README.md

[Reproducibilty Challenge] RigL

This repository hosts source code for our reproducibility report on Rigging the Lottery: Making all Tickets Winners, published at ICML 2020.

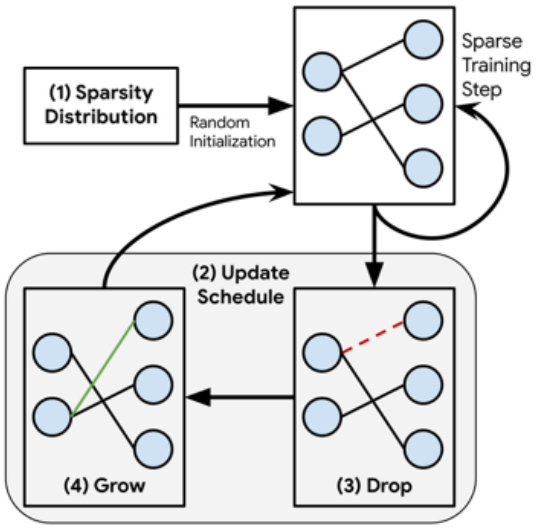

Figure Courtsey: Evci et al. 2020.

Getting Started

▼ Install

- `python3.8`
- `pytorch` : 1.7.0+ (GPU support preferable).

Then,

- `make install`

▼ W&B API key

Copy your WandB API key to `wandb_api.key` . Will be used to login to your dashboard for visualisation. Alternatively, you can skip W&B visualisation, and set `wandb.use=False` while running the python code or `USE_WANDB=False` while running make commands.

▼ Unit Tests

`make test` . Run `make help` to see specific make commands.

▼ API Documentation

See `docs/index.html` , autogenerated via Sphinx. Run `make docs.github` to refresh.

Example Code

▼ Train WideResNet-22-2 with RigL on CIFAR10

```
make cifar10.ERK.RigL DENSITY=0.2 SEED=0
```

Change `DENSITY` incase you want to use a different density (1 - sparsity) level. See `outputs/CIFAR10/RigL_ERK/0.2/` for checkpoints etc.

▼ **Train ResNet-50 with SNFS on CIFAR100**

```
make cifar100.ERK.SNFS DENSITY=0.2 SEED=0
```

See `outputs/CIFAR100/SNFS_ERK/0.2` for checkpoints etc.

▼ **Evaluate WideResNet-22-2 with RigL on CIFAR10**

Either train WRN-22-2 with RigL as described above, or download checkpoints from here. Place under `outputs/CIFAR10/RigL_ERK/0.2/+specific=cifar10_wrn_22_2_masking,seed=0` .

```
make cifar10.ERK.RigL DENSITY=0.2 SEED=0
```

▼ **Evaluate ResNet-50 with SNFS on CIFAR100**

Either train ResNet-50 with SNFS as described above, or download checkpoints from here. Place under `outputs/CIFAR100/SNFS_ERK/0.2/+specific=cifar100_resnet50_masking,seed=0` .

```
make cifar100.ERK.SNFS DENSITY=0.2 SEED=0
```

Main Results

▼ **Pre-trained Models**

All checkpoints can be found here. Place folders under `outputs/` .

▼ **Commands**

The following make command runs all the main results described in our reproducibility report.

```
make cifar10 DENSITY=0.05,0.1,0.2,0.5
make cifar100 DENSITY=0.05,0.1,0.2,0.5
make cifar10_tune DENSITY=0.05,0.1,0.2,0.5
```

Use the `-n` flag to see which commands are executed. Note that these runs are executed sequentially, although we include parallel processes for cifar10 runs of a particular method. Eg: `cifar10.Random.RigL` runs RigL Random for densities `0.05,0.1,0.2,0.5` , `seed=0` in parallel.

It may be preferable to run specific make commands in parallel for this reason. See `make help` for an exhaustive list.

▼ **Table of Results**

Shown for 80% sparsity (20% density) on CIFAR10. For exhaustive results and their analysis refer to our report.

Method	Accuracy (Test)	FLOPS (Train, Test)
Small Dense	91.0 ± 0.07	0.20x, 0.20x
Static	91.2 ± 0.16	0.20x, 0.20x
SET	92.7 ± 0.28	0.20x, 0.20x
RigL	92.6 ± 0.10	0.20x, 0.20x
SET (ERK)	92.9 ± 0.16	0.35x, 0.35x
RigL (ERK)	**93.1 ± 0.09**	0.35x, 0.35x
Pruning	93.2 ± 0.27	0.41x, 0.27x
RigL_2x	93.0 ± 0.21	0.41x, 0.20x

RigL_2x (ERK)	**93.3 ± 0.09**	0.70x, 0.35x

▼ **Visualization & Plotting Code**

Run `make vis`.

Misc

This section may be useful if you desire to extend this code base or understand its structure. `main.py` is the python file used for training-evaluating, and the `make` commands serve as a wrapper for it.

▼ **Print current config**

We use hydra to handle configs.

```
python main.py --cfg job
```

See `conf/configs` for a detailed list of default configs, and under each folder of `conf` for possible options.

▼ **Understanding the config setup**

We split configs into various config groups for brevity.

Config groups (example):

- masking
- optimizer
- dataset etc.

Hydra allows us to override these either group-wise or globally as described below.

▼ **Overrriding options / group configs**

```
python main.py masking=RigL wandb.use=True
```

Refer to hydra's documentation for more details.

▼ **Exhaustive config options**

See `conf/config.yaml` and the defaults it uses (eg: `dataset: CIFAR10`, `optimizer: SGD`, etc.).

▼ **Using specific configs**

Sometimes, we want to store the specific config of a run with tuned options across mutliple groups (masking, optimizer etc.)

To do so:

- store your config under `specific/`.
- each YAML file must start with a `# @package _global_` directive. See `specific/` for existing examples.
- override only what has changed, i.e., donot keep redundant arguments, which the base config (`config.yaml`) already covers.

Syntax:

```
python main.py +specific=cifar_wrn_22_2_rigl
```

References

1. Rigging the Lottery: Making All Tickets Winners, Original Paper.
2. Our report on OpenReview.