Avalanche

An End-to-End Library for Continual Learning

avalanche.continualai.org

powered by

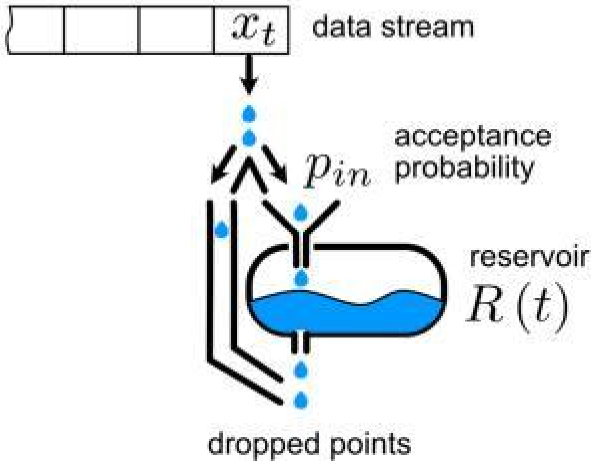
ContinualAI

Antonio Carta, Andrea Cossu, Lorenzo Pellegrini, Gabriele Graffieti, Hamed Hemati, Vincenzo Lomonaco
and many more contributors…

CL - Strategies

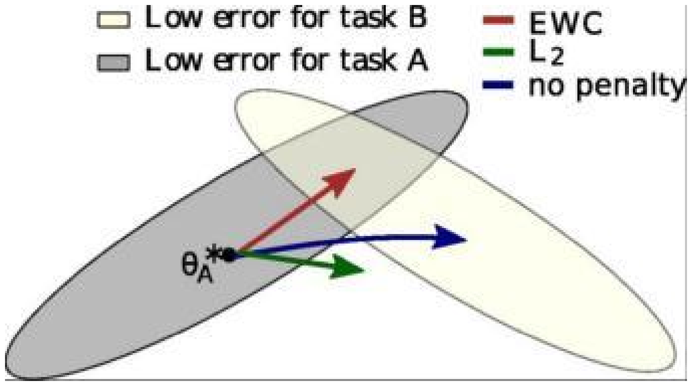
Replay

- Keep a buffer of old samples
- Rehearse old samples



Regularization

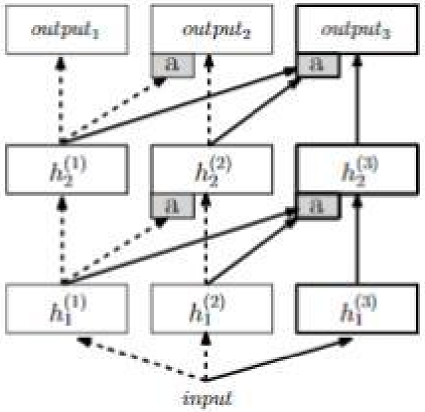
- Regularize the model to balance learning and forgetting



Elastic Weight Consolidation

Architectural

- Expand the model over time with new units/ layers



Progressive Neural Networks

Image from
https://towardsdatascience.com/reservoir-sampling-for-efficient-stream-processing-97f47f85c11b

Avalanche – Design Principles

1. Comprehensive and Consistent
2. Easy to Use high-level APIs
3. Reproducibility and Portability
4. Easy to Extend - modularity and independence of low-level APIs 5.

