# Towards Continually Learning Application Performance Models

**Ray A. O. Sinurat**
Department of Computer Science
University of Chicago
rayandrew@cs.uchicago.edu

**Anurag Daram**
Department of Electrical & Computer Engineering
University of Texas at San Antonio
anurag.daram@utsa.edu

**Haryadi S. Gunawi**
Department of Computer Science
University of Chicago
haryadi@cs.uchicago.edu

**Robert B. Ross**
Mathematics & Computer Science Division
Argonne National Laboratory
rross@mcs.anl.gov

**Sandeep Madireddy***
Mathematics & Computer Science Division
Argonne National Laboratory
smadireddy@anl.gov

## Abstract

Machine learning-based performance models are increasingly being used to build critical job scheduling and application optimization decisions. Traditionally, these models assume that data distribution does not change as more samples are collected over time. However, owing to the complexity and heterogeneity of production HPC systems, they are susceptible to hardware degradation, replacement, and/or software patches, which can lead to drift in the data distribution that can adversely affect the performance models. To this end, we develop continually learning performance models that account for the distribution drift, alleviate catastrophic forgetting, and improve generalizability. Our best model was able to retain accuracy, regardless of having to learn the new distribution of data inflicted by system changes, while demonstrating a $2\times$ improvement in the prediction accuracy of the whole data sequence in comparison to the naive approach.

## 1 Introduction

The complexity of leadership-class high-performance computing (HPC) systems is increasing rapidly due to the need to handle diverse workloads and applications. In particular, storage systems and I/O architectures are integrating different heterogeneous storage technologies to maximize the price-performance trade-off. This complexity entails the need for sophisticated empirical/machine learning models to accurately predict the application performance. The accuracy of the performance model is crucial due to its use in making decisions about job scheduling, application optimization, and capacity planning of facilities.

It is commonly assumed that the data used to learn the performance models do not undergo any distribution shift. Under this stationarity assumption, the observation of more data on the application performance in the HPC system should increase the predictive performance of these machine learning models. However, as reported in recent works [1], factors such as hardware degradation [2], replacement, anomalies [3] or software upgrades [4] can affect the state of the system and, in turn, lead to a change/drift in the underlying distribution. Performance models trained on the past data will be degraded due to this data distribution shift. Such distribution shifts have been handled either by updating the model ad hoc without drift detection [5] using a sliding window of

*Contact for Correspondence

Workshop on Machine Learning for Systems at 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

data or by ignoring those data before the drift occurred. More recently, Madireddy et al. [6] proposed a moment-matching transform to correct for data drift post-detection. A more general approach to train/adapt the performance model in the presence of this drift is using *continual learning* [7] algorithms, which acknowledge the presence of data distribution shifts and are designed to prevent catastrophic forgetting of the models on the data observed before the shift when training on data observed post-drift, thus generalizing across the distribution shifts. The ability to learn continuously from an incoming data stream without catastrophic forgetting is critical for designing intelligent systems.

However, prior research in continuous learning has focused mainly on *virtual concept drift* [8] (or label drift) where there exists drift in the distribution of targets ($P(y)$) without affecting the functional relationship between the inputs and outputs ($P(y|x)$) of the model. For example, this scenario is encountered in training classification models, which should learn a new class without forgetting the previous ones when presented sequentially, one after the other. However, for performance modeling, we are interested in the *real concept drift* scenario in which learning occurs in a sequence of tasks where the input distribution ($P(x)$) remains the same but the functional relationship ($P(y|x)$) changes across tasks due to the change in the state of the system. This scenario has been less explored with few studies pertaining to image segmentation [9] and improvement in label precision [10].

To this end, we make the following contributions to this work: **(i)** we formulate performance modeling in the presence of data drifts as a *real concept drift* continuous learning scenario; **(ii)** adapted several *virtual concept drift* continuous learning approaches to performance modeling and compared to naive learning that ignores catastrophic forgetting; finally, **(iii)** our results show a **2-fold** improvement in the accuracy of the continual learning models compared to their naive learning counterparts after learning all tasks.

