

SequeL: A Continual Learning Library in PyTorch and JAX

Nikolaos Dimitriadis
EPFL

nikolaos.dimitriadis@epfl.ch

François Fleuret
University of Geneva

francois.fleuret@unige.ch

Pascal Frossard
EPFL

pascal.frossard@epfl.ch

Abstract

Continual Learning is an important and challenging problem in machine learning, where models must adapt to a continuous stream of new data without forgetting previously acquired knowledge. While existing frameworks are built on PyTorch, the rising popularity of JAX might lead to divergent codebases, ultimately hindering reproducibility and progress. To address this problem, we introduce SequeL, a flexible and extensible library for Continual Learning that supports both PyTorch and JAX frameworks. SequeL provides a unified interface for a wide range of Continual Learning algorithms, including regularization-based approaches, replay-based approaches, and hybrid approaches. The library is designed towards modularity and simplicity, making the API suitable for both researchers and practitioners. We release SequeL¹ as an open-source library, enabling researchers and developers to easily experiment and extend the library for their own purposes.

1. Introduction

The field of Continual Learning (CL), also known as Lifelong Learning [31], Incremental Learning [33], or Sequential Learning, has seen fast growth in recent years. Continual Learning addresses the important setting of incrementally learning from a stream of data sources, disposing of the long-standing i.i.d. assumption of traditional machine learning. However, the pace of innovation has led to diverging settings in terms of datasets, assumptions, and requirements. As a consequence, several works have attempted to unify the Continual Learning paradigms [3, 24].

The plethora of Continual Learning settings is accompanied by a variety of Deep Learning libraries, such as PyTorch, JAX, and TensorFlow, leading to further division. Each Deep Learning library has different advantages, and researchers opt for the one that better suits their needs and prior experience. Over time, the influx of new methods results in disconnected repositories, stagnating progress due

to limited reusability and lack of reproducibility.

In this work, we propose *SequeL*, i.e., Sequential Learning, a Continual Learning framework written in both PyTorch and JAX. *SequeL* aims to unite the divergent codebases while allowing researchers to prototype fast without delving into engineering code, e.g., training loops and metric tracking. Users can develop in the framework of their choosing while accessing the already implemented baselines. For example, consider the case where one researcher wants to implement a novel algorithm in JAX, but all the baselines are in PyTorch. Reimplementing everything from scratch is time-consuming and prone to mistakes. Instead, they can use our proposed framework to integrate their method and compare with baselines in an equal footing.

Overall, *SequeL* offers a unified and flexible framework for Continual Learning research, which is easily extensible

```
import sequel
import optax
# PyTorch -> from torch.optim import optim

benchmark = sequel.benchmarks.SplitMNIST(
    num_tasks=5, batch_size=10)

model = sequel.backbones.jax.MLP(
    widths=[200], num_classes=10)
optimizer = optax.sgd(learning_rate=0.1)
# for PyTorch -> optimizer = optim.SGD(lr=0.1)

algo = sequel.algorithms.jax.EWC(
    model, benchmark, optimizer,
    callbacks=[
        sequel.callbacks.JaxMetricCallback(),
        TqdmCallback()],
    loggers=[
        WandbLogger(...), CometLogger(...)],
    # additional arguments specific to EWC
    ewc_lambda=1,
    # optional arguments for all algorithms
    lr_decay=0.8)

algo.fit(epochs_per_task=1)
```

Code 1. Example of an experiment in *SequeL*. By changing the highlighted `jax` to `pytorch`, and modifying the optimizer definition, the user can opt to run the experiment in either framework.

¹<https://github.com/nik-dim/sequel>

and accessible in order to foster reproducibility. We believe *SequeL* can help researchers to better compare methods and scale up to more complex Continual Learning settings.