Community-driven – 40+ contributors from different institutions

Install with **pip install avalanche-lib**

A Minimal Example

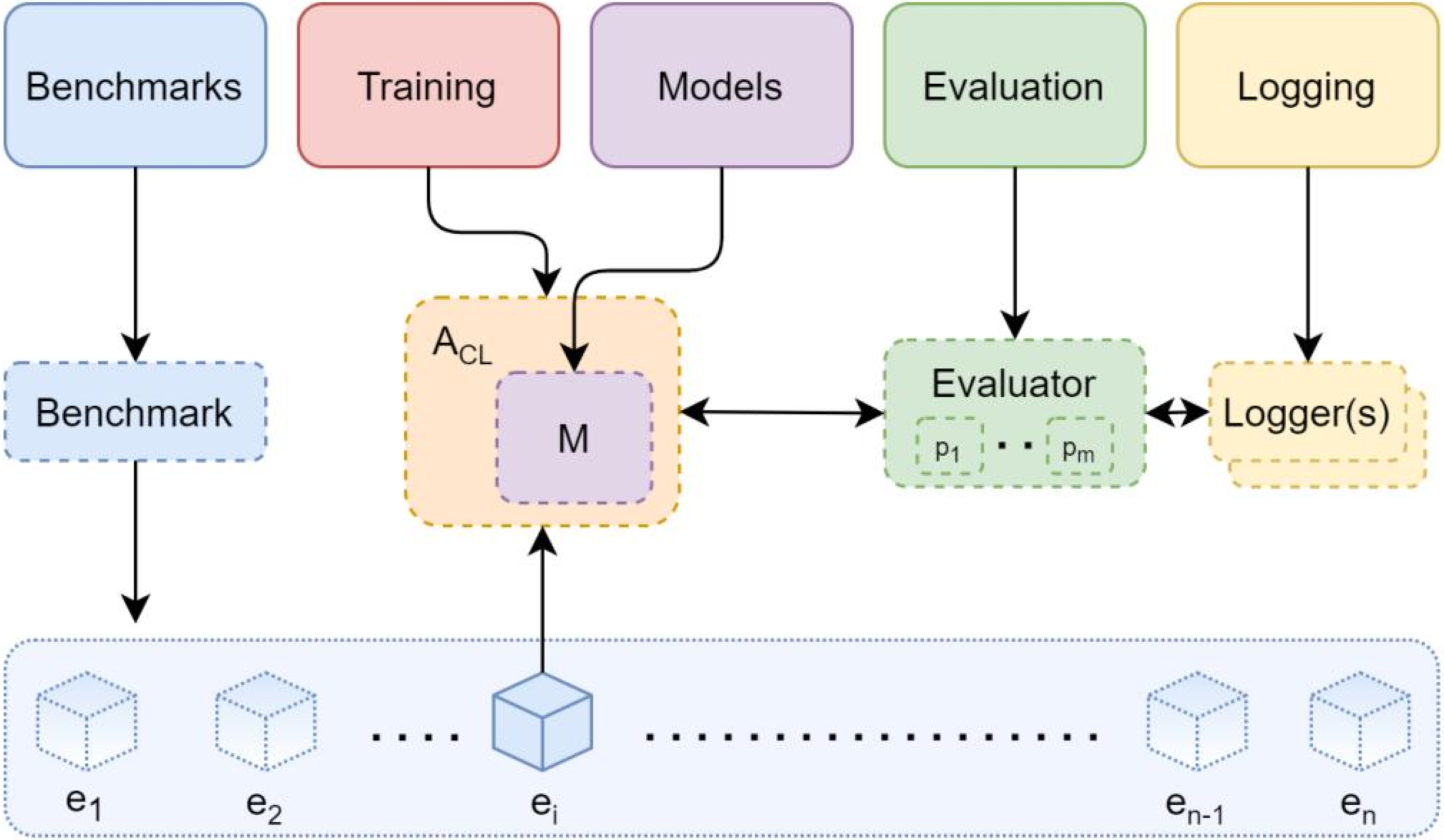
```python
# CL Benchmark Creation
benchmark = PermutedMNIST(n_experiences=3)
train_stream = benchmark.train_stream
test_stream = benchmark.test_stream

# Prepare model, optimizer, criterion (standard pytorch)
model = SimpleMLP(num_classes=10)
optimizer = SGD(model.parameters(), lr=0.001, momentum=0.9)
criterion = CrossEntropyLoss()

# Continual learning strategy
cl_strategy = Naive(
    model, optimizer, criterion,
    train_mb_size=32, train_epochs=2,
    eval_mb_size=32, device=device)

# train and test loop over the stream of experiences
results = []
for train_exp in train_stream:
    cl_strategy.train(train_exp)
    results.append(cl_strategy.eval(test_stream))
```

Avalanche modules



Classic Benchmarks

Most common benchmarks from the literature are available

- MNIST: SplitMNIST, RotatedMNIST, PermutedMNIST, SplitFashionMNIST
- CIFAR10: SplitCIFAR10, SplitCIFAR100, SplitCIFAR110.
- CORe50: all the CORe50 scenarios are supported.
- Others: SplitCUB200, CLStream51, CLEAR.

```
benchmark = SplitMNIST(
      n_experiences=5,
      seed=1,
      return_task_id=False,
      fixed_class_order=[5,0,9, ...],
      train_transform=ToTensor(),
      eval_transform=ToTensor()
)
```