## 2 I/O Performance Modeling

**Data Preparation:** We collected our data from Cori, a production supercomputing system at the National Energy Research Scientific Computing Center that is used to run high-performance applications (HPC) tasks. Cori is a Cray XC40 system that comprises of 12,000 compute node and luster file systems with a 30PB capacity and peak performance of 700 GB/sec. The data used in our experiments consist of eight production applications with diverse application workloads representative of the science application typically run. We used TOKIO (Total Knowledge of I/O; [11]) framework for I/O performance profiling. As part of TOKIO, the application performance statistics is provided by Darshan [12] logs, I/O traffic in Lustre file systems is obtained using LMT [13] (Lustre Monitoring Tools), and scheduling information using Slurm [14].

Table 1: Input features extracted from the I/O monitoring data.

| Metric | Description | Units | Tool |
|---|---|---|---|
| Perc_OST_Full | Average % OST fullness | – | LMT |
| Ave_OSS_CPU | Average OSS CPU load | – | LMT |
| Ave_MDS_CPU | Average MDS CPU Load | – | LMT |
| Num_Conc_Jobs | Number of concurrent jobs | – | Slurm |
| FS_Read_Vol | Total read volume across FS | GB | LMT |
| FS_Read_Vol | Total write volume across FS | GB | LMT |
| Num_Mkdir_Op | Number of mkdir operations | – | LMT |
| Num_Rename_Op | Number of rename operations | – | LMT |
| Num_Rmdir_Op | Number of rmdir operations | – | LMT |
| Num_Unlink_Op | Number of unlink operations | – | LMT |

All of these applications were run contemporaneously on Cori; hence, the temporal location of the system changes should be consistent across the applications. Specifically, two major software upgrades were performed during the course of this study. Both upgrades affected the I/O performance of some applications. Therefore, it can be assumed that the data are divided into three different regimes by the two software upgrades as seen in Table 1 that show the normalized I/O performance for a single application at the top and all eight applications at the bottom. The magenta line indicates the time that the software updates were performed, and the top figure shows the shift in the I/O performance distribution. We consider metrics that capture filesystem traffic (using LMT) and system load (using Slurm) as features used by the performance

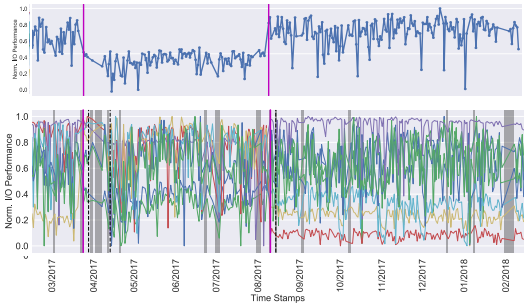



Figure 1: I/O performance as a function of time for all eight applications (bottom) and one of the applications (top); Magenta lines show the location of the software upgrade.

model to predict I/O performance. We also do not preprocess these features to choose a subset of these metrics that are not collinear. A summary of the adopted features is shown in Table 1.

## 3 Continual Learning

The goal of continual learning (CL) is to improve the model over time while retaining prior knowledge or experiences. CL came to fruition as a solution when the model forgets prior knowledge due to a dynamic data stream that changes over time [8]. This change is typically in the data distribution which leads to data drift/non-stationarity and ignoring it can lead to catastrophic forgetting [15]. Formally, continual learning on a sequence of tasks ($T^t, t = 1, 2, .., N \; \forall \, T \in \mathscr{T}$) consisting of ordered pairs of input data points and their corresponding targets $\{\mathcal{X}^t, \mathcal{Y}^t\}$ aims to maximize the performance of a system across all tasks $\mathscr{T}$, when trained sequentially. There are different kinds of data drifts that introduce non-stationarity in learning and motivate the need for continual learning approaches. Some well-known drifts that affect supervised learning include: (1) **real concept drift**: occurs when the distribution of inputs remain same across tasks, i.e, $P_t(x) = P_{t+1}(x), \forall x \in \mathcal{X}$ but the functional relation changes, i.e., $P_t(y|x) \neq P_{t+1}(y|x), \forall (x, y) \in (\mathcal{X}, \mathcal{Y})$; (2) **virtual concept drift** happens when the distribution shifts only happen inside target variables ($P_t(y) \neq P_{t+1}(y)$) without altering the relationship towards input variables; (3) **domain drift** exists if the drift occurs in the input variables ($P_t(x) \neq P_{t+1}(x)$) without affecting the relationship towards target variables.

