Continual Learning: Overview

COMS 6988: CLMM

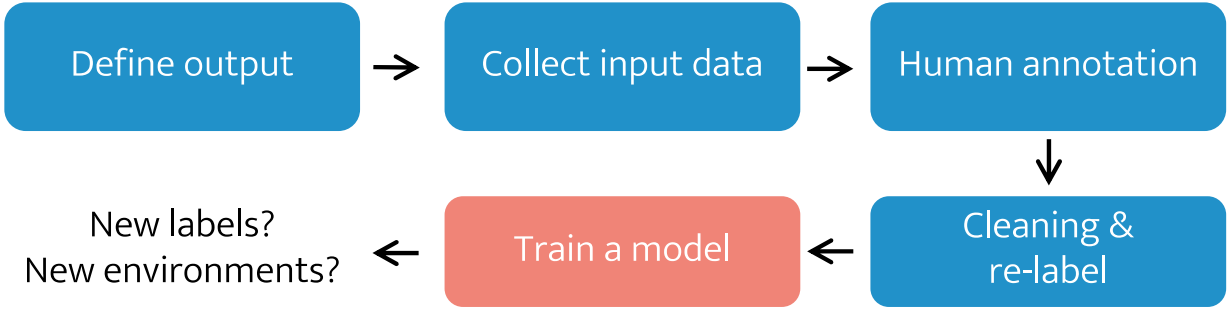
January 2026

Many slides based on:
A Comprehensive Survey of Continual Learning: Theory, Method and Application 2024

Thanks to the TAs: Tom Zollo & Todd Morrill

Classic learning pipeline

- Machine learning works well in a closed-world
  - fixed set of classes
  - lots of labelled data
- But, narrow and brittle once deployed

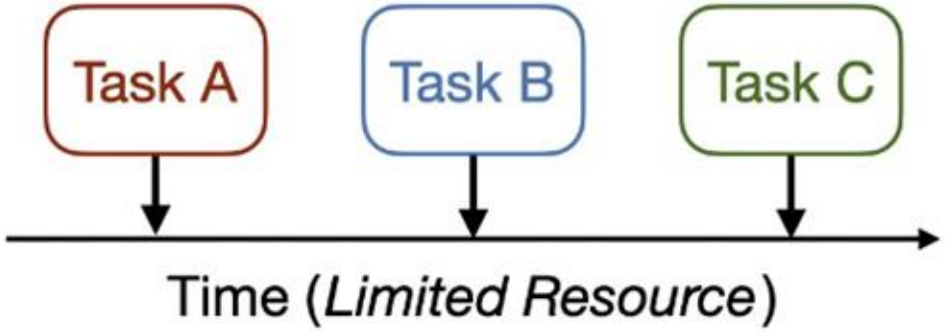


Learning in Open World

- Quickly adapt to novel environments
- Learn in an open world: new tasks
- Encounter new concepts through online experience

Continual Learning

- Evolution has empowered human and other organisms with strong adaptability to continually acquire, update, accumulate and exploit knowledge



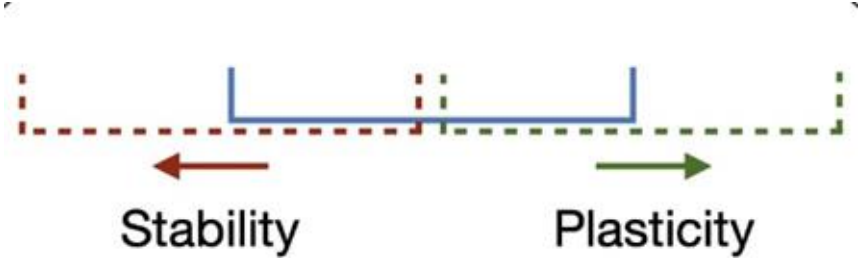
- This motivates the study of continual learning in AI systems, where the typical setting is to *learn a sequence of tasks one by one* and **hope the model behaves as if they were observed simultaneously**

Challenges

- Machine learning models typically capture *static* data distributions
- Continual learning is characterized by learning from *dynamic* data distributions.

Major challenge = **catastrophic forgetting**: when adaptation to a new distribution results in a largely reduced ability to master the old ones.

Arises from trade-off between learning **plasticity** and **memory stability**



Challenges

A desirable solution for continual learning should also obtain strong **generalizability** → accommodate distribution differences within and between tasks

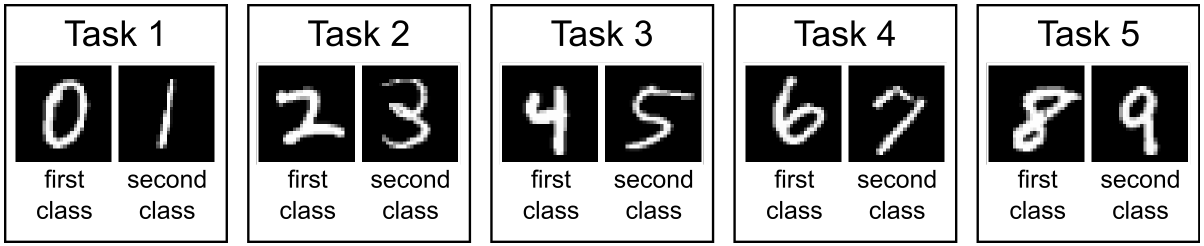
- Shifts from training to test set within a task
- Shifts in task distribution

Continual Learning for Classification

Definition: data stream provides samples, learner aims to continually upgrade knowledge about old classes and learn new ones

Many different approaches, each with own simplifying assumptions:

- Are the task boundaries known?
- Is the task identified with the sample?
- Online vs. offline learning
- Can the system handle unknown/new classes? unlabelled examples?
- Are labels available during training?



[van de Ven & Tolias, 2019]

Setup

Task is defined by its training samples $\mathcal{D}_t$ following the distribution $p(\mathcal{X}_t, \mathcal{Y}_t)$

For task $t \in \mathcal{T} = \{1, ..., k\}$, batch is denoted as $\mathcal{D}_{t,b} = \{\mathcal{X}_{t,b}, \mathcal{Y}_{t,b}\}$ for input data $\mathcal{X}_{t,b}$, label data $\mathcal{Y}_{t,b}$, batch index $b \in \mathcal{B}_t$

…but many different settings…

- Under realistic constraints, data labels and task identity may not be available at train and/or test time
- Training samples for a task might arrive in batches or all at once

Common Settings

Scenario	Training	Testing
IIL [279]	$\{\{\mathcal{D}_{t,b}, t\}_{b\in\mathcal{B}_t}\}_{t=j}$	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required
DIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i = \mathcal{Y}_j$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$, t is not required
TIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is available
CIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
TFCL [15]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
OCL [16]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$, $\lvert b\rvert = 1$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
BBCL [27], [46]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$, $\mathcal{Y}_i \neq \mathcal{Y}_j$ and $\mathcal{Y}_i \cap \mathcal{Y}_j \neq \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
CPT [409]	$\{\mathcal{D}_t^{pt}, t\}_{t\in\mathcal{T}^{pt}}$, followed by a downstream task j	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required

Instance-Incremental Learning (IIL): All training samples belong to the same task and arrive in batches.

Domain-Incremental Learning (DIL): Tasks have the same data label space but different input distributions. Task identities are not required.

Common Settings

Scenario	Training	Testing
IIL [279]	$\{\{\mathcal{D}_{t,b}, t\}_{b\in\mathcal{B}_t}\}_{t=j}$	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required
DIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i = \mathcal{Y}_j$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$, t is not required
TIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is available
CIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
TFCL [15]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
OCL [16]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$, $\lvert b\rvert = 1$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
BBCL [27], [46]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$, $\mathcal{Y}_i \neq \mathcal{Y}_j$ and $\mathcal{Y}_i \cap \mathcal{Y}_j \neq \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
CPT [409]	$\{\mathcal{D}_t^{pt}, t\}_{t\in\mathcal{T}^{pt}}$, followed by a downstream task j	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required

Task-Incremental Learning (TIL): Tasks have disjoint data label spaces. Task identities are provided in both training and testing.

Class-Incremental Learning (CIL): Tasks have disjoint data label spaces. Task identities are only provided in training.

Common Settings

Scenario	Training	Testing
IIL [279]	$\{\{\mathcal{D}_{t,b}, t\}_{b\in\mathcal{B}_t}\}_{t=j}$	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required
DIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i = \mathcal{Y}_j$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$, t is not required
TIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is available
CIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
TFCL [15]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
OCL [16]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$, $\lvert b\rvert = 1$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
BBCL [27], [46]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$, $\mathcal{Y}_i \neq \mathcal{Y}_j$ and $\mathcal{Y}_i \cap \mathcal{Y}_j \neq \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
CPT [409]	$\{\mathcal{D}_t^{pt}, t\}_{t\in\mathcal{T}^{pt}}$, followed by a downstream task j	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required

Task-Free Continual Learning (TFCL): Tasks have disjoint data label spaces. Task identities are not provided in either training or testing.

Online Continual Learning (OCL): Tasks have disjoint data label spaces. Training samples for each task arrive as a one-pass data stream.

