Deep Learning System and Parallel Computing HW1

- Author: 黃柏瑄 (P78081528)

1. Experiment description

- Framework: Chainer
- Dataset: Cifar10
- Model: ResNet20

2. Environment setting

- Hardware:
  - CPU: Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz (`$ lscpu | grep 'Model name'`)
  - Memory: 32 GB
- Software:
  - OS: Ubuntu 18.04.5 LTS (`$ lsb_release -d`)
  - Python 3.6.9
  - Chainer 7.7.0

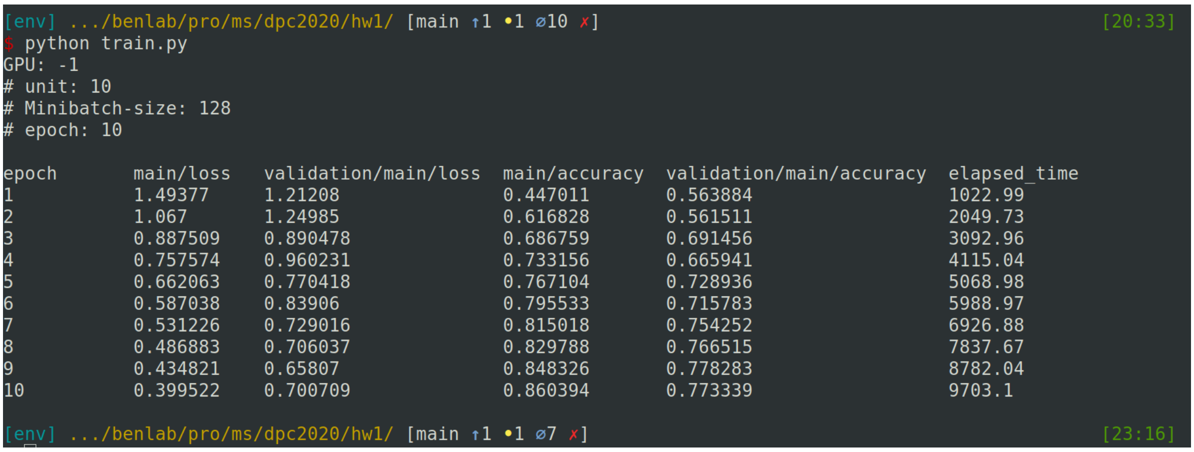
```
$ virtualenv --python python3.6 env
$ source env/bin/activate
$ pip install chainer==7.7.0
$ pip list
Package           Version
----------------- -------
chainer           7.7.0
filelock          3.0.12
numpy             1.19.2
pip               20.2.2
protobuf          3.13.0
setuptools        50.2.0
six               1.15.0
typing-extensions 3.7.4.3
wheel             0.35.1
```

3. Result

Code can be found at Appendix section.

3.1. Train

```
$ python train.py
```



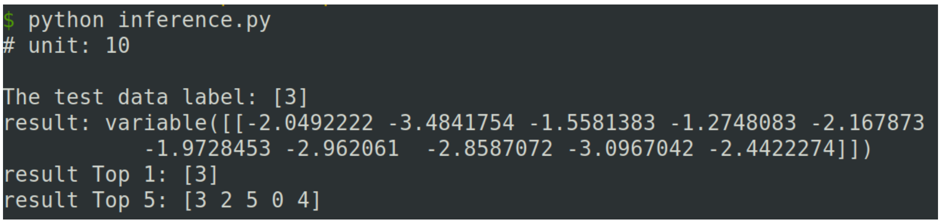
```
[env] .../benlab/pro/ms/dpc2020/hw1/ [main ↑1 •1 ⌀10 ✗]                                                    [20:33]
$ python train.py
GPU: -1
# unit: 10
# Minibatch-size: 128
# epoch: 10

epoch       main/loss   validation/main/loss  main/accuracy  validation/main/accuracy  elapsed_time
1           1.49377     1.21208               0.447011       0.563884                  1022.99
2           1.067       1.24985               0.616828       0.561511                  2049.73
3           0.887509    0.890478              0.686759       0.691456                  3092.96
4           0.757574    0.960231              0.733156       0.665941                  4115.04
5           0.662063    0.770418              0.767104       0.728936                  5068.98
6           0.587038    0.83906               0.795533       0.715783                  5988.97
7           0.531226    0.729016              0.815018       0.754252                  6926.88
8           0.486883    0.706037              0.829788       0.766515                  7837.67
9           0.434821    0.65807               0.848326       0.778283                  8782.04
10          0.399522    0.700709              0.860394       0.773339                  9703.1

[env] .../benlab/pro/ms/dpc2020/hw1/ [main ↑1 •1 ⌀7 ✗]                                                     [23:16]
```