2. Framework

The framework contains the following modules: Benchmarks (Section 2.1), Backbones (Section 2.2), Callbacks (Section 2.3), Loggers (Section 2.4), and Algorithms (Section 2.5). Code 1 shows the interplay between the different modules; a benchmark and a backbone along with loggers are fed into the algorithm instance that is endowed with additional custom functionalities via callbacks. The algorithm serves as a trainer module and also houses the conceptual details of CL methodologies, such as Averaged-Gradient Episodic Memory [8]. *SequeL* provides flexibility by allowing users to develop their algorithm in either JAX or PyTorch. The framework has been geared towards ease-of-use and division of engineering and algorithmic details.

2.1. Benchmarks

The *Benchmarks* module provides several widely-used Continual Learning benchmarks, both in the New Instance (NI) and New Class (NC) scenarios. The currently supported benchmarks are Split/Permuted/Rotated MNIST, Split CIFAR10/100 and Split TinyImageNet. The *Benchmarks* module is implemented in PyTorch, since it is better suited for dynamically generating and handling data streams. During training, the input and targets are transformed to the appropriate format to ensure compatibility with both PyTorch and JAX. All supported benchmarks are based on `BaseBenchmark` class that handles most use cases such as loading training and validation streams for task t , for all tasks up to t , i.e., $\{1, 2, \dots, t\}$. Similar functionalities are provided for memory capabilities, such as loading memory streams for one or more tasks, augmenting the current task dataset with the memories of all previous tasks etc. The benchmark module also handles dataloader construction, and the input to the algorithm is a tuple (x, y, t) of inputs x , targets y and task IDs t .

Implementing new benchmarks Let T be the number of tasks. Each Benchmark must implement the method `prepare_datasets` that returns two dictionaries of T key-value pairs that contain the dataset for the corresponding task $t \in [T]$, for training and validation. For New Class scenarios, such as `SplitMNIST` or `SplitCIFAR100`, the method creates a disjoint datasets and for New Instance scenarios, such as `PermutedMNIST` or `RotatedMNIST`, each dataset is coupled with a specific `torchvision` transformation, regarding the fixed permutation or rotation.

2.2. Models/Backbones

The *Models* module contains neural networks widely used in the literature, such as MultiLayer Perceptrons and Convolutional Neural Networks, Residual Networks [18]. For both JAX and PyTorch, a `BaseBackbone` class is defined that inherits from `flax.nn.Module` and `torch.nn.Module`, respectively and endows the model with the functionality of selecting the output head for NC benchmarks. The user can easily extend the module with models stemming from the literature or that are custom-made by changing the base class to `BaseBackbone`. A utility model is also provided that receives as input a `torch/flax.nn.Module` and wraps it with the appropriate `BaseBackbone` to facilitate importability.

2.3. Callbacks

A callback provides hooks for any point in training and validation, similar to Pytorch Lightning [15]. It offers the ability to extend or probe the algorithm and/or model during fitting. Metric callbacks have been implemented for both JAX and PyTorch and handle the monitoring of metrics, e.g., accuracy and forgetting, and the ad hoc tracking via the *Loggers* module. Utility callbacks can be implemented, such as the `TqdmCallback` that provides additional information during training via a progress bar.

2.4. Loggers

Experiment tracking has become an indispensable part of the ML pipeline. Hence, *SequeL* includes five different loggers: `LocalLogger`, `ConsoleLogger`, `WandbLogger`², `TensorBoardLogger` and `CometLogger`, allowing users to track their runs with the preferred service. Specifically, `LocalLogger` saves the evolution and final metrics in a local file, while `ConsoleLogger` prints information as a table in the console. `WandbLogger`, `TensorBoardLogger` and `CometLogger` use the APIs of the homonym services, allowing the tracking of images, tables etc. and the integration with powerful visualization tools and dashboards.