- Example: OAK dataset, egocentric video streams collected over nine months; focus on learning new objects as they appear in daily routines

Common Settings

Scenario	Training	Testing
IIL [279]	$\{\{\mathcal{D}_{t,b}, t\}_{b\in\mathcal{B}_t}\}_{t=j}$	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required
DIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i = \mathcal{Y}_j$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$, t is not required
TIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is available
CIL [167], [423]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
TFCL [15]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
OCL [16]	$\{\{\mathcal{D}_{t,b}\}_{b\in\mathcal{B}_t}\}_{t\in\mathcal{T}}$, $\lvert b\rvert = 1$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$ and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is optionally available
BBCL [27], [46]	$\{\mathcal{D}_t, t\}_{t\in\mathcal{T}}$; $p(\mathcal{X}_i) \neq p(\mathcal{X}_j)$, $\mathcal{Y}_i \neq \mathcal{Y}_j$ and $\mathcal{Y}_i \cap \mathcal{Y}_j \neq \emptyset$ for $i \neq j$	$\{p(\mathcal{X}_t)\}_{t\in\mathcal{T}}$; t is unavailable
CPT [409]	$\{\mathcal{D}_t^{pt}, t\}_{t\in\mathcal{T}^{pt}}$, followed by a downstream task j	$\{p(\mathcal{X}_t)\}_{t=j}$; t is not required

Blurred Boundary Continual Learning (BBCL): Task boundaries are blurred, characterized by distinct but overlapping data label spaces.

Continual Pre-training (CPT): Pre-training data arrives in sequence. The goal is to improve knowledge transfer to downstream tasks.

Related Areas & Paradigms

- Transfer learning: uses past experience from a source task to improve learning (speed, data efficiency, accuracy) on a target task.
- Domain Adaptation (DA) and Domain Generalization (DG): Domain-shift is where source and target problems share the same objective, but input distribution shifted. DA adapts source-trained model using sparse or unlabeled data from the target. DG trains source model to be robust to domain-shift without target data.
- Multi-Task Learning: Aims to jointly learn several related tasks, to benefit from regularization due to parameter sharing diversity of the resulting shared representation, and compute/memory savings.
- Meta-learning: Inner learning algorithm solves a task such as image classification, defined by a dataset and objective. Outer algorithm updates the inner algorithm such that the model it learns improves an outer objective (e.g., generalization, learning speed)
- Continual Learning: ability to learn on a sequence of tasks drawn from a potentially non-stationary distribution, accelerating learning new tasks and without forgetting old tasks.

Approaches

Naive solution → reuse all old training samples

Makes it easy to address the previously mentioned challenges, but creates huge **computational** and **storage** overheads, as well as potential **privacy** issues

So we must come up with better approaches…

Approaches

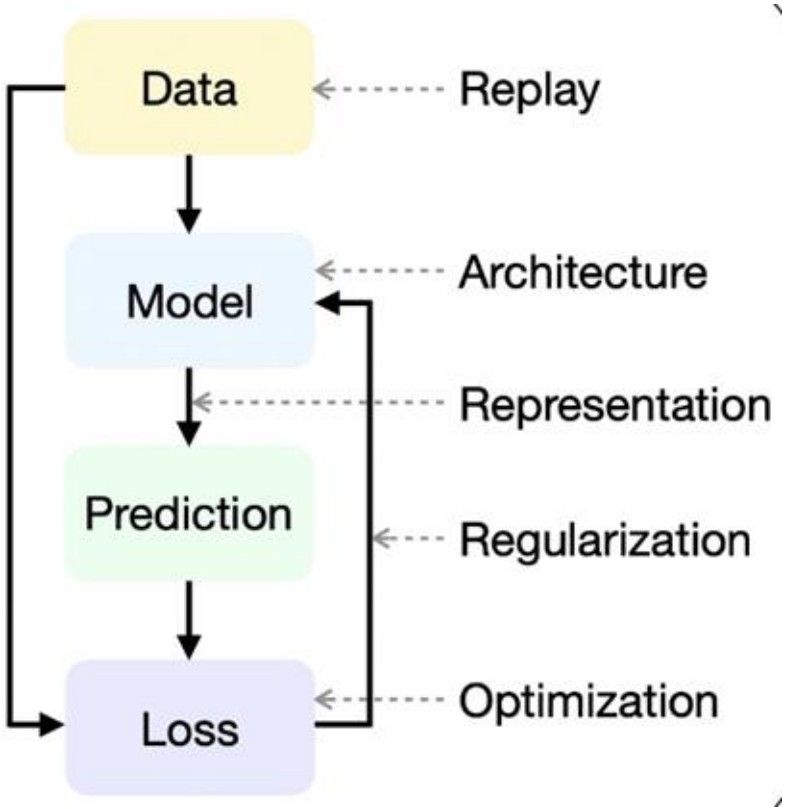
Regularization: Add regularization terms with reference to the old model

Replay: Approximate and recover the old data distributions

Optimization: Explicitly manipulate the optimization programs

Representation: Learn robust and well-distributed representations

Architecture: Construct task-adaptive parameters with a tailored architecture

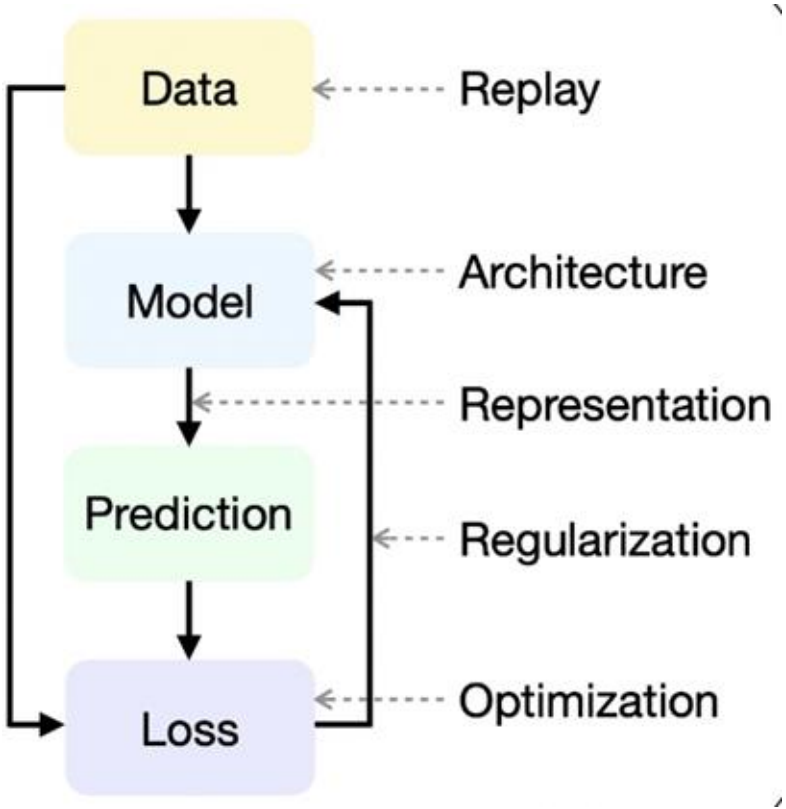


Approaches

These strategies are:

Connected → e.g., regularization and replay guide gradients in optimization

Synergistic → e.g., knowledge distillation for effective replay



Evaluation Metrics

First need to define accuracy during continual learning:

- $a_{k,j}$ denotes accuracy on test set of task j after learning task k
- models updated on tasks in sequence: after learning task k evaluating k^{th} model
- most measures are based on this evaluation matrix:

	t_1	t_2	t_3
m_1	0.67	0.55	0.82
m_2	0.70	0.61	0.85
m_3	0.68	0.57	0.89

Evaluation Metrics: Overall Performance

Evaluated by **Average Accuracy (AA)** and **Average Incremental Accuracy (AIA)**

- AA represents overall performance at current moment
- AIA reflects historical performance

$$\mathrm{AA}_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j},$$

$$\mathrm{AIA}_k = \frac{1}{k}\sum_{i=1}^{k} \mathrm{AA}_i$$

	t_1	t_2	t_3
m_1	0.67	0.55	0.82
m_2	0.70	0.61	0.85
m_3	0.68	0.57	0.89

$\Longrightarrow$

k	AA	AIA
1	0.67	0.67
2	0.66	0.665
3	0.71	0.68

Evaluation Metrics: Memory Stability

Evaluated by **Forgetting Measure (FM)** and **Backward Transfer (BWT)**

Forgetting of a task is the difference between its maximum performance obtained in the past and its current performance:

$$f_{j,k} = \max_{i \in \{1,\ldots,k-1\}} (a_{i,j} - a_{k,j}), \forall j < k. \tag{3}$$

FM at the k-th task is the average forgetting of all old tasks:

$$\mathrm{FM}_k = \frac{1}{k-1} \sum_{j=1}^{k-1} f_{j,k}. \tag{4}$$

As for the latter, BWT evaluates the average influence of learning the k-th task on all old tasks:

$$\mathrm{BWT}_k = \frac{1}{k-1} \sum_{j=1}^{k-1} (a_{k,j} - a_{j,j}), \tag{5}$$

where the forgetting is usually reflected as a negative BWT.