3.2. Inference

```
$ python inference.py
```



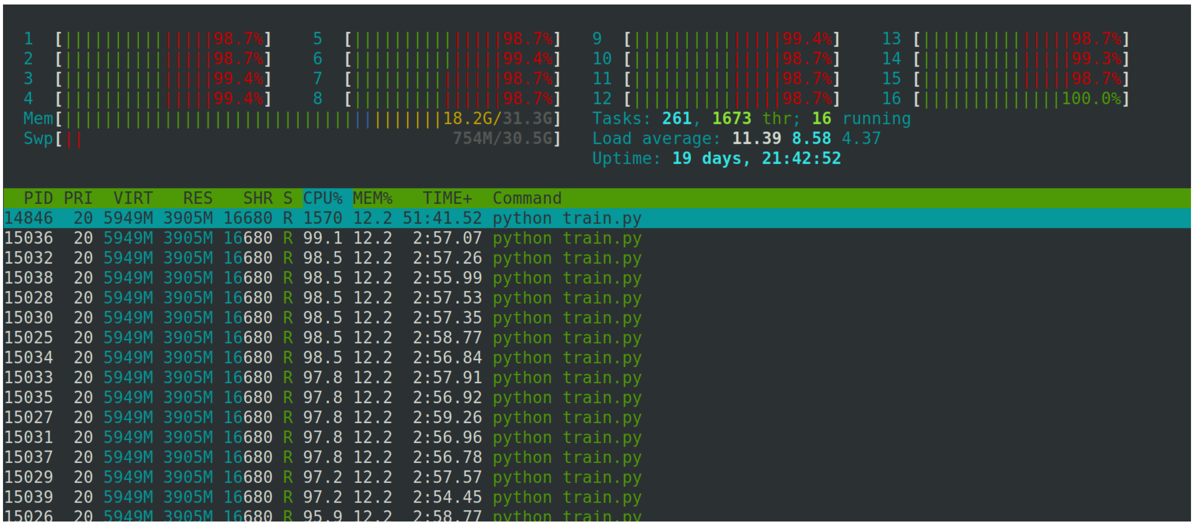
```
$ python inference.py
# unit: 10

The test data label: [3]
result: variable([[-2.0492222 -3.4841754 -1.5581383 -1.2748083 -2.167873
           -1.9728453 -2.962061  -2.8587072 -3.0967042 -2.4422274]])
result Top 1: [3]
result Top 5: [3 2 5 0 4]
```

4. Insight

4.1. CPU utilization

The CPU utilization is good during training phase.