2.5. Algorithms

The *Algorithms* module controls the program flow and incorporates all the aforementioned modules. By calling the `fit` method, training with validation occurs for the selected backbone for the given benchmark, tracking metrics via the corresponding callback and logging them to the desired service via a logger. The parent class `BaseAlgorithm` handles the engineering code, while the algorithmic parts are implemented by the children classes. This design choice is motivated by the desire to have access to all internal variables, such as the input x and task ID t of the current batch,

²Wandb refers to Weights & Biases [4]

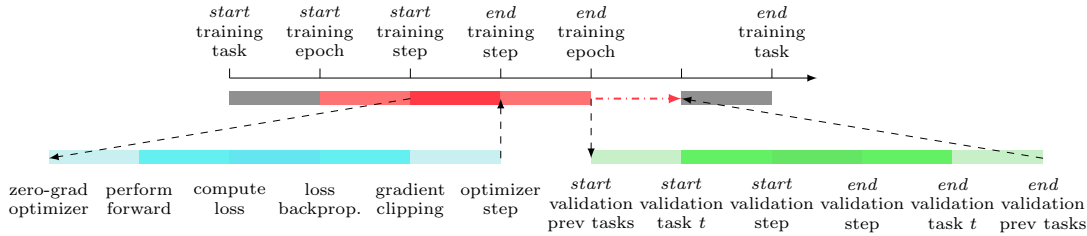


Figure 1. Control flow of the fitting process. Every point is surrounded by callback hooks. For instance, `training_step()` is preceded by `on_before_training_step()` and `on_before_training_step_callbacks()` and proceeded by `on_after_training_step()` and `on_after_training_step_callbacks()`.

without using a separate training module. As a result, the engineering logic is kept separate from research code via inheritance. Figure 1 shows a simplified version of the program flow. Each event is encircled by the homonym callbacks.

The `BaseAlgorithm` class is framework agnostic and primarily sets the control flow of the program, such as training for one task and then validating current and preceding tasks. The peculiarities and design constraints imposed by the PyTorch and JAX philosophy are handled by the corresponding base classes, `PyTorchBaseAlgorithm` and `JaxBaseAlgorithm`. For instance, for PyTorch the current batch is moved to the appropriate CUDA device, while for JAX it is converted to the NumPy format.

The `BaseAlgorithm` offers basic functionality and uses callbacks for specific and custom functionalities. It also inherits from `BaseCallback` and provides the same hooks outlined in Section 2.3. Overall, each event E in training, e.g., `training_epoch()` is surrounded by four hooks in the following sequence: `on_before_E`, `on_before_E_callbacks`, E , `on_after_E` and `on_after_E_callbacks`. Hence, the user can choose to implement an algorithm via specific callbacks or in child classes so that the research code is concentrated in a single file. For improved readability, the currently supported methods opt for the latter. Calculation of metrics, utilities for printing to console etc. are reserved for callbacks and the corresponding hooks.

The framework includes implementations for Naive Finetuning, Elastic Weight Consolidation (EWC) [22], Synaptic Intelligence (SI) [38], Memory Aware Synapses (MAS) [3], Averaged-Gradient Episodic Memory (A-GEM) [8], Less-Forgetting Learning (LFL) [21], Experience Replay (ER) [9], Dark Experience Replay (DER and DER++) [7], Stable SGD [27], Kernel Continual Learning (KCL) [11], Look Ahead Model Agnostic Meta Learning (LaMAML) [17], and Mode Connectivity Stochastic Gradient Descent (MC-SGD) [26].

Implementing new algorithms *SequeL* supports regu-

larization and replay algorithms, via out-of-the-box components. Parent classes are implemented for the specific realizations of regularization-based algorithms such as Elastic Weight Consolidation. For replay methods, the `MemoryMechanism` class and the corresponding callback handle saving samples in the memory and their selection process. For regularization algorithms, the overall loss for sample (\mathbf{x}, y, t) of a classification problem is $\mathcal{L}(\mathbf{x}, y) = \mathcal{L}_{CE}(f_{\theta}(\mathbf{x}), y) + \lambda \sum_i \Omega_i (\theta_i - \theta_{i,old})^2$ where f is a neural network parameterized by θ , θ_{old} are the parameters at the end of training of the previous task, Ω_i refers to the importance of parameter i and λ is the regularization coefficient. To add a new regularization method, the user needs only to implement the `calculate_parameter_importance` method to calculate Ω_i , while the storing of the old parameters and the calculation of the regularization loss is handled by the parent class. In case of algorithms such as Synaptic Intelligence [38] that keep an online internal parameter ω_i that is later used to compute Ω_i , the method `on_after_training_step` houses the corresponding algorithmic details.