We observed that the I/O performance data undergoes *real concept drift*, which is much harder and less studied in the context of continual learning. Most works have considered *virtual concept drift* in the classification scenario where data from disjoint classes form new tasks. Although I/O performance modeling is a regression problem, we formulate it as a classification problem by dividing the continuous output range into an equally spaced interval for simplicity, in addition to the potential to adapt the high-performing algorithms from the literature as well as software that have been primarily targeting the classification scenarios.

**Formulation and implementation of learning with real concept drift:** To this end, we use the API provided by Avalanche [16], a continual learning framework based on PyTorch [17], and adopt the existing implementation of the classification-based continual learning algorithms to learn in the presence of real concept drift. Specifically, we incorporated the class-incremental setting [18] from *virtual concept drift* works but modified the learning so that each task's output spans the entire space as opposed to each of them accumulating from disjoint classes in the previous case. In addition to that, we keep the task id in training and inference consistent with the traditional class-incremental learning, where the task id is provided at training, but not inference. In this paper, the task id is an integer that provides essential information when the software upgrade occurs. We also note that as we described in Section 2, we used the aggregation of performance statistics provided by Darshan [12] as our target variable ($y$) which results in a single floating number that should be restricted to regression problems. We converted regression problems into classification problems by splitting the $y$, which infinitely spans from $[0.0, 1.0]$, into 10 regimes equally.

To train our model, we need to provide features ($x$), targets ($y$) at training time, and additional task id ($t$) while splitting the data. We achieve this by providing task id ($t$) to Avalanche data loader and treating each dataset with a different task id as a separate dataset. This separation of the dataset is required for informing Avalanche to split the data correctly based on its performance shifts after upgrading software. Our model is implemented in PyTorch which then will be continually learning inside Avalanche based on the chosen methodology.

**Continual learning strategies:** In this work, we adopt six different methodologies that are popular benchmarks and have an existing implementation within Avalanche [16]. These approaches include (1) **Elastic Weight Consolidation (EWC)** [19], which is a regularization-based approach that measures the importance of the parameters for the current task and penalizes future updates. The regularization term of EWC consists of a quadratic penalty term for each previously learned task, whereby each task's term penalizes the parameters for how different they are compared to their value directly after finishing the training on that task. The strength of each parameter's penalty depends for every task on how important that parameter was estimated to be for that task, with higher penalties for more important parameters; (2) **Synaptic Intelligence (SI)** [20] is another regularization-based approach that consists of only one quadratic term that penalizes changes to important parameters which are identified by tracking each synapse's credit assignment during the task. The importance parameter is measured by computing the per parameter contribution to the change of loss for the current task and thus strongly contributing parameters are heavily penalized in subsequent tasks. (3) **Learning without Forgetting (LwF)** [21] is a distillation-based approach towards continual learning, wherein previous model outputs are used as soft labels for previous tasks. For this, each