Evaluation Metrics: Learning Plasticity

Evaluated by intransience measure (IM) and forward transfer (FWT)

IM is the inability of a model to learn new tasks, i.e. the difference between its joint training performance and continual learning performance

$$\mathrm{IM}_k = a_k^* - a_{k,k}, \tag{6}$$

where a_k^* is the classification accuracy of a randomly-initialized reference model jointly trained with $\cup_{j=1}^{k} \mathcal{D}_j$ for the k-th task. In comparison, FWT evaluates the average influence of all old tasks on the current k-th task:

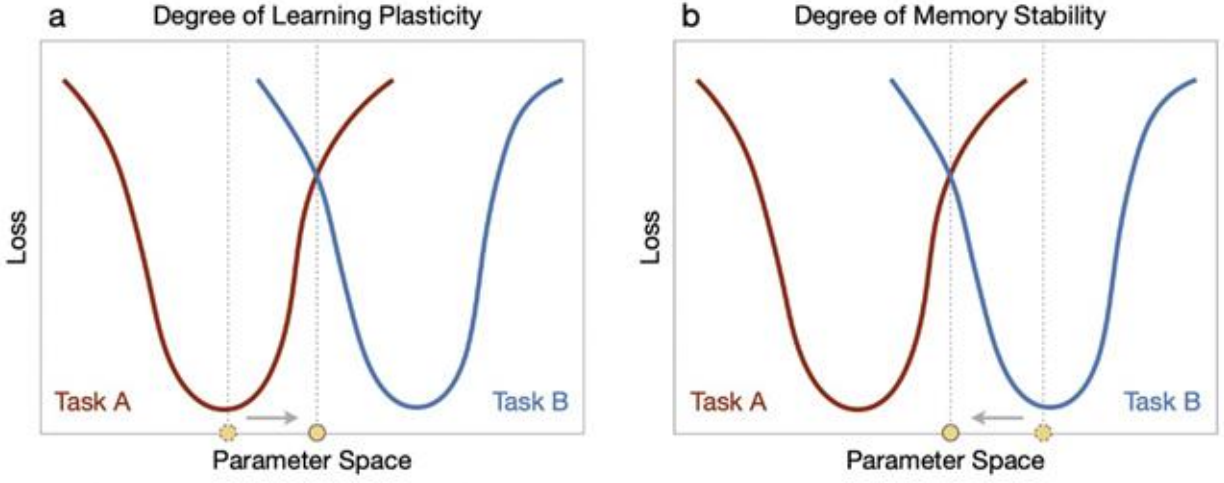
$$\mathrm{FWT}_k = \frac{1}{k-1} \sum_{j=2}^{k} (a_{j,j} - \tilde{a}_j), \tag{7}$$

where $\tilde{a}_j$ is the classification accuracy of a randomly-initialized reference model trained with $\mathcal{D}_j$ for the j-th task.

Stability-Plasticity Trade-off

In continual learning, it is critical but difficult to capture the distributions of both old and new tasks in a balanced manner

Goal: ensure an appropriate stability-plasticity trade-off, where excessive learning plasticity or memory stability can largely compromise each other



Stability-Plasticity Trade-off

Idea 1: Approximate and recover the old data distributions by storing a few old training samples or training a generative model, known as the **replay-based approach**

→ Helps increase stability

→ Replaying old training samples helps old tasks, but this results in potential privacy issues and a linear increase in resource overhead.

→ The use of generative models is also limited by a huge additional resource overhead, as well as their own catastrophic forgetting and expressiveness.

Stability-Plasticity Trade-off

Idea 2: Propagate the old data distributions by formulating continual learning in a Bayesian framework, known as a **regularization-based approach**

$$p(\theta|\mathcal{D}_{1:k}) \propto p(\theta) \prod_{t=1}^{k} p(\mathcal{D}_t|\theta) \propto p(\theta|\mathcal{D}_{1:k-1}) p(\mathcal{D}_k|\theta),$$

→ The posterior of the (k-1)th task becomes the prior of the kth task, and thus enables the new posterior to be computed with only the current training set D_k

→ However, the posterior is generally intractable (except very special cases)

1) Online Laplace approximation
2) Online variational inference

Stability-Plasticity Trade-off

Idea 3: Directly manipulate the gradient-based optimization process, known as the **optimization-based approach**

→ Old training samples M_t for task t are maintained in a memory buffer, gradient directions of the new training samples are encouraged to stay close to that of M_t

→ Alternatively, gradient projection can also be performed without storing old training samples

Idea 4: Incremental tasks can also be learned in a (partially) separated way, which is the dominant idea of the **architecture-based approach**

Generalizability

Since the objective for learning multiple tasks is typically highly non-convex, there are many local optimal solutions that perform similarly on each training set but have significantly different generalizability on test sets and new tasks.

Intra-task → generalizability from training sets to test sets

e.g. test images may be corrupted

Inter-task → generalizability to accommodate incremental changes of task distributions

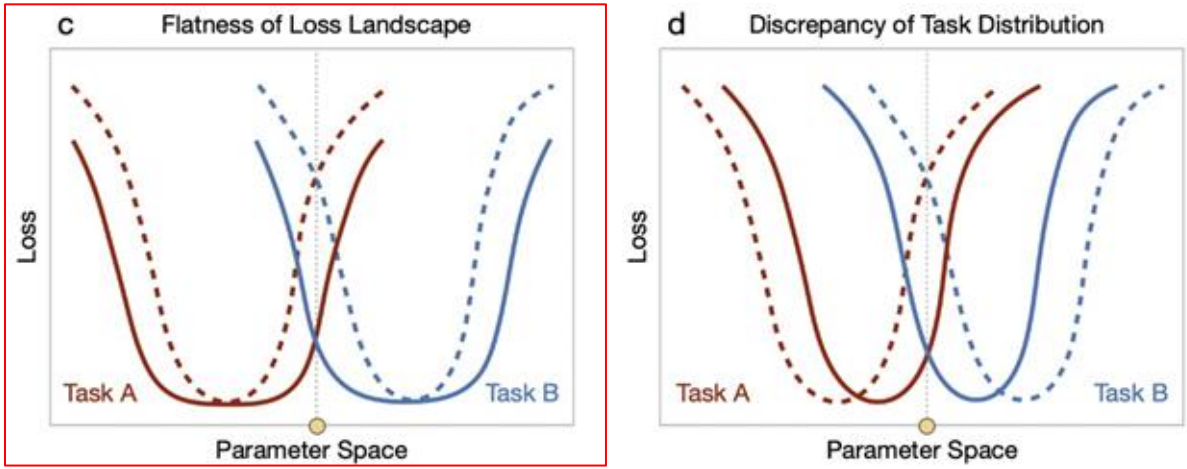
e.g. learning different object classes over time

Generalizability

Factor 1: Loss landscape

Convergence to a local minima with a flatter loss landscape will be less sensitive to modest parameter changes and thus benefit both old and new tasks

→ Can achieve via optimization-based or representation-based approaches

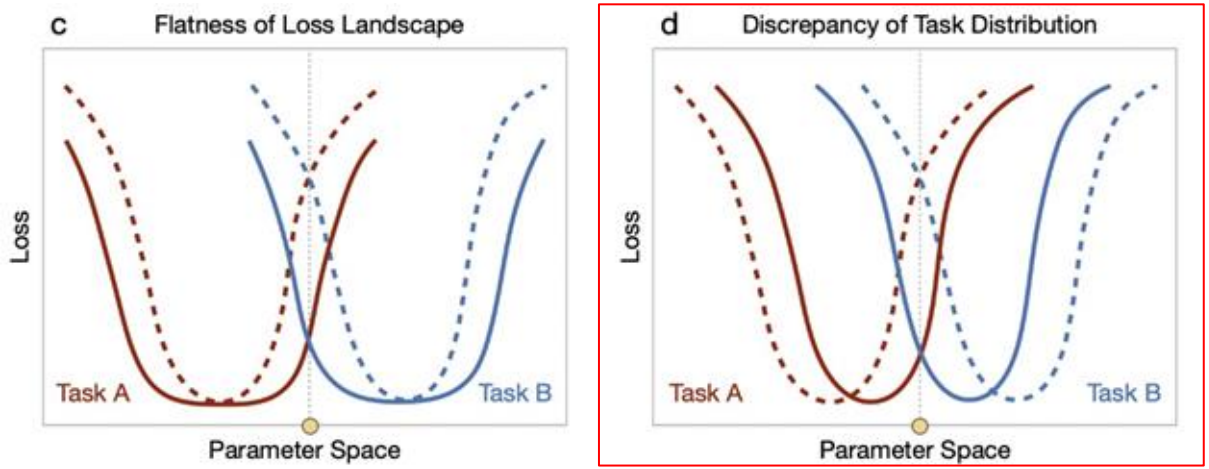


Generalizability

Factor 2: Discrepancy of task distributions

Upper bound of performance degradation also depends on the difference of the empirical optimal solution for each task