Reproducibility To encourage transparency, *SequeL* uses Hydra [37] configuration files to formalize experiments. While an experiment can be constructed as in Code 1, an alternative lies in defining a configuration file, as in Code 2. Instead of obfuscating hyperparameters and impeding reproducible results, an experiment defined with Hydra can be easily shared and reported. This feature is enabled by a series of routers that select the correct benchmark or model, and the implementation of the `from_config` method for all related module classes. *SequeL* includes such configuration files reproducing Continual Learning baselines reported in various papers. Example configuration files along with reproducibility runs tracked via Weights&Biases are provided; the experiments focus on `RotatedMNIST` and include classic algorithms, such as EWC [22] and Naive SGD, as well as more involved baselines in MCSGD [26] and LaMAML [17]. The list will be expanded to ensure correctness. See Appendix A for more details.

Hyperparameter Tuning Another benefit of the Hydra-based [37] setup is its out-of-the-box hyperparameter tuning capabilities, allowing for the quick setup of ablation studies. Specifically, the user picks as a basis the aforementioned config file and defines the settings of a grid search, such as $\text{batch_size} \in \{10, 20, 30\}$ and $\text{lr} \in \{0.01, 0.1\}$.

3. Related Work

The progress of the Machine Learning community can be attributed to large extent to the development of Deep Learning libraries, such as PyTorch [32], TensorFlow [2] and JAX [5], which abstract low-level engineering code and provide a high-level API to the user. Thus, researchers and practitioners can reliably develop new methodologies by focusing on the algorithmic inner workings.

The progress of the field in conjunction with the fact that majority of the ML pipelines are similar has pushed for the creation of frameworks that provide further abstractions. Pytorch Lightning [15] and fastai [19] are general ML libraries that extend flexibility via a wide range of callbacks and loggers, while minimizing the engineering overhead.

The progressive increase in abstraction has led to the development of libraries specialized towards specific ML sub-fields and that use the aforementioned software as building blocks. For instance, the HuggingFace [36] library for Transformers [34] includes pretrained models for Natural Language Processing and, more recently, Computer Vision

```
version: 0.0.1 # of SequeL framework
mode: pytorch # or jax
algo:
  name: ewc
  ewc_lambda: 1.0
benchmark:
  name: rotatedmnist
  batch_size: 10
  num_tasks: 20
  per_task_rotation: 9
backbone:
  type: mlp
  n_hidden_layers: 2
  width: 256
  num_classes: 10
  dropout: 0.2
optimizer:
  type: sgd
  lr: 0.01
  lr_decay: 0.8
training:
  epochs_per_task: 1
wandb: # enables Weights and Biases tracking
  entity: ENTITY
  project: PROJECT
```

Code 2. Example of a configuration file. Experiments in *SequeL* can be fully expressed and started easily by defining the settings in a yaml file. The configuration file includes the version of the framework as well, in this case 0.0.1.

[13]. PyTorch Geometric [16] offers a comprehensive suite of tools geared towards Graph Neural Networks (GNNs). Deep Graph Library [35] also focuses on deep learning for graphs and is also framework agnostic, i.e., it supports PyTorch, Apache MXNet and TensorFlow. *SequeL* shares this trait and offers the user the flexibility of two ecosystems in PyTorch and JAX. Multiple packages exist in the Reinforcement Learning literature, e.g., OpenAI Gym [6] and OpenAI baselines [12]. MMSegmentation [29] and Segmentation-Models-PyTorch [20] are semantic segmentation toolboxes. *SequeL*'s Hydra integration shares the design philosophy of the configuration files used in the former.