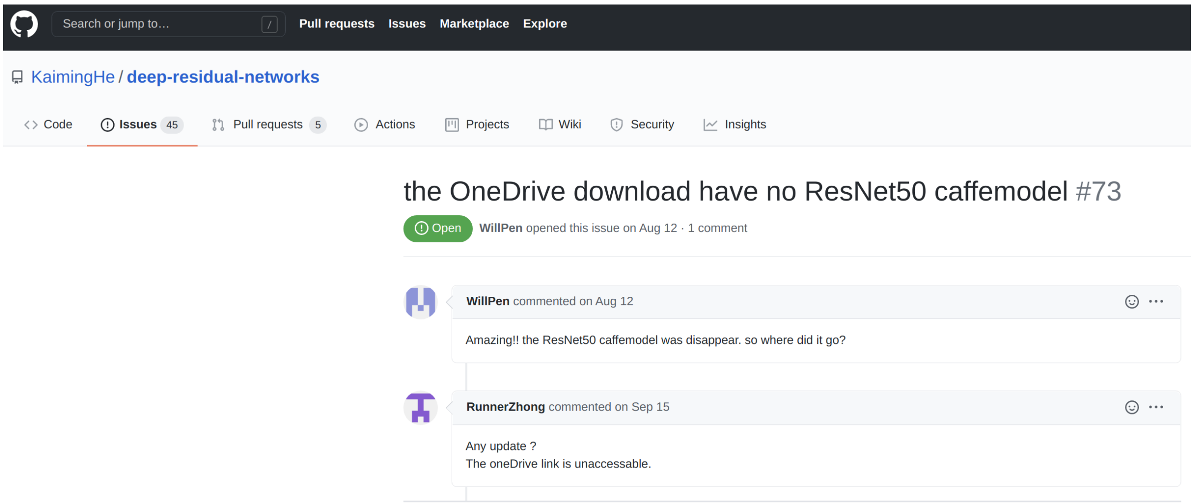
```
  1  [||||||||||||||98.7%]     5  [||||||||||||||98.7%]    9  [||||||||||||||99.4%]     13 [||||||||||||||98.7%]
  2  [||||||||||||||98.7%]     6  [||||||||||||||99.4%]    10 [||||||||||||||98.7%]     14 [||||||||||||||99.3%]
  3  [||||||||||||||99.4%]     7  [||||||||||||||98.7%]    11 [||||||||||||||98.7%]     15 [||||||||||||||98.7%]
  4  [||||||||||||||99.4%]     8  [||||||||||||||98.7%]    12 [||||||||||||||98.7%]     16 [|||||||||||||100.0%]
  Mem[|||||||||||||||||||||||||||||||||||||||18.2G/31.3G]    Tasks: 261, 1673 thr; 16 running
  Swp[||                                      754M/30.5G]    Load average: 11.39 8.58 4.37
                                                             Uptime: 19 days, 21:42:52

  PID PRI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
14846  20 5949M 3905M 16680 R 1570 12.2 51:41.52 python train.py
15036  20 5949M 3905M 16680 R 99.1 12.2  2:57.07 python train.py
15032  20 5949M 3905M 16680 R 98.5 12.2  2:57.26 python train.py
15038  20 5949M 3905M 16680 R 98.5 12.2  2:55.99 python train.py
15028  20 5949M 3905M 16680 R 98.5 12.2  2:57.53 python train.py
15030  20 5949M 3905M 16680 R 98.5 12.2  2:57.35 python train.py
15025  20 5949M 3905M 16680 R 98.5 12.2  2:58.77 python train.py
15034  20 5949M 3905M 16680 R 98.5 12.2  2:56.84 python train.py
15033  20 5949M 3905M 16680 R 97.8 12.2  2:57.91 python train.py
15035  20 5949M 3905M 16680 R 97.8 12.2  2:56.92 python train.py
15027  20 5949M 3905M 16680 R 97.8 12.2  2:59.26 python train.py
15031  20 5949M 3905M 16680 R 97.8 12.2  2:56.96 python train.py
15037  20 5949M 3905M 16680 R 97.8 12.2  2:56.78 python train.py
15029  20 5949M 3905M 16680 R 97.2 12.2  2:57.57 python train.py
15039  20 5949M 3905M 16680 R 97.2 12.2  2:54.45 python train.py
15026  20 5949M 3905M 16680 R 95.9 12.2  2:58.77 python train.py
```

4.2. Fun facts

4.2.1. It is not easy to find some pre-trained weights on the Internet

Many pre-trained models are disappear. There may be many reasons such as privacy issue, business considerations, or lack of maintenance.



the OneDrive download have no ResNet50 caffemodel #73

WillPen opened this issue on Aug 12 · 1 comment

WillPen commented on Aug 12

Amazing!! the ResNet50 caffemodel was disappear. so where did it go?

RunnerZhong commented on Sep 15

Any update ?
The oneDrive link is unaccessable.