Avalanche datasets add:

- train/eval transforms
- Management of class and task labels

Benchmark – Data Iteration

```python
train_stream = benchmark_instance.train_stream
test_stream = benchmark_instance.test_stream

for idx, experience in enumerate(train_stream):
    dataset = experience.dataset

    print('Train dataset contains',
        len(dataset), 'patterns')

    for x, y, t in dataset:
        ...

    test_experience = test_stream[idx]
    cumulative_test = test_stream[:idx+1]
```

Avalanche – Strategies

- ➢ Methods from the literature.
- ➢ Different methods can be combined together using plugins.
- ➢ You can also implement custom plugins to define your own strategies.

```
strategy = Replay(model, optimizer,
                  criterion, mem_size)
for train_exp in scenario.train_stream:
    strategy.train(train_exp)
    strategy.eval(scenario.test_stream)
```

- **Baselines**: Naive, JointTraining, Cumulative.
- **Rehearsal**: Replay with reservoir sampling and balanced buffers, GSS greedy, CoPE, Generative Replay.
- **Regularization**: EWC, LwF, GEM, AGEM, CWR*, Synaptic Intelligence, MAS.
- **Architectural**: Progressive Neural Networks, multi-head, incremental classifier.
- **Others**: GDumb, iCaRL, AR1, Streaming LDA, LFL.

```
replay = ReplayPlugin(mem_size)
ewc = EWCPlugin(ewc_lambda)
strategy = BaseStrategy(
    model, optimizer,
    criterion, mem_size,
    plugins=[replay, ewc])
```

Example of Custom Plugin

```
from avalanche.training.plugins import StrategyPlugin

class ReplayPlugin(StrategyPlugin):
    """ Experience replay plugin. """

    def __init__(self, mem_size=200):
        super().__init__()
        self.mem_size = mem_size
        self.ext_mem = {}  # a Dict<task_id, Dataset>
        self.rm_add = None

    def adapt_train_dataset(self, strategy, **kwargs):
        """
        Expands the current training set with datapoints from
        the external memory before training.
        """
        ...

    def after_training_exp(self, strategy, **kwargs):
        """
        After training we update the external memory with the patterns of
         the current training batch/task.
        """
        ...
```

Models – Multi-Task

- Avalanche supports multitask models
- One task labels for each sample
- Standard models, like Multi-head classifiers, are already implemented
- You can also implement custom modules. You implement the single-task forward, and Avalanche splits by task automatically

```python
class MTSimpleMLP(MultiTaskModule):
    """Multi-layer perceptron with multi-head classifier"""

    def __init__(self, input_size=28 * 28, hidden_size=512):
        super().__init__()

        self.features = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(inplace=True),
            nn.Dropout(),
        )
        self.classifier = MultiHeadClassifier(hidden_size)
        self._input_size = input_size

    def forward(self, x, task_labels):
        x = x.contiguous()
        x = x.view(x.size(0), self._input_size)
        x = self.features(x)
        x = self.classifier(x, task_labels)
        return x
```

Models – Dynamic (growing) Modules

- Dynamic modules grow overtime by adding units/layers
  - Incremental classifier
  - Progressive neural network
- Adaptation is called automatically by the strategy

```python
class IncrementalClassifier(DynamicModule):
    """
    Output layer that incrementally adds units whenever new classes are
    encountered.
    """

    def __init__(self, in_features, initial_out_features=2):
        """
        :param in_features: number of input features.
        :param initial_out_features: initial number of classes (can be
            dynamically expanded).
        """
        super().__init__()
        self.classifier = torch.nn.Linear(in_features, initial_out_features)

    @torch.no_grad()
    def adaptation(self, dataset: AvalancheDataset):
        """If `dataset` contains unseen classes the classifier is expanded.

        :param dataset: data from the current experience.
        :return:
        """
        in_features = self.classifier.in_features
        old_nclasses = self.classifier.out_features
        new_nclasses = max(
            self.classifier.out_features, max(dataset.targets) + 1
        )

        if old_nclasses == new_nclasses:
            return
        old_w, old_b = self.classifier.weight, self.classifier.bias
        self.classifier = torch.nn.Linear(in_features, new_nclasses)
        self.classifier.weight[:old_nclasses] = old_w
        self.classifier.bias[:old_nclasses] = old_b

    def forward(self, x, **kwargs):
        return self.classifier(x)
```