An important aspect of the ML toolbox focuses on experiment tracking and monitoring and is becoming more important given the increasing complexity of models and methods, the need of rigorous ablation studies and hyperparameter tunings. Several frameworks address these imperatives, such as MLFlow [10], Weights and Biases [4], Comet and TensorBoard [2]. *SequeL* incorporates the logging capabilities of such libraries and allows users to track and visualize their experiments with the service of their choosing.

The overarching effort to create easy-to-use and reliable tools has also been observed in the Continual Learning realm. Several libraries have been proposed, such as Avalanche [23], CL-Gym [28] and Sequoia [30]. Avalanche and CL-Gym share a similar design in terms of module structure and focus on the supervised setting. The algorithm selection in CL-Gym is limited and, while Avalanche offers a wide range of algorithms, the focus lies on more classical algorithms. For instance, Kernel Continual Learning [11] and Dark Experience Replay [7] are not implemented in either framework. AvalancheRL [25] extends Avalanche with functionalities for Reinforcement Learning. Sequoia [30] focuses on the Reinforcement Learning perspective of Continual Learning and uses components of OpenAI Gym [6], Avalanche [1] and Continuum [14]. Compared to the aforementioned libraries, *SequeL* supports both PyTorch and JAX, simplifying the comparison and importability of novel methods and implementations of existing approaches irrespective of framework.

4. Conclusion

In conclusion, we have presented *SequeL*, a novel Continual Learning framework written in both PyTorch and JAX, aimed at unifying divergent codebases and facilitating reproducible research in the field of Continual Learning. Our library provides a convenient and flexible platform for researchers to prototype and test their novel algorithms, as well as compare them to existing state-of-the-art methods. We believe that our library will contribute to the growth of Continual Learning research and provide a valuable resource for the community.

References

- [1] *Avalanche: an End-to-End Library for Continual Learning*, 2nd Continual Learning in Computer Vision Workshop. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2021. 4
- [2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 4
- [3] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory Aware Synapses: Learning What (not) to Forget. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, volume 11207, pages 144–161. Springer International Publishing, Cham, 2018. 1, 3
- [4] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. 2, 4
- [5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 4
- [6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 4
- [7] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in neural information processing systems*, volume 33, page 15920–15930. Curran Associates, Inc., 2020. 3, 4
- [8] Arslan Chaudhry, Marc’Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient Lifelong Learning with A-GEM. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 2, 3
- [9] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc’Aurelio Ranzato. On tiny episodic memories in continual learning. (arXiv:1902.10486), Jun 2019. arXiv:1902.10486 [cs, stat]. 3
- [10] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. Developments in mlflow: A system to accelerate the machine learning lifecycle. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning, DEEM20*, pages 1–4, New York, NY, USA, Jun 2020. Association for Computing Machinery. 4
- [11] Mohammad Mahdi Derakhshani, Xiantong Zhen, Ling Shao, and Cees Snoek. Kernel Continual Learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2621–2631. PMLR, 2021. 3, 4
- [12] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. <https://github.com/openai/baselines>, 2017. 4
- [13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 4
- [14] Arthur Douillard and Timothée Lesort. Continuum: Simple management of complex continual learning scenarios, 2021. 4
- [15] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. 2, 4
- [16] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, 5 2019. 4
- [17] Suyog Gupta and Berkin Akin. Accelerator-aware Neural Network Design using AutoML. *arXiv:2003.02838 [cs, eess, stat]*, Mar. 2020. 3, 7
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 2
- [19] Jeremy Howard and Sylvain Gugger. Fastai: A layered api for deep learning. *Information*, 11(2), 12020. 4
- [20] Pavel Iakubovskii. Segmentation models pytorch. https://github.com/qubvel/segmentation_models_pytorch, 2019. 4
- [21] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetful learning for domain expansion in deep neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr 2018. 3
- [22] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, Mar. 2017. 3
- [23] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. FractalNet: Ultra-Deep Neural Net-