> source: https://github.com/KaimingHe/deep-residual-networks/issues/73

4.2.2. Preferred Networks Migrates its Deep Learning Research Platform to PyTorch

The company who developed Chainer changed to use PyTorch as their DL framework.

5. Appendix

5.1. Code

To avoid duplicate code, I isolated the model's code into independent file and import it in train and inference code.

- Code architecture

```
$ tree . -I 'env|result|__pycache__|images|*.md|*.pdf'
.
├── inference.py
├── resnet_cifar10.py
└── train.py
```

- Model (`resnet_cifar10.py`)
  - Reference:
    1. https://github.com/akamaster/pytorch_resnet_cifar10/blob/master/resnet.py
    2. https://github.com/mitmul/chainer-cifar10/blob/master/models/ResNet.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import chainer
import chainer.functions as F
import chainer.links as L


class BottleNeck(chainer.Chain):
    def __init__(self, n_in, n_out, stride=1):
        self.shortcut = stride != 1
        super(BottleNeck, self).__init__()
        with self.init_scope():
            self.conv1 = L.Convolution2D(
                n_in, n_out, ksize=3, stride=stride, pad=1, nobias=True
            )
            self.bn1 = L.BatchNormalization(n_out)
            self.conv2 = L.Convolution2D(
                n_out, n_out, ksize=3, stride=1, pad=1, nobias=True
            )
            self.bn2 = L.BatchNormalization(n_out)
            self.conv3 = L.Convolution2D(
                n_in, n_out, ksize=1, stride=stride, pad=0, nobias=True
            )
            self.bn3 = L.BatchNormalization(n_out)

    def __call__(self, x):
        h = F.relu(self.bn1(self.conv1(x)))
        h = self.bn2(self.conv2(h))
        if self.shortcut:
            h += self.bn3(self.conv3(x))
        h = F.relu(h)
        return h


class Block(chainer.ChainList):
    def __init__(self, n_in, n_out, n_bottlenecks, stride):
        super(Block, self).__init__()
        self.in_planes = n_in
        strides = [stride] + [1] * (n_bottlenecks - 1)
        for stride in strides:
            self.add_link(BottleNeck(self.in_planes, n_out, stride))
            self.in_planes = n_out

    def __call__(self, x):
        for f in self:
            x = f(x)
        return x


class ResNet(chainer.Chain):
    def __init__(self, n_class=10, n_blocks=[3, 3, 3]):
        super(ResNet, self).__init__()

        with self.init_scope():
            self.conv1 = L.Convolution2D(None, 16, 3, 1, 1, nobias=True)
            self.bn2 = L.BatchNormalization(16)
            self.res3 = Block(16, 16, n_blocks[0], 1)
            self.res4 = Block(16, 32, n_blocks[1], 2)
            self.res5 = Block(32, 64, n_blocks[2], 2)
            self.fc7 = L.Linear(64, n_class)

    def __call__(self, x):
        h = F.relu(self.bn2(self.conv1(x)))
        h = self.res3(h)
        h = self.res4(h)
        h = self.res5(h)
        h = F.average_pooling_2d(h, h.shape[2:])
        h = self.fc7(h)
        return h


class ResNet20(ResNet):
    def __init__(self, n_class=10):
        super(ResNet20, self).__init__(n_class, [3, 3, 3])
```

- Train script (`train.py`)
  - Reference: "Homework 1 - building_model.pdf"

```
from __future__ import print_function
import chainer
from chainer import training
from chainer.training import extensions
from chainer import serializers
import chainer.links as L
import argparse
from resnet_cifar10 import ResNet20


parser = argparse.ArgumentParser(description="Chainer example: Cifar-10")
parser.add_argument(
    "--batchsize",
    "-b",
    type=int,
    default=128,
    help="Number of images in each mini-batch",
)
parser.add_argument(
    "--epoch",
    "-e",
    type=int,
    default=10,
    help="Number of sweeps over the dataset to train",
)
parser.add_argument(
    "--gpu", "-g", type=int, default=0, help="GPU ID (negative value indicates CPU)"
)  # Set the initial matrixformat(numpy/cupy)
parser.add_argument(
    "--out", "-o", default="result/4", help="Directory to output the result"
)
parser.add_argument(
    "--resume", "-r", default="", help="Resume the training from snapshot"
)
parser.add_argument(
    "--unit", "-u", type=int, default=10, help="Number of output layer units"
)