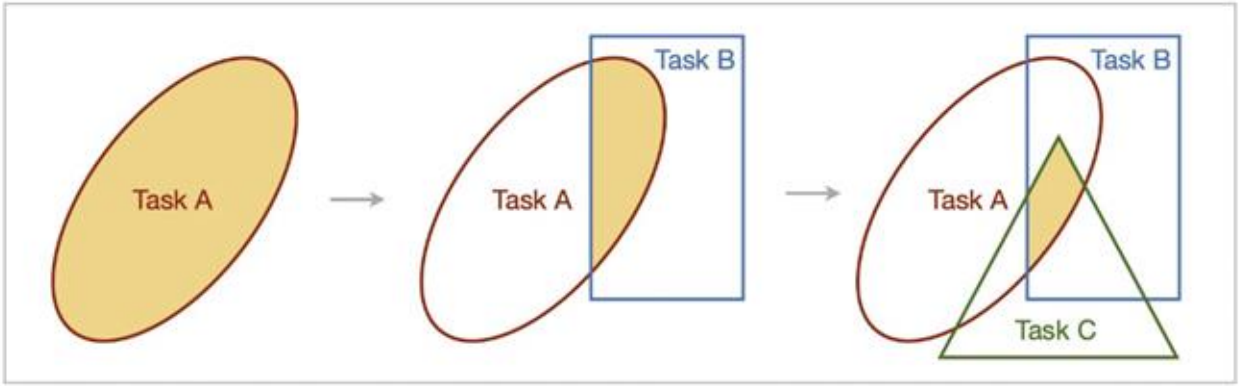
→ Generalization error for each task can improve with synergistic tasks



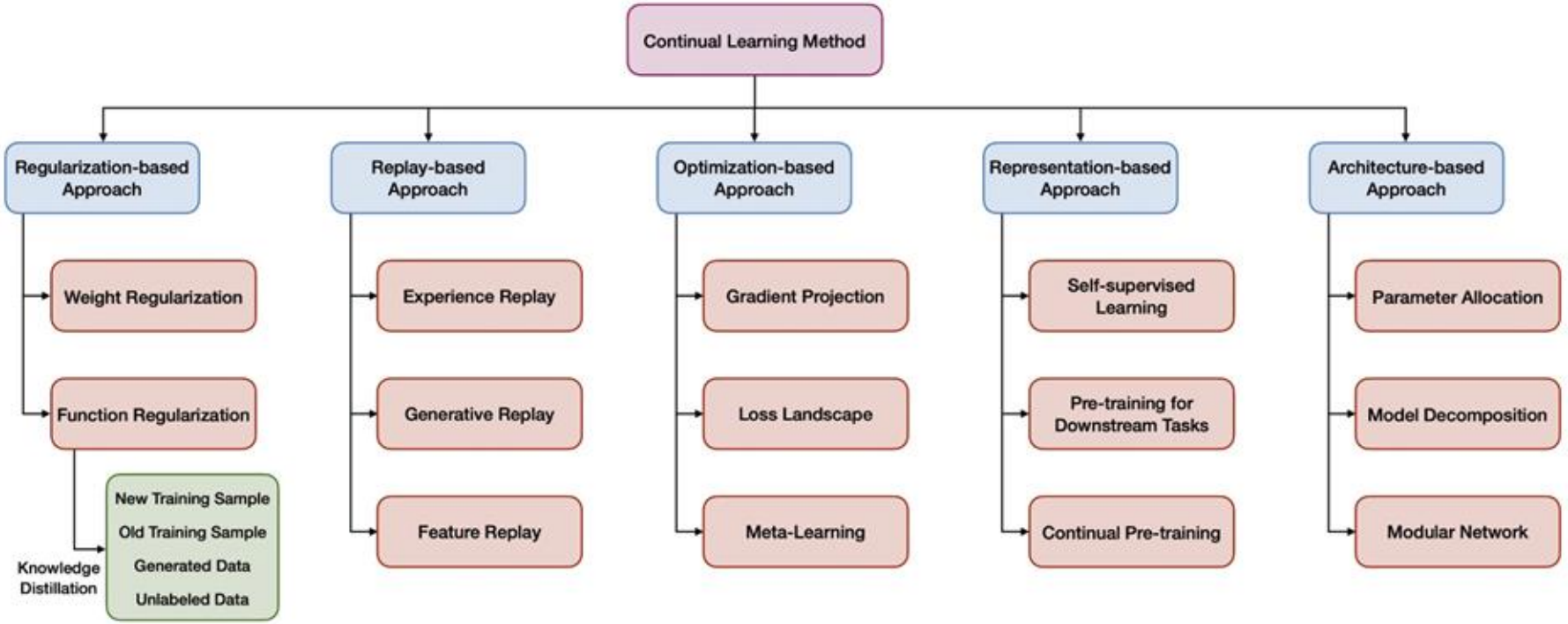
Generalizability

Factor 3: Structure of Parameter Space

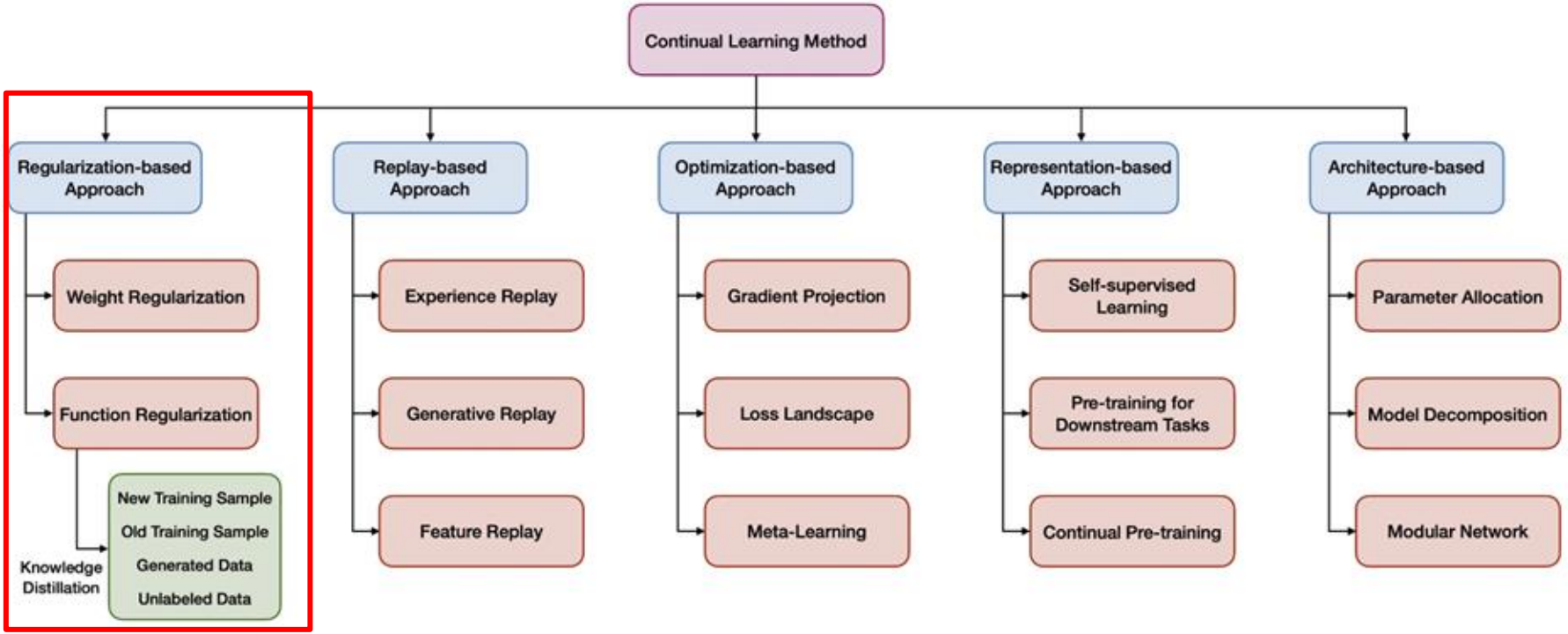
Learning all incremental tasks with a shared solution is equivalent to learning each new task in a constrained parameter space that prevents performance degradation of all old tasks (NP-hard)



Approach Taxonomy



Approach Taxonomy

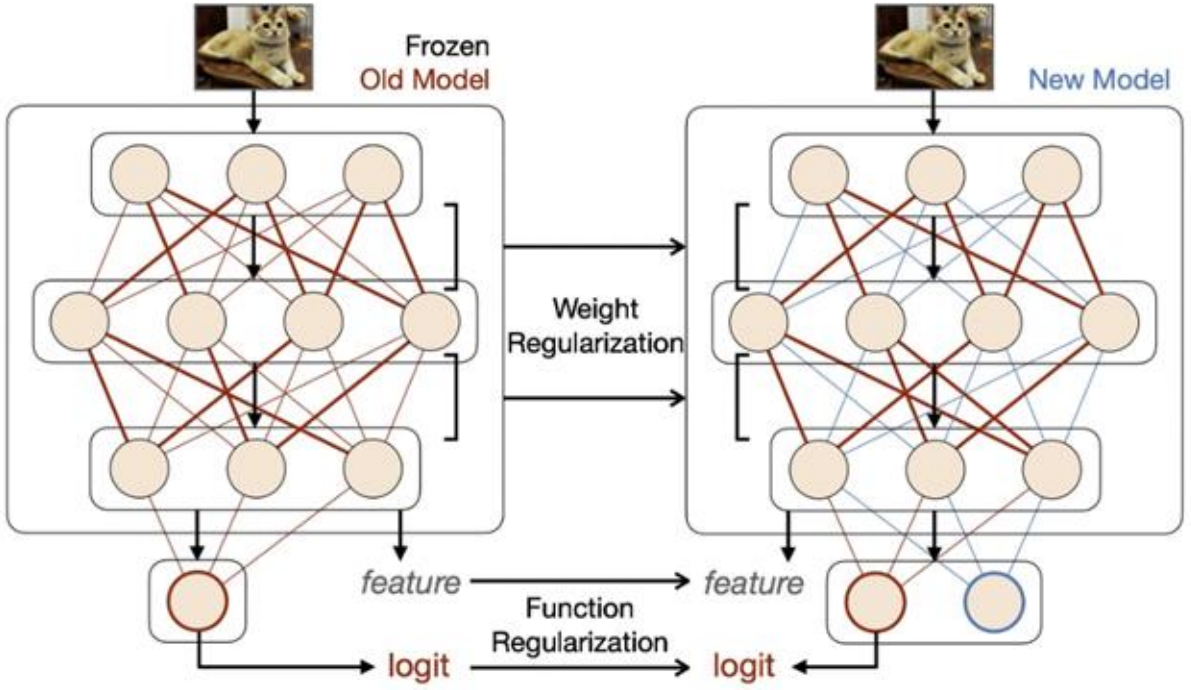


Regularization-Based Approach

Add explicit regularization terms for **stability** with respect to the old tasks

