Introducing My PhD Work in 20 Minutes

Pengxiang Wang

Peking University, School of Mathematical Sciences

2025-10-21

My Work in Continual Learning

My PhD research mainly focuses on continual learning with:

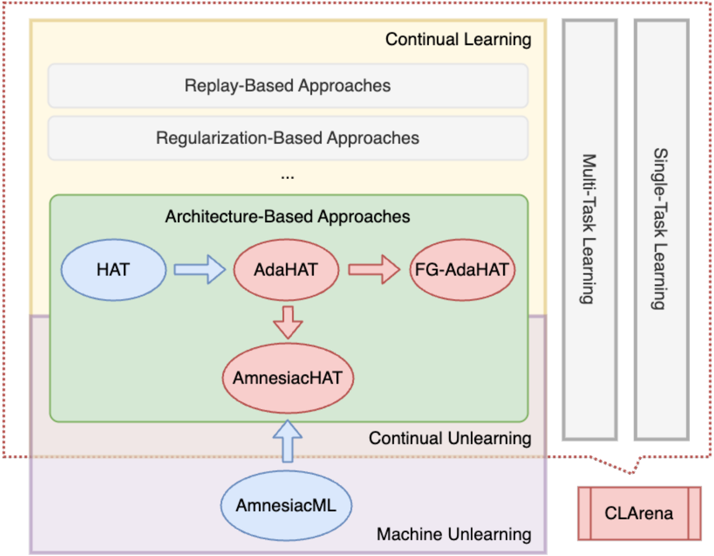
3 papers:

1. **AdaHAT**: Adaptive Hard Attention to the Task in Task-Incremental Learning
2. **FG-AdaHAT**: Fine-Grained Neuron Importance Guided Architecture-Based Continual Learning
3. **AmnesiacHAT**: Continual Unlearning via Capacity Recycling in Architecture-Based Continual Learning

1 open-source software:

- **CLArena**: A Python package for continual learning research

Connections Among My Work



My Work in Continual Learning

Work	Scope	Approach
AdaHAT	Task-Incremental Learning	Architecture-Based
FG-AdaHAT	Task-Incremental Learning	Architecture-Based
AmnesiacHAT	Continual Unlearning	Architecture-Based
CLArena	Continual Learning / Unlearning	All Kinds

Task-Incremental Learning (TIL): A typical setting in continual learning

Continual Unlearning (CUL): A new area combining machine unlearning and continual learning, hardly explored before

Architecture-Based Approach: One of the main branches of continual learning approaches

Continual Learning Background

Definition of Continual Learning

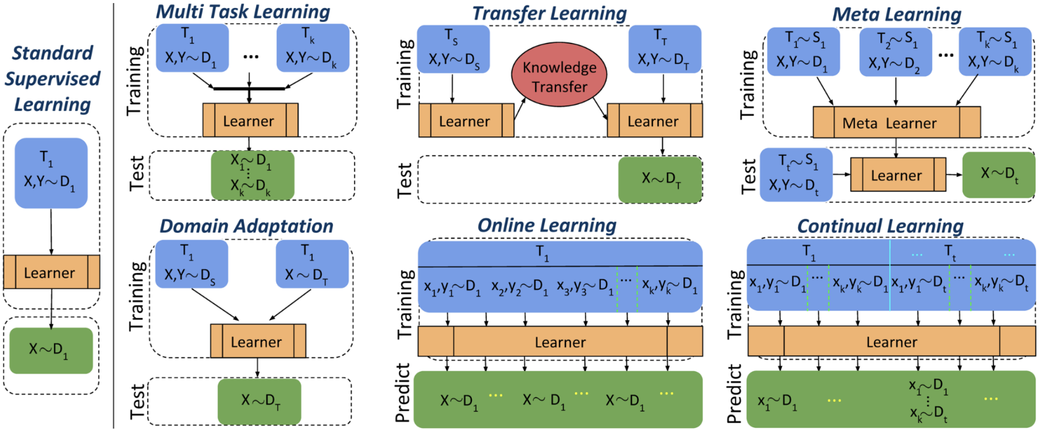
Continual Learning (CL), which becomes popular after 2017, is a machine learning paradigm where an algorithm receives the data from tasks sequentially without the access to previous ones to learn a model that performs the best for all tasks.

(a.k.a. lifelong learning, incremental learning, sequential learning)

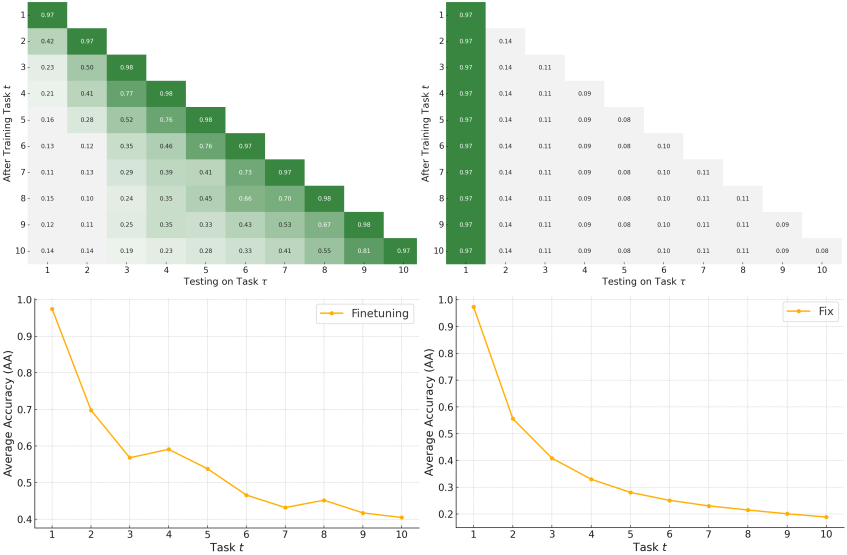
Key assumptions:

- Sequential tasks from different distributions
- Test for all tasks, but no access to previous task's data

Differences from Other Paradigms



Baselines: Finetuning and Fix



Challenges

Catastrophic Forgetting

- Previous knowledge can hardly be preserved within neural networks after training different tasks
- The main problem that most CL algorithms seek to address

