.json настроен верно. Особенно следует обратить внимание на параметр «backend», он отвечает за то, каким фреймворком пользуется Keras:

```
{
    "image_dim_ordering": "tf",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

Для проверки установки достаточно написать «import keras». Если команда выполнилась без ошибок, значит Keras установлен правильно.

Чтобы снимать показания во время эксперимента будем использовать инструмент мониторинга dstat, позволяющий анализировать производительность системы. Для его установки достаточно ввести команду:

sudo apt-get install dstat

Теперь для проведения эксперимента нужен скрипт Python, который бы содержал в себе описание модели нейронной сети и команду обучения. Пример такого скрипта можно найти в официальном репозитарии Keras. Следующая строка запускает скрипт и dstat, который сохранит результат в файл в формате cvs.

*dstat -cmdr --output *файл для записи результата.cvs* | python *скрипт*,*

где ключ *c* выводит процессорную статистику, *m* – статистику памяти, *d* – диска, *r* - запросов ввода-вывода.

Вторая часть эксперимента проводится с помощью контейнеров Docker. Чтобы его установить необходимо добавить отдельный репозитарий [4]. Для этого установим следующие пакеты:

```
sudo apt-get install \
   apt-transport-https \
   ca-certificates \
   curl \
   software-properties-common
```

Добавим официальные GPG ключи Docker:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

Следующая команда добавит стабильный репозитарий для Docker:

```
sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Теперь можно установить Docker, версию Community Edition:

sudo apt-get install docker-ce

Чтобы убедиться, что установка прошла успешно, запустим образ hello-world. Контейнер должен работать без ошибок и печатать сообщение «Hello, world!».

sudo docker run hello-world

Для начала работы контейнера Docker с TensorFlow, поддерживающего только CPU и запускающего оболочку bash, нужно ввести команду:

docker run -it gcr.io/tensorflow/tensorflow bash

Затем следует установить Keras, так же как было описано выше, но только внутри этого контейнера. Контейнер готов для проведения эксперимента.

Во второй части эксперимента обучение нейронной сети нужно проводить с использованием графических процессоров. Потребуется другое программное обеспечение. Для графических процессоров компании NVIDIA существуют готовые решения: CUDA – это платформа для параллельного вычисления на графических процессорах, cuDNN – мощная библиотека для машинного обучения. Дистрибутивы распространяются бесплатно на официальном сайте.

На странице CUDA Toolkit необходимо выбрать операционную систему (Linux), архитектуру (X86_64), дистрибутив (Ubuntu), версию (16.04), тип дистрибутива (runfile) и скачать его [5]. Для выбранных нами параметров установка библиотеки начнется после введения следующей строки:

sudo sh cuda_9.1.85_387.26_linux.run

Для того чтобы скачать дистрибутив cuDNN необходимо войти под своей учетной записью на официальный сайт (или зарегистрироваться) [6]. После этого нужно подтвердить свое согласие с лицензионным соглашением и выбрать версию дистрибутива. Мы выбрали библиотеку для Linux версию 7.0.5 для CUDA 9.1. Следующая строка распакует скачанный архив:

tar -xzvf cudnn-9.1-linux-x64-v7.tgz

Файлы нужно скопировать в директорию CUDA Toolkit:

sudo cp cuda/include/cudnn.h /usr/local/cuda/include
sudo cp cuda/lib64/libcudnn /usr/local/cuda/lib64*
sudo chmod a+r /usr/local/cuda/include/cudnn.h
*/usr/local/cuda/lib64/libcudnn**

Установка библиотек для графических процессоров завершена. Последующая установка вспомогательных библиотек и фрэймфорков ничем не отлича-

ется от описанных выше, кроме версии TensorFlow [7]. Если установлена версия без поддержки графических процессоров, то ее нужно предварительно удалить. Команду установки TensorFlow следует заменить на следующую:

pip install tensorflow-gpu

Вместо обычного будем ставить контейнер Docker с поддержкой графических процессоров – nvidia-docker [8]. Сначала нужно добавить репозитарии:

curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -

curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu16.04/amd64/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list

sudo apt-get update

Затем устанавливается nvidia-docker2 и перегружается конфигурация демона Docker:

sudo apt-get install -y nvidia-docker2
sudo pkill -SIGHUP dockerd

Чтобы проверить, что установка прошла успешно, запустим утилиту для мониторинга графических процессоров nvidia-smi:

docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi

Теперь можно запустить контейнер TensorFlow с поддержкой графических процессоров:

nvidia-docker run -it gcr.io/tensorflow/tensorflow:latest-gpu bash

Установка Keras осуществляется аналогично тому, как было написано выше. На этом подготовка к эксперименту завершена.

Эксперимент проводился с помощью услуг публичных провайдеров – 1Cloud, Azure, Google Cloud. У них можно заказывать виртуальные сервера нужной конфигурации с поддержкой GPU и без нее. В качестве набора данных использовались набор изображений 28x28 с рукописными цифрами MNIST и набор цветных изображений 32x32 с 10 классами CIFAR-10. Однако для первой части эксперимента использовался только MNIST (рис.1) , а для второй – оба набора (рис. 2, 3), чтобы лучше проследить зависимость времени от конфигурации системы. Это объясняется тем, что нейронной сети проще обрабатывать MNIST, чем CIFAR-10, а мощности графических процессоров больше, чем центральных.