→ usually requires storing a frozen copy of the old model for reference

→ can regularize based on **weights** or **function (i.e., output)**



Weight Regularization

Regularize using the variation of **network parameters.**

→ typical implementation is to add a quadratic penalty in loss function that penalizes the variation of each parameter depending on its contribution or “importance” to performing the old tasks (e.g. based on Fisher information Matrix)

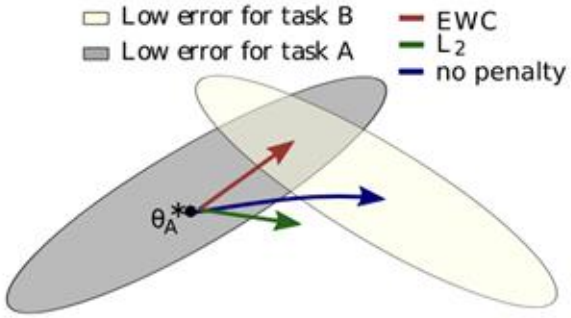
→ efforts to refine importance measure and quadratic penalty

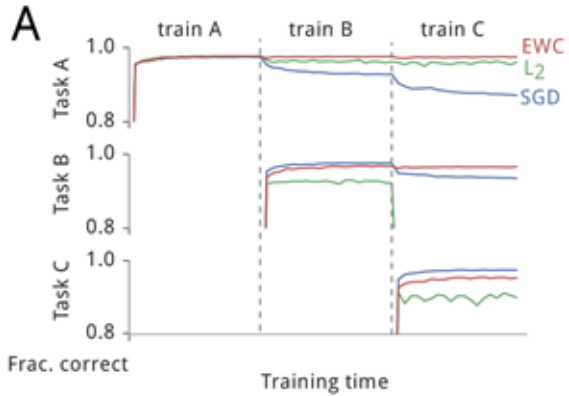
→ expansion-renormalization process of obtaining separately the new task solution and renormalizing it with the old model has been shown to provide a better stability-plasticity trade-off

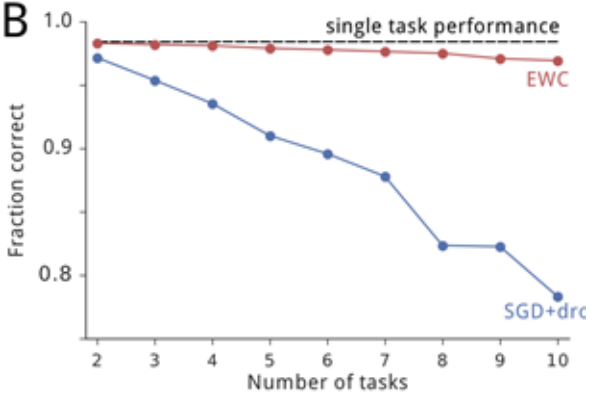
Weight Regularization Example - EWC

EWC protects the performance in task A by constraining the parameters to stay in a region of low error for task A centered around θ^*_A

Constraint can be imagined as a spring anchoring the parameters to the previous solution, where the stiffness is greater for params that matter most to task A.







Source: Overcoming catastrophic forgetting in neural Networks (Kirkpatrick et al.)

EWC Objective

Elastic Weight Consolidation (EWC) [Kirkpatrick et al. 2017] adds a quadratic penalty to the loss function $\mathcal{L}$ to prevent the weights from changing too much from their original values. In particular, we have

$$\mathcal{L}_{EWC}(\theta) = \underbrace{\mathcal{L}_k(\theta)}_{\text{standard loss}} + \underbrace{\frac{\lambda}{2}(\theta - \theta_{\text{old}})^\intercal \hat{\mathbf{F}}_{1:k-1}(\theta - \theta_{\text{old}})}_{\text{regularization term}},$$

where

- $\mathcal{L}_k(\theta)$ is the loss function for task k
- λ is a hyperparameter
- θ_{old} is the parameter vector of the old model
- and $\hat{\mathbf{F}}_{1:k-1}$ is the Fisher Information matrix for the old tasks, which measures the importance of each weight for the old tasks.

A naive alternative to EWC would be to use the L2 norm of the weights as a regularization term: $\mathcal{L}_{L2}(\theta) = \mathcal{L}_k(\theta) + \frac{\lambda}{2}\|\theta - \theta_{\text{old}}\|_2^2$.

Function Regularization

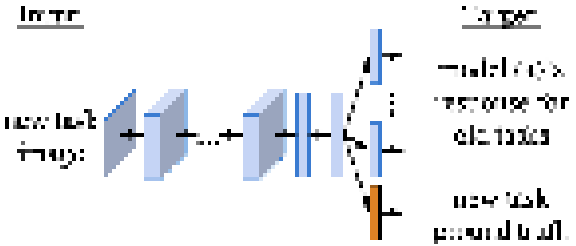
Regularize using **intermediate or final output of the prediction function.**

→ typically employs the previously-learned model as the teacher and the currently-trained model as the student, while implementing knowledge distillation (KD) to mitigate catastrophic forgetting.

→ ideally, the target of KD should be all old training samples, which are unavailable in continual learning. The alternatives can be: new training samples, a small fraction of old training samples, external unlabeled data, generated data, etc., which incur different degrees of distribution shift

Function Regularization Example - LWF

Function Based Regularization uses intermediate or final outputs of the model based on e.g., new training samples [Li and Hoiem 2017], a small fraction of old training samples, generated data, etc.



Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ (output of old tasks for new data)

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ (randomly initialize new parameters)

Train:

$\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ (old task output)

$\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ (new task output)

$$\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \arg\min_{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n} \left(\mathcal{L}_{new}(Y_n, \hat{Y}_n) + \lambda \mathcal{L}_{old}(Y_o, \hat{Y}_o) \right)$$

Function Regularization Example - LWF

$\mathcal{L}_{new}(Y_n, \hat{Y}_n)$ is simply the cross-entropy loss between the ground truth Y_n and the model's prediction $\hat{Y}_n$ for the new task.

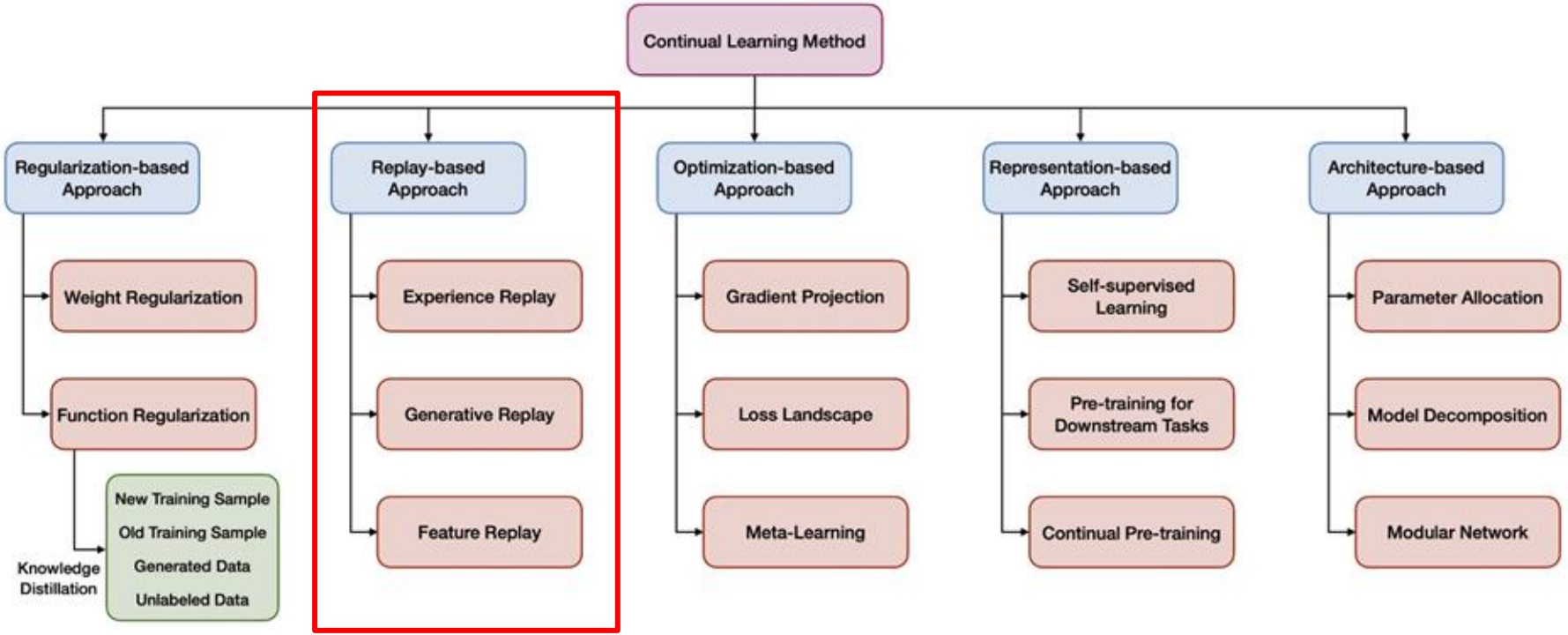
$$\mathcal{L}_{new}(Y_n, \hat{Y}_n) = -\sum_{i=1}^{N} Y_n^{(i)} \log(\hat{Y}_n^{(i)})$$