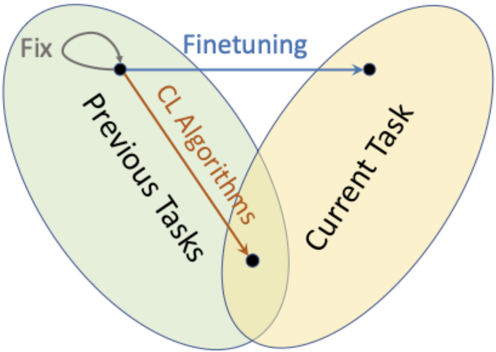
Stability-Plasticity Dilemma

- The model cannot achieve both stability and plasticity at the same time
- Need to trade off the balance of stability-plasticity

Network Capacity Problem

- Any fixed model will eventually get full as infinite tasks arrive
- Cannot select a proper-sized network beforehand under the infinite task assumption

Challenges



For more about continual learning, check out my article:
pengxiang-wang.com/posts/continual-learning-beginners-guide

Solved Problems of My Work

AdaHAT / FG-AdaHAT:

- Balances the stability-plasticity trade-off in continual learning
- Alleviates the network capacity problem that architecture-based approaches generally suffer from
- Addresses the challenges of continual learning on long task sequences

AmnesiacHAT:

- Provides a solution to continual unlearning problem, which is: how to erase knowledge learned from specific tasks without affecting other learned tasks
- Addresses the network capacity problem from a new perspective: capacity recycling through unlearning

Significance of My Work

AdaHAT / FG-AdaHAT:

- Provides effective architecture-based approaches for continual learning
- Proposes a unified gradient adjustment algorithm framework
- Paves the way for leveraging task information (especially fine-grained) in architecture-based approaches

AmnesiacHAT:

- Could be one of the foundational works in the new continual unlearning area
- Could be one of the first to explore the potential benefits from unlearning

CLArena:

- Another open-source package in the research community of continual learning
- Solid and robust codebase supporting my 3 papers above

I will introduce them as follows. You can ask questions at any time!

AdaHAT: Adaptive Hard Attention to the Task in Task-Incremental Learning

Background: Architecture-Based Approaches

Architecture-based approaches (a.k.a. parameter isolation) are one of the main branches of continual learning algorithms:

- A distinctly different strategy that decomposes the network architecture
- Dedicate different parts of a neural network to different tasks

HAT (Hard Attention to the Task) is one of the most representative architecture-based approaches:

- Hard (binary) masks on layers
- Treat the masks as model parameters
- Masks condition on gradients directly

Network Capacity Problem in HAT

HAT's hard gradient clipping mechanism allows no update for parameters masked by previous tasks:

$$g'_{l,ij} = a_{l,ij} \cdot g_{l,ij},\ a_{l,ij} \in \{0, 1\}$$

More tasks come in

$\downarrow$

More active parameters become static

$\downarrow$

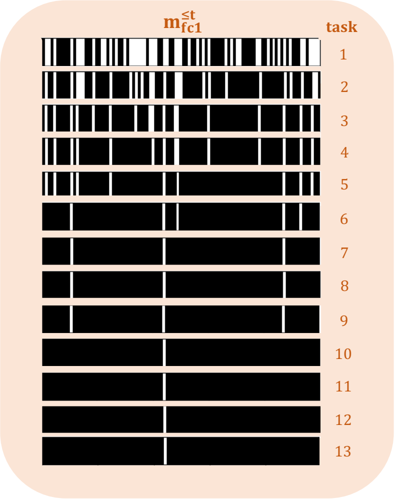
Insufficient network capacity

$\downarrow$

Significantly affect performance on new tasks

$\downarrow$

Learning plasticity reduced, inbalanced stability-plasticity trade-off



AdaHAT: Adaptive Hard Attention to the Task

AdaHAT **soft-clips gradients**, allowing minor updates for parameters masked by previous tasks:

$$g'_{l,ij} = a^{\star}_{l,ij} \cdot g_{l,ij}, \; a^{\star}_{l,ij} \in [0, 1]$$

The adjustment rate $a^{\star}_{l,ij}$ now is an adaptive controller, guided by two pieces of information about previous tasks:

- **Parameter Importance**: how many tasks the target parameter has been used for
- **Network Sparsity**: how many parameters are currently used for all previous tasks

AdaHAT: Adaptive Hard Attention to the Task

Adaptive Adjustment Rate (AdaHAT)

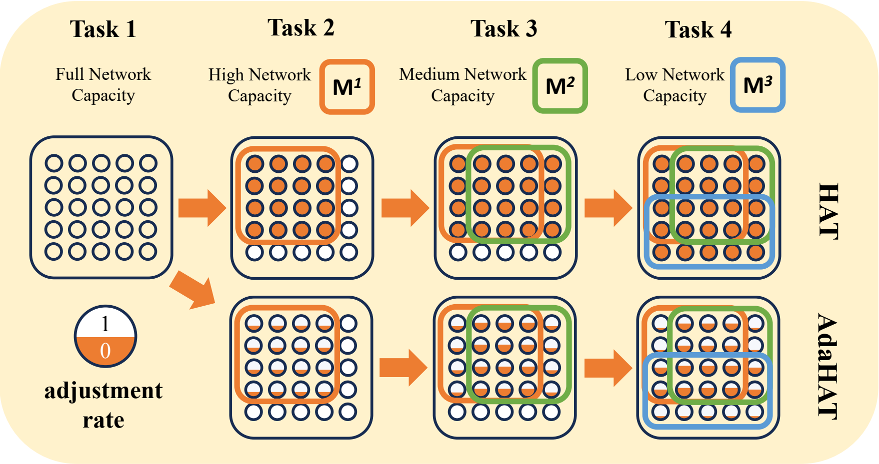
$$a^{\star}_{l,ij} = \frac{r_l}{\min\left(m_{l,i}^{<t,\text{sum}}, m_{l-1,j}^{<t,\text{sum}}\right) + r_l}, \quad r_l = \frac{\alpha}{R\left(\mathsf{M}^t, \mathsf{M}^{<t}\right) + \epsilon}$$