- works without Residuals. *arXiv:1605.07648 [cs]*, May 2017. 4
- [24] David Lopez-Paz and Marc’Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6467–6476, 2017. 1
- [25] Nicolò Lucchesi, Antonio Carta, Vincenzo Lomonaco, and Davide Bacciu. Avalanche RL: A continual reinforcement learning library. In Stan Sclaroff, Cosimo Distante, Marco Leo, Giovanni Maria Farinella, and Federico Tombari, editors, *Image Analysis and Processing - ICIAP 2022 - 21st International Conference, Lecce, Italy, May 23-27, 2022, Proceedings, Part I*, volume 13231 of *Lecture Notes in Computer Science*, pages 524–535. Springer, 2022. 4
- [26] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Dilan Görür, Razvan Pascanu, and Hassan Ghasemzadeh. Linear Mode Connectivity in Multitask and Continual Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 3, 7
- [27] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 3
- [28] Seyed Iman Mirzadeh and Hassan Ghasemzadeh. Cl-gym: Full-featured pytorch library for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3621–3627, June 2021. 4
- [29] MMSegmentation Contributors. OpenMMLab Semantic Segmentation Toolbox and Benchmark, 7 2020. 4
- [30] Fabrice Normandin, Florian Golemo, Oleksiy Ostapenko, Pau Rodriguez, Matthew D Riemer, Julio Hurtado, Khimya Khetarpal, Dominic Zhao, Ryan Lindeborg, Thimothée Lesort, Laurent Charlin, Irina Rish, and Massimo Caccia. Sequoia: A Software Framework to Unify Continual Learning Research. 4
- [31] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. `tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/journals/nn/ParisiKPKW19.bib tex.timestamp: Fri, 09 Apr 2021 18:26:43 +0200`. 1
- [32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4
- [33] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental Classifier and Representation Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5533–5542. IEEE Computer Society, 2017. 1
- [34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. 4
- [35] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019. 4
- [36] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. pages 38–45. Association for Computational Linguistics, 10 2020. 4
- [37] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. 3, 4
- [38] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995. PMLR, 2017. 3

A. Reproducibility

SequeL offers configuration files for reproducibility purposes and to ensure the correctness of the framework. Two examples based on experimental results reported for RotatedMNIST [17, 26] are shown in [Code 3](#) and [Code 4](#). Instructions on how to run the experiments along with more examples are provided in the repository.

```
version: 0.0.1
expected:
  avg_acc: 82.3 # as reported in the paper
benchmark:
  name: rotatedmnist
  num_tasks: 20
  per_task_rotation: 9
  batch_size: 64
  eval_batch_size: 1024
source: mcsgd paper
algo:
  name: mcsgd
  per_task_memory_samples: 100
  lmc_policy: offline
  lmc_interpolation: linear
  lmc_lr: 0.05
  lmc_momentum: 0.8
  lmc_batch_size: 64
  lmc_init_position: 0.1
  lmc_line_samples: 10
  lmc_epochs: 1
  lr_decay: 0.8
backbone:
  type: mlp
  n_hidden_layers: 2
  width: 256
  num_classes: 10
  dropout: 0.2
optimizer:
  type: sgd
  lr: 0.1
  momentum: 0.8
training:
  epochs_per_task: 1
```

Code 3. Reproducibility Experiment for MCSGD.

```
version: 0.0.1
expected:
  avg_acc: 77.42 # as reported in the paper
source: original paper
algo:
  name: lamaml
  glances: 5
  n_inner_updates: 5
  second_order: false
  grad_clip_norm: 2.0
  learn_lr: true
  lr_alpha: 0.3
  sync_update: false
  mem_size: 200
backbone:
  type: mlp
  n_hidden_layers: 2
  width: 100
optimizer:
  type: sgd
  lr: 0.1
benchmark:
  name: rotatedmnist
  per_task_rotation: 9
  num_tasks: 20
  batch_size: 10
  eval_batch_size: 10000
  subset: 1000
training:
  epoch_per_task: 1
```

Code 4. Reproducibility Experiment for LaMAML.