Readme

Anonymous Authors

January 2019

Contents

1 Introduction

This file contains details of setting up and running our code. We tried to provide a very thorough documentation to help reviewers. We start by installation section which explains what are the requirements for code and how to run it. Then we will explain the code base structure and configurations needed to run the code. After getting familiar with code structure, we will explain different configurations needed for training and then how to run the code. Finally, we will provide additional examples for three different scenarios you might need to validate the results.

2 Installation

First, we need to make sure these requirements are satisfied:

1. **Operating System**: The code is tested on MacOS and Linux(Ubuntu and Debian).
2. **Python**: we tested our code and all versions of Python 3.5, 3.6, and 3.7 will work.
3. **Pytorch(1.0.0)**: We used Pytorch for implementation of our computation graph and details of installation can be found on the official website. For most users installing with pip should work:

```
pip3 install torch torchvision
```

4. **Microsoft NNI toolkit**: We used NNI for hyper parameter optimization. The details of installation can be found at its official repository. However, for most users installing using pip would work:

```
pip3 install nni
```

3 Codebase Details

The important files inside the code directory are:

1. **train.py**: The main code. Given a training configuration, which we will explain below, it will train models.
2. **model_factory.py**: Creates neural networks(resnet and plane CNN) models to be used by trainer.

3. **data_loader.py**: Loads and prepares datasets(CIFAR10/CIFAR100) to be used by trainer.

4. **search_space.json**: Defines hyper parameter search space for the hyper parameter optimizer.

5. **config.yml**: The configuration file used by the hyper parameter optimizer to run an experiment.

4 Training Configurations

There are two kinds of configurations.
The first type, is used for the general training of models and are fixed during an experiment. Examples are batch size and number of epochs. These configurations are passed as arguments when running the code. For example, to define that number of epochs for training is 150, we will add $--\text{epochs } 150$ when we call the main training file.
The second type of configurations, is used by the hyper-parameter optimizer and are inside *config.yml* and *search_space.json*. These configurations will be read by the hyper-parameter optimizer when setting up an experiment. We will explore these configuration types more in the following subsections.

4.1 General Configurations

- **epochs**: Number of training epochs.
- **dataset**: Dataset name, can be either 'cifar10' or 'cifar100'. The default value is 'cifar100'.
- **batch-size**: Mini batch size used in training. The default value is 128.
- **learning-rate**: Initial learning rate for the SGD optimizer. Depending on models in might be changed during training.
- **momentum**: momentum for SGD optimizer. The default value is 0.9.
- **weight-decay**: Weight decay for SGD optimizer. The default value is 0.0001.
- **teacher**: Teacher model name. Can be 'resnetX' for resnet models and X can be any value in (8, 14, 20, 26, 32, 44, 56, 110) or 'PlaneY' for plane CNN model where Y can be any value in (2, 4, 6, 8, 10). For details of network please refer to the paper.
- **teacher-checkpoint**: Path for a file which has pre-trained teacher. The default value is empty which means there's no pre-trained teacher and we need to train the teacher.

- **student**: Student model name. Values can be in forms of ‘resnetX‘ or ‘planeY‘ as explained before.
- **cuda**: Defines whether or not train on GPU. (must have GPU supportive pytorch installed). Values ’1’ and ’true’ can be used for training on GPU.
- **dataset-dir**: location of the dataset. default is ’./data/’.

4.2 Hyper-parameter Configurations

As explained before, NNI toolkit needs a search space file (like ‘search_space.json‘ in the code directory) consists ‘T‘ and ‘lambda‘ in **equation (3)** of the paper. Also, in order to get reliable results, there will be multiple ‘seeds‘ to avoid bad runs. For more details on search space file, please refer to the example sections or refer to the official documentation.

5 Running The Code

Because the main code will be run by the hyper-parameter optimizer(NNI Toolkit), we only need to run the optimizer. After making sure the requirements are satisfied, you can run the optimizer using:

```
nnictl create --config config.yml
```

The hyper-parameter search space is defined inside this ”config.yml” and also general configurations such as number of epochs, dataset, and models will be inside the ”command” section of the ”config.yml” file as you can see in the code base.
So, in order to run different experiments, you need to only change ”search_space.json” file to reflect hyper-parameter changes. Let us review important parts of the default ”config.yml” file as an illusteration:

```
searchSpacePath: search_space.json
trial:
  command: python3 train.py --epochs 160 --teacher resnet110 --
student resnet8 --cuda 1
```

Inside this file, there is a *searchSpacePath* inside of which we have already defined our **hyper-parameters**. Also, There is trial part inside of which there is a command part where you can see **general configurations** are passed.

To be more helpful, we will provide more examples in the next section to help you run different experiments. In all of those examples, you need to change the ”command” line of the ”config.yml” file to tell the the hyper-parameter optimizer how to run the code.

6 Examples

As explained before, in order to run different experiments, you only need to change the "command" line of the "config.yml" file. Here are some scenarios we think they might be most helpful:

6.1 Solo Training

The simplest kind of training we need is train a single neural network without knowledge distillation.

There two reasons you might need to run the code:

1. Because we need a pre-trained teacher for knowledge distillation, the output model of this phase can be used as teacher for knowledge distillation. Although this is not mandatory because our code can work with and without the pre-trained teacher file, doing so will make the training of knowledge distillation students faster because in those trainings, we will only train students.
2. This is needed for comparing knowledge distillation(referred as KD in the paper) with a network trained without any teacher(referred NOKD in the paper).

For example the following code says: train a resnet110 model on GPU and the dataset is cifar100. The output will be a file of like "resnet110_xyzw_best.pth.tar" which contains weights of the best model(best validation accuracy) we found during the training.

```
command: python3 train.py --epochs 160 --model resnet110 --cuda
 1 --dataset cifar100
```

After running the code. There will be multiple

6.2 Baseline Knowledge Distillation

'resnet 110' as teacher, 'resnet8' as student on CIFAR100:
In this scenario, you need to define which model will be "teacher" model and which model will be "student" model. There are also some other parameters that can be passed to override default configurations.

```
command: python3 train.py --epochs 160 --teacher resnet110 \
--student resnet8 --cuda 1 --dataset cifar10
```

6.3 TA improved Knowledge Distillation

Because our algorithm is based on multiple runs of knowledge distillation, in this section you need to run the knowledge distillation two times: first, distillate knowledge from teacher to teacher assistant and second, distillate knowledge

from student. Also, if we don't have a pretrained teacher, we can train it in step 1.

1. Train Teacher(Resnet110): This phase is not knowledge distillation. So there's no teacher and only a student trained alone.

```
command: python3 train.py --epochs 160 --model resnet110 \
--cuda 1 --dataset cifar100 '
```

2. After first step, we choose the weights which had best accuracy on valdiation data and train TA(Resnet20) with teacher (Resnet110) weights. Say the best resnet110 weights file was resnet110_XXXX_best.pth.tar.

```
command: python3 train.py --epochs 160 --teacher resnet110 \
--teacher-checkpoint ./resnet110_XXXX_best.pth.tar \
--student resnet20 --cuda 1 --dataset cifar100
```

3. Repeat like step two, distillate knowledge from TA to student (Teacher is resnet20, student is resnet8). Also, we assume the best weights from step two was resnet20_XXXX_best.pth.tar. The we need to change "config.yml" file to:

```
command: python3 train.py --epochs 160 --teacher resnet20 \
--teacher-checkpoint ./resnet20_XXXX_best.pth.tar \
--student resnet8 --cuda 1 --dataset cifar100
```