Adjustment Rate (HAT)

$$a_{l,ij} = 1 - \min\left(m_{l,i}^{<t}, m_{l-1,j}^{<t}\right)$$

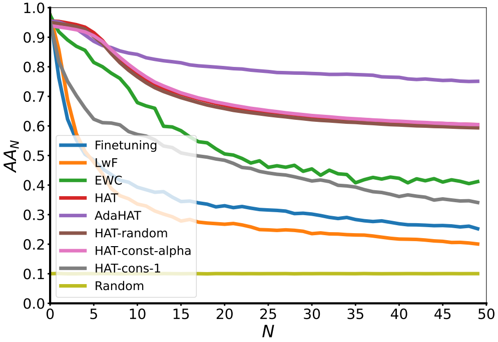
For more about this work, check out my website: pengxiang-wang.com/papers/adahat

AdaHAT: Adaptive Hard Attention to the Task

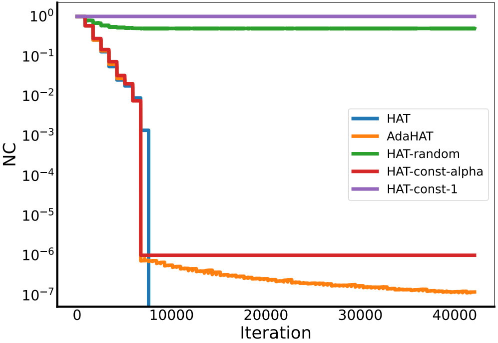


HAT $\Rightarrow$ **AdaHAT (Adaptive HAT)**

Experiment Highlights



Results on long task sequences



Network capacity usage

FG-AdaHAT: Fine-Grained Neuron Importance Guided Architecture-Based Continual Learning

Limitations of AdaHAT

The adaptive gradient clipping mechanism in AdaHAT is coarse:

- Parameter Importance is an integer value from 0 to $t-1$
- Network Sparsity is a single scalar value for the whole network

Fine-grained task information plays a crucial role among other continual learning approaches, meaningfully contributing to guiding the training of new tasks.

AdaHAT $\Rightarrow$ **FG-AdaHAT (Fine-Grained AdaHAT)**

$$a_{l,ij}^{\text{FG-AdaHAT}} = \alpha \left[c^t \cdot \text{Agg}\left(I_{l,i}^{<t}, I_{l-1,j}^{<t} \right) + \alpha \right]^{-1}, c^t = (t + b_L) \cdot \left[R\left(\mathsf{M}^t, \mathsf{M}^{<t} \right) + b_R \right]$$

Fine-Grained Neuron Importance in FG-AdaHAT

Neuron Importance measures how important a neuron is to previous tasks, which is finer-grained than Parameter Importance and Network Sparsity.

$$I_{l,i}^{<t} = \sum_{\tau<t} I_{l,i}^{\tau}, \; I_{l,i}^{\tau} = \frac{1}{|D^{\tau}|} \sum_{(\mathbf{x},y)\in D^{\tau}} I_{l,i}^{\tau}(\mathbf{x}, y)$$

It can be constructed from training information:

- Input Gradients (IG): $I_{l,i}^{\tau}(\mathbf{x}, y) = \sum_j |g'_{l,ij}|$
- Output Gradients (OG): $I_{l,i}^{\tau}(\mathbf{x}, y) = \sum_i |g'_{l+1,ij}|$
- Contribution Utility (CU): $I_{l,i}^{\tau}(\mathbf{x}, y) = |h_{l,j}(\mathbf{x})| \sum_i |w_{l+1,ij}|$
- Input Contribution Utility (ICU): $I_{l,i}^{\tau}(\mathbf{x}, y) = |h_{l,i}(\mathbf{x})| \sum_j |w_{l,ij}|$
- Activation Fisher Information (AFI): $I_{l,i}^{\tau}(\mathbf{x}, y) = |h_{l,i}(\mathbf{x})| \sum_j |g'_{l,ij}|^2$

Fine-Grained Neuron Importance in FG-AdaHAT

Layer attribution methods from **Explainable AI (XAI)** provides more fine-grained neuron importance measures:

- Feature Ablation (FA): perturbation-based attribution
- Internal Influence (II): gradient-based attribution
- Gradient SHAP (GS): gradient-based attribution, using Shapley values
- DeepLIFT (DL): backpropagation-based attribution

For more about this work, check out my website: pengxiang-wang.com/papers/fg-adahat

Experiment Highlights

Approach	Permuted MNIST AA (↑)	Permuted MNIST FR (↑)	Split CIFAR-100 AA (↑)	Split CIFAR-100 FR (↑)	Combined-20 AA (↑)	Combined-20 FR (↑)
Finetuning	32.06 ± 0.96	-73.19 ± 1.11	34.19 ± 1.84	-72.47 ± 3.34	10.88 ± 0.44	-74.45 ± 1.72
LwF	32.95 ± 1.35	-72.12 ± 1.55	31.78 ± 2.36	-77.20 ± 5.34	10.92 ± 0.91	-74.36 ± 1.04
HAT	67.10 ± 0.56	-30.68 ± 0.65	40.12 ± 2.17	-59.12 ± 4.01	17.05 ± 2.57	-74.63 ± 4.16
AdaHAT	79.91 ± 1.47	-12.43 ± 6.46	44.93 ± 1.29	-50.28 ± 2.51	16.46 ± 2.49	-68.31 ± 4.74
FG-AdaHAT (IG)	84.05 ± 1.71	-10.12 ± 1.86	45.41 ± 3.30	-49.36 ± 5.87	$\mathbf{24.67 \pm 4.15}$	-56.65 ± 4.74
FG-AdaHAT (OG)	$\mathbf{85.05 \pm 0.71}$	$\mathbf{-8.92 \pm 0.82}$	45.54 ± 2.19	-49.55 ± 3.69	22.05 ± 2.77	-59.11 ± 4.17
FG-AdaHAT (CU)	84.11 ± 0.91	-10.05 ± 1.05	45.71 ± 3.47	-49.12 ± 6.42	22.26 ± 1.69	-59.98 ± 3.76
FG-AdaHAT (ICU)	84.97 ± 0.63	-9.01 ± 0.72	$\mathbf{49.66 \pm 1.88}$	$\mathbf{-40.93 \pm 3.75}$	19.95 ± 3.93	-62.37 ± 4.80
FG-AdaHAT (AFI)	83.43 ± 1.28	-10.88 ± 1.47	47.00 ± 1.79	-46.66 ± 3.31	20.09 ± 5.16	-61.29 ± 5.90
FG-AdaHAT (FA)	84.86 ± 0.89	-9.14 ± 1.03	—	—	—	—
FG-AdaHAT (II)	83.66 ± 0.74	-10.60 ± 0.85	45.70 ± 1.66	-48.85 ± 2.85	22.32 ± 1.01	-59.10 ± 4.24
FG-AdaHAT (GS)	83.91 ± 0.70	-10.30 ± 0.81	46.43 ± 3.83	-47.40 ± 7.32	24.25 ± 6.52	$\mathbf{-56.05 \pm 9.23}$
FG-AdaHAT (DL)	84.70 ± 0.57	-9.34 ± 0.66	47.70 ± 4.64	-45.02 ± 7.92	23.71 ± 4.49	-57.79 ± 5.34