input to be replayed is labeled with a "soft target", which is a vector containing a probability for each active class; (4) **Averaged Gradient Episodic Memory (A-GEM)** [22] uses episodic memory as an optimization constraint to avoid catastrophic forgetting. The sample handling of A-GEM avoids solving a quadratic optimization problem for handling samples in the buffer and, instead uses the mean gradient of such samples from the buffer. The sample is selected, if they point in the same direction in which the current gradient is applied, otherwise an orthogonal projection to the averaged gradient is performed; (5) **Gradient-based Sample Selection (GSS) Greedy** [23] is a sample selection strategy for a setup without task boundaries or at least knowledge about these. Each seen sample is regarded as an individual constraint, to which every following sample must be compatible. Sample selection in this context is identical to a constraint reduction problem which is solved by a greedy strategy. This strategy selects $n$ random samples from the buffer and calculates the cosine-similarity between the gradient of the current sample and the gradients of the selected samples. Samples of the buffer are only replaced if the similarity falls below a defined threshold, that is, the sample with maximal cosine-similarity is replaced; (6) **Greedy Sampler and Dumb learner (GDumb)** [24] greedily stores samples balanced over the observed classes and at test time learns a new model from scratch with the rehearsal memory. This approach consists of two components, namely the gradient balancing sampler and learner. The sampler greedily creates a new bucket for that class and starts removing samples from the old ones, in particular, from the one with a maximum number of samples.

We also adopt the baseline/**Naive**, which is a simple methodology in continual learning that does not do any optimization to prevent catastrophic forgetting. In Naive, a model is just incrementally fine-tuned depending on the new (drifted) data. Naive is provided natively inside Avalanche to give the lowest baseline of a model that suffers from catastrophic forgetting and mimics typical (stationary) supervised learning.

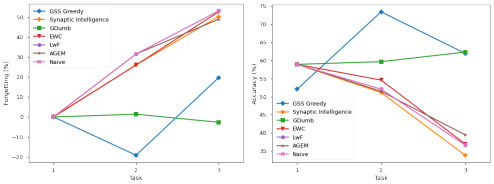
## 4 Results and Discussions

The following section defines the experiment setup and establishes the improvement induced by the use of performance modeling as proposed. All experiments were carried out inside *bare metal* machines with a single Intel (R) Xeon (R) CPU @ 2.00GHZ CPUs consisting of 8 cores with hyper-threading enabled, 12 GiB of memory, and 200 GB of disk space. The configurations of the model and the continual learning methodologies can be seen in Appendix A.1. For memory-based methodologies such as GSS Greedy and GDumb, we fixed the memory size to obtain fair results and comparisons. To evaluate the efficacy of our proposed model, we used two different metrics: (1) **Average Forgetting** that measures how bad a model forgets about prior tasks after being trained on a new task [25]; and (2) **Average Accuracy** is used to determine the accuracy of the trained model to predict current and prior data after training [25].

Table 2: Accuracy and forgetting metrics.

| Method | Task # | Train Acc | Avg Acc | Avg Forgetting |
|---|---|---|---|---|
| Naive | 1 | 0.668 | 0.589 | 0 |
| | 2 | 0.857 | 0.521 | 0.315 |
| | 3 | 0.879 | 0.367 | **0.531** |
| Synaptic Intelligence | 1 | 0.668 | 0.589 | 0 |
| | 2 | 0.702 | 0.511 | 0.260 |
| | 3 | 0.776 | 0.338 | 0.503 |
| EWC | 1 | 0.668 | 0.589 | 0 |
| | 2 | 0.850 | 0.546 | 0.260 |
| | 3 | 0.952 | 0.370 | 0.526 |
| LwF | 1 | 0.668 | 0.589 | 0 |
| | 2 | 0.857 | 0.521 | 0.315 |
| | 3 | 0.879 | 0.367 | **0.531** |
| AGEM | 1 | 0.668 | 0.589 | 0 |
| | 2 | 0.837 | 0.515 | 0.315 |
| | 3 | 0.924 | 0.395 | 0.489 |
| GSS Greedy | 1 | 0.617 | 0.521 | 0 |
| | 2 | 0.984 | 0.734 | -0.192 |
| | 3 | 0.990 | **0.619** | 0.197 |
| GDumb | 1 | 0.694 | 0.589 | 0 |
| | 2 | 0.712 | 0.596 | 0.014 |
| | 3 | 0.703 | **0.623** | -0.027 |