while $\mathcal{L}_{old}(Y_o, \hat{Y}_o)$ is the distillation loss between the old task's output and the model's prediction for the old task.

$$\mathcal{L}_{old}(Y_o, \hat{Y}_o) = -\sum_{i=1}^{N} \sum_{j=1}^{\ell} Y_o^{(i,j)} \log(\hat{Y}_o^{(i,j)})$$

where ℓ is the number of class for an old task.

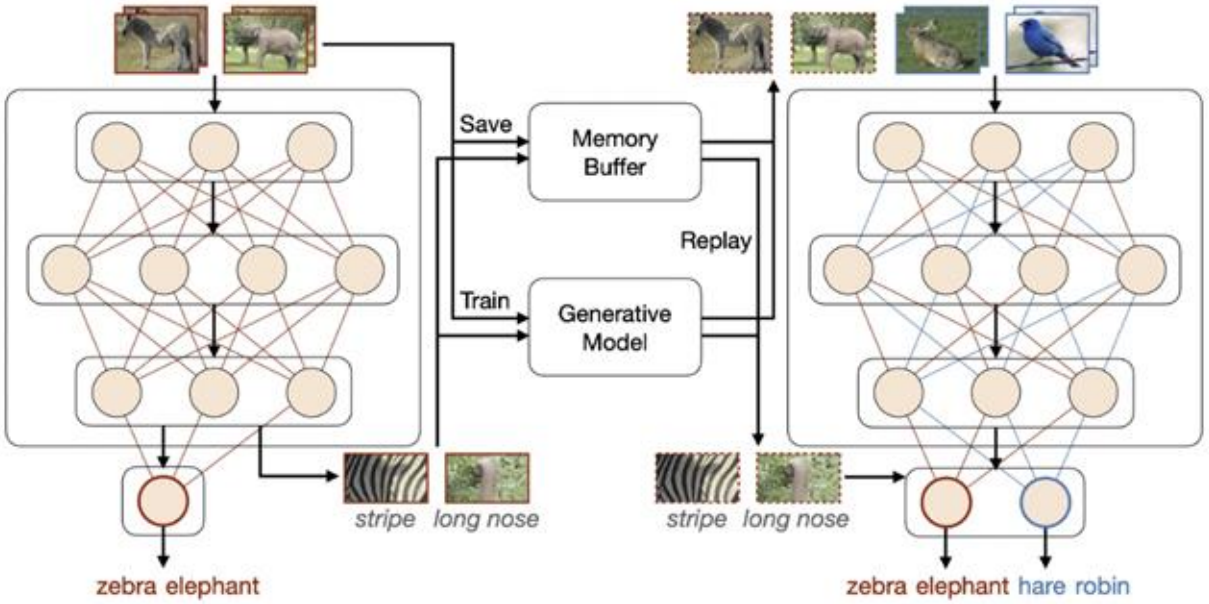
Approach Taxonomy



Replay-Based Approach

Approximate, recall, and/or recover old data distributions to avoid forgetting

Depending on the content of replay, these methods can be further divided into three sub-directions: **experience, generative, and feature-based**



Experience Replay

Stores a few old training samples within a small memory buffer.

→ Due to the extremely limited storage space, a key challenge is how to construct and exploit the memory buffer.

→ The preserved old training samples should be carefully selected, compressed, augmented, and updated, in order to recover adaptively the past information.

Selection and storage: equal number per class, prototypical examples, maximize sample diversity in terms of parameter gradients, compression for storage efficiency, augmentation, etc.

Exploitation: How do we make use of the memory buffer to recover the past information? Fine-tuning, knowledge distillation, etc.

Replay Example - GEM

Gradient Episodic Memory (GEM) constructs constraints for each task based on the old training samples to ensure non-increase in their losses

Key idea:

- Can diagnose increases in the loss of previous tasks by computing the angle between their loss gradient vector and the proposed update → project to nearest non-loss-increasing solution

Source: Gradient Episodic Memory for Continual Learning
David Lopez-Paz, Marc'Aurelio Ranzato

Replay Example - GEM

Gradient Episodic Memory (GEM) [Lopez-Paz and Ranzato 2017] constructs individual constraints based on the old training samples for each task to ensure non-increase in their losses. When observing the triplet (x, t, y), we solve the following problem:

$$\begin{aligned} \min_{\theta} \quad & \mathcal{L}(f_\theta(x,t), y) \\ \text{s.t.} \quad & \mathcal{L}(f_\theta, \mathcal{M}_k) \leq \mathcal{L}(f_\theta^{t-1}(x,t), \mathcal{M}_k) \quad \forall k < t \end{aligned}$$

where $\mathcal{M}_k$ is the memory buffer for task k and f_θ^{t-1} is the model at the end of task $t-1$. Rewrite as

$$\langle g, g_k \rangle \equiv \langle \nabla_\theta \mathcal{L}(f_\theta(x,t), y), \nabla_\theta \mathcal{L}(f_\theta, \mathcal{M}_k) \rangle \geq 0 \quad \forall k < t$$

and we encode this constraint to the following quadratic program:

$$\begin{aligned} \min_{\tilde{g}} \quad & \frac{1}{2}\|\tilde{g} - g\|_2^2 \\ \text{s.t.} \quad & \langle \tilde{g}, g_k \rangle \geq 0 \quad \forall k < t \end{aligned}$$

Generative Replay

Train an additional generative model to replay data from old task distributions

→ closely related to continual learning of generative models themselves, as these models also require incremental updates

e.g. learning each generation task is accompanied with replaying generated data sampled from the old generative model, so as to inherit the previously-learned knowledge.

→ often uses GANs, VAEs

However, continual learning of generative models is extremely difficult and requires significant resource overhead, so generative replay is often limited to relatively simple datasets

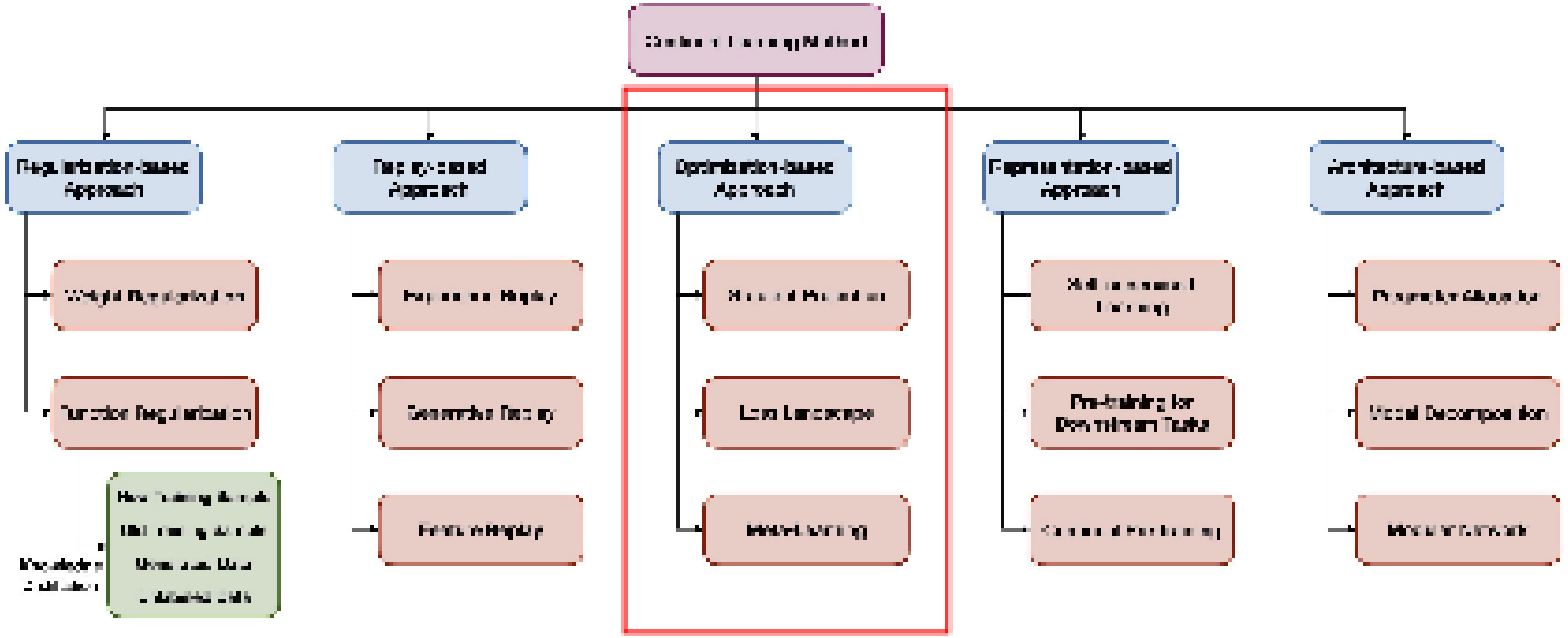
Feature-Based Replay

Maintaining feature-level rather than data-level distributions enjoys numerous benefits in terms of efficiency and privacy.

→ central challenge is the representation shift caused by sequentially updating the feature extractor, which reflects the feature-level catastrophic forgetting

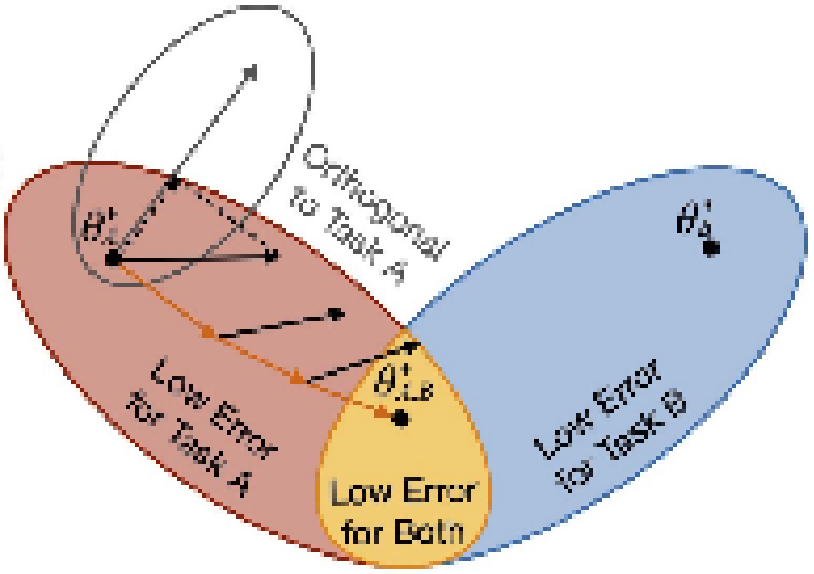
→ the use of strong pretraining can provide robust representations that are generalizable to downstream tasks and remain relatively stable in continual learning.

Optimization-Based Approach



Gradient Projection

- **Gradient projection** methods project the gradient of the new task onto the subspace spanned by the gradients of the old tasks (e.g., **GEM**)
- **Orthogonal** methods remove the component of the gradient of the new task in the orthogonal subspace spanned by the gradients of the old tasks



Gradient Projection

Gradient Projection: Orthogonal Gradient Descent

Orthogonal Gradient Descent (OGD) [Farajtabar et al. 2019] preserves previously acquired knowledge by maintaining a space consisting of the gradient directions of previous tasks. Any update orthogonal to this gradient space change the output of the network minimally.

Consider a first-order Taylor expansion of the loss function $\mathcal{L}_A$ for task A around the current parameter w_A^* at a new set of parameters $w = w_A^* + \Delta w$:

$$\begin{aligned}\mathcal{L}_A(w) &\approx \mathcal{L}_A(w_A^*) + \nabla \mathcal{L}_A(w_A^*)^T \Delta w \\ &= \mathcal{L}_A(w_A^*) + 0\end{aligned}$$

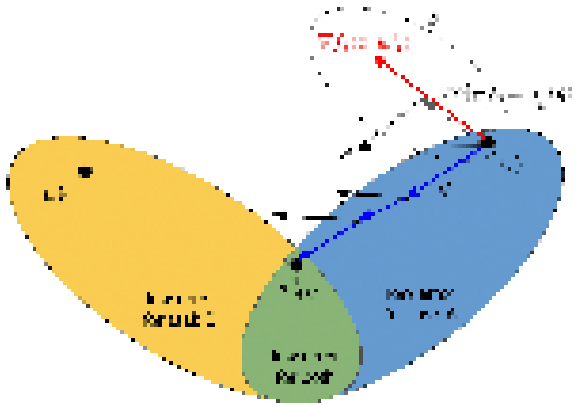
by our choice of Δw orthogonal to the gradient space of the old tasks.

Gradient Projection: Orthogonal Gradient Descent

Algorithm 1 Orthogonal Gradients Descent

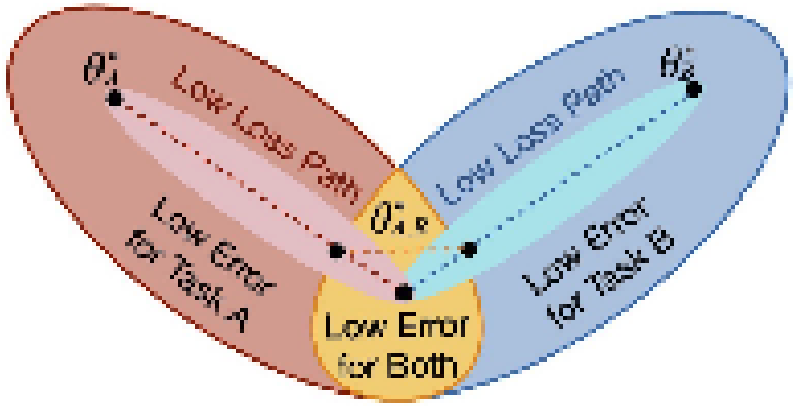
Input Task sequence $T_1, T_2, T_3, \ldots$, learning rate η
Output The optimal parameter w.

Initialize $S \leftarrow \{\}$; $w \leftarrow w_0$
for Task ID $k = 1, 2, 3, \ldots$ **do**
  repeat
    $g \leftarrow$ Stochastic/Batch Gradient for T_k at w
    $\tilde{g} = g - \sum_{v \in S} \text{proj}_v(g)$
    $w \leftarrow w - \eta\tilde{g}$
  until convergence
  for $(x, y) \in T_k$ and $j \in [1, c]$ s.t. $y_j = 1$ **do**
    $u \leftarrow \nabla f_j(x; w) - \sum_{v \in S} \text{proj}_v(\nabla f_j(x; w))$
    $S \leftarrow S \cup \{u\}$
  end for
end for



Loss Landscape Approaches

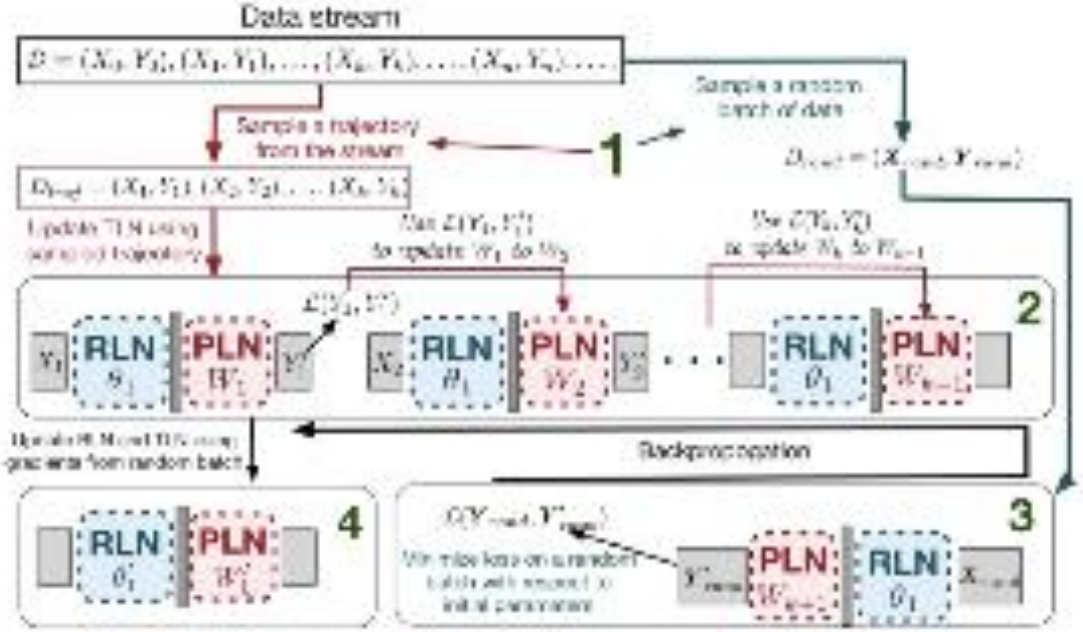
- **Curvature** methods attempt to "widen" the minima of the loss landscape to prevent catastrophic forgetting [Mirzadeh, Farajtabar, Pascanu, et al. 2020]
  - learning rate, batch size, regularization method impact the geometry of the loss landscape
  - *this was the paper I enjoyed reading the most*
- **Loss landscape** methods aim to find a low-error path (in the loss landscape) to a joint solution for several tasks [Mirzadeh, Farajtabar, Gorur, et al. 2020]



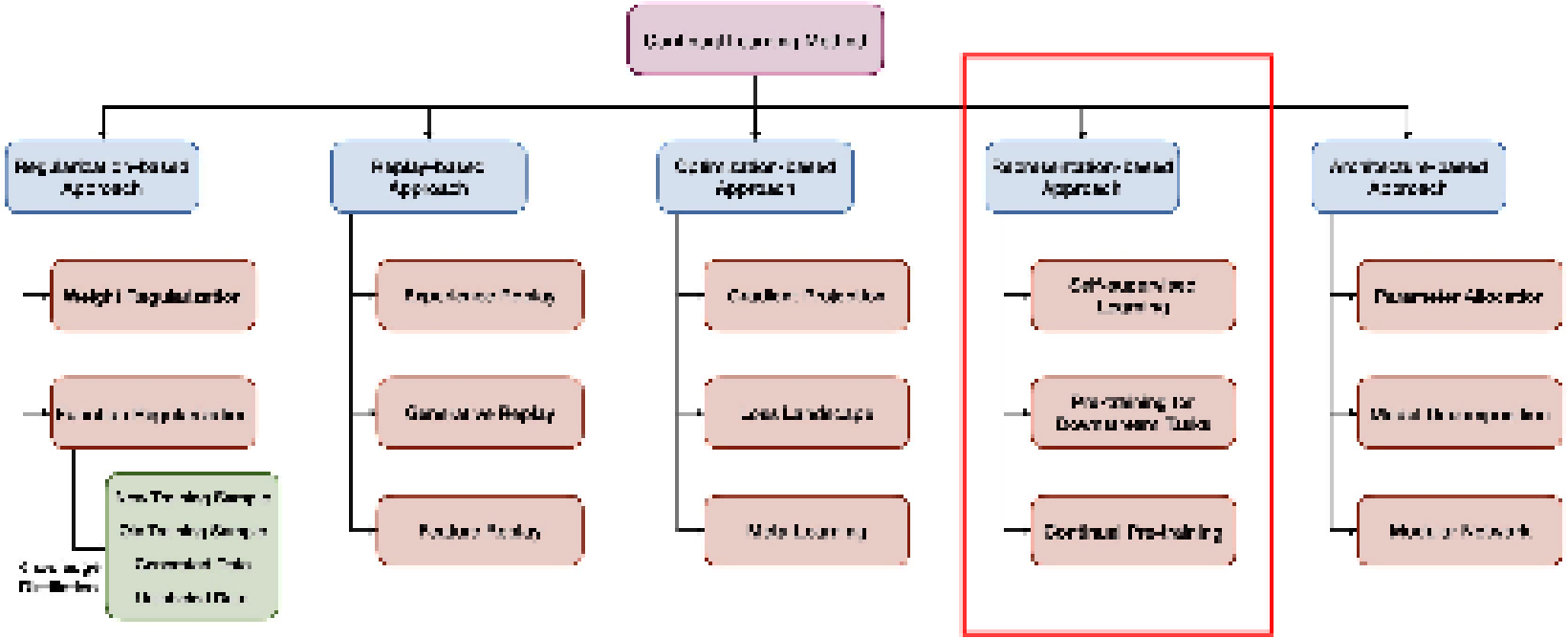
Loss Landscape

Meta-Learning Approaches

- **Meta learning** methods in continual learning aim to learn about how tasks interfere with each other
- Online Aware Meta-Learning [Javed and White 2019] computes the degree of interference and uses it as a training signal.
- **It learns sparse representations**

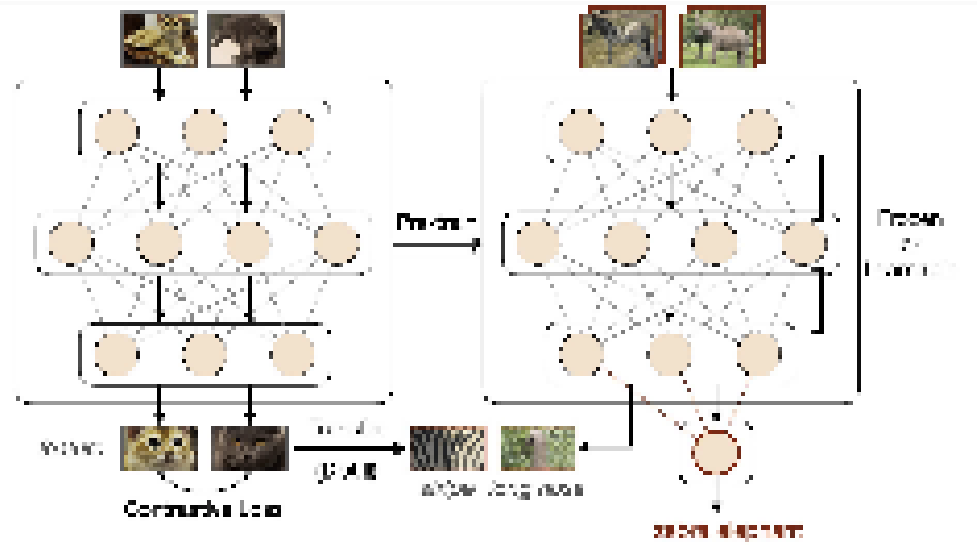


Representation-Based Approach

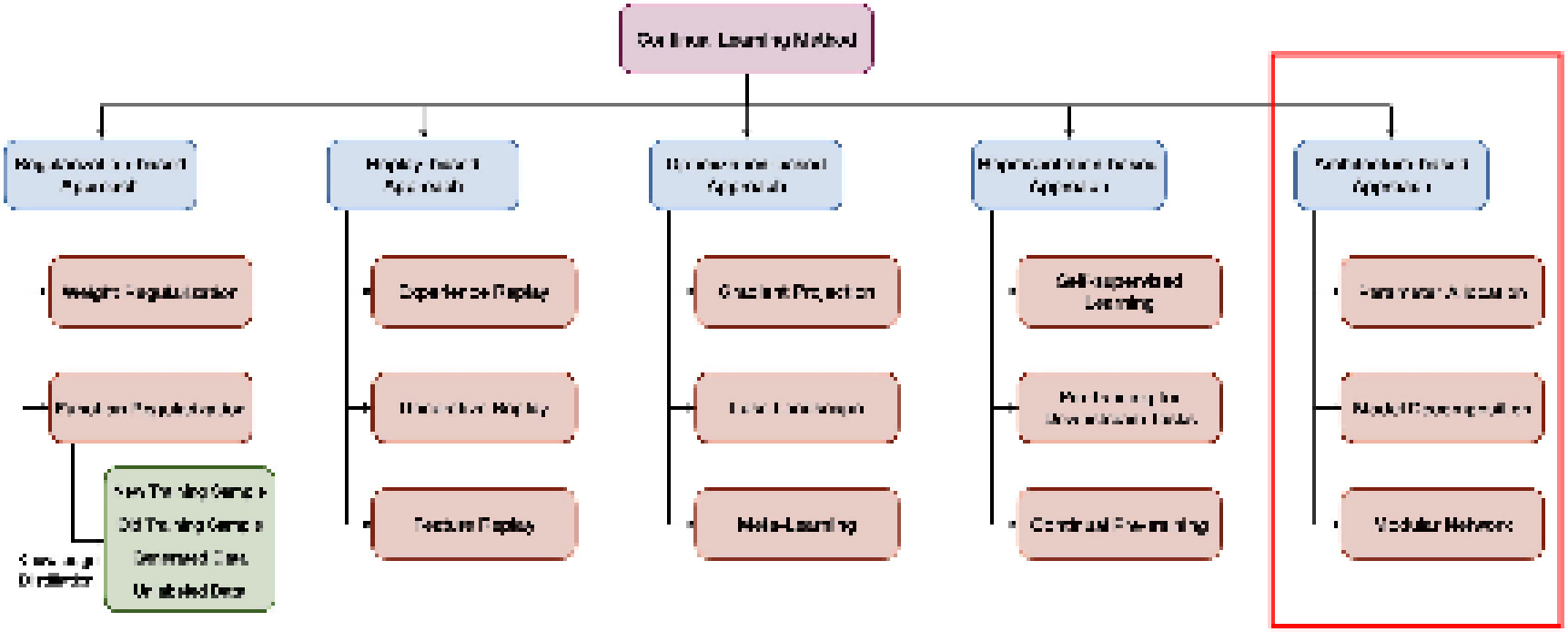


Representation-Based Approach

- **Self-supervised learning/pre-training** methods aim to learn a representation that is invariant to task identity. Such representations can be frozen or tuned for downstream tasks.
- **Continual pre-training** methods aim to continually refine foundation models for new tasks as new (potentially unlabeled) data becomes available.



Architecture-Based Approach



Architecture-Based Approach

- **Parameter Allocation** assigns dedicated parameters to each task, e.g., LoRA [Hu et al. 2021]
- **Modular Networks** construct task-adaptive sub-modules or subnetworks
- **Model Decomposition** breaks the model into tasksharing and task-specific components, e.g.,
  - low-rank factorization
  - masking of intermediate features