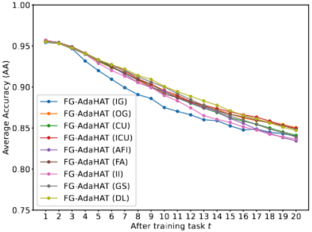
Performance on 3 continual learning benchmarks

Experiment Highlights

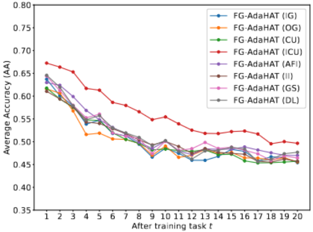
Approach	Permuted MNIST BWT (↑)	Permuted MNIST FWT (↑)	Split CIFAR-100 BWT (↑)	Split CIFAR-100 FWT (↑)	Combined-20 BWT (↑)	Combined-20 FWT (↑)
Finetuning	-68.52 ± 0.96	0.42 ± 0.02	-51.17 ± 1.87	10.70 ± 0.40	-64.88 ± 0.57	9.87 ± 0.22
LwF	-67.42 ± 1.34	0.25 ± 0.05	-51.78 ± 3.40	8.88 ± 1.06	-64.75 ± 0.81	9.70 ± 0.21
HAT	-0.00 ± 0.00	-31.12 ± 0.57	-21.40 ± 2.40	-12.43 ± 0.72	-19.18 ± 3.98	-27.31 ± 4.40
AdaHAT	-12.46 ± 1.42	-5.19 ± 0.09	-27.23 ± 1.06	-1.60 ± 0.97	-40.91 ± 2.29	-6.16 ± 1.29
FG-AdaHAT (IG)	-7.73 ± 1.67	-5.55 ± 0.17	-19.90 ± 3.13	-8.13 ± 0.82	-33.90 ± 2.45	-5.11 ± 2.62
FG-AdaHAT (OG)	-5.33 ± 0.70	-6.90 ± 0.19	-19.66 ± 2.08	-8.12 ± 1.01	-34.55 ± 2.13	-6.56 ± 1.80
FG-AdaHAT (CU)	-6.09 ± 0.90	-7.14 ± 0.11	-22.61 ± 2.62	-5.00 ± 0.77	-35.31 ± 1.57	-5.79 ± 0.71
FG-AdaHAT (ICU)	-5.57 ± 0.67	-6.75 ± 0.11	-23.60 ± 1.70	-0.41 ± 0.69	-38.10 ± 2.72	-5.37 ± 1.55
FG-AdaHAT (AFI)	-7.32 ± 1.24	-6.62 ± 0.14	-21.92 ± 2.23	-4.40 ± 1.18	-37.06 ± 2.86	-6.58 ± 2.71
FG-AdaHAT (FA)	-6.71 ± 0.87	-5.73 ± 0.09	–	–	–	–
FG-AdaHAT (II)	-6.02 ± 0.70	-7.69 ± 0.12	-20.57 ± 1.54	-7.01 ± 1.58	-34.48 ± 4.14	-6.59 ± 3.23
FG-AdaHAT (GS)	-4.75 ± 0.69	-8.68 ± 0.25	-20.92 ± 2.61	-6.06 ± 1.83	-34.58 ± 4.19	-4.83 ± 1.80
FG-AdaHAT (DL)	-3.89 ± 0.57	-8.70 ± 0.25	-20.03 ± 3.92	-5.63 ± 1.11	-34.65 ± 2.26	-5.29 ± 1.70

Stability-plasticity trade-off on 3 continual learning benchmarks

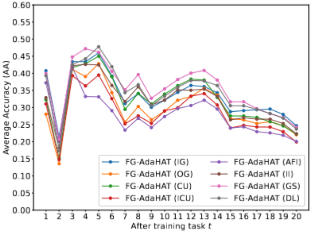
Experiment Highlights



(a) Permuted MNIST



(b) Split CIFAR-100



(c) Combined-20

Comparison of 9 fine-grained neuron importance on 3 benchmarks

AmnesiacHAT: Continual Unlearning via Capacity Recycling in Architecture-Based Continual Learning

Background: Continual Unlearning

Machine unlearning, which becomes popular after 2020, deliberately removes the influence of specific data from a trained AI model.

Applications: data privacy, security, data quality, fairness, etc.