Among all of the methodologies tested, GDumb and GSS Greedy are able to retain prior knowledge from the previous tasks while learning our I/O profiling data. Interestingly, the results are different for the other methodologies used, such as EWC, LwF, Synaptic Intelligence, and AGEM. These methodologies still suffer catastrophic forgetting after Task 3 even though the Naive methodology was provided as the lowest baseline (Fig. 2a), which can be attributed to the fact that learning in the presence of *real concept drift* is a much harder problem.




(a) Average Forgetting (b) Average Accuracy

Figure 2: Comparison of the accuracy and forgetting metrics for all the approaches as a function of the task sequence.

*GSS Greedy* is the best methodology while training from Task 1 to Task 2. However, it suffers greatly from catastrophic forgetting after training Task 3. Furthermore, GSS Greedy also has the best accuracy when training a single task (Table 2). Consequently, after Task 3, GSS Greedy is not able to

maintain its performance due to its memory size limitation and poor greedy selection of prior task knowledge. Despite its performance degradation after learning Task 3, GSS Greedy is still one of the best results we got in this experiment. *GDumb* performs best on our data by successfully retaining knowledge from Task 1 to Task 3. The adaptation of GDumb to new tasks without forgetting its previous tasks can be clearly seen in (Fig. 2a). GDumb gets the best result after training Task 3 and performed better than the other methodologies, albeit not getting the best accuracy for training a single task, in contrast to GSS Greedy.

## 5 Conclusions and Future Works

In this work, we focus on modeling applications performance inside HPC production systems which usually suffer from catastrophic forgetting due to data distribution drifts as a result of system changes over time. We highlight that the distribution drift for performance modeling follows *real concept drift* and present a continual learning-based approach to performance modeling that is able to retain prior knowledge while maintaining the ability to learn new data. We incorporate real concept drift within the Avalanche framework to evaluate on various continual learning benchmarks. We propose a novel approach towards modeling continuous concept drift as a class incremental learning problem, and observe that the best performing GDumb model provides a $2\times$ improvement in accuracy over the naive approach.

So far, we have not yet performed further hyperparameter (Appendix A.1) optimization of the baseline model and the different continual learning methodologies used in this work. We believe that the performance of the model can be greatly improved by optimizing the hyperparameters. To date, the task id as described in Section 2 was collected manually from the server operators. This can be automated by using the concept drift-aware modeling created by Madireddy et al. [6] which automatically identifies the points where concept drift occurs in near-real time. We hope to integrate this modeling along with our predictor to continually and adaptively learn the data in the production systems.

In the near future, we hope that the improved model can be used in the production systems. Our model needs more data with more concept drifts to learn the characteristics of applications inside the production system. Furthermore, our model can benefit other software, such as job schedulers and monitoring tools, by providing them with enough information to aid their decision-making. All in all, by integrating our model into the production system and its software, we hope that we can improve the workload optimization that spans across production systems.

## References


[1] M. Mundt, Y. W. Hong, I. Pliushch, and V. Ramesh, "A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning," *arXiv preprint arXiv:2009.01797*, 2020.

[2] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan, "Black-Box problem diagnosis in parallel file systems," in *FAST*, 2010, pp. 43–56.

[3] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, D. Srinivasan, B. Panda, A. Baptist, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-Slow at scale: Evidence of hardware performance faults in large production systems," *ACM Trans. Storage*, vol. 14, no. 3, pp. 1–26, Oct. 2018.

[4] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. J. Wright, "A year in the life of a parallel file system," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp. 931–943.

[5] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 1–37, Mar. 2014.