Metrics and Evaluation

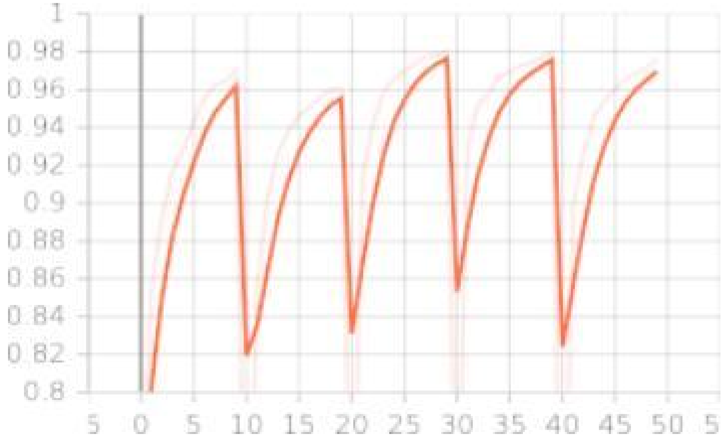
Avalanche provides continuous evaluation of CL strategies with a large set of Metrics. They are collected and logged automatically by the strategy during the training and evaluation loops.

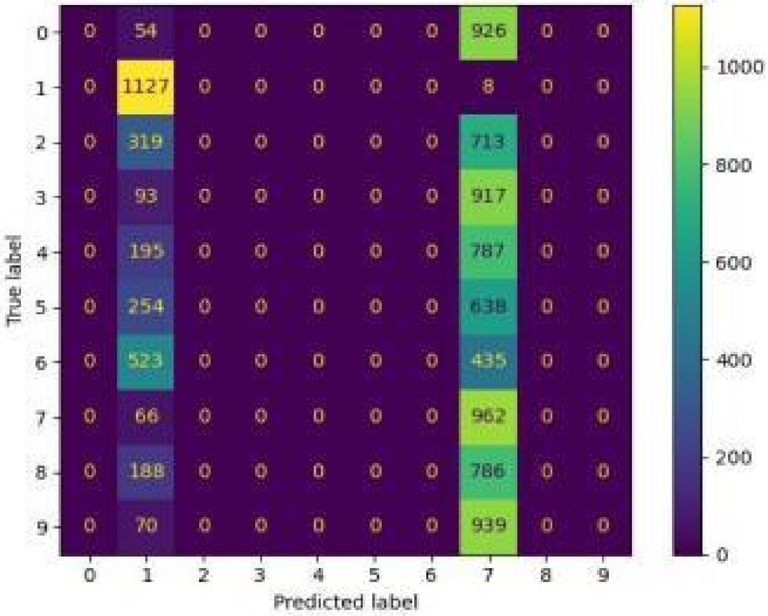
- **Standard Performance Metrics:** accuracy, loss, confusion (averaged over streams or experiences).
- **CL-Metrics:** backward/forward transfer, forgetting.
- **Computational Resources**: CPU and RAM usage, MAC, execution times.

Metrics and Evaluation

Evaluation module provides:

- **Metrics** (accuracy, forgetting, CPU Usage…) - you can create your own!
- **Loggers** to report results in different ways - you can create your own!
- Automatic integration in the training and evaluation loop through the **Evaluation Plugin**
- A **dictionary** with all recorded metrics always available for custom use





Some Useful Links

1. Related Projects:
    1. Reproducibility: https://github.com/ContinualAI/reproducible-continual-learning
    2. Continual Reinforcement Learning: https://github.com/ContinualAI/avalanche-rl
2. Become familiar with all the avalanche features.
    a. Official documentation: **https://avalanche.continualai.org/**
    b. Learn avalanche in 5 minutes (link here).
    c. From zero to hero tutorial (link here)
    d. API doc can be consulted at https://avalanche-api.continualai.org/
3. If you don’t understand something or you want to discuss a new feature or a possible improvement:
    a. Join our slack channel #avalanche (here)
    b. Open a discussion ongithub (here)
4. If you find an issue on Avalanche open an issue ongithub (here)