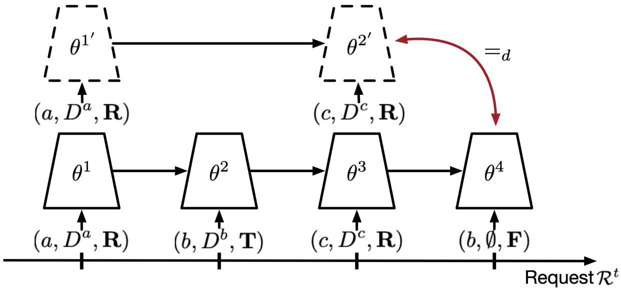
Unlearning is difficult, because:

- AI models like neural networks are usually holistic, where unlearning causes side effects
- Retraining from scratch without the unlearning data is:
  1. Computationally expensive
  2. Not always feasible when the original data is unavailable

Background: Continual Unlearning

Continual unlearning: Unlearning specific tasks from a continual learning model

(A field that has been hardly explored before)

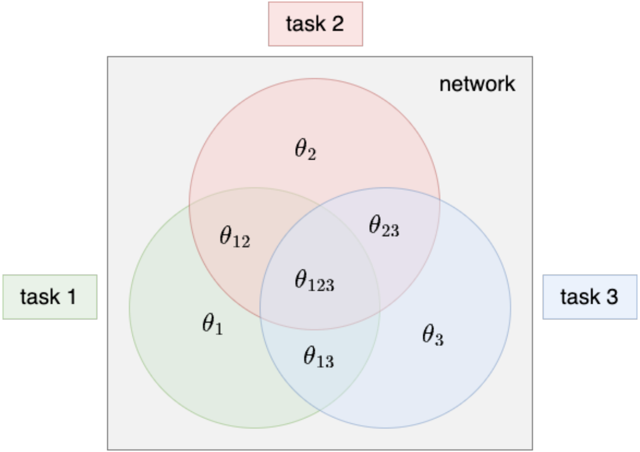


Objectives:

1. **Learning** objective: Maximal average performance on the remaining tasks
2. **Unlearning** objective: Minimal difference from retraining without the unlearning tasks

For more about unlearning, check out my slides:
pengxiang-wang.com/slides/slides-continual-learning-and-machine-unlearning.pdf

Motivation



Architecture-based approaches are naturally suitable for continual unlearning:

- Less overlapping, less task interference
- We choose our state-of-the-art fix-sized architecture-based approach AdaHAT

AdaHAT $\Rightarrow$ **AmnesiacHAT**

Unlearning Algorithm in AmnesiacHAT

The AdaHAT architecture needs to be modified to enable unlearning:

1. Store task-wise update record (inspired by AmnesiacML):

$$\theta_{l,ij}^{(t)} = \theta_{l,ij}^{(0)} + \Delta\theta_{l,ij}^{(1)} + \cdots + \Delta\theta_{l,ij}^{(t)}$$

The update in HAT is sparse which can be stored efficiently by sparse storage format and pruning.

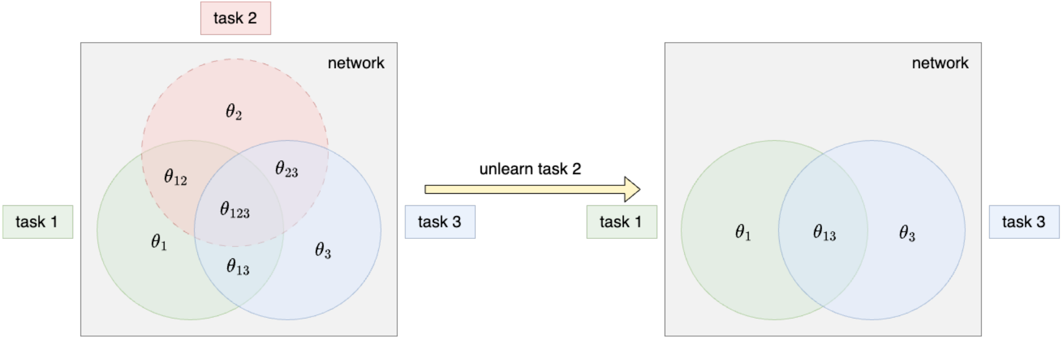
2. Unlearn task τ = Remove its parameter update $\Delta\theta_{l,ij}^{(\tau)}$

3. Deal with the side effects on overlapping parameters

If the parameter is first used for task τ, compensate the adjustment rate back in its next task's update. Small-scale retraining for fixing is applied after.

Unintended Benefit: Capacity Recycling

When a task is unlearned, the parameters that were dedicated to that task become free and can be reused for future tasks.

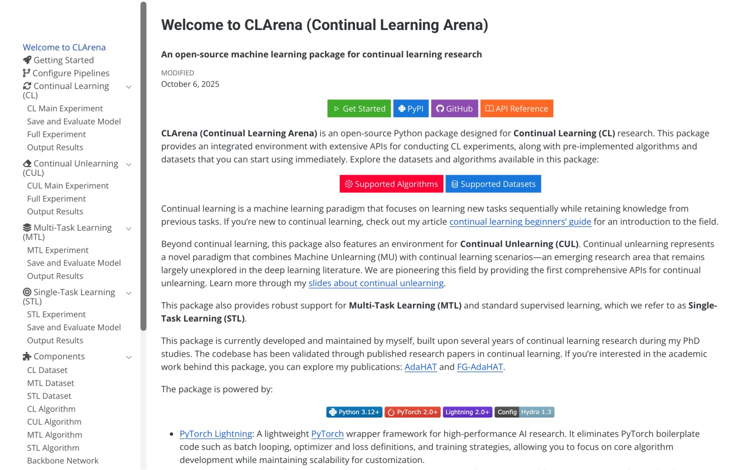


How to measure it? *Compare the performance without unlearning*

Continual Learning Arena (CLArena)

Continual Learning Arena (CLArena)

Continual Learning Arena (CLArena) is an open-source Python package for continual learning research, developed and documented entirely by myself.


Welcome to CLArena
Getting Started
Configure Pipelines
Continual Learning (CL)
CL Main Experiment
Save and Evaluate Model
Full Experiment
Output Results
Continual Unlearning (CUL)
CUL Main Experiment
Full Experiment
Output Results
Multi-Task Learning (MTL)
MTL Experiment
Save and Evaluate Model
Output Results
Single-Task Learning (STL)
STL Experiment
Save and Evaluate Model
Output Results
Components
CL Dataset
MTL Dataset
STL Dataset
CL Algorithm
CUL Algorithm
MTL Algorithm
STL Algorithm
Backbone Network

Welcome to CLArena (Continual Learning Arena)

An open-source machine learning package for continual learning research

MODIFIED
October 6, 2025

Get Started | PyPI | GitHub | API Reference

CLArena (Continual Learning Arena) is an open-source Python package designed for **Continual Learning (CL)** research. This package provides an integrated environment with extensive APIs for conducting CL experiments, along with pre-implemented algorithms and datasets that you can start using immediately. Explore the datasets and algorithms available in this package:

Supported Algorithms | Supported Datasets

Continual learning is a machine learning paradigm that focuses on learning new tasks sequentially while retaining knowledge from previous tasks. If you're new to continual learning, check out my article continual learning beginners' guide for an introduction to the field.