[6] S. Madireddy, P. Balaprakash, P. Carns, R. Latham, G. K. Lockwood, R. Ross, S. Snyder, and S. M. Wild, "Adaptive learning for concept drift in application performance modeling," in *Proceedings of the 48th International Conference on Parallel Processing*. New York, NY, USA: ACM, Aug. 2019.

[7] S. Thrun and L. Pratt, *Learning to Learn*. Springer Science & Business Media, Dec. 2012.

[8] T. Lesort, M. Caccia, and I. Rish, "Understanding continual learning settings with data distribution drift analysis," Apr. 2021.

[9] F. Cermelli, M. Mancini, S. Rota Bulò, E. Ricci, and B. Caputo, "Modeling the background for incremental learning in semantic segmentation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 9230–9239.

[10] M. Abdelsalam, M. Faramarzi, S. Sodhani, and S. Chandar, "IIRC: Incremental Implicitly-Refined classification," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 11 033–11 042.

[11] G. K. Lockwood, N. J. Wright, S. Snyder, P. Carns, G. Brown, and K. Harms, "TOKIO on ClusterStor: Connecting standard tools to enable holistic I/O performance analysis," Lawrence Berkeley National Lab. (LBNL), Berkeley, CA (United States), Tech. Rep., Jan. 2018.

[12] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Trans. Storage*, vol. 7, no. 3, pp. 1–26, Oct. 2011.

[13] Garlick and Morrone, "Lustre monitoring tools," https://github.com/LLNL/lmt.

[14] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, 2003, pp. 44–60.

[15] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends Cogn. Sci.*, vol. 3, no. 4, pp. 128–135, Apr. 1999.

[16] V. Lomonaco, L. Pellegrini, A. Cossu, A. Carta, G. Graffieti, T. L. Hayes, M. D. Lange, M. Masana, J. Pomponi, G. van de Ven, M. Mundt, Q. She, K. Cooper, J. Forest, E. Belouadah, S. Calderara, G. I. Parisi, F. Cuzzolin, A. Tolias, S. Scardapane, L. Antiga, S. Amhad, A. Popescu, C. Kanan, J. van de Weijer, T. Tuytelaars, D. Bacciu, and D. Maltoni, "Avalanche: an end-to-end library for continual learning," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, ser. 2nd Continual Learning in Computer Vision Workshop, 2021.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and Others, "Pytorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[18] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer, "Class-incremental learning: survey and performance evaluation on image classification," Oct. 2020.

[19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, no. 13, pp. 3521–3526, 2017.

[20] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. JMLR. org, 2017, pp. 3987–3995.

[21] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[22] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-gem," *arXiv preprint arXiv:1812.00420*, 2018.

[23] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," *Advances in neural information processing systems*, vol. 32, 2019.

[24] A. Prabhu, P. H. Torr, and P. K. Dokania, "Gdumb: A simple approach that questions our progress in continual learning," in *European conference on computer vision*. Springer, 2020, pp. 524–540.

[25] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner, "Online continual learning in image classification: An empirical survey," Jan. 2021.

# A Appendix

## A.1 Hyperparameters

In Table 3, we defined the hyperparameters of neural network models, training and evaluation configurations, and continual learning approaches parameter that we used through the experiments.

Table 3: Hyperparameters

| | Parameters | Value |
|---|---|---|
| General | Epochs | 60 |
| | Training Batch Size | 4 |
| | Test Batch Size | 4 |
| | Learning Rate | 0.001 |
| | Optimizer | Adam |
| | Hidden Layers | 3 |
| | Hidden Size | 400 |
| Synaptic Intelligence | Lambda | 1.0 |
| | Eps | $1 \times 10^{-7}$ |
| EWC | Mode | Separate |
| | Lambda | 0.5 |
| LwF | Alpha | 1.0 |
| | Temperature | 2.0 |
| AGEM | Patterns Per Exp | 2 |
| GSS Greedy | Mem Size | 5000 |
| GDumb | Mem Size | 5000 |