華中科技大学
人工智能与自动化学院
SCHOOL OF ARTIFICIAL INTELLIGENCE AND AUTOMATION.HUST

HUST BCI&ML Lab
脑机接口与机器学习实验室
BRAIN-COMPUTER INTERFACE AND MACHINE LEARNING LABORATORY

Continual Learning

Dongrui Wu
School of Artificial Intelligence and Automation
Huazhong University of Science and Technology
drwu@hust.edu.cn

Outline

Motivation

◆**Machine learning state-of-the-art**

- Deep learning achieves state-of-the-art performance in many tasks
- Mainly supervised training with huge and fixed datasets

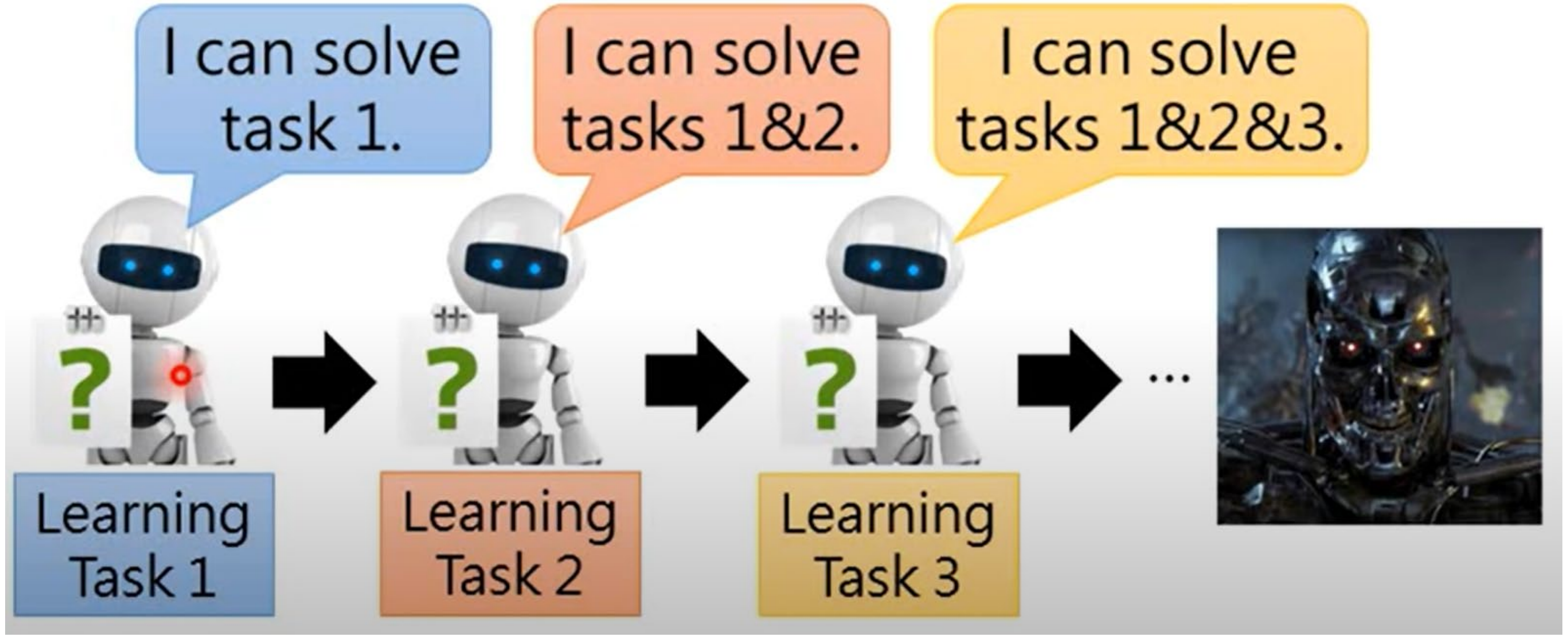
◆**Limitation:**

- Need to restart the training process each time new data become available
- Intractable due to storage constraints or privacy issues

Call for systems that adapt continually and keep on learning over time

Continual Learning (CL)

- **Goal:** Gradually extend acquired knowledge and use it for future learning
- Learn from an infinite stream of data
- Also referred to as **lifelong learning**, **sequential learning**, or **incremental learning**



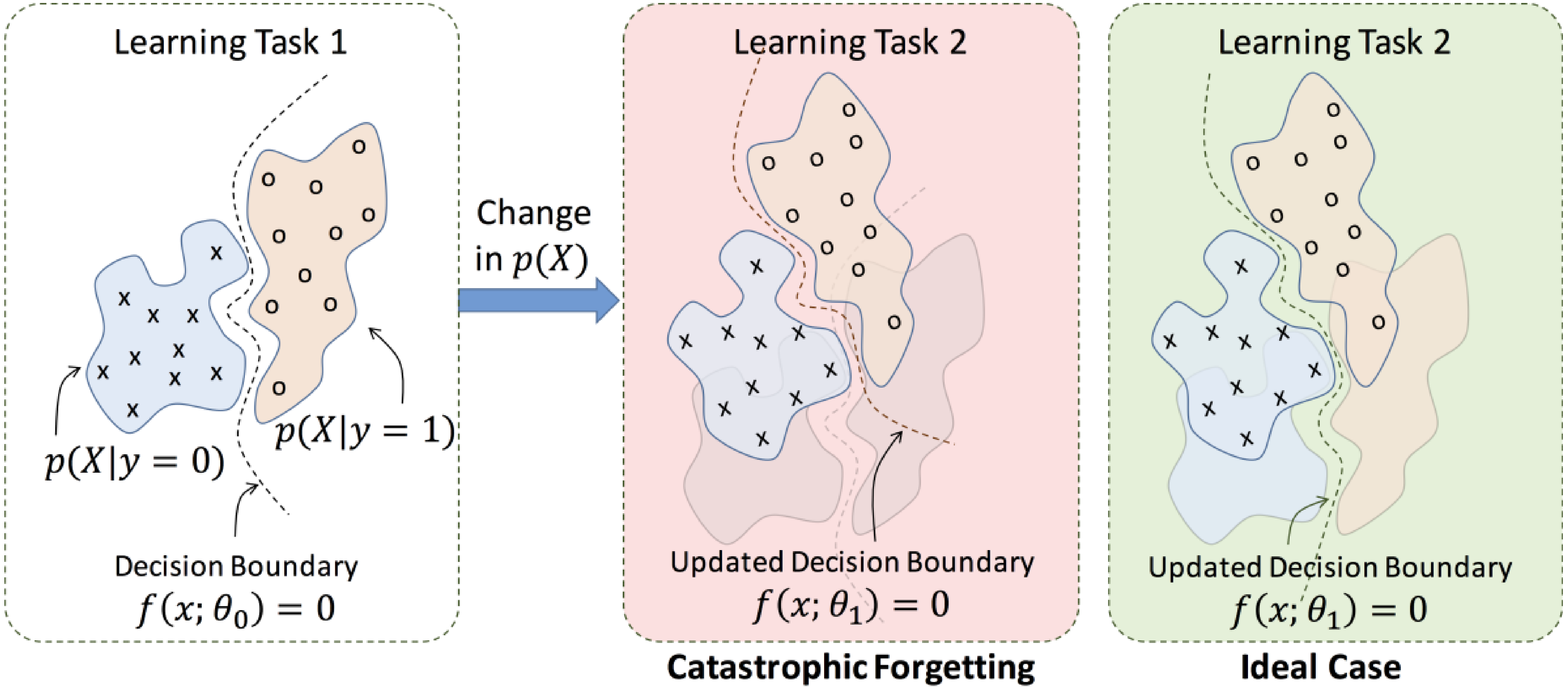
De Lange M, Aljundi R, Masana M, et al. A continual learning survey: Defying forgetting in classification tasks. IEEE TPAMI 2021.

Major Challenge: Catastrophic Forgetting

- Performance on a previously learned task or domain should not significantly degrade over time as new tasks or domains are added

- **Stability-Plasticity dilemma:**
  - ✓**Stability**: Retain previous knowledge
  - ✓**Plasticity**: Ability to learn new knowledge

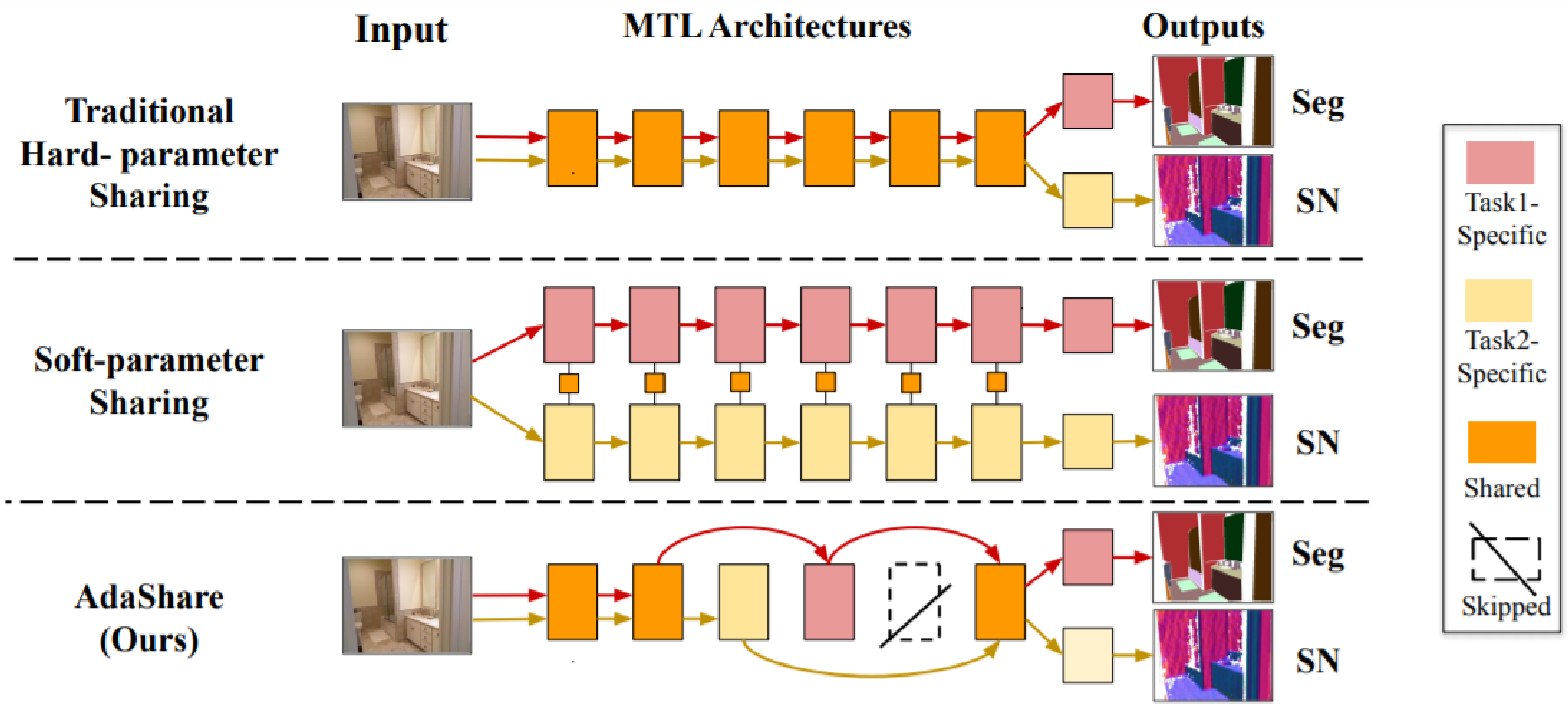
Catastrophic Forgetting



Kolouri S, Ketz N, Zou X, et al. Attention-based selective plasticity. arXiv preprint arXiv:1903.06070, 2019.

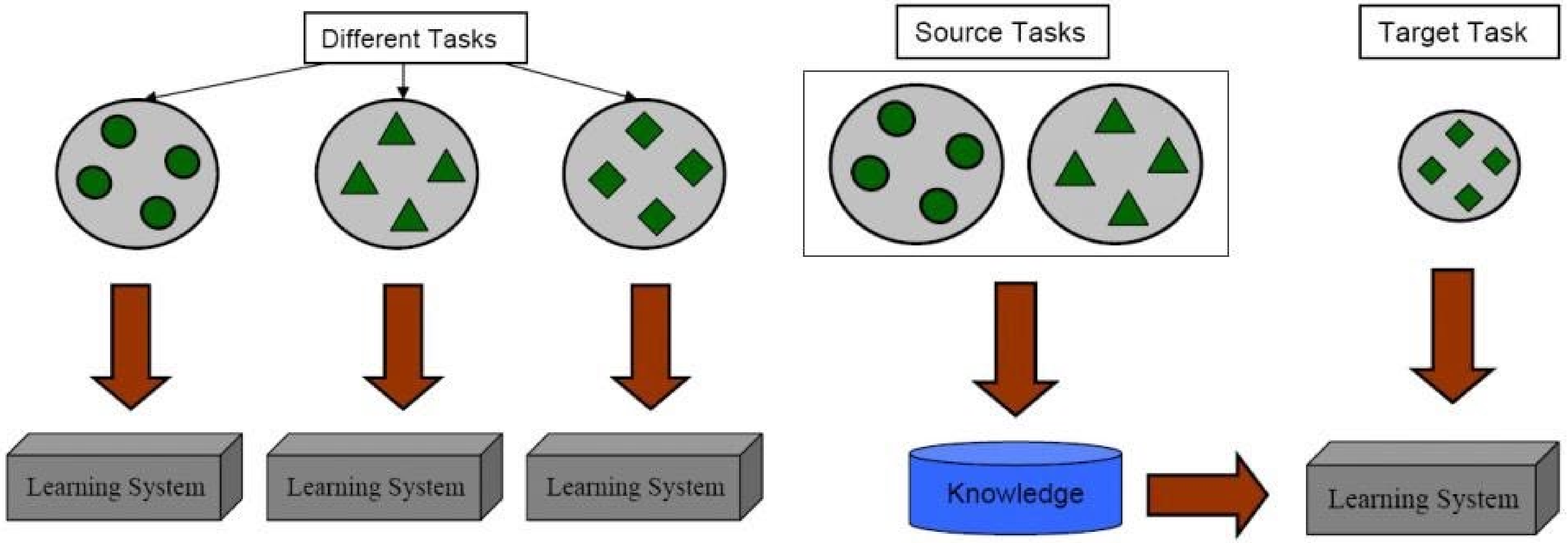
Related Field: Multi-Task Learning

- **Multi-Task Learning**: Learn multiple related tasks simultaneously using a set or subset of shared parameters
- Aims for a better generalization and a reduced overfitting using shared knowledge extracted from related tasks



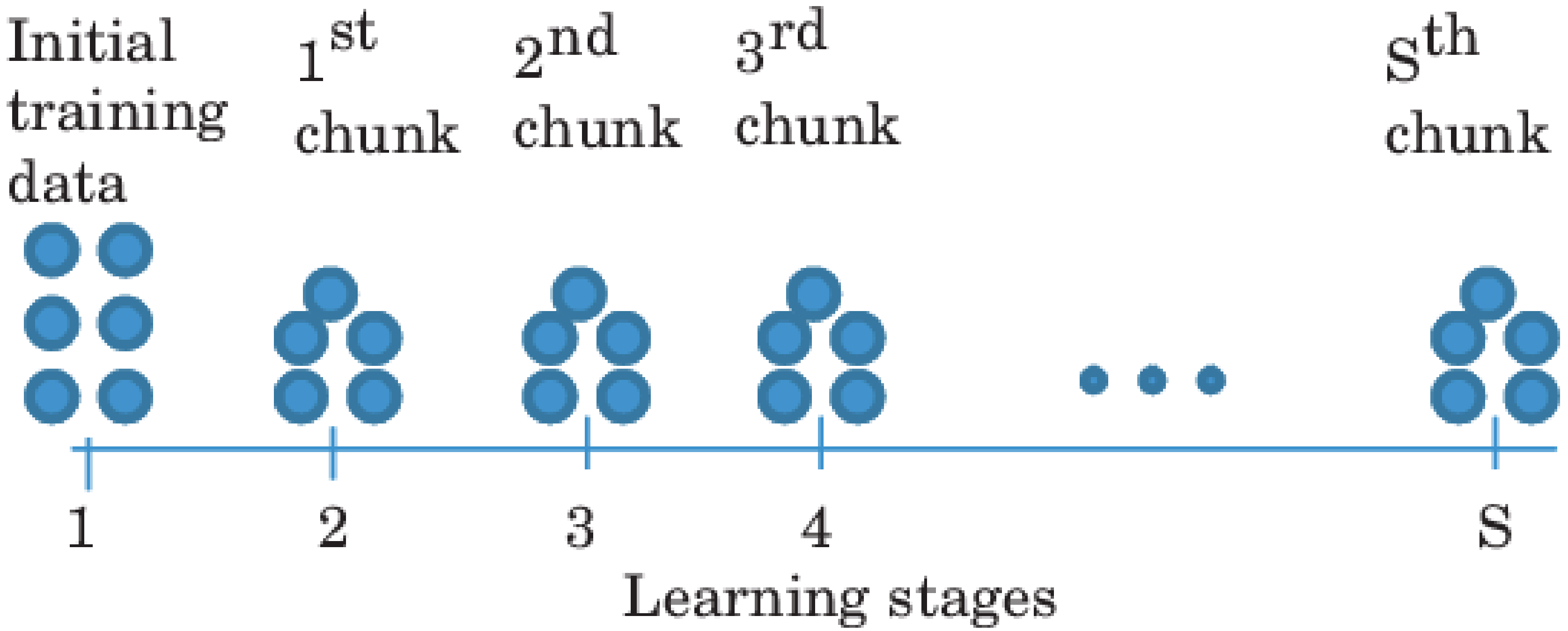
Related Field: Transfer Learning

- **Transfer Learning**: Aid the learning process of a given task (the target) by exploiting knowledge acquired from another task or domain (the source)
- Performance on the source task(s) is not important



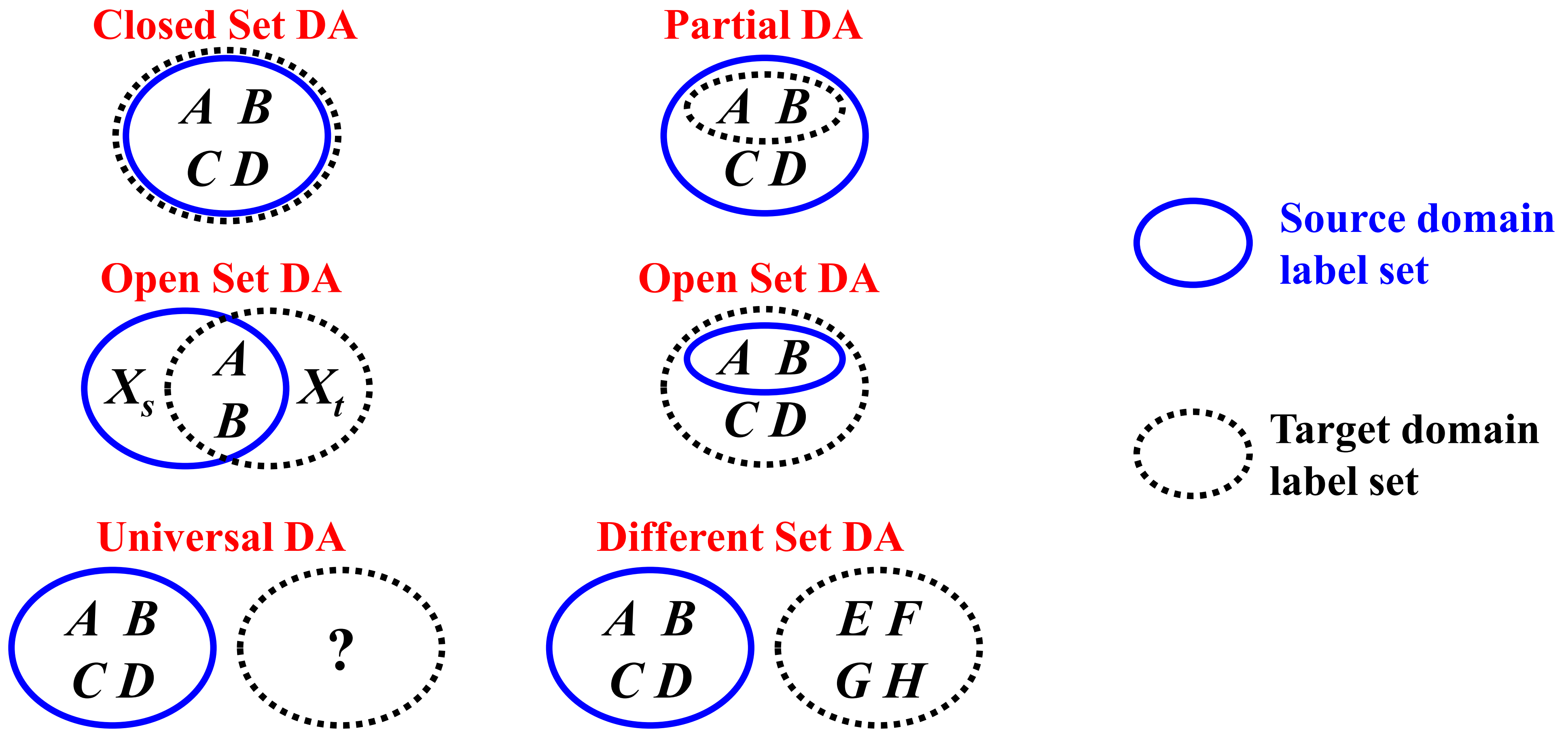
Related Field: Online Learning

- **Online Learning**: Optimize predictive models over a stream of data instances sequentially
- Assumes an i.i.d data sampling procedure and considers a single task domain



Related Field: Open World Learning

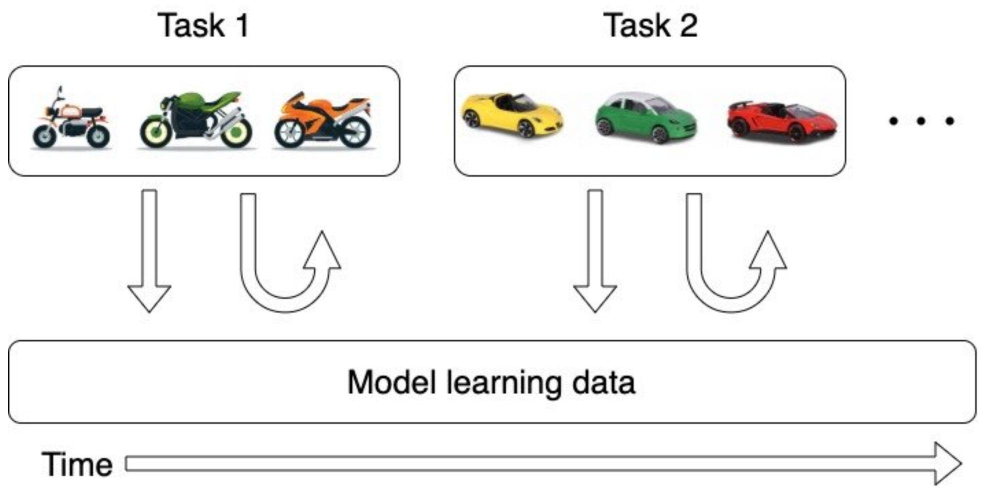
- **Open World Learning**: Detect new classes at test time
- When those new classes are then integrated into the model, it becomes continual learning



3 Scenarios of CL: Task-Incremental Learning

Task-Incremental Learning: Incrementally learn a set of distinct tasks

- ✓ Always clear which task being performed during training and testing
- ✓ Task identity is explicitly provided, or the tasks are clearly distinguishable
- ✓ Train models with task-specific components or a completely separate network for each task (no forgetting)



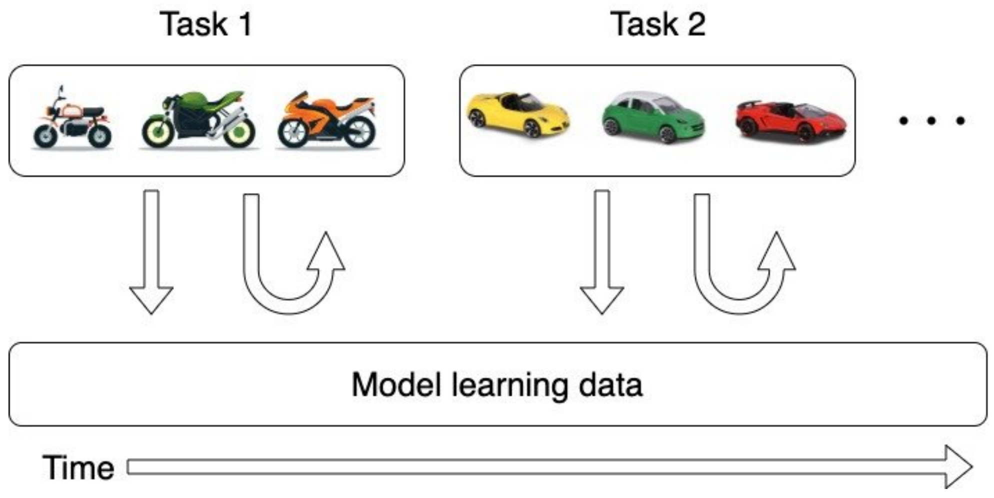
- **Challenge**: Find effective ways to share learned representations across tasks
- **Example:** Learn to play different sports or different musical instruments (it is always clear which sport or instrument should be played)

van de Ven G M, Tuytelaars T, Tolias A S. Three types of incremental learning. Nature Machine Intelligence, 2022: 1-13.

3 Scenarios of CL: Domain-Incremental Learning

Domain-Incremental Learning: The structure of the problem is the same, but the input-distribution changes (domain-shifts)

- ✓ The algorithm does not know which task a sample belongs to (no task identity)
- ✓ If using task-specific components, the algorithm must first identify the task

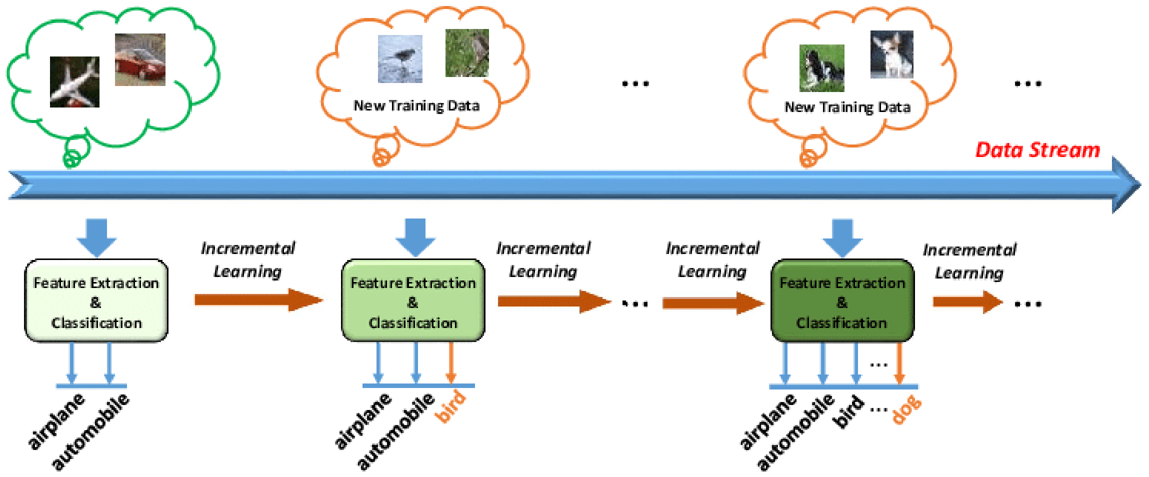


- **Challenge**: Alleviate catastrophic forgetting
- **Example:** Incrementally learn to recognize objects under variable lighting conditions (for example, indoor versus outdoor), or learn to drive under different weather conditions

3 Scenarios of CL: Class-Incremental Learning

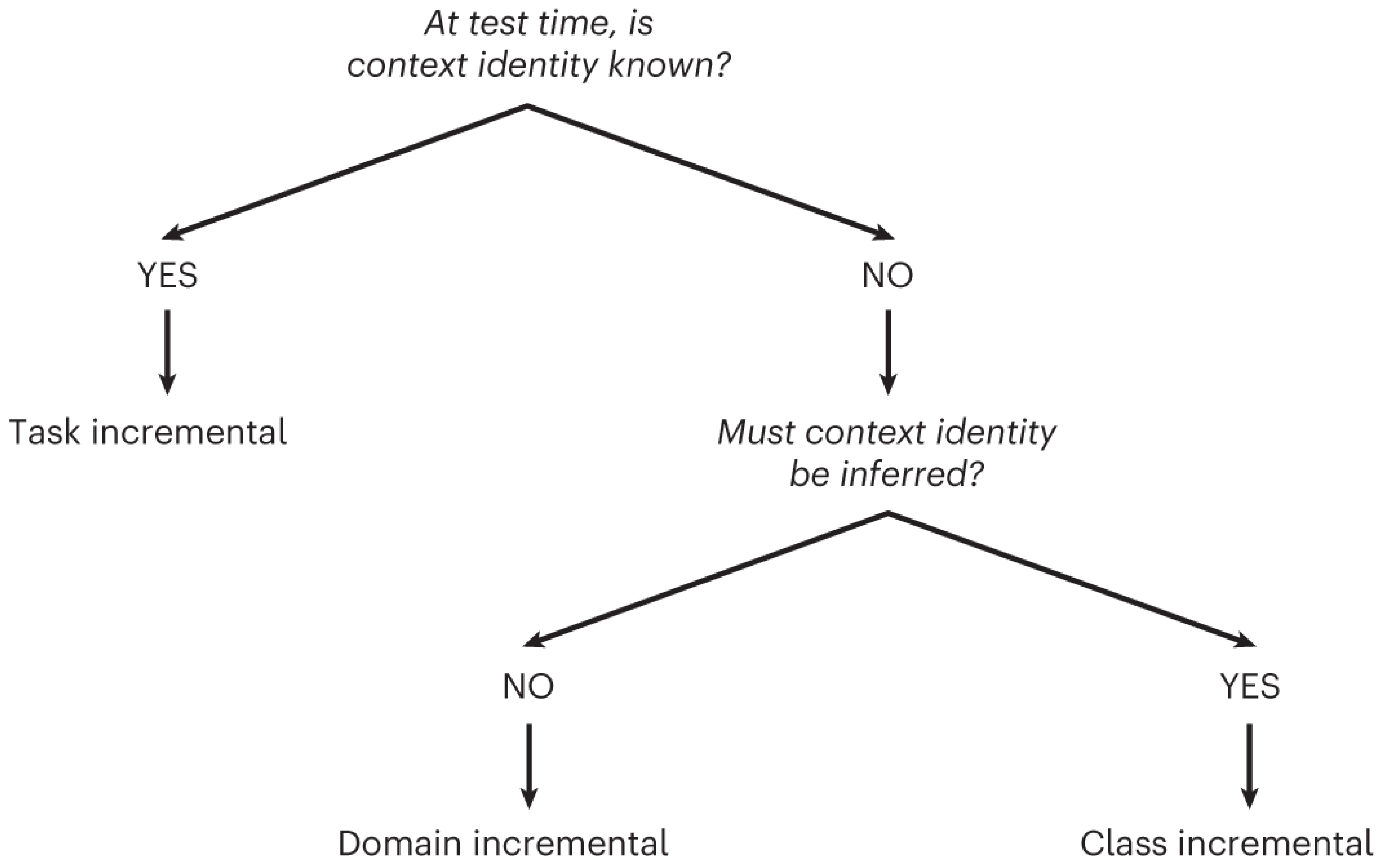
Class-Incremental Learning: Algorithm must incrementally learn to discriminate between a growing number of objects or classes

✓ A sequence of classification-based tasks, each task contains different classes and the algorithm must learn to distinguish between all classes



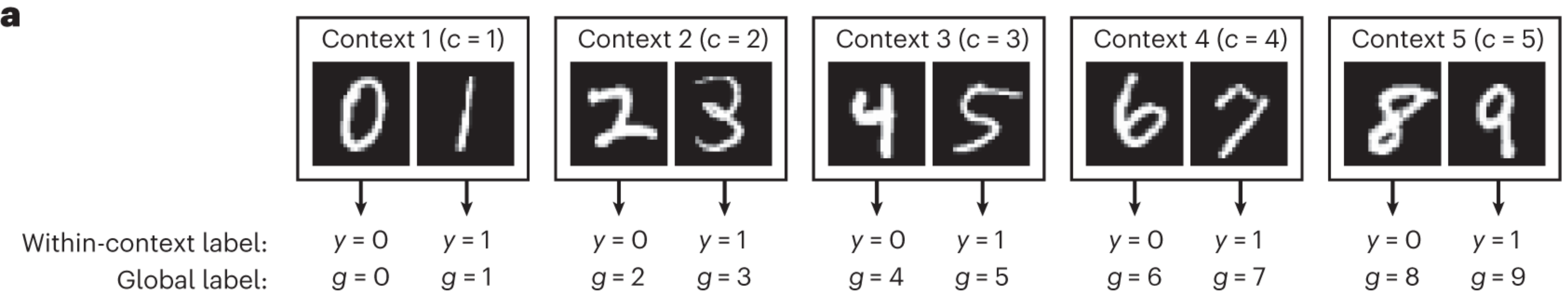
- **Challenge**: Learn to discriminate between classes that are not observed together
- **Example:** An algorithm first learn about airplane and automobile, and later about birds and dogs; the algorithm needs to complete a 4-class classification task

3 Scenarios of CL: Summary


At test time, is context identity known?
YES
NO
Task incremental
Must context identity be inferred?
NO
YES
Domain incremental
Class incremental

van de Ven G M, Tuytelaars T, Tolias A S. Three types of incremental learning. Nature Machine Intelligence, 2022: 1-13.

3 Scenarios of CL: Example & Comparison



b

	Input (at test time)	Expected output	Intuitive description
Task-incremental learning	Image + context label	Within-context label[a]	Choice between two digits of same context (e.g. 0 or 1)
Domain-incremental learning	Image	Within-context label	Is the digit odd or even?
Class-incremental learning	Image	Global label	Choice between all ten digits

van de Ven G M, Tuytelaars T, Tolias A S. Three types of incremental learning. Nature Machine Intelligence, 2022: 1-13.

Outline

- Basic Concepts
- **Rehearsal Methods**
- Regularization-Based Methods
- Parameter Isolation Methods
- Discussion

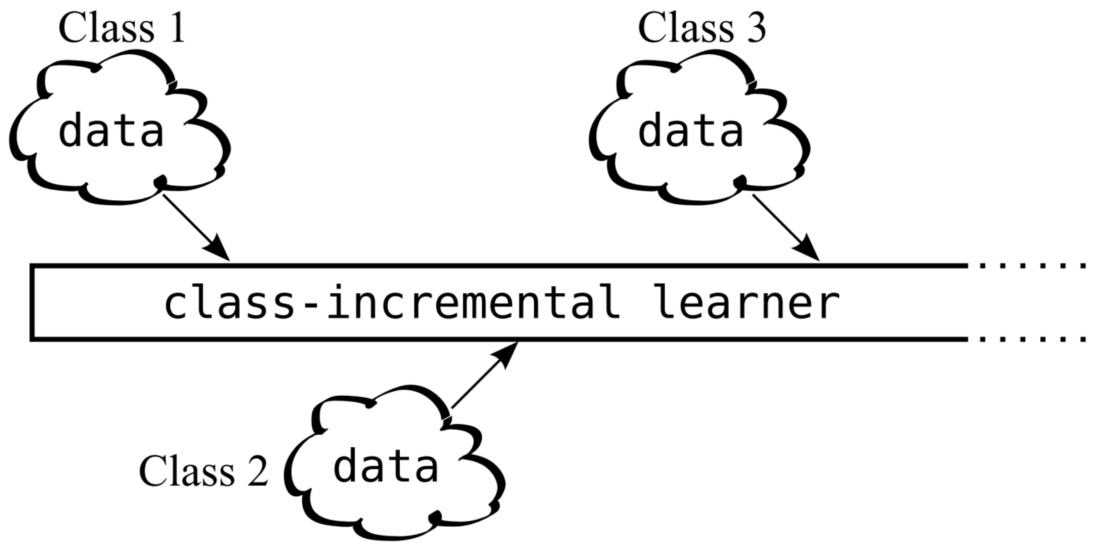
Rehearsal (Replay) Methods

- **Basic idea:**
  - ✓ Store samples in raw format or generate pseudo-samples
  - ✓ The samples are reused as model inputs for rehearsal, or to constrain the optimization of the new task loss

- **Three main solutions**:
  - ✓ **Data Rehearsal**: Retrain on a limited subset of stored samples while training on new tasks
  - ✓ **Pseudo Rehearsal**: Generate pseudo-samples with a generative model
  - ✓ **Constrained Optimization**: Constrain new task updates using stored samples

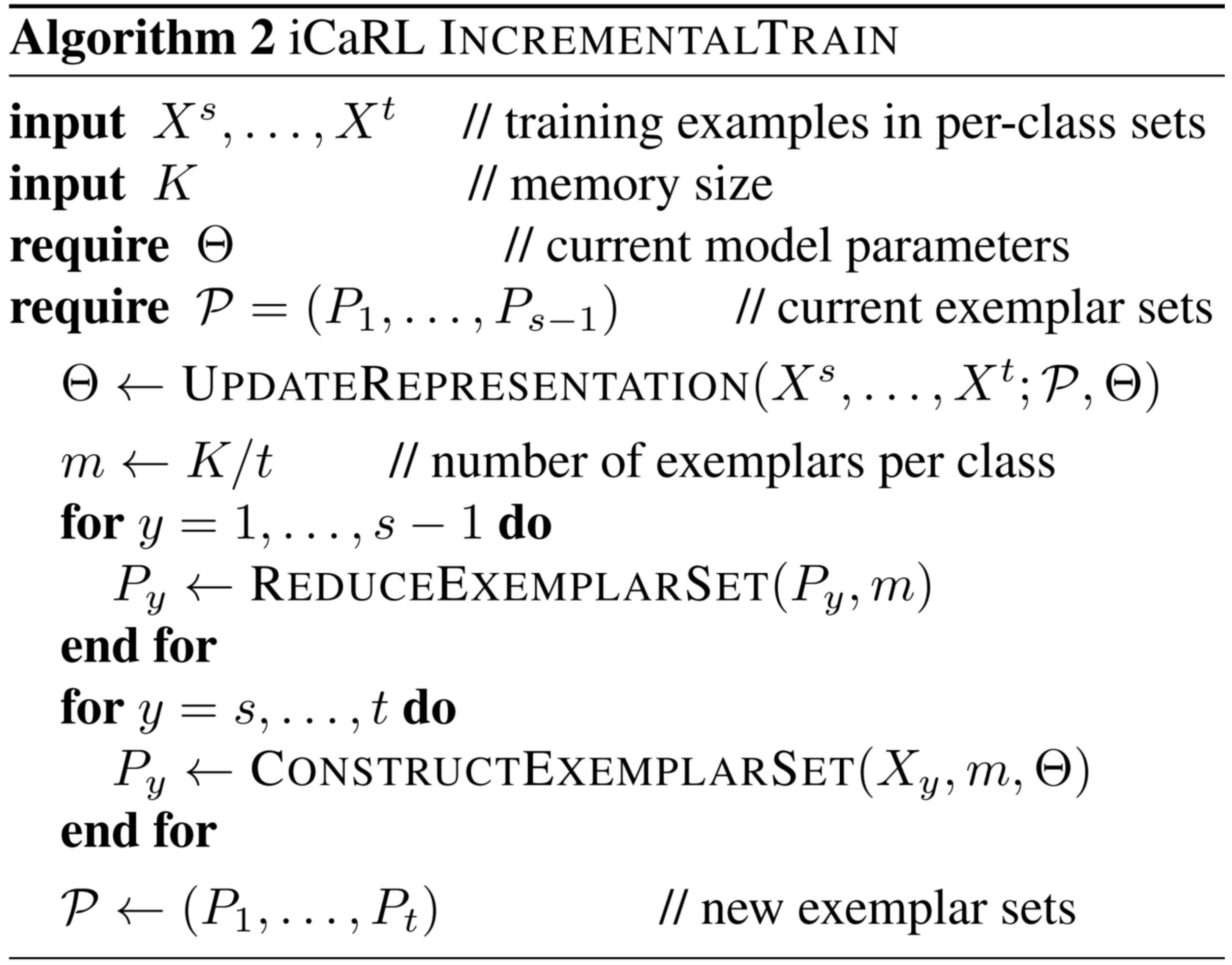
iCaRL — Data Rehearsal

- **Scenario:** Class incremental learning
- **Settings:**
  - ✓ Different classes occur at different time
  - ✓ A single-head classifier
  - ✓ Limited computational resources and memory
- **Three main components:**
  1. Classification by **nearest-mean-of-exemplars**
  2. Prioritized exemplar selection based on **herding**
  3. Representation learning using **knowledge distillation** and prototype rehearsal



Rebuffi S A, Kolesnikov A, Sperl G, Lampert C H. iCaRL: Incremental classifier and representation learning. CVPR, 2017.

Pseudo code



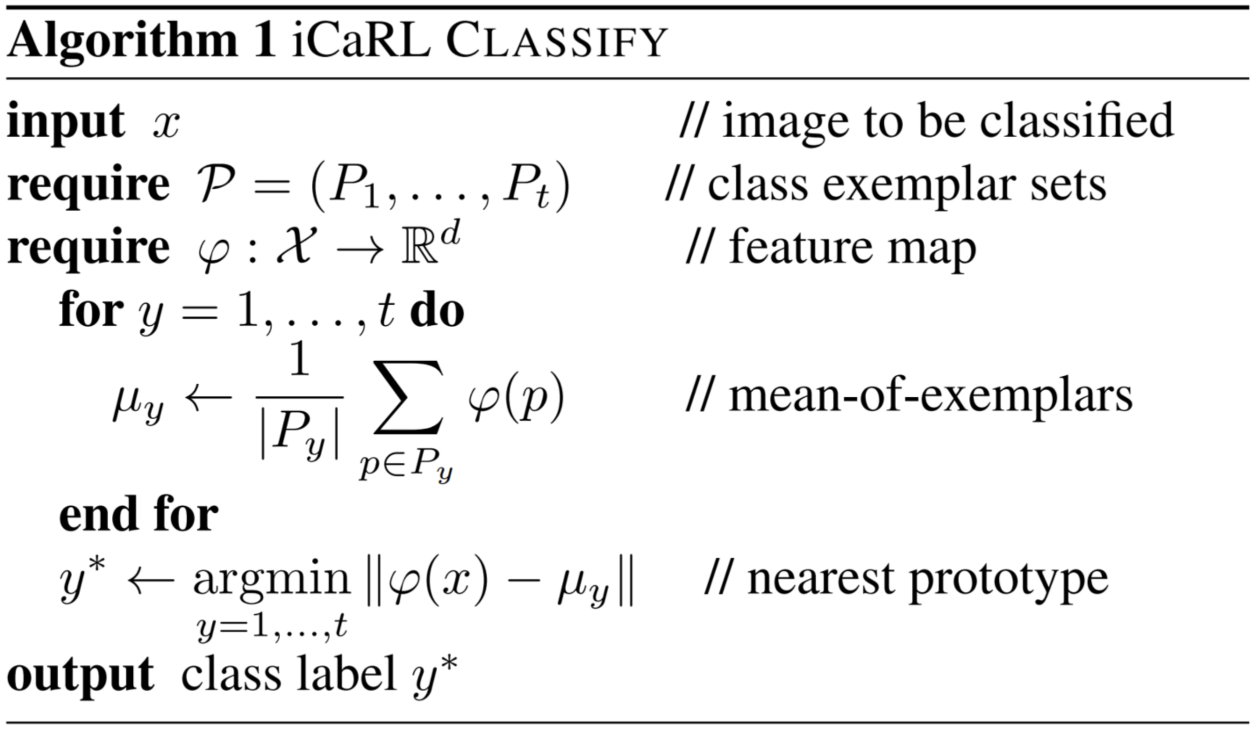
Algorithm 2 iCaRL IncrementalTrain

```
input $X^s, \dots, X^t$                // training examples in per-class sets
input $K$                              // memory size
require $\Theta$                       // current model parameters
require $\mathcal{P} = (P_1, \dots, P_{s-1})$   // current exemplar sets
  $\Theta \leftarrow$ UpdateRepresentation($X^s, \dots, X^t; \mathcal{P}, \Theta$)
  $m \leftarrow K/t$                   // number of exemplars per class
  for $y = 1, \dots, s-1$ do
    $P_y \leftarrow$ ReduceExemplarSet($P_y, m$)
  end for
  for $y = s, \dots, t$ do
    $P_y \leftarrow$ ConstructExemplarSet($X_y, m, \Theta$)
  end for
  $\mathcal{P} \leftarrow (P_1, \dots, P_t)$        // new exemplar sets
```

Nearest-Mean-of-Exemplars (NME) Classification

- **Limitation of linear classifier**
  - ✓ Whenever the feature map φ changes, all weights $\omega_1, \dots, \omega_t$ must be updated
  - ✓ The outputs change uncontrollably

- **Nearest-Mean-of-Exemplars (NME)**
  - ✓ No weight vectors
  - ✓ Class-prototypes automatically change with φ



Algorithm 1 iCaRL CLASSIFY

```
input $x$                                                // image to be classified
require $\mathcal{P} = (P_1, \dots, P_t)$                // class exemplar sets
require $\varphi : \mathcal{X} \to \mathbb{R}^d$         // feature map
  for $y = 1, \dots, t$ do
    $\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$   // mean-of-exemplars
  end for
  $y^* \leftarrow \operatorname*{argmin}_{y=1,\dots,t} \|\varphi(x) - \mu_y\|$   // nearest prototype
output class label $y^*$
```

Representation Learning

Steps:

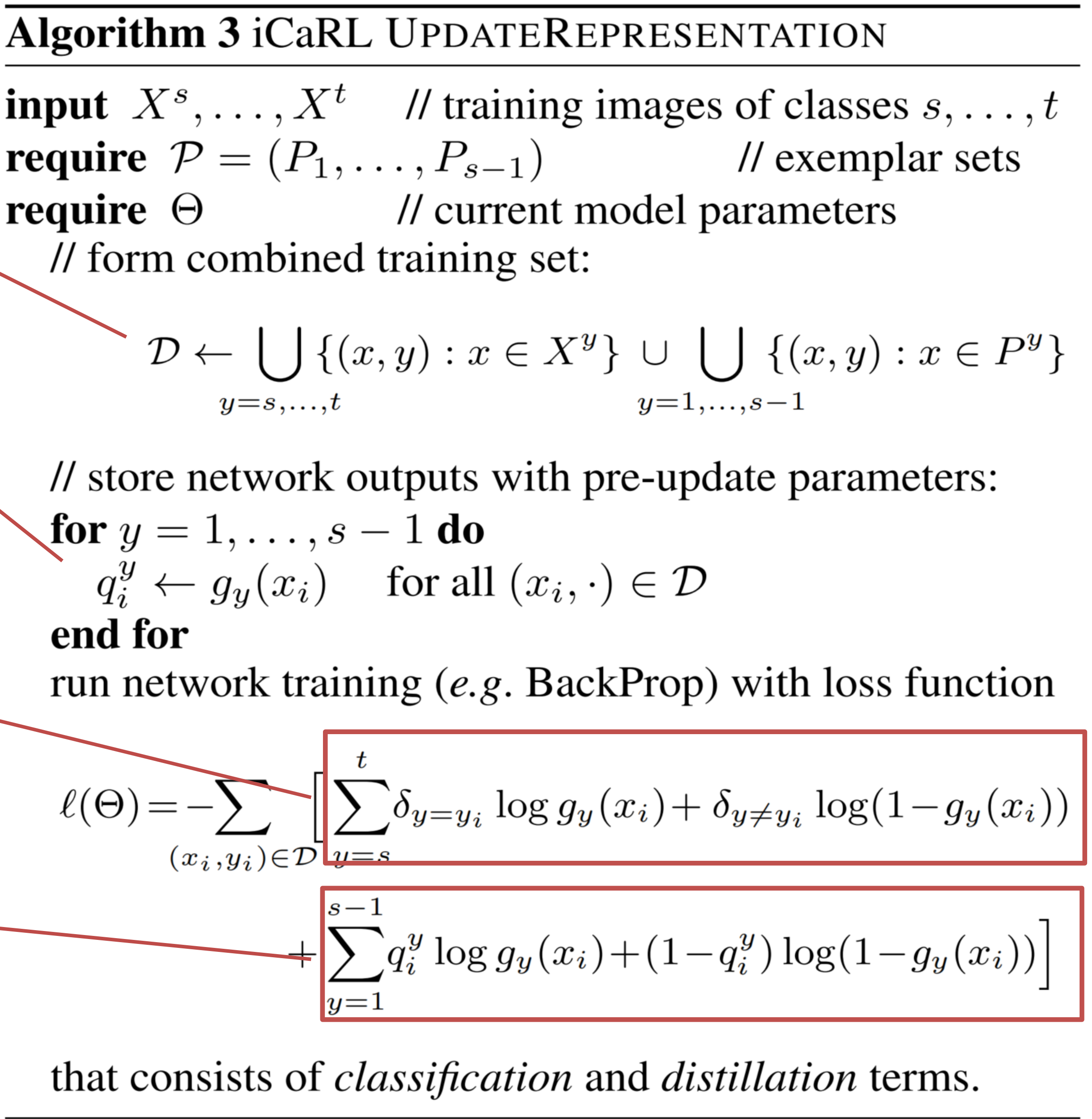
1. Construct an augmented training set consisting of new training examples and stored exemplars

2. Store network output for all previous classes

3. Construct loss function:

   ✓ **Classification loss** encourages the network to output the correct class for new classes

   ✓ **Distillation loss** encourages the network to reproduce the scores stored in the previous step



Algorithm 3 iCaRL UPDATEREPRESENTATION

```
input $X^s, \dots, X^t$   // training images of classes $s, \dots, t$
require $\mathcal{P} = (P_1, \dots, P_{s-1})$   // exemplar sets
require $\Theta$   // current model parameters
  // form combined training set:
  $\mathcal{D} \leftarrow \bigcup_{y=s,\dots,t} \{(x, y) : x \in X^y\} \cup \bigcup_{y=1,\dots,s-1} \{(x, y) : x \in P^y\}$
  // store network outputs with pre-update parameters:
  for $y = 1, \dots, s-1$ do
    $q_i^y \leftarrow g_y(x_i)$   for all $(x_i, \cdot) \in \mathcal{D}$
  end for
  run network training (e.g. BackProp) with loss function
```

$$\ell(\Theta) = -\sum_{(x_i,y_i)\in\mathcal{D}} \Big[\boxed{\sum_{y=s}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y\neq y_i} \log(1-g_y(x_i))} \\ \boxed{+ \sum_{y=1}^{s-1} q_i^y \log g_y(x_i) + (1-q_i^y) \log(1-g_y(x_i))} \Big]$$

that consists of *classification* and *distillation* terms.

Exemplar Management

- **Steps:**
  1. Construct exemplar set
  2. Reduce exemplar set
- **Basic idea:**
  - ✓ The initial exemplar set should approximate the class mean vector
  - ✓ Remove exemplars at any time during the algorithm's runtime without violating this property (challenging) : Remove elements in fixed order starting at the end

Algorithm 4 iCaRL ConstructExemplarSet

```
input  image set $X = \{x_1, \dots, x_n\}$ of class $y$
input  $m$ target number of exemplars
require  current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$
  $\mu \leftarrow \frac{1}{n}\sum_{x \in X} \varphi(x)$   // current class mean
  for $k = 1, \dots, m$ do
    $p_k \leftarrow \underset{x \in X}{\operatorname{argmin}} \left\| \mu - \frac{1}{k}[\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right\|$
  end for
  $P \leftarrow (p_1, \dots, p_m)$
output  exemplar set $P$
```

Algorithm 5 iCaRL ReduceExemplarSet

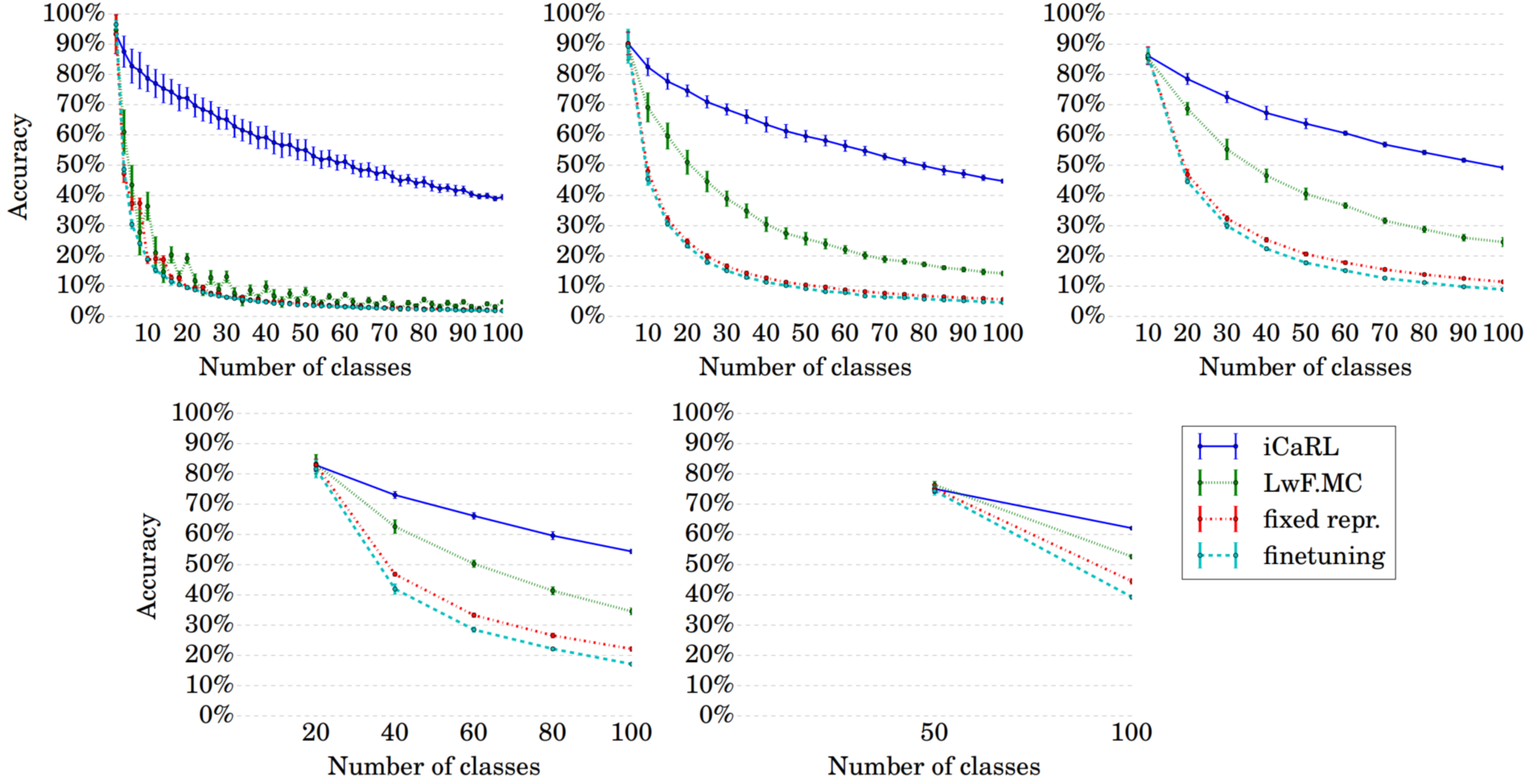
```
input  $m$                          // target number of exemplars
input  $P = (p_1, \dots, p_{|P|})$   // current exemplar set
  $P \leftarrow (p_1, \dots, p_m)$    // i.e. keep only first $m$
output  exemplar set $P$
```

Experiments

- **Dataset:**
  - ✓ CIFAR-100: Train all 100 classes in batches of 2/5/10/20/50 classes at a time
  - ✓ ImageNet ILSVRC 2012
    - iILSVRC-small: A subset of 100 classes, trained in batches of 10
    - iILSVRC-full: All 1000 classes, trained in batches of 100
- **Metric:** Accuracy on classes that have already been trained

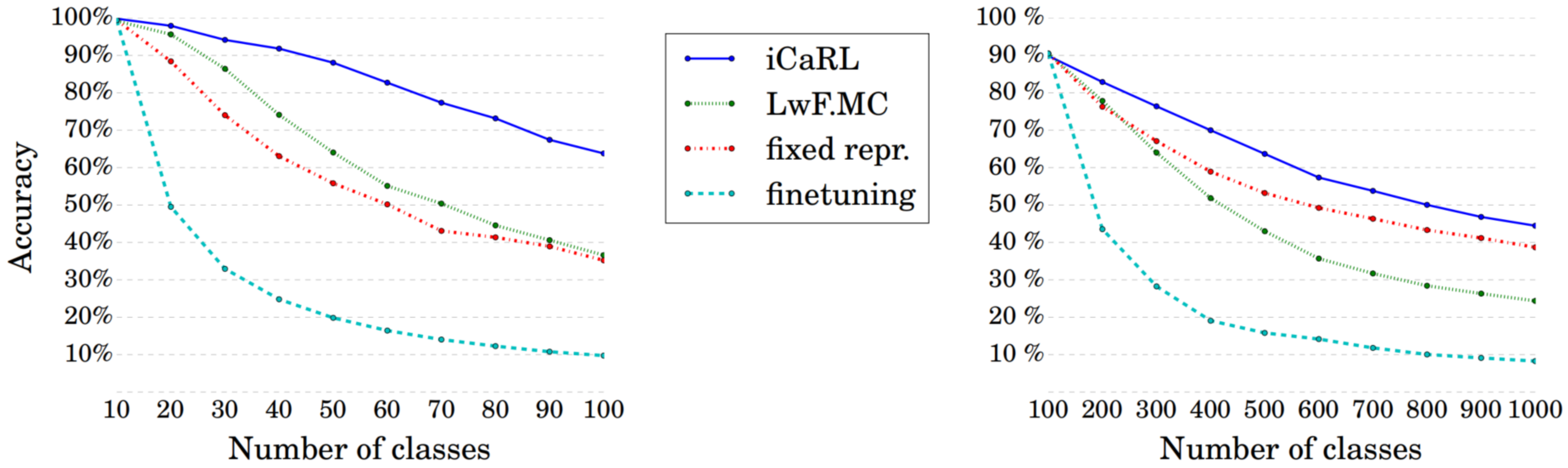
  ImageNet ILSVRC 2012 uses the ***top-5*** **accuracy**
- **Baselines:**
  - ✓ **Finetuning**: Finetune the model using new incoming classes
  - ✓ **Fixed representation**: Only train the weight vectors of new classes
  - ✓ **LwF.MC**: Apply LWF to class incremental learning

Results: iCIFAR-100



(a) Multi-class accuracy (averages and standard deviations over 10 repeats) on iCIFAR-100 with 2 (top left), 5 (top middle), 10 (top right), 20 (bottom left) or 50 (bottom right) classes per batch.

Results: iILSVRC-full



(b) Top-5 accuracy on iILSVRC-small (top) and iILSVRC-full (bottom).

Effect of Memory Budgets *K*

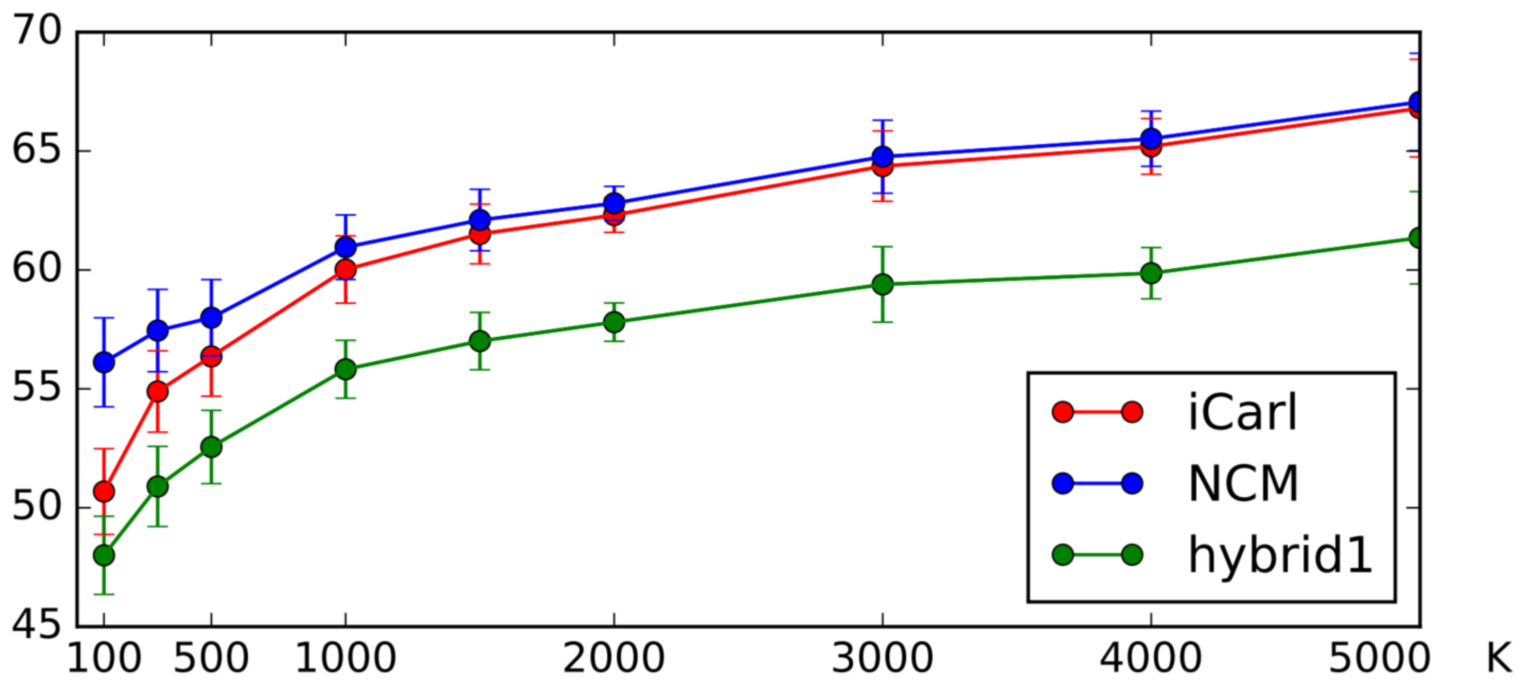


Figure 4: Average incremental accuracy on iCIFAR-100 with 10 classes per batch for different memory budgets K.

- **NCM** (Nearest-class-mean): Exact class mean instead of means-of-exemplars
- **Hybrid 1**: Linear classifier instead of NME classifier

Deep Generative Replay (DGR) — Pseudo Rehearsal

Cooperative dual model architecture: Deep generative model (generator) + task solver (solver)

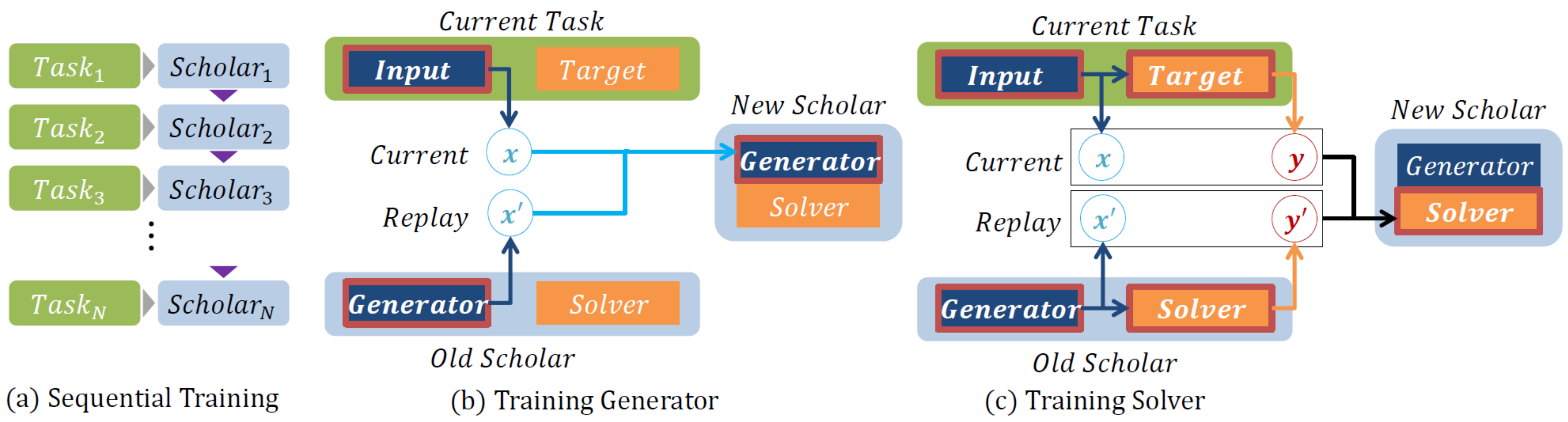
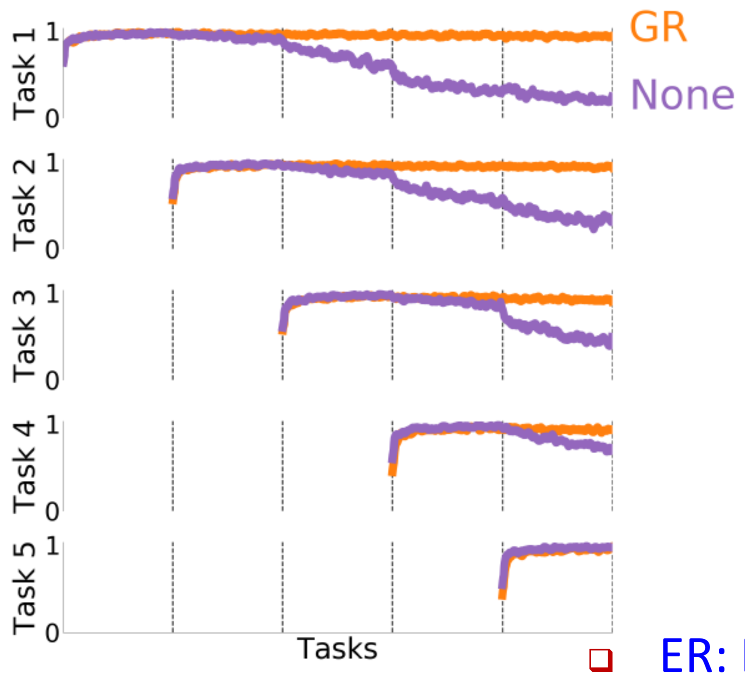


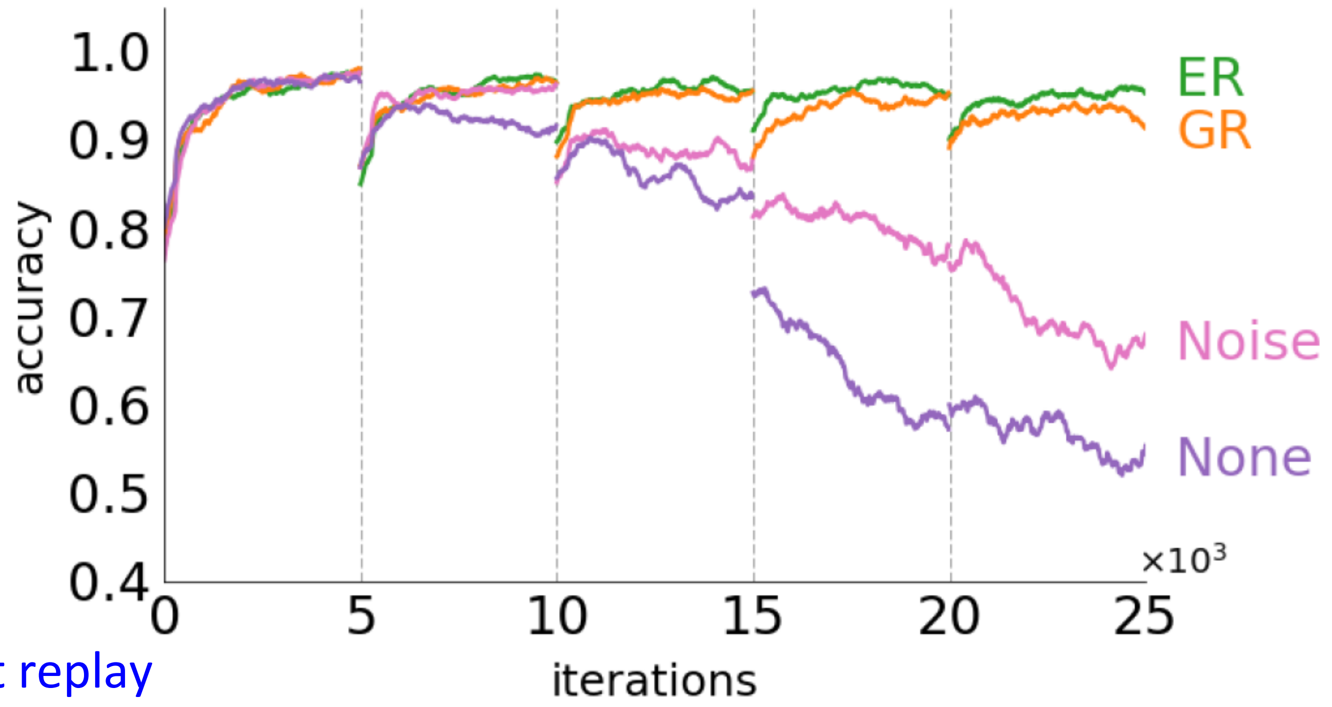
Figure 1: Sequential training of scholar models. (a) Training a sequence of scholar models is equivalent to continuous training of a single scholar while referring to its most recent copy. (b) A new generator is trained to mimic a mixed data distribution of real samples x and replayed inputs x' from previous generator. (c) A new solver learns from real input-target pairs $(\boldsymbol{x}, \boldsymbol{y})$ and replayed input-target pairs $(\boldsymbol{x}', \boldsymbol{y}')$, where replayed response $\boldsymbol{y}'$ is obtained by feeding generated inputs into previous solver.

H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," NeurIPS 2017.

DGR: Experiment Results



(a)

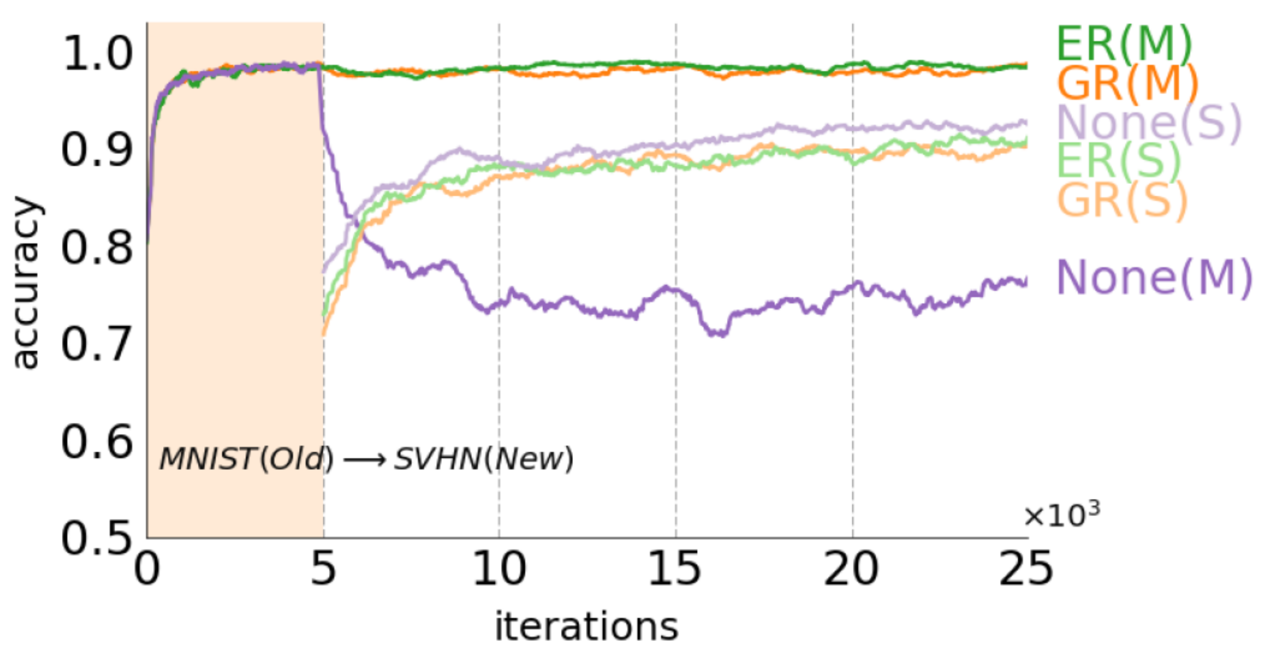


(b)

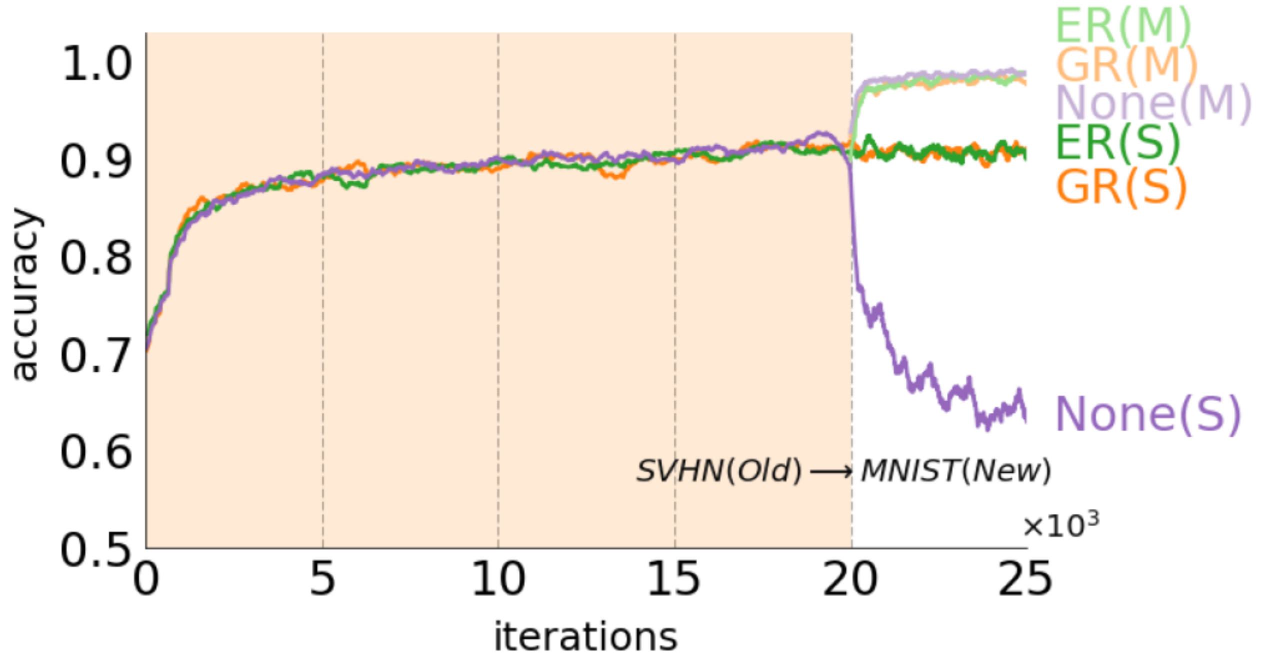
- ER: Exact replay
- Noise: The generated samples do not resemble the real distribution
- None: No samples are stored

Figure 2: Results on MNIST pixel permutation tasks. (a) Test performances on each task during sequential training. Performances for previous tasks dropped without replaying real or meaningful fake data. (b) Average test accuracy on learnt tasks. Higher accuracy is achieved when the replayed inputs better resembled real data.

DGR: Experiment Results



(a) MNIST to SVHN



(b) SVHN to MNIST

Figure 3: Accuracy on classifying samples from two different domains. (a) The models are trained on MNIST then on SVHN dataset or (b) vice versa. When the previous data are recalled by generative replay (orange), knowledge of the first domain is retained as if the real inputs with predicted responses are replayed (green). Sequential training on the solver alone incurs forgetting on the former domain, thereby resulting in low average performance (violet).

DGR: Experiment Results

❖ Samples from trained generator

MNIST to SVHN experiment after training on SVHN dataset for different iterations

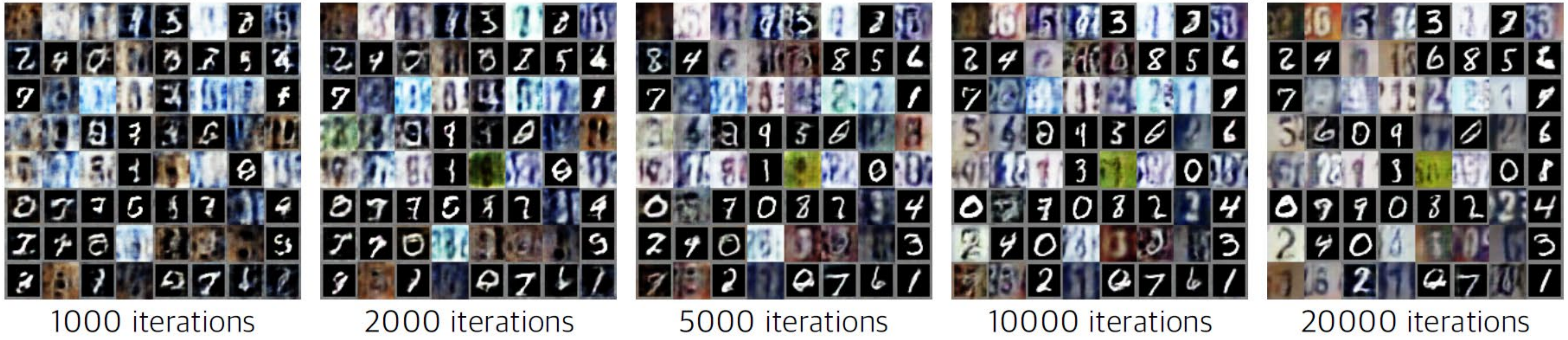


Figure 4: Samples from trained generator in MNIST to SVHN experiment after training on SVHN dataset for 1000, 2000, 5000, 10000, and 20000 iterations. The samples are diverted into ones that mimic either SVHN or MNIST input images.

Gradient Episodic Memory (GEM) — Constrained Optimization

- **Basic idea:** Constrain the gradient to improve the previous tasks
- **Settings:**
  - ✓ A fix total memory $\mathcal{M}$
  - ✓ M_t stores examples in Task t
- **Implementation:**
  - ❑ Loss on memories from the *k*-th task $$\ell(f_\theta, \mathcal{M}_k) = \frac{1}{|\mathcal{M}_k|} \sum_{(x_i,k,y_i)\in\mathcal{M}_k} \ell(f_\theta(x_i,k), y_i).$$
  - ❑ Minimizing the above loss results in overfitting to examples in M_k , so a better optimization problem is

$$\begin{aligned} \text{minimize}_\theta \quad & \ell(f_\theta(x,t), y) \\ \text{subject to} \quad & \ell(f_\theta, \mathcal{M}_k) \le \ell(\boxed{f_\theta^{t-1}}, \mathcal{M}_k) \text{ for all } k < t, \end{aligned}$$

Model after learning task $t-1$

D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," NeurIPS 2017.

GEM — Constrained Optimization

Implementation:

- To avoid storing the old model, computing the angle between loss gradient vector on new task data and M_k

$$\langle g, g_k \rangle := \left\langle \frac{\partial \ell(f_\theta(x,t), y)}{\partial \theta}, \frac{\partial \ell(f_\theta, \mathcal{M}_k)}{\partial \theta} \right\rangle \geq 0, \text{ for all } k < t.$$

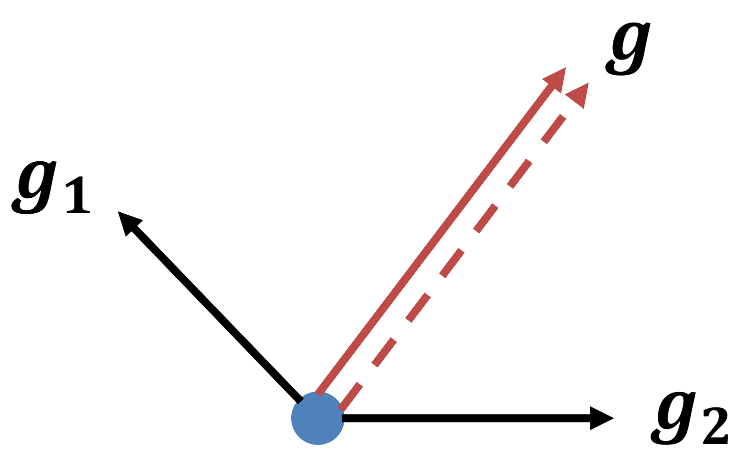
The proposed parameter update g is unlikely to increase the loss on previous tasks.

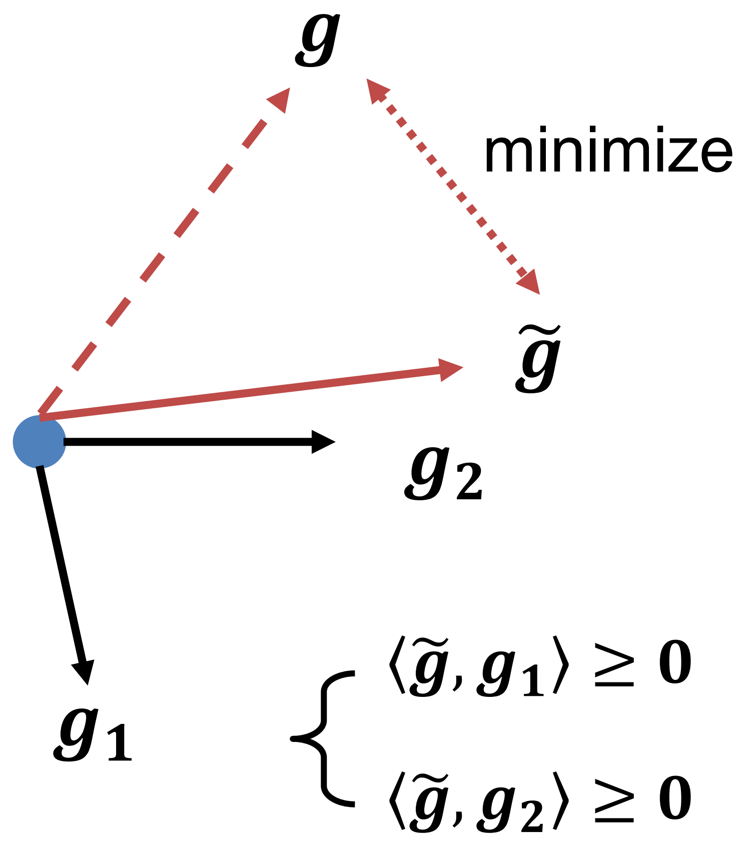
- If violations occur, project gradient g to the closest gradient $\tilde{g}$ satisfying all the constraints above:

$$\begin{aligned} \text{minimize}_{\tilde{g}} \frac{1}{2} \quad & \|g - \tilde{g}\|_2^2 \\ \text{subject to} \quad & \langle \tilde{g}, g_k \rangle \geq 0 \text{ for all } k < t. \end{aligned}$$

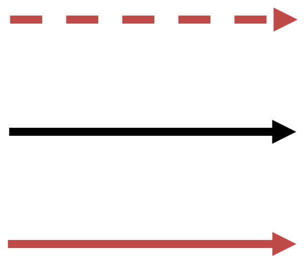
D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," NeurIPS 2017.

GEM: Illustration





$$\begin{cases} \langle \tilde{\boldsymbol{g}}, \boldsymbol{g}_1 \rangle \geq \mathbf{0} \\ \langle \tilde{\boldsymbol{g}}, \boldsymbol{g}_2 \rangle \geq \mathbf{0} \end{cases}$$



https://www.youtube.com/watch?v=7qT5P9KJnWo&list=PLJV_el3uVTsOK_ZK5L0Iv_EQoL1JefRL4&index=25

GEM: Experiments

Dataset:

- ✓ MNIST Permutations: A fixed permutation of pixels
- ✓ MNIST Rotations: Digits rotated by a fixed angle between 0 and 180
- ✓ CIFAR100: 100 classes

Total number of tasks $T = 20$

Performance Metrics:

$$\textbf{Average Accuracy:} \quad \text{ACC} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}$$

$R_{i,j}$ Accuracy of testing on task t_j data using the model of task t_i

$$\textbf{Backward Transfer:} \quad \text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$$

$$\textbf{Forward Transfer:} \quad \text{FWT} = \frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - \boxed{\bar{b}_i}.$$

Random accuracy

GEM: Experiment Results

Baselines:

- **Single**: A single model
- **Independent**: One model per task, with T times fewer hidden units than "single"
- **Multimodal**: A dedicated input layer per task
- **EWC**
- **iCaRL**

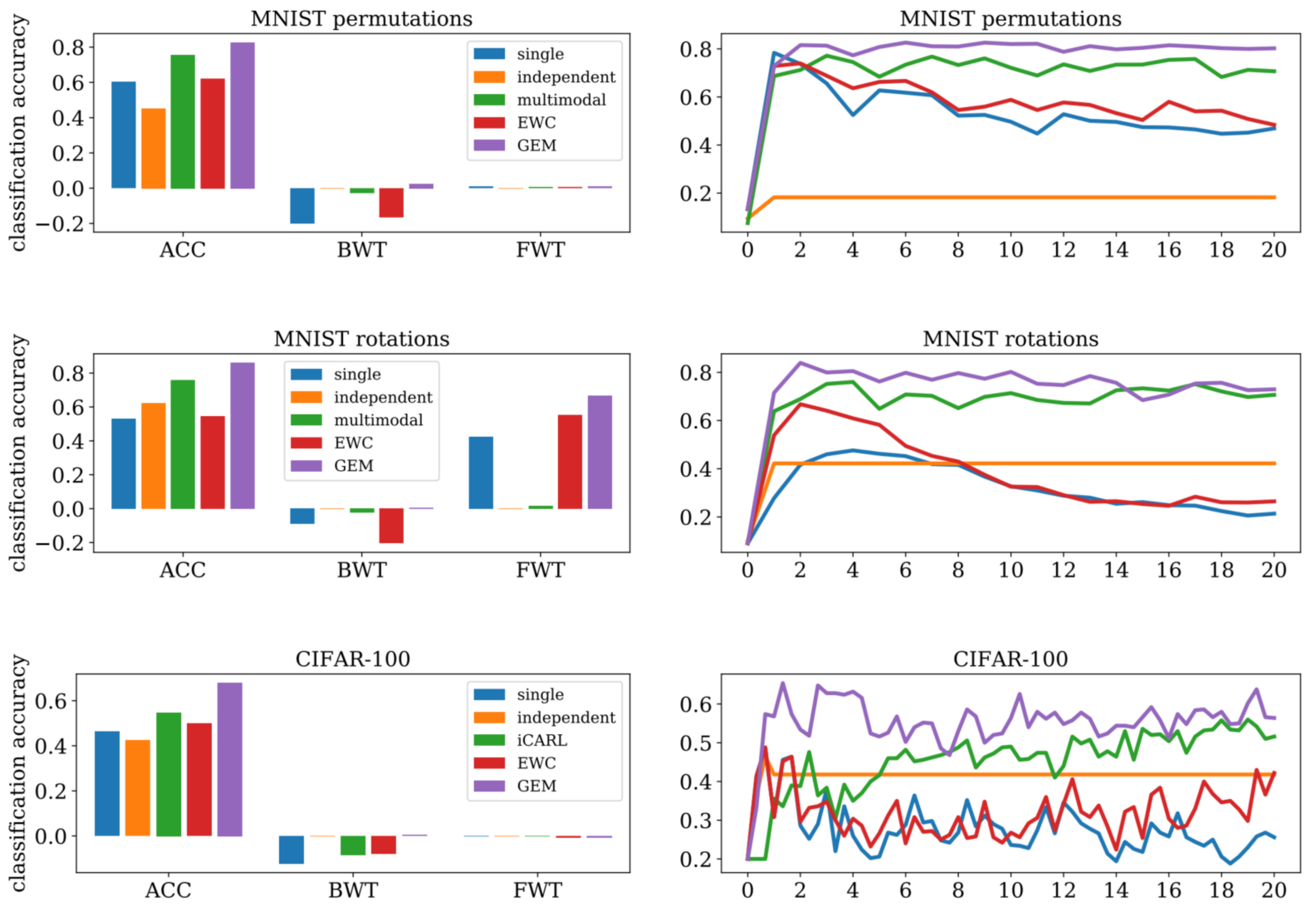
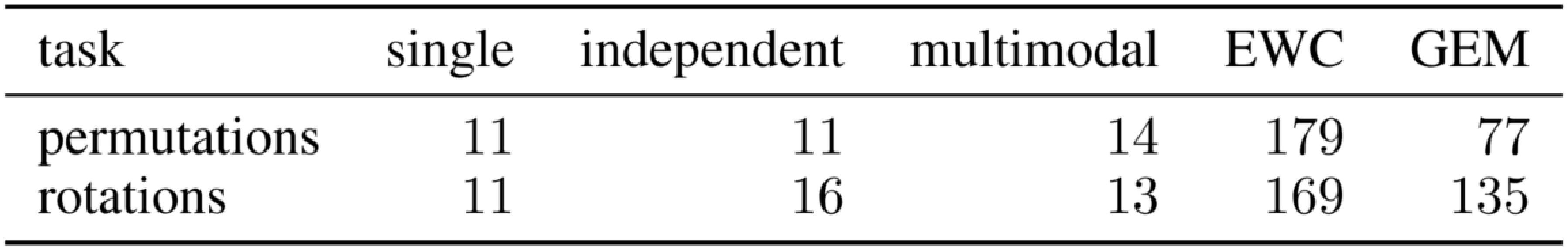


Figure 1: Left: ACC, BWT, and FWT for all datasets and methods. Right: evolution of the test accuracy at the first task, as more tasks are learned.

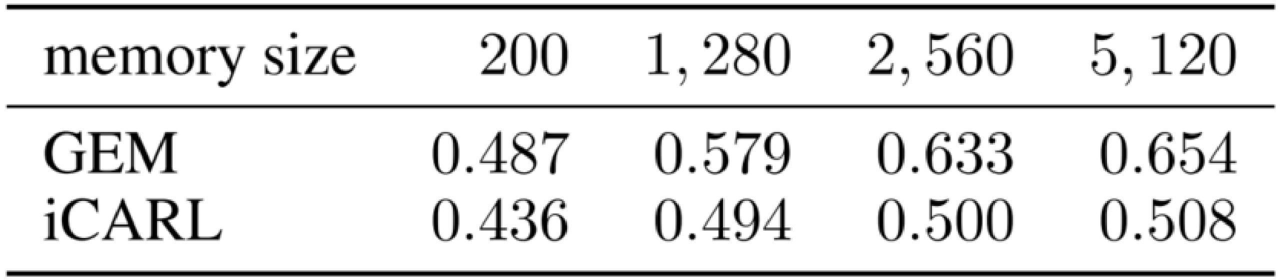
GEM: Experiment Results

Table 1: CPU Training time (s) of MNIST experiments for all methods.



task	single	independent	multimodal	EWC	GEM
permutations	11	11	14	179	77
rotations	11	16	13	169	135

Table 2: ACC as a function of the episodic memory size for GEM and iCARL, on CIFAR100.



memory size	200	1,280	2,560	5,120
GEM	0.487	0.579	0.633	0.654
iCARL	0.436	0.494	0.500	0.508

Averaged GEM (A-GEM)

- **An improved version of GEM**
- **Basic idea:** Project on a direction estimated by some randomly selected samples from a previous task data buffer
  - Current task t

$$\text{minimize}_{\theta} \quad \ell(f_\theta, \mathcal{D}_t) \quad \text{s.t.} \quad \ell(f_\theta, \mathcal{M}) \leq \ell(f_\theta^{t-1}, \mathcal{M}) \qquad \text{where} \boxed{\mathcal{M} = \cup_{k<t}\mathcal{M}_k}$$

  - The corresponding optimization problem reduces to:

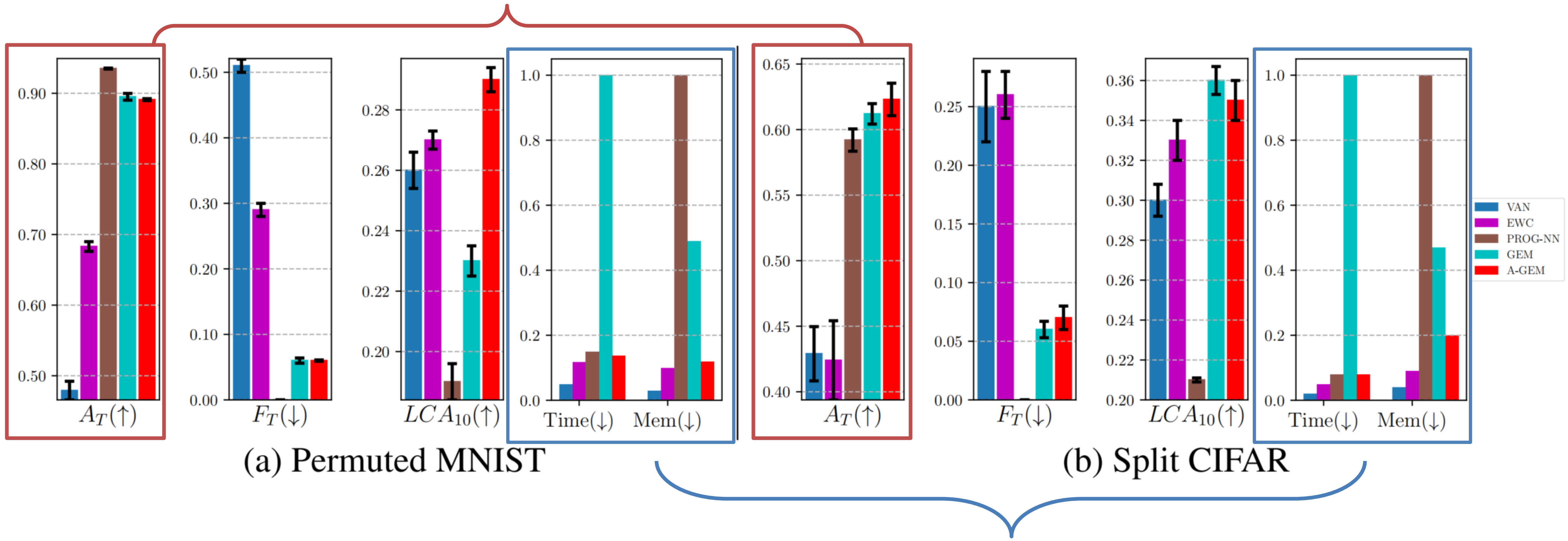
$$\text{minimize}_{\tilde{g}} \quad \frac{1}{2}||g - \tilde{g}||_2^2 \quad \text{s.t.} \quad \tilde{g}^\top \boxed{g_{ref}} \geq 0$$

The gradient computed using a batch randomly sampled from $(\mathbf{x}_{ref}, y_{ref}) \sim \mathcal{M}$

A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, “Efficient lifelong learning with A-GEM,” ICLR 2018.

A-GEM: Experiment Results

A-GEM and GEM perform comparably in terms of average accuracy



(a) Permuted MNIST

(b) Split CIFAR

A-GEM has much lower time (about 100 times faster) and memory cost (about 10 times lower)

Outline

Regularization-Based Methods

- **Basic idea:**
  - ✓ Add an extra regularization item to the loss function

- **Two main solutions:**
  1. **Prior-Focused Methods**: Changes to important parameters are penalized during the training of later tasks
  2. **Data-Focused Methods**: Knowledge distillation from a previous model to the model being trained on the new data

Elastic Weight Consolidation (EWC) — Prior-Focused Method

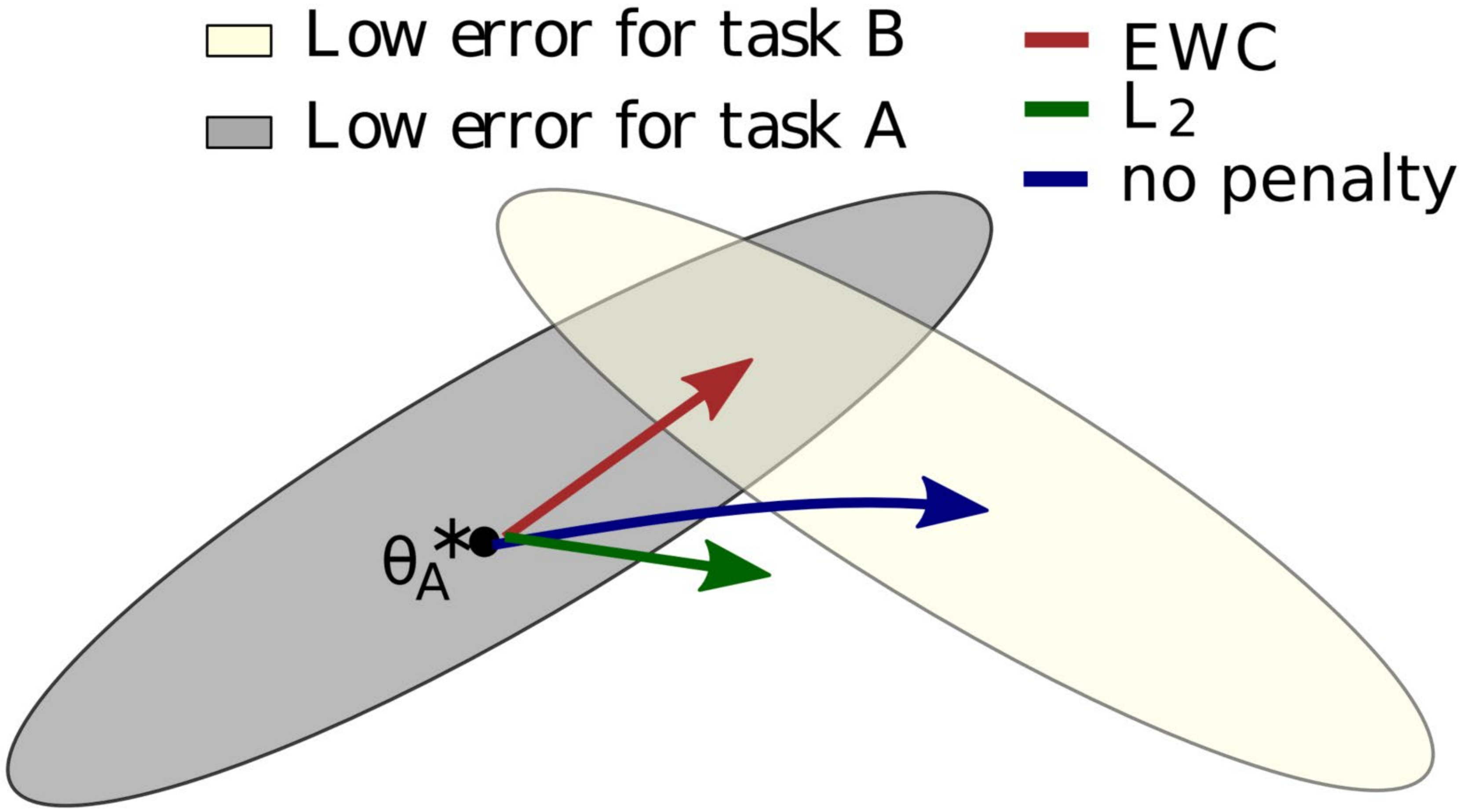
Basic idea:

- ✓ Determine the parameters that are important for previous tasks
- ✓ Constrain these important parameters to stay close to their old values

$$\mathcal{L}(\theta) = \overbrace{\boxed{\mathcal{L}_B(\theta)}}^{\text{Loss of current task}} + \sum_i \frac{\lambda}{2} \overbrace{\boxed{F_i}}^{\substack{\text{Importance of each parameter} \\ \text{(Fisher information matrix)}}} (\underbrace{\boxed{\theta_i}}_{\substack{\text{Parameters of} \\ \text{current task}}} - \underbrace{\boxed{\theta^*_{A,i}}}_{\substack{\text{Parameters learned} \\ \text{from previous tasks}}})^2,$$

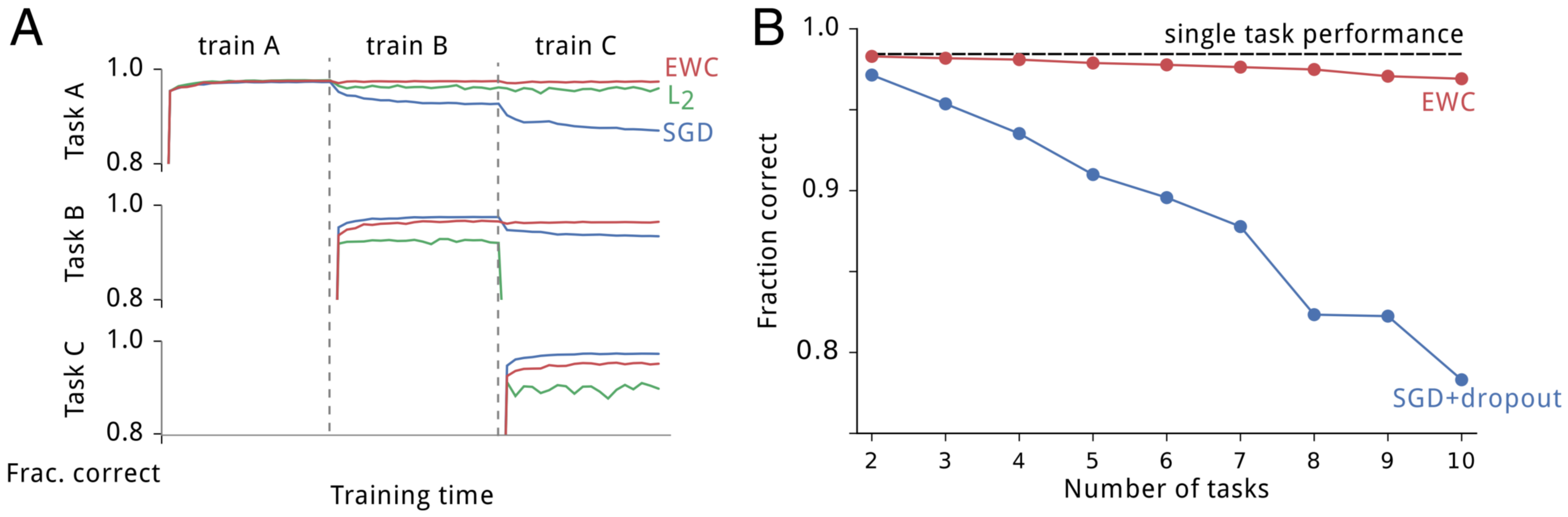
Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu A A, et al. Overcoming catastrophic forgetting in neural networks. PNAS, 2017, 114(13): 3521−3526

EWC: Illustration



Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu A A, et al. Overcoming catastrophic forgetting in neural networks. PNAS, 2017, 114(13): 3521−3526

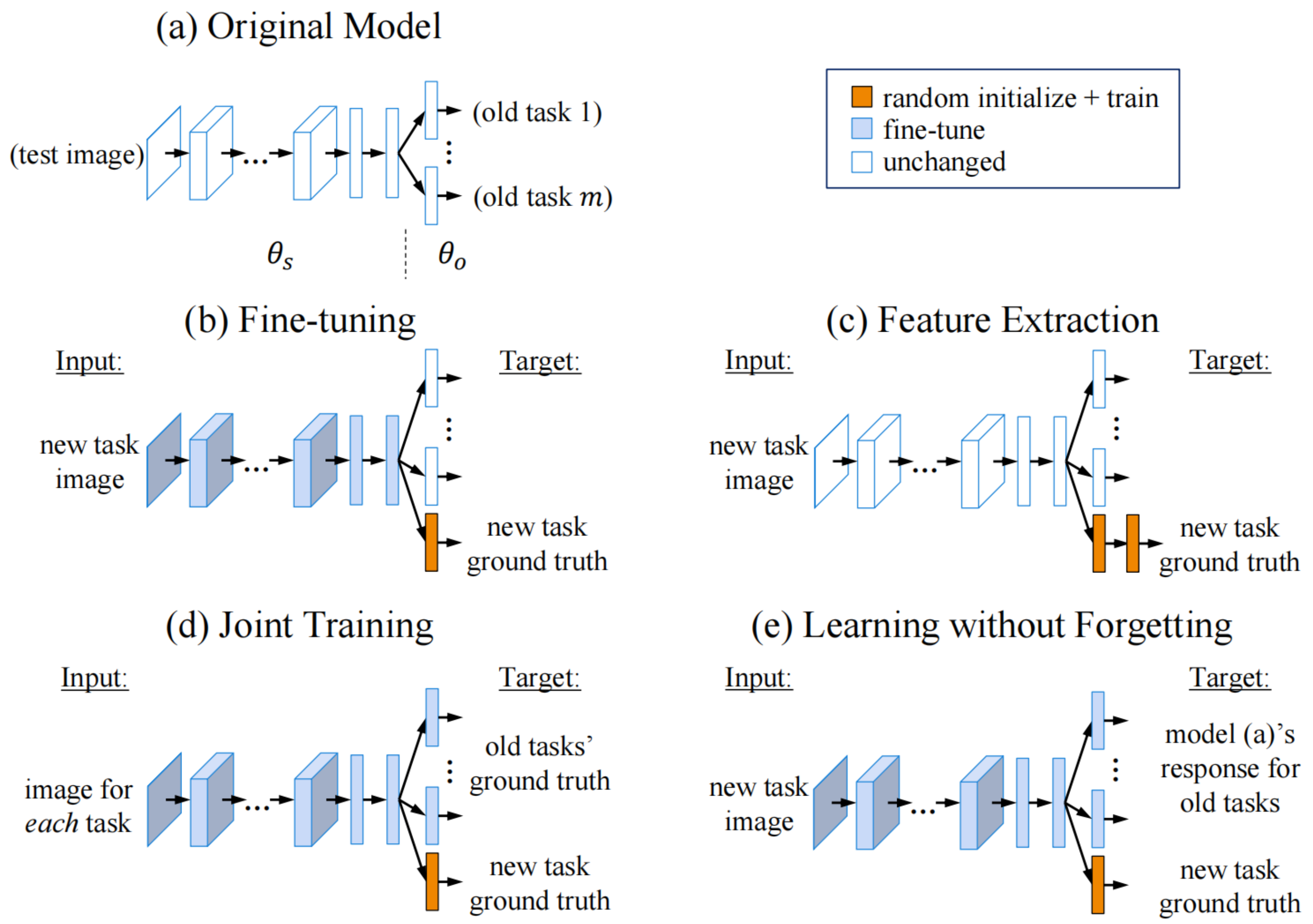
EWC: Experiment Results



Limitation: EWC can alleviate catastrophic forgetting; however, important parameters will slowly deviate from their optimal parameters learned from previous tasks.

Learning Without Forgetting (LWF) — Data-Focused Method

Basic idea: Minimize the KL divergence between the probability distributions of the old and new model outputs



Z. Li and D. Hoiem, "Learning without forgetting," ECCV 2016, pp. 614–629.

LWF: Loss Function

1. Loss on new data:

$$\mathcal{L}_{new}(\mathbf{y}_n, \hat{\mathbf{y}}_n) = -\mathbf{y}_n \cdot \log \hat{\mathbf{y}}_n$$

True label

Prediction probability from the updated model

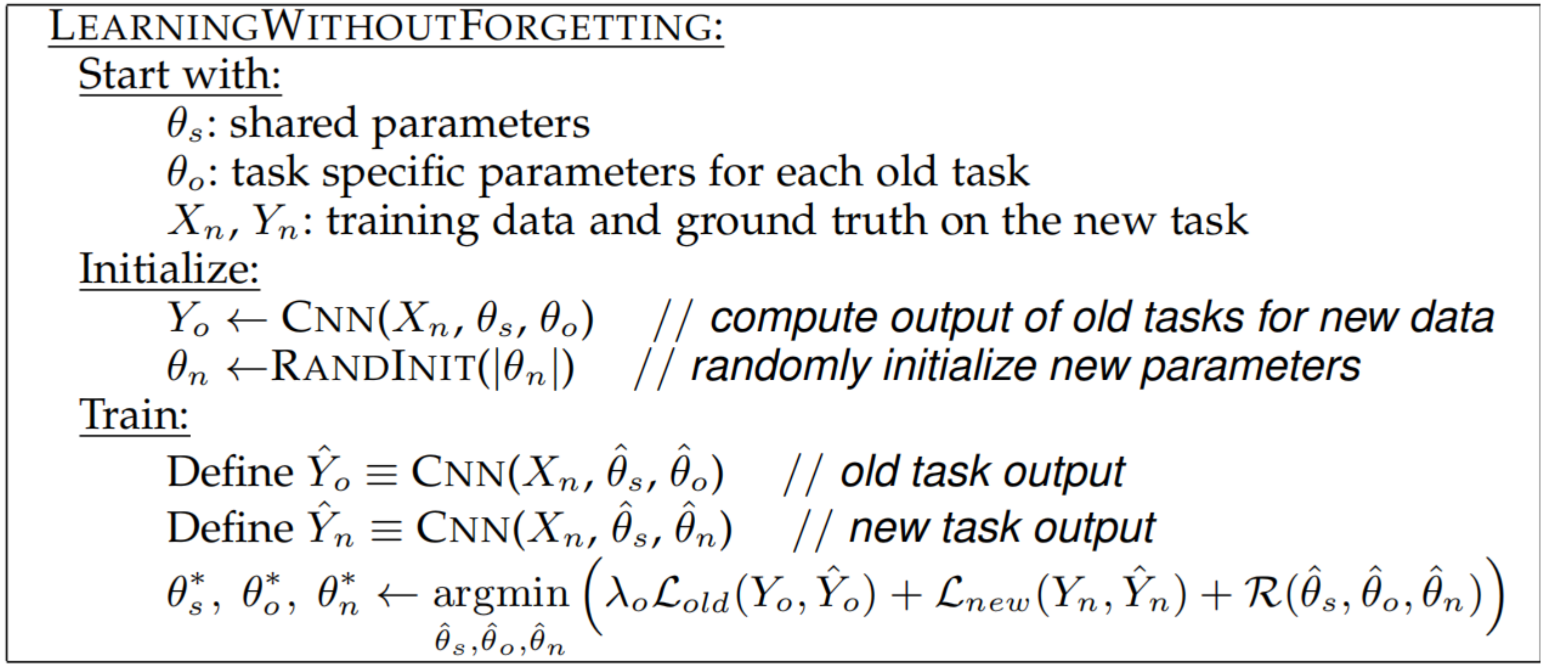
2. Distillation loss:

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o)$$

$$= -\sum_{i=1}^{l} y'^{(i)}_o \log \hat{y}'^{(i)}_o$$

Prediction probability on the old task model

Prediction probability from the updated model

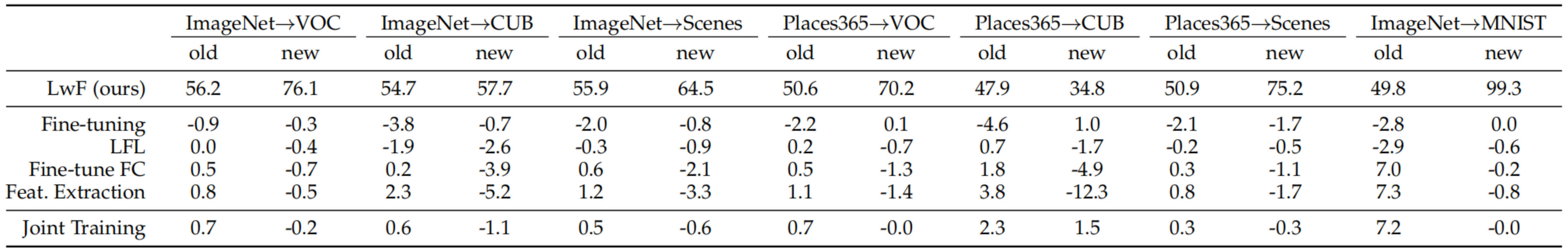


```
LEARNINGWITHOUTFORGETTING:
  Start with:
      $\theta_s$: shared parameters
      $\theta_o$: task specific parameters for each old task
      $X_n, Y_n$: training data and ground truth on the new task
  Initialize:
      $Y_o \leftarrow$ CNN($X_n, \theta_s, \theta_o$)    // compute output of old tasks for new data
      $\theta_n \leftarrow$ RANDINIT($|\theta_n|$)    // randomly initialize new parameters
  Train:
      Define $\hat{Y}_o \equiv$ CNN($X_n, \hat{\theta}_s, \hat{\theta}_o$)    // old task output
      Define $\hat{Y}_n \equiv$ CNN($X_n, \hat{\theta}_s, \hat{\theta}_n$)    // new task output
      $\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n)\right)$
```

Z. Li and D. Hoiem, "Learning without forgetting," ECCV 2016, pp. 614–629.

LWF: Experimental Results

(a) Using AlexNet structure (validation performance for ImageNet/Places365/VOC)



	ImageNet→VOC		ImageNet→CUB		ImageNet→Scenes		Places365→VOC		Places365→CUB		Places365→Scenes		ImageNet→MNIST	
	old	new	old	new	old	new	old	new	old	new	old	new	old	new
LwF (ours)	56.2	76.1	54.7	57.7	55.9	64.5	50.6	70.2	47.9	34.8	50.9	75.2	49.8	99.3
Fine-tuning	-0.9	-0.3	-3.8	-0.7	-2.0	-0.8	-2.2	0.1	-4.6	1.0	-2.1	-1.7	-2.8	0.0
LFL	0.0	-0.4	-1.9	-2.6	-0.3	-0.9	0.2	-0.7	0.7	-1.7	-0.2	-0.5	-2.9	-0.6
Fine-tune FC	0.5	-0.7	0.2	-3.9	0.6	-2.1	0.5	-1.3	1.8	-4.9	0.3	-1.1	7.0	-0.2
Feat. Extraction	0.8	-0.5	2.3	-5.2	1.2	-3.3	1.1	-1.4	3.8	-12.3	0.8	-1.7	7.3	-0.8
Joint Training	0.7	-0.2	0.6	-1.1	0.5	-0.6	0.7	-0.0	2.3	1.5	0.3	-0.3	7.2	-0.0

Difference between joint training and LWF:

- Joint training requires the data and labels from the old task
- LWF only uses the new task data and prediction probability on the old task model

Comparison: Confusion Matrices

- **iCaRL**

  Roughly uniform over all classes

- **LwF.MC**

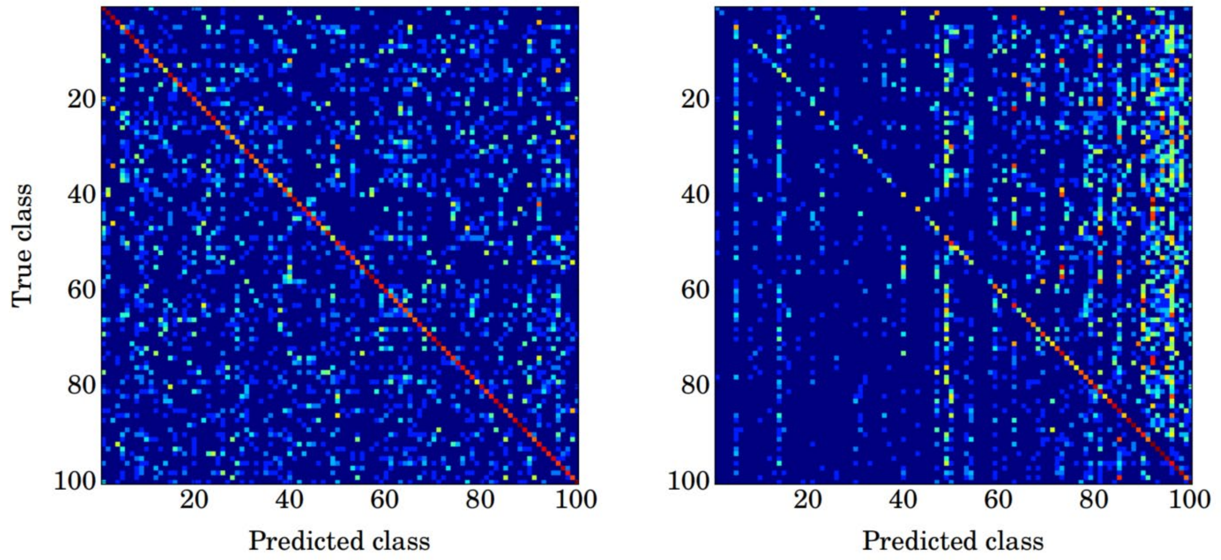
  Bias towards classes from recent batches

- **Fixed representation**

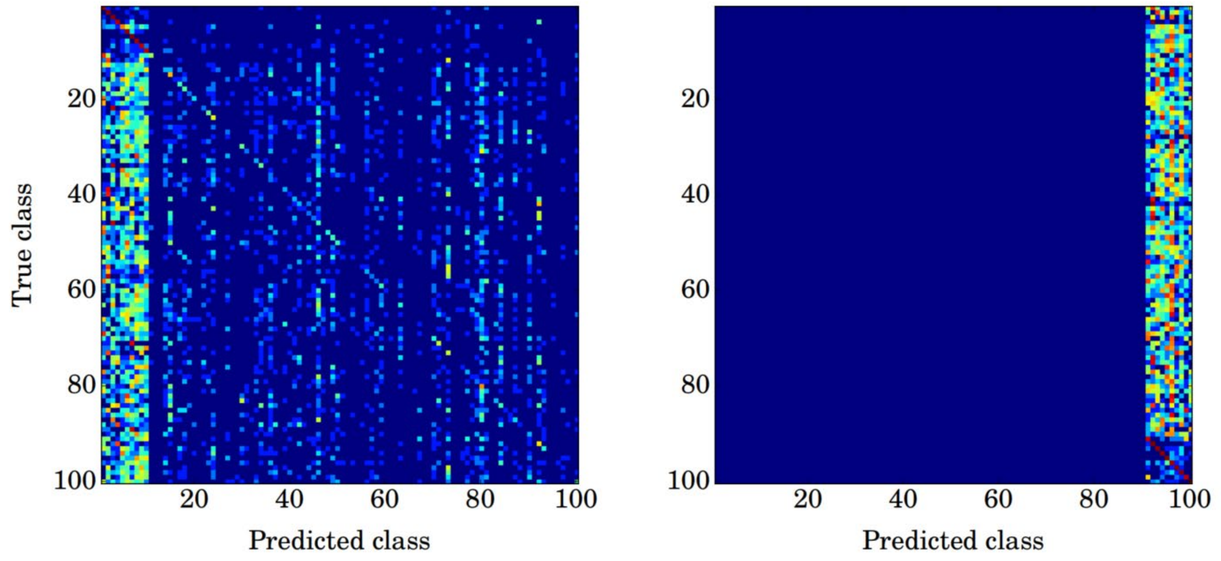
  Bias towards classes from the first batch

- **Finetuning**

  Bias towards classes from the last batch



(a) iCaRL (b) LwF.MC



(c) fixed representation (d) finetuning

Outline

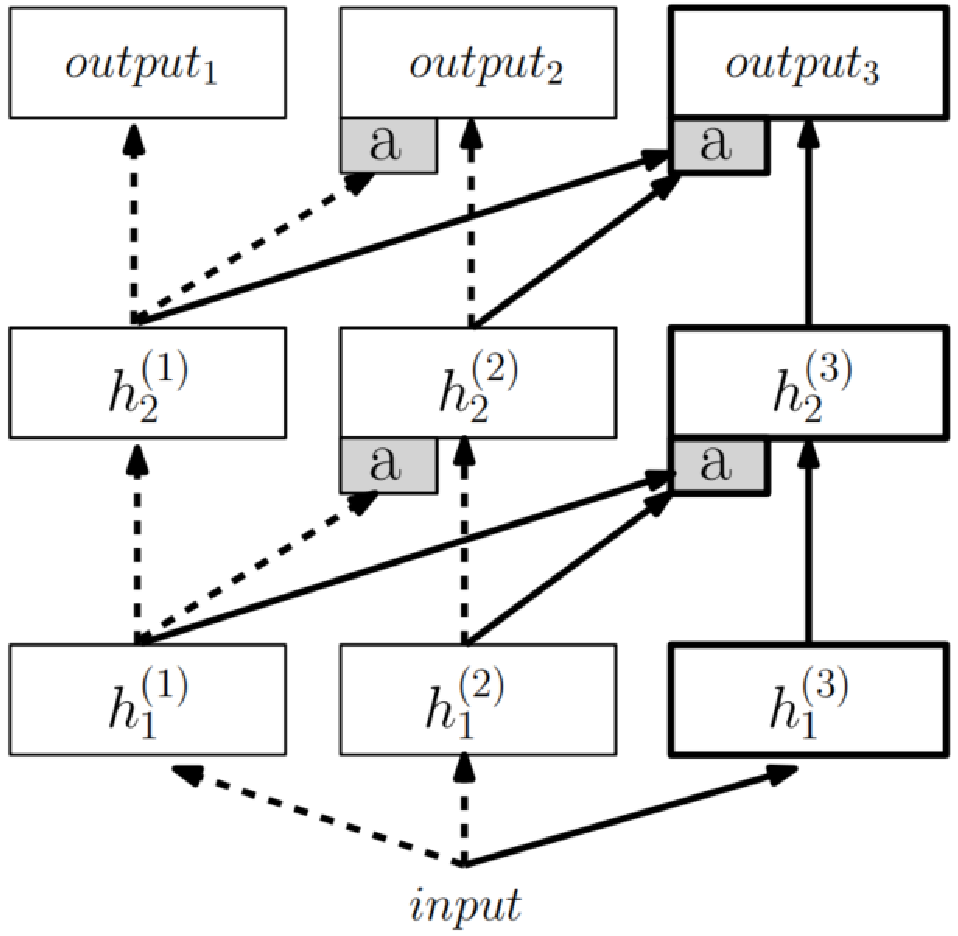
Parameter Isolation Methods

- **Basic idea:**
  - ✓ Different model parameters for each task

- **Two main solutions:**
  - ✓ **Dynamic architecture**: No constraint on model size
  - ✓ **Static network**: Fixed parts allocated to each task

Progressive Neural Networks (PNN) — Dynamic Architecture

- Instantiate a new neural network for each task
- Freeze the weights of previous tasks and transfer knowledge via lateral connections

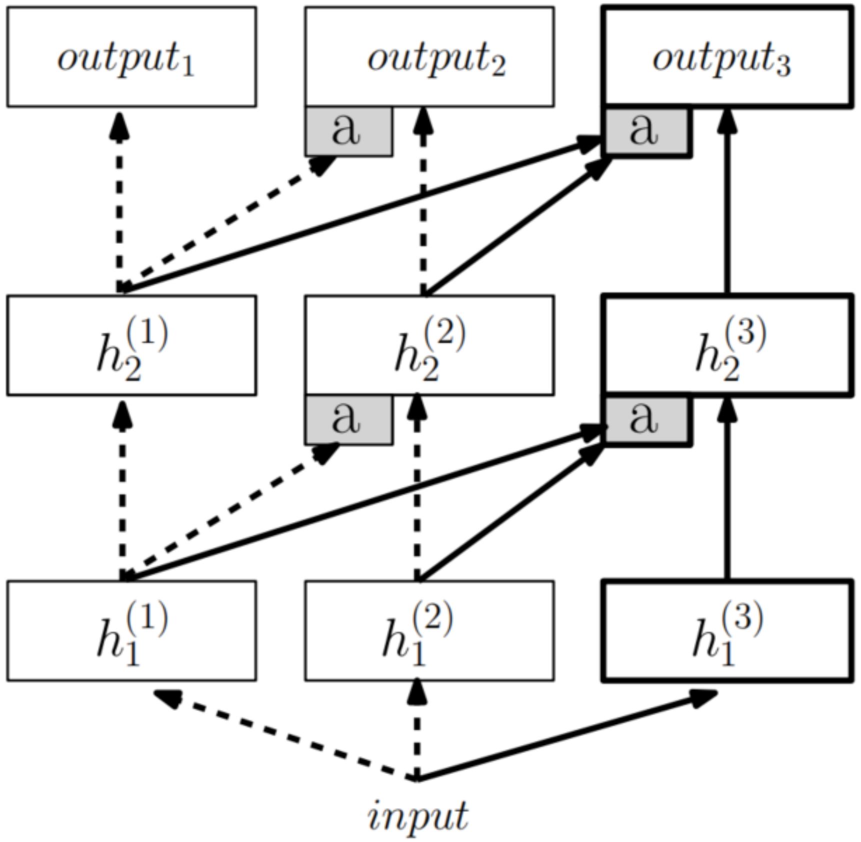


A. A. Rusu et al., "Progressive neural networks," 2016, arXiv:1606.04671.

Progressive Neural Networks (PNN)

Train network of Task k:

$$h_i^{(k)} = f\left(\overbrace{\boxed{W_i^{(k)}}}^{\text{Weight of current network}}\underbrace{h_{i-1}^{(k)}}_{\text{Hidden activation of current network}} + \sum_{j<k} \overbrace{\boxed{U_i^{(k:j)}}}^{\text{Weight of lateral connections}}\underbrace{h_{i-1}^{(j)}}_{\text{Hidden activation of previous network}}\right),$$



A. A. Rusu et al., "Progressive neural networks," 2016, arXiv:1606.04671.

PNN: Experiment Setup

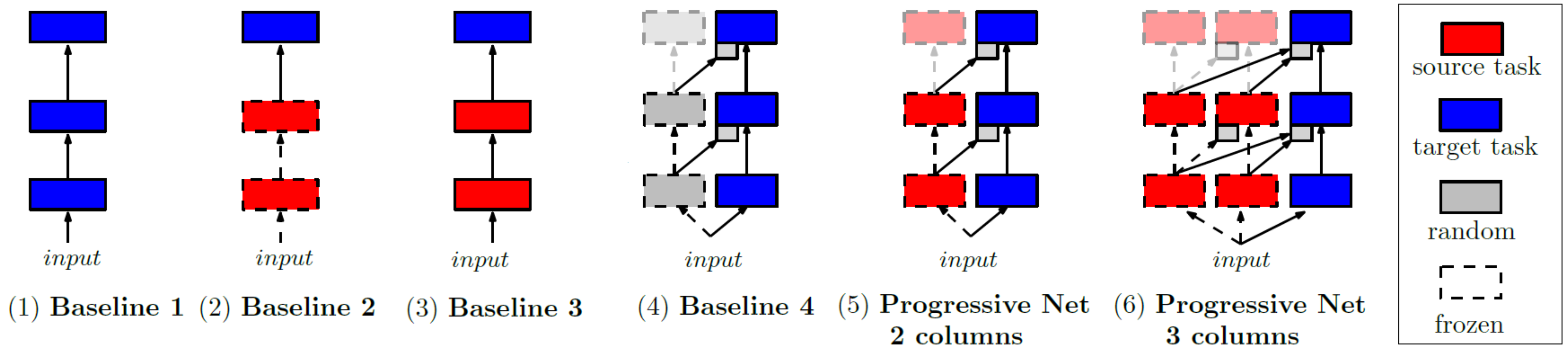
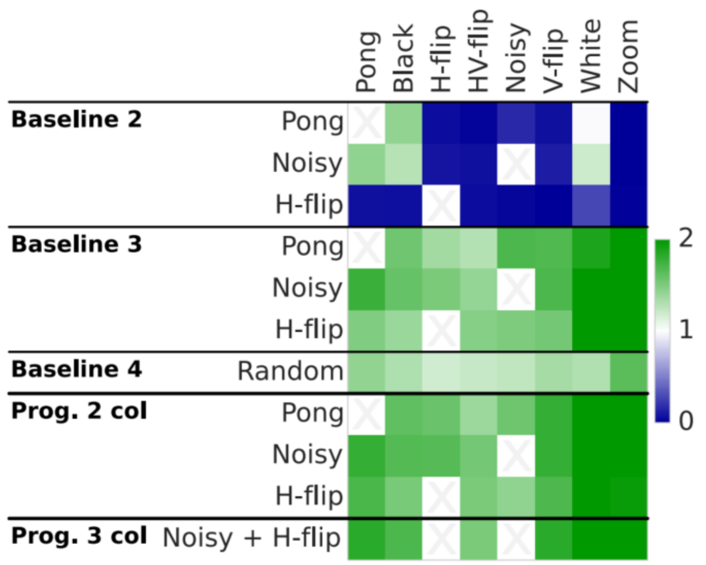


Figure 3: Illustration of different baselines and architectures. *Baseline 1* is a single column trained on the target task; *baseline 2* is a single column, pretrained on a source task and finetuned on the target task (output layer only); *baseline 3* is the same as baseline 2 but the whole model is finetuned; and *baseline 4* is a 2 column progressive architecture, with previous column(s) initialized randomly and frozen.

A. A. Rusu et al., "Progressive neural networks," 2016, arXiv:1606.04671.

PNN: Experiment Results

- **Evaluation Domain:** Atari game where the visuals and gameplay have been altered

(a)


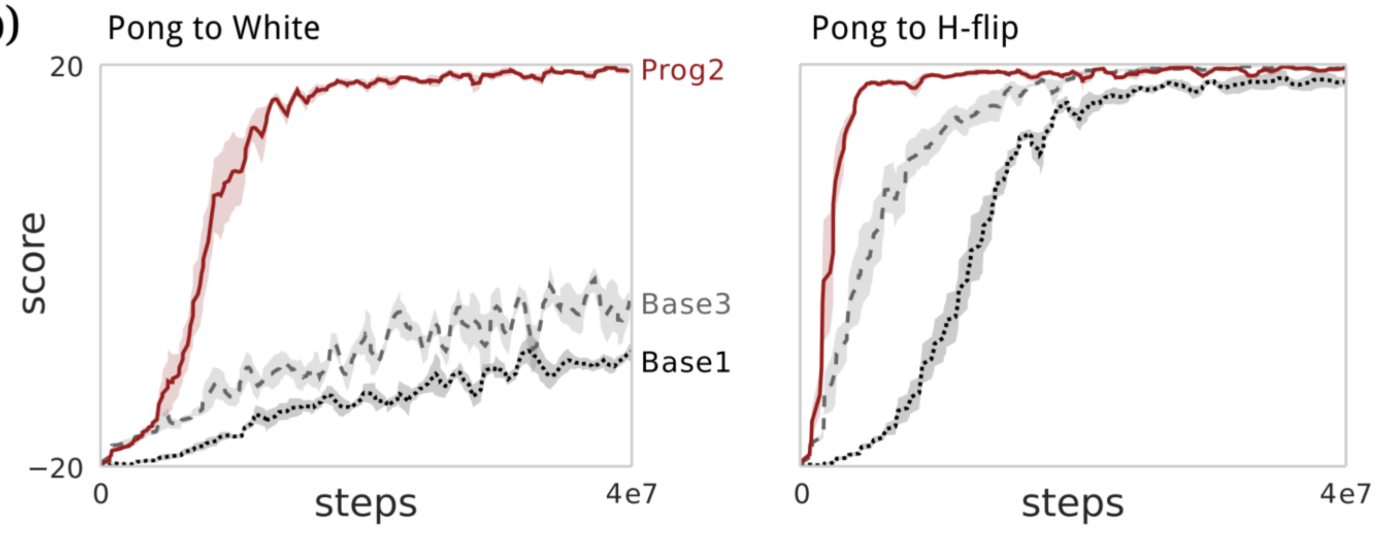