Beyond continual learning, this package also features an environment for **Continual Unlearning (CUL)**. Continual unlearning represents a novel paradigm that combines Machine Unlearning (MU) with continual learning scenarios—an emerging research area that remains largely unexplored in the deep learning literature. We are pioneering this field by providing the first comprehensive APIs for continual unlearning. Learn more through my slides about continual unlearning.

This package also provides robust support for **Multi-Task Learning (MTL)** and standard supervised learning, which we refer to as **Single-Task Learning (STL)**.

This package is currently developed and maintained by myself, built upon several years of continual learning research during my PhD studies. The codebase has been validated through published research papers in continual learning. If you're interested in the academic work behind this package, you can explore my publications: AdaHAT and FG-AdaHAT.

The package is powered by:

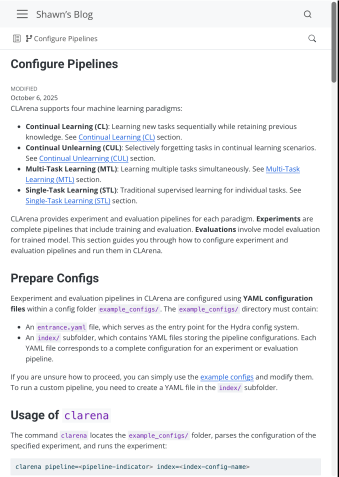
Python 3.12+ | PyTorch 2.0+ | Lightning 2.0+ | Config Hydra 1.3

- PyTorch Lightning: A lightweight PyTorch wrapper framework for high-performance AI research. It eliminates PyTorch boilerplate code such as batch looping, optimizer and loss definitions, and training strategies, allowing you to focus on core algorithm development while maintaining scalability for customization.

Key Features of CLArena

1. Provide pipelines for various machine learning paradigms:
    - Continual Learning (CL)
    - Continual Unlearning (CUL)
    - Multi-Task Learning (MTL)
    - Single-Task Learning (STL)
2. Provide implementations of:
    - Mainstream CL algorithms: LwF, EWC, HAT, WSN, CBP, etc.
    - Common CL / MTL / STL benchmarks: permuted, split, combined, etc.
    - Neural network architectures: MLP, ResNet, HAT-based models, etc.
3. Provide framework for building custom datasets, models, algorithms, metrics, callbacks, etc.
4. Full code support for my 3 papers above, ensuring reproducibility

Ease-of-Use Features



Shawn's Blog

Configure Pipelines

Configure Pipelines

MODIFIED
October 6, 2025

CLArena supports four machine learning paradigms:

- **Continual Learning (CL)**: Learning new tasks sequentially while retaining previous knowledge. See Continual Learning (CL) section.
- **Continual Unlearning (CUL)**: Selectively forgetting tasks in continual learning scenarios. See Continual Unlearning (CUL) section.
- **Multi-Task Learning (MTL)**: Learning multiple tasks simultaneously. See Multi-Task Learning (MTL) section.
- **Single-Task Learning (STL)**: Traditional supervised learning for individual tasks. See Single-Task Learning (STL) section.

CLArena provides experiment and evaluation pipelines for each paradigm. **Experiments** are complete pipelines that include training and evaluation. **Evaluations** involve model evaluation for trained model. This section guides you through how to configure experiment and evaluation pipelines and run them in CLArena.

Prepare Configs

Experiment and evaluation pipelines in CLArena are configured using **YAML configuration files** within a config folder `example_configs/`. The `example_configs/` directory must contain:

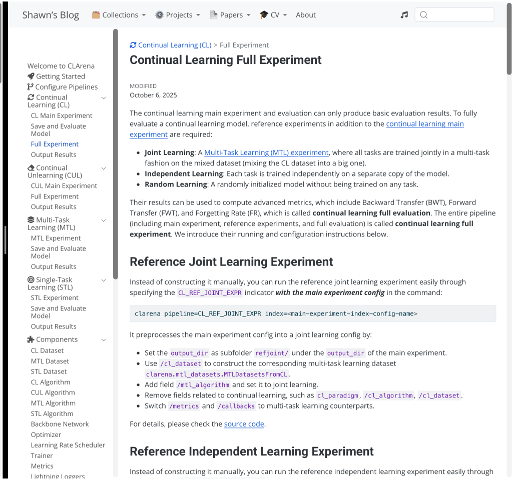
- An `entrance.yaml` file, which serves as the entry point for the Hydra config system.
- An `index/` subfolder, which contains YAML files storing the pipeline configurations. Each YAML file corresponds to a complete configuration for an experiment or evaluation pipeline.

If you are unsure how to proceed, you can simply use the example configs and modify them. To run a custom pipeline, you need to create a YAML file in the `index/` subfolder.

Usage of `clarena`

The command `clarena` locates the `example_configs/` folder, parses the configuration of the specified experiment, and runs the experiment:

```
clarena pipeline=<pipeline-indicator> index=<index-config-name>
```



Shawn's Blog   Collections   Projects   Papers   CV   About

Welcome to CLArena
Getting Started
Configure Pipelines
Continual Learning (CL)
CL Main Experiment
Save and Evaluate Model
Full Experiment
Output Results
Continual Unlearning (CUL)
CUL Main Experiment
Full Experiment
Output Results
Multi-Task Learning (MTL)
MTL Experiment
Save and Evaluate Model
Output Results
Single-Task Learning (STL)
STL Experiment
Save and Evaluate Model
Output Results
Components
CL Dataset
MTL Dataset
STL Dataset
CL Algorithm
CUL Algorithm
MTL Algorithm
STL Algorithm
Backbone Network
Optimizer
Learning Rate Scheduler
Trainer
Metrics
Lightning Loggers

Continual Learning (CL) > Full Experiment

Continual Learning Full Experiment

MODIFIED
October 6, 2025

The continual learning main experiment and evaluation can only produce basic evaluation results. To fully evaluate a continual learning model, reference experiments in addition to the continual learning main experiment are required:

- **Joint Learning**: A Multi-Task Learning (MTL) experiment, where all tasks are trained jointly in a multi-task fashion on the mixed dataset (mixing the CL dataset into a big one).
- **Independent Learning**: Each task is trained independently on a separate copy of the model.
- **Random Learning**: A randomly initialized model without being trained on any task.