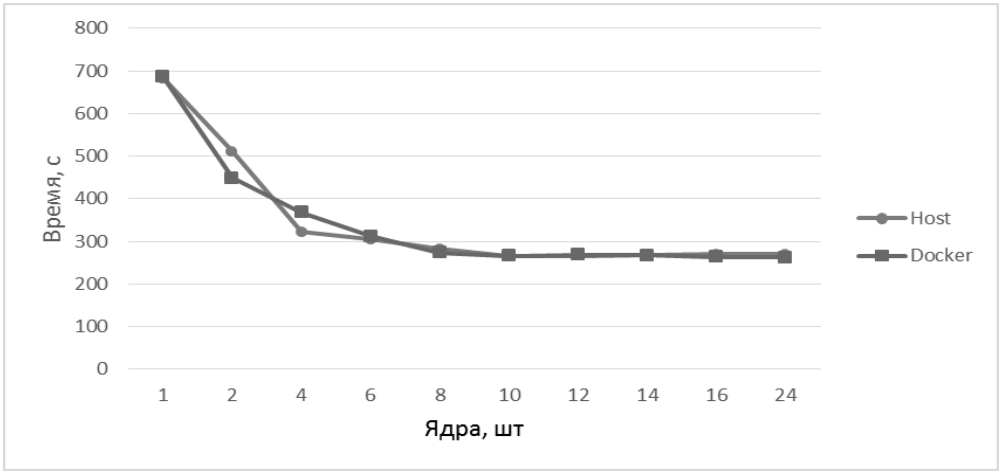

Рисунок 1 – Время обучение модели нейронной сети на наборе MNIST

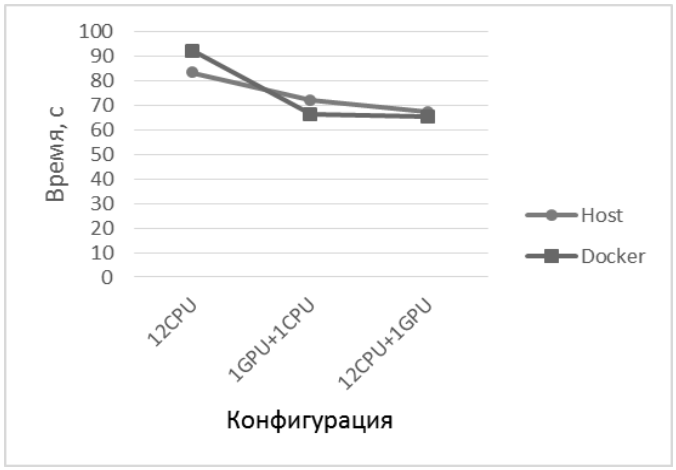

Рисунок 2 – Время обучение модели нейронной сети на наборе MNIST

Рисунок 3 – Время обучение модели нейронной сети на наборе CIFAR-10

Результаты показали, что использование контейнеризации Docker при достаточном количестве вычислительных ядер позволяет не ухудшить показатели производительности и стоимости, при этом существенно упрощает первоначальный запуск модели. Однако Docker с поддержкой графических процессоров не готов к автоматизации на сегодняшний день, так как требуется слишком много индивидуальных настроек вручную. Тем не менее, при написании статьи был подготовлен скрипт, устанавливающий нужные драйвера и вспомогательные пакеты.

Исследование выполнено при финансовой поддержке Правительства Оренбургской области и РФФИ (проекты №17-47-560046, №16-29-09639 и №18-07-01446), Президента Российской Федерации в рамках стипендии для молодых ученых и аспирантов (СП-2179.2015.5).

Список литературы

1. Чернова Е.В., Полежаев П.Н. Анализ существующих технологий нейронных сетей // IV Международная научно-техническая конференция «Новые информационные технологии и системы» (НИТиС-2017), 2017. – С. 232-236.

2. Installing TensorFlow [Электронный ресурс] //Режим доступа: https://www.tensorflow.org/install/ – Загл. с экрана. – (Дата обращения: 22.12.2017)

3. Get Started [Электронный ресурс] //Режим доступа: https://docs.docker.com/get-started/ – Загл. с экрана. – (Дата обращения: 24.12.2017)

4. Keras: The Python Deep Learning library [Электронный ресурс] //Режим доступа: https://keras.io/ – Загл. с экрана. – (Дата обращения: 24.12.2017)

5. CUDA Zone [Электронный ресурс] //Режим доступа: https://developer.nvidia.com/cuda-zone – Загл. с экрана. – (Дата обращения: 25.12.2017)

6. NVIDIA cuDNN [Электронный ресурс] //Режим доступа: https://developer.nvidia.com/cudnn – Загл. с экрана. – (Дата обращения: 25.12.2017)

7. Чернова Е.В., Полежаев П.Н. Архитектура распределенной библиотеки Distributed TensorFlow // Компьютерная интеграция производства и ИПИ-технологии: материалы VIII Всероссийской научно-практической конференции. – Оренбург, 2017. – С. 354-357.

8. NVIDIA Docker: GPU Server Application Deployment Made Easy [Электронный ресурс] //Режим доступа: https://devblogs.nvidia.com/parallelforall/nvidia-docker-gpu-server-application-deployment-made-easy/ – Загл. с экрана. – (Дата обращения: 28.12.2017)