if __name__ == "__main__":
    args = parser.parse_args(["-g", "-1"])  # The negative device number means CPU.
    print("GPU: {}".format(args.gpu))
    print("# unit: {}".format(args.unit))
    print("# Minibatch-size: {}".format(args.batchsize))
    print("# epoch: {}".format(args.epoch))
    print("")

    model = ResNet20(args.unit)
    classifier_model = L.Classifier(model)
    # if args.gpu >= 0:
    # chainer.cuda.get_device(args.gpu).use() # Make a specified GPU current
    # classifier_model.to_gpu() # Copy the model to the GPU

    optimizer = (
        chainer.optimizers.Adam()
    )  # Adam is one of the Gradient descent optimizationalgorithms.
    optimizer.setup(classifier_model)

    train, test = chainer.datasets.get_cifar10()
    train_iter = chainer.iterators.SerialIterator(
        train, args.batchsize
    )  # Set Training data batchiterater
    test_iter = chainer.iterators.SerialIterator(
        test, args.batchsize, repeat=False, shuffle=False
    )
    # Forward the test data after each of training to calcuat the validation loss/arruracy.
    # updater and trainer
    updater = training.StandardUpdater(train_iter, optimizer, device=args.gpu)
    trainer = training.Trainer(updater, (args.epoch, "epoch"), out=args.out)
    # extension objects
    trainer.extend(extensions.Evaluator(test_iter, classifier_model, device=args.gpu))
    trainer.extend(extensions.dump_graph("main/loss"))
    trainer.extend(extensions.snapshot(), trigger=(1, "epoch"))
    trainer.extend(extensions.LogReport())
    trainer.extend(
        extensions.PrintReport(
            [
                "epoch",
                "main/loss",
                "validation/main/loss",
                "main/accuracy",
                "validation/main/accuracy",
                "elapsed_time",
            ]
        )
    )
    trainer.extend(extensions.ProgressBar())

    if args.resume:
        # Resume from a snapshot
        serializers.load_npz(args.resume, trainer)
    trainer.run()
    serializers.save_npz(
        "{}/resnet20.model".format(args.out), model
    )  # Save the model(all trainedweights)
```

- Inference (`inference.py`)
    - Reference: "Homework 1 - building_model.pdf"

```
from resnet_cifar10 import ResNet20
import argparse
import chainer
from chainer import serializers
import numpy as np

parser = argparse.ArgumentParser(description="Chainer example: Cifar-10")
parser.add_argument(
    "--out", "-o", default="result/4/resnet20.model", help="Directory to output the result"
)
parser.add_argument(
    "--unit", "-u", type=int, default=10, help="Number of output layer units"
)
# parser.add_argument('--gpu', '-g', type=int, default=0,
# help='GPU ID (negative value indicates CPU)')#Set the initial matrixformat(numpy/cupy)
if __name__ == "__main__":
    args = parser.parse_args()  # The negative device number means CPU.
    # print('GPU: {}'.format(args.gpu))
    print("# unit: {}".format(args.unit))
    print("")

    # Load the dataset
    _, test = chainer.datasets.get_cifar10()
    # Load trained model
    model = ResNet20(args.unit)
    # if args.gpu >= 0:
    # chainer.cuda.get_device(args.gpu).use() # Make a specified GPU current
    # model.to_gpu() # Copy the model to the GPU
    # xp = np if args.gpu < 0 else chainer.cuda.cupy
    xp = np

    serializers.load_npz(args.out, model)

    x = chainer.Variable(xp.asarray([test[0][0]]))  # test data
    t = chainer.Variable(xp.asarray([test[0][1]]))  # labels
    y = model(x)  # Inference result
    print("The test data label:", xp.asarray([test[0][1]]))
    print("result:", y)
    y_top5 = y.array[0].argsort()[-5:][::-1]
    print("result Top 1:", [y_top5[0]])
    print("result Top 5:", y_top5)
```