Their results can be used to compute advanced metrics, which include Backward Transfer (BWT), Forward Transfer (FWT), and Forgetting Rate (FR), which is called **continual learning full evaluation**. The entire pipeline (including main experiment, reference experiments, and full evaluation) is called **continual learning full experiment**. We introduce their running and configuration instructions below.

Reference Joint Learning Experiment

Instead of constructing it manually, you can run the reference joint learning experiment easily through specifying the `CL_REF_JOINT_EXPR` indicator ***with the main experiment config*** in the command:

```
clarena pipeline=CL_REF_JOINT_EXPR index=<main-experiment-index-config-name>
```

It preprocesses the main experiment config into a joint learning config by:

- Set the `output_dir` as subfolder `refjoint/` under the `output_dir` of the main experiment.
- Use `/cl_dataset` to construct the corresponding multi-task learning dataset `clarena.mtl_datasets.MTLDatasetsFromCL`.
- Add field `/mtl_algorithm` and set it to joint learning.
- Remove fields related to continual learning, such as `cl_paradigm`, `/cl_algorithm`, `/cl_dataset`.
- Switch `/metrics` and `/callbacks` to multi-task learning counterparts.

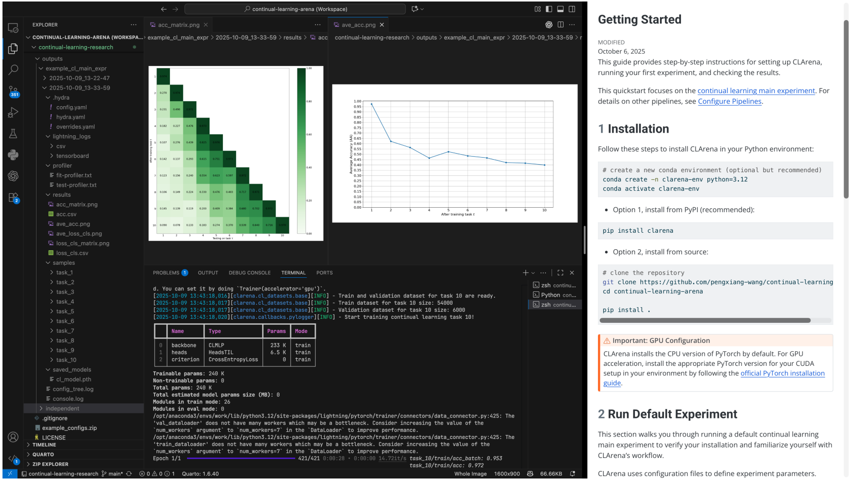
For details, please check the source code.

Reference Independent Learning Experiment

Instead of constructing it manually, you can run the reference independent learning experiment easily through

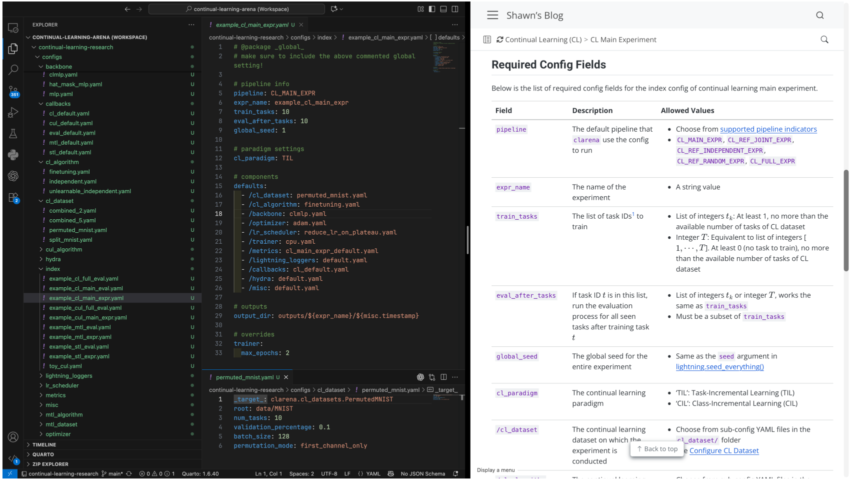
Well-structured and detailed documentation

Ease-of-Use Features



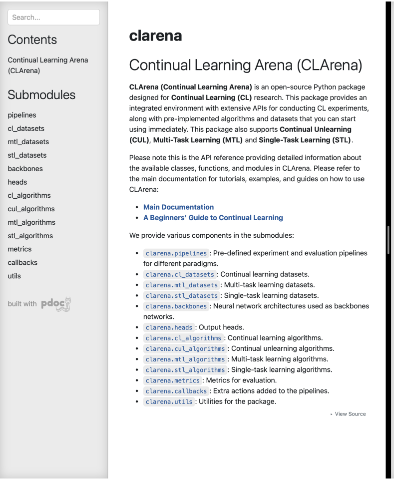
Get started quickly with tutorials and examples

Ease-of-Use Features



An easy configuration system based on YAML, each config field clearly explained

Ease-of-Use Features


Search...

Contents

Continual Learning Arena (CLArena)

Submodules

pipelines
cl_datasets
mtl_datasets
stl_datasets
backbones
heads
cl_algorithms
cul_algorithms
mtl_algorithms
stl_algorithms
metrics
callbacks
utils

built with pdoc

clarena

Continual Learning Arena (CLArena)

CLArena (Continual Learning Arena) is an open-source Python package designed for **Continual Learning (CL)** research. This package provides an integrated environment with extensive APIs for conducting CL experiments, along with pre-implemented algorithms and datasets that you can start using immediately. This package also supports **Continual Unlearning (CUL), Multi-Task Learning (MTL)** and **Single-Task Learning (STL)**.

Please note this is the API reference providing detailed information about the available classes, functions, and modules in CLArena. Please refer to the main documentation for tutorials, examples, and guides on how to use CLArena:

- **Main Documentation**
- **A Beginners' Guide to Continual Learning**

We provide various components in the submodules:

- `clarena.pipelines` : Pre-defined experiment and evaluation pipelines for different paradigms.
- `clarena.cl_datasets` : Continual learning datasets.
- `clarena.mtl_datasets` : Multi-task learning datasets.
- `clarena.stl_datasets` : Single-task learning datasets.
- `clarena.backbones` : Neural network architectures used as backbones networks.
- `clarena.heads` : Output heads.
- `clarena.cl_algorithms` : Continual learning algorithms.
- `clarena.cul_algorithms` : Continual unlearning algorithms.
- `clarena.mtl_algorithms` : Multi-task learning algorithms.
- `clarena.stl_algorithms` : Single-task learning algorithms.
- `clarena.metrics` : Metrics for evaluation.
- `clarena.callbacks` : Extra actions added to the pipelines.
- `clarena.utils` : Utilities for the package.

View Source


clarena.cl_algorithms

Search...

API Documentation

class HAT
HAT()
adjustment_mode
s_max
clamp_threshold
mask_sparsity_reg_factor
mask_sparsity_reg_mode
mark_sparsity_reg
task_embedding_init_mode
alpha
cumulative_mask_for_previous_tasks
automatic_optimization
sanity_check()
on_train_start()
clip_grad_by_adjustment()
compensate_task_embedding_gradients()
forward()
training_step()
on_train_end()
validation_step()
test_step()

built with pdoc

clarena.cl_algorithms.hat

The submodule in `cl_algorithms` for HAT (Hard Attention to the Task) algorithm.

View Source

class HAT(clarena.cl_algorithms.base.CLAlgorithm): View Source

HAT (Hard Attention to the Task) algorithm.

An architecture-based continual learning approach that uses learnable hard attention masks to select task-specific parameters.

```
HAT(
    backbone: clarena.backbones.HATMaskBackbone,
    heads: clarena.heads.HeadsTIL,
    adjustment_mode: str,
    s_max: float,
    clamp_threshold: float,
    mask_sparsity_reg_factor: float,
    mask_sparsity_reg_mode: str = 'original',
    task_embedding_init_mode: str = 'N01',
    alpha: float | None = None,
    non_algorithmic_hparams: dict[str, typing.Any] = {}
)
```

View Source

Initialize the HAT algorithm with the network.

Args:

- **backbone** (`HATMaskBackbone`): must be a backbone network with the HAT mask mechanism.
- **heads** (`HeadsTIL`): output heads. HAT only supports TIL (Task-Incremental Learning).
- **adjustment_mode** (`str`): the strategy of adjustment (i.e., the mode of gradient clipping), must be one of:
    1. 'hat': set gradients of parameters linking to masked units to zero. This is how HAT fixes the part of the network for previous tasks completely. See Eq. (2) in Sec. 2.3 "Network Training" in the HAT paper.
    2. 'hat_random': set gradients of parameters linking to masked units to random 0–1 values. See "Baselines" in Sec. 4.1 in the AdaHAT paper.
    3. 'hat_const_alpha': set gradients of parameters linking to masked units to a constant value `alpha`. See "Baselines" in Sec. 4.1 in the AdaHAT paper.
    4. 'hat_const_1': set gradients of parameters linking to masked units to a constant value of 1 (i.e., no gradient constraint). See "Baselines" in Sec.

Complete API reference for custom implementation in the framework

Package Information

Main Page / Documentation

pengxiang-wang.com/projects/continual-learning-arena

GitHub (feel free to give it a star! :D)

github.com/pengxiang-wang/continual-learning-arena

PyPI (‘pip install clarena‘)

pypi.org/project/clarena

API Reference

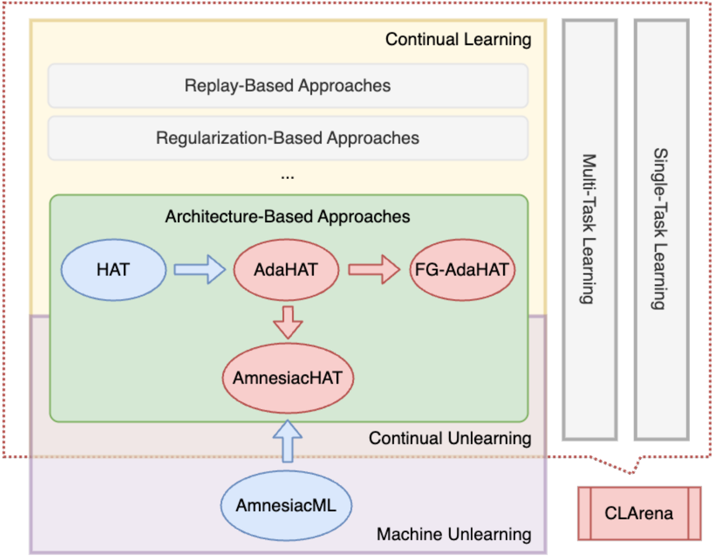
pengxiang-wang.com/projects/continual-learning-arena/docs/api-reference

Thank You

Overall, my PhD work dives deep into continual learning through architecture-based approaches, across comprehensive contributions to this area in both width and depth:

- New well-performed continual learning approaches: **AdaHAT**, **FG-AdaHAT** variants
- New algorithm framework: **FG-AdaHAT**
- New paradigm problem hardly explored before: **continual unlearning**, and its solution: **AmnesiacHAT**
- New software platform for continual learning research: **CLArena**

Thank You



Continual Learning
Replay-Based Approaches
Regularization-Based Approaches
...
Architecture-Based Approaches
HAT
AdaHAT
FG-AdaHAT
AmnesiacHAT
Continual Unlearning
AmnesiacML
Machine Unlearning
Multi-Task Learning
Single-Task Learning
CLArena

Thank You

Thank you for your attention!

My website: pengxiang-wang.com

My email: wangpengxiang@stu.pku.edu.cn

北京大學
PEKING UNIVERSITY