VDL Documentation

Weichao, Vincent, Kaiyue

Jul 05, 2018
Contents:

1 Multiple learners component  3
  1.1 Multiple learners on Tensorflow MNIST  3
  1.2 Multiple learners on Neonrace task  4

2 Run 5 learners and 10 actors in a cluster  5
  2.1 Install dependencies for the code  5
  2.2 Modify the cluster configuration  5
  2.3 Start the parameter server  6
  2.4 Start five learners  6
  2.5 Start all actors and start learning  6
  2.6 Check the learning result  6

3 The virtual robot arm  7

4 Files in this project  9
This documentation documents how to reproduce the result in the VDL project. The most update-to-date version can be found in the VDL github repository.

The final presentation is hosted here. The pdf version can be downloaded from here.

![Diagram of multiple learners and actors](image.png)

**Fig. 1:** The architecture with multiple learners and multiple actors

This project implements the distributed learning system as shown in the above figure. How to run and benchmark each component and how to run the complete system are documented in the following sections.
1.1 Multiple learners on TensorFlow MNIST

First, please go to tensorflow_MNIST folder. To run P2P multi-learner code on TensorFlow MNIST task, you can use either unreliable Python multicast (slightly faster) or reliable Spread multicast. To run Python multicast, do:

```
python mnist_mcast_peer.py <num_peers> <my_peer_ID> <batch_size> <num_rounds>
```

For instance, if you want to run 4 parallel learners, each with a data batch size of 100 and 250 rounds each, do on four different machines:

```
python mnist_mcast_peer.py 4 1 100 250
python mnist_mcast_peer.py 4 2 100 250
python mnist_mcast_peer.py 4 3 100 250
python mnist_mcast_peer.py 4 4 100 250
```

To synchronize the training, you also need to run a short script after all 4 learner programs have been started. Do:

```
python start_mcast.py
```

To run Spread multicast, do:

```
python mnist_spread_peer.py <num_peers> <my_peer_ID> <batch_size> <num_rounds>
```

The corresponding commands for 4 parallel learners will be:

```
python mnist_spread_peer.py 4 1 100 250
python mnist_spread_peer.py 4 2 100 250
python mnist_spread_peer.py 4 3 100 250
python mnist_spread_peer.py 4 4 100 250
```

Similarly, to synchronize the training, do:
1.2 Multiple learners on Neonrace task

First, please go to `universe-starter-agent` folder. To run P2P multi-learner code on Neonrace task, say for 3 parallel learners for example, do:

```python
python run.py --num-workers 3 --log-dir train-log/pong-multi-learners-0 -id 0
python run.py --num-workers 3 --log-dir train-log/pong-multi-learners-1 -id 1
python run.py --num-workers 3 --log-dir train-log/pong-multi-learners-2 -id 2
```

In any machine

```python
python start_spread.py
```

To clean up the train log

```bash
rm train-log/pong-multi-learners* -r
```
Run 5 learners and 10 actors in a cluster

The setup and execution is a complex procedure. If not clear, please report an issue in the issue tracker.

### 2.1 Install dependencies for the code

The complete system requires several dependencies. The dependencies are:

- **docker**
- **python libraries**
  - OpenAI gym
  - OpenAI universe
  - Tensorflow
- The learning code for neonrace is modified from openai/universe-starter-agent

An installation script is provided in `universe-starter-agent/install/install.sh`

### 2.2 Modify the cluster configuration

The multiple-learner component is implemented with distributed tensorflow.

The learner configuration is hard coded in `universe-starter-agent/ccvl_cluster_spec.py`. Modify this file according to your cluster spec.

In the following document, the parameter server will be `ccvl2`. And the other five machines `ccvl1-5` will run learners. The parameter server is responsible for coordinating weights between learners.
2.3 Start the parameter server

In the machine for parameter server, **ccvl2**, start the parameter server with

```
cd universe-starter-agent/
sh run_ps.sh
```

`universe-starter-agent/run_ps.sh` will start `ps_run.py` with proper parameters.

2.4 Start five learners

In each machine from **ccvl1-5**, start the learner with

```
sh run_learner.sh 0
```

The number 0 is the worker id for ccvl1, number 1 will be the id for ccvl2.

The learner will wait until all actors are connected.

2.5 Start all actors and start learning

Start docker which contains the **neonrace** virtual environment. This script will start two docker containers, each running a neonrace virtual environment.

```
sh run_docker.sh
```

Start the actor code with

```
sh run_actor.sh
```

`run_actor.sh` will run `actor.py` with proper parameters.

2.6 Check the learning result

The learning procedure can be visualized by connecting to the docker container through vnc.

Use TurboVNC client to connect to **ccvl1.ccvl.jhu.edu:13000**. Change the url to your own configuration.

The learnt models will be stored in **train-log** folder. Use *tensorboard* to visualize the result, or use the code in **neonrace** to use trained model.
The virtual robot arm

The virtual arm is stored in a different repository qiuwch/UE4VirtualArm. This is because the 3D CAD models are very large. This repository is still in private mode, because we are still finalizing the design and will release it together with a publication. If you are interested in this project, please send an email to qiuwch@gmail.com to request access.

A compiled virtual arm binary can be downloaded.

- **Windows version**
- **Linux version**

The screenshot of the virtual arm
The arm is placed in an empty environment. If you want to place the arm to a different virtual environment, the access to the source project is required.

This virtual arm can be controlled with the unrealcv project.

How to use Maya and UE4 to create a virtual replica of a $30 OWI arm.

The CAD model is downloaded from 3D warehouse.

You can buy a real arm from here.

Due to the very large file size, all files are tracked with git lfs. Make sure lfs is installed, otherwise all files will just be a link.

In Maya.

In UE4.
Files in this project

# Main
- docs/ # Documentation files in reStructuredText format
- universe-starter-agent/ # Virtual distributed learning system, the code is modified from https://github.com/openai/universe-starter-agent, which provides the baseline learning algorithm.

# Components
- learner-actor/ # Experiment code for learner-actor communication
- tensorflow_MNIST/ # Experiment code for P2P-multi-learner

# Utility
- gym-demo/ # Virtual environment demos to make sure the dev boxes are correctly configured.
- benchmark/ # Benchmark code to evaluate the network speed and speed of different virtual environments
- neonrace/ # Code to run trained neonrace auto-driving model
- spread/ # Compiled spread and its python wrapper

# Virtual arm
- arm-pose/ # Pose estimation code trained on the virtual arm and test on the real arm.
- owi-arm/ # Code to control real and the virtual arm