GMvandeVen / continual-learning

PyTorch implementation of various methods for continual learning (XdG, EWC, online EWC, SI, LwF, DGR, DGR+distill, RtF, iCaRL).

#deep-learning #artificial-neural-networks #continual-learning #lifelong-learning #incremental-learning #replay #distillation #generative-models #variational-autoencoder #elastic-weight-consolidation #replay-through-feedback #icarl

11 commits | 1 branch | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

GMvandeVen Merge branch 'master' of https://github.com/GMvandeVen/continual-lear… ··· Latest commit `ff0e03c` on May 17

.gitignore	Initial upload.	last year
LICENSE	Initial upload.	last year
README.md	Update README.md	6 months ago
_compare.py	Made files executable.	6 months ago
_compare_replay.py	Made files executable.	6 months ago
_compare_taskID.py	Made files executable.	6 months ago
_compare_time.py	Made files executable.	6 months ago
callbacks.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
continual_learner.py	Adapted code for PyTorch 0.4.	10 months ago
data.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
encoder.py	Fixed issue with distill and BCE loss that summation over classes was…	5 months ago
evaluate.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
excitability_modules.py	Adapted code for PyTorch 0.4.	10 months ago
exemplars.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
linear_nets.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
main.py	Made files executable.	6 months ago
param_stamp.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
replayer.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
train.py	Fixed issue with iCaRL (stored exemplars were not actually added to t…	6 months ago
utils.py	Fixed issue with distill and BCE loss that summation over classes was…	5 months ago
vae_models.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago

visual_plt.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago
visual_visdom.py	Added iCaRL, made replay more efficient, included extra comparisons.	6 months ago

README.md

Continual Learning

This is a PyTorch implementation of the continual learning experiments described in the following papers:

- Three scenarios for continual learning (link)
- Generative replay with feedback connections as a general strategy for continual learning (link)

Requirements

The current version of the code has been tested with:

- `pytorch 0.4.1`
- `torchvision 0.2.1`

Running the experiments

Individual experiments can be run with `main.py`. Main options are:

- `--experiment` : which task protocol? (`splitMNIST` | `permMNIST`)
- `--scenario` : according to which scenario? (`task` | `domain` | `class`)
- `--tasks` : how many tasks?

To run specific methods, use the following:

- Context-dependent-Gating (XdG): `./main.py --xdg=0.8`
- Elastic weight consolidation (EWC): `./main.py --ewc --lambda=5000`
- Online EWC: `./main.py --ewc --online --lambda=5000 --gamma=1`
- Synaptic intelligenc (SI): `./main.py --si --c=0.1`
- Learning without Forgetting (LwF): `./main.py --replay=current --distill`
- Deep Generative Replay (DGR): `./main.py --replay=generative`
- DGR with distillation: `./main.py --replay=generative --distill`
- Replay-trough-Feedback (RtF): `./main.py --replay=generative --distill --feedback`
- iCaRL: `./main.py --icarl --budget=2000`

For information on further options: `./main.py -h` .

Running comparisons from the papers

"Three CL scenarios"-paper

This paper describes three scenarios for continual learning (Task-IL, Domain-IL & Class-IL) and provides an extensive comparion of recently proposed continual learning methods. It uses the permuted and split MNIST task protocols, with both performed according to all three scenarios.

A comparison of all methods included in this paper can be run with `_compare.py`. The comparison in Appendix B can be run with `_compare_taskID.py`, and Figure C.1 can be recreated with `_compare_replay.py`.

"Replay-through-Feedback"-paper

The three continual learning scenarios were actually first identified in this paper, after which this paper introduces the Replay-through-Feedback framework as a more efficent implementation of generative replay.

A comparison of all methods included in this paper can be run with `_compare_time.py` . This includes a comparison of the time these methods take to train (Figures 4 and 5).

We should note that the results reported in this paper were obtained with this earlier version of the code.

On-the-fly plots during training

With this code it is possible to track progress during training with on-the-fly plots. This feature requires `visdom` , which can be installed as follows:

```
pip install visdom
```

Before running the experiments, the visdom server should be started from the command line:

```
python -m visdom.server
```

The visdom server is now alive and can be accessed at `http://localhost:8097` in your browser (the plots will appear there). The flag `--visdom` should then be added when calling `./main.py` to run the experiments with on-the-fly plots.

For more information on `visdom` see https://github.com/facebookresearch/visdom.

Citation

Please consider citing our papers if you use this code in your research:

```
@article{vandeven2019three,
  title={Three scenarios for continual learning},
  author={van de Ven, Gido M and Tolias, Andreas S},
  journal={arXiv preprint arXiv:1904.07734},
  year={2019}
}

@article{vandeven2018generative,
  title={Generative replay with feedback connections as a general strategy for continual learning},
  author={van de Ven, Gido M and Tolias, Andreas S},
  journal={arXiv preprint arXiv:1809.10635},
  year={2018}
}
```

Acknowledgments

The research projects from which this code originated have been supported by an IBRO-ISN Research Fellowship, by the Lifelong Learning Machines (L2M) program of the Defence Advanced Research Projects Agency (DARPA) via contract number HR0011-18-2-0025 and by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. Disclaimer: views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, IARPA, DoI/IBC, or the U.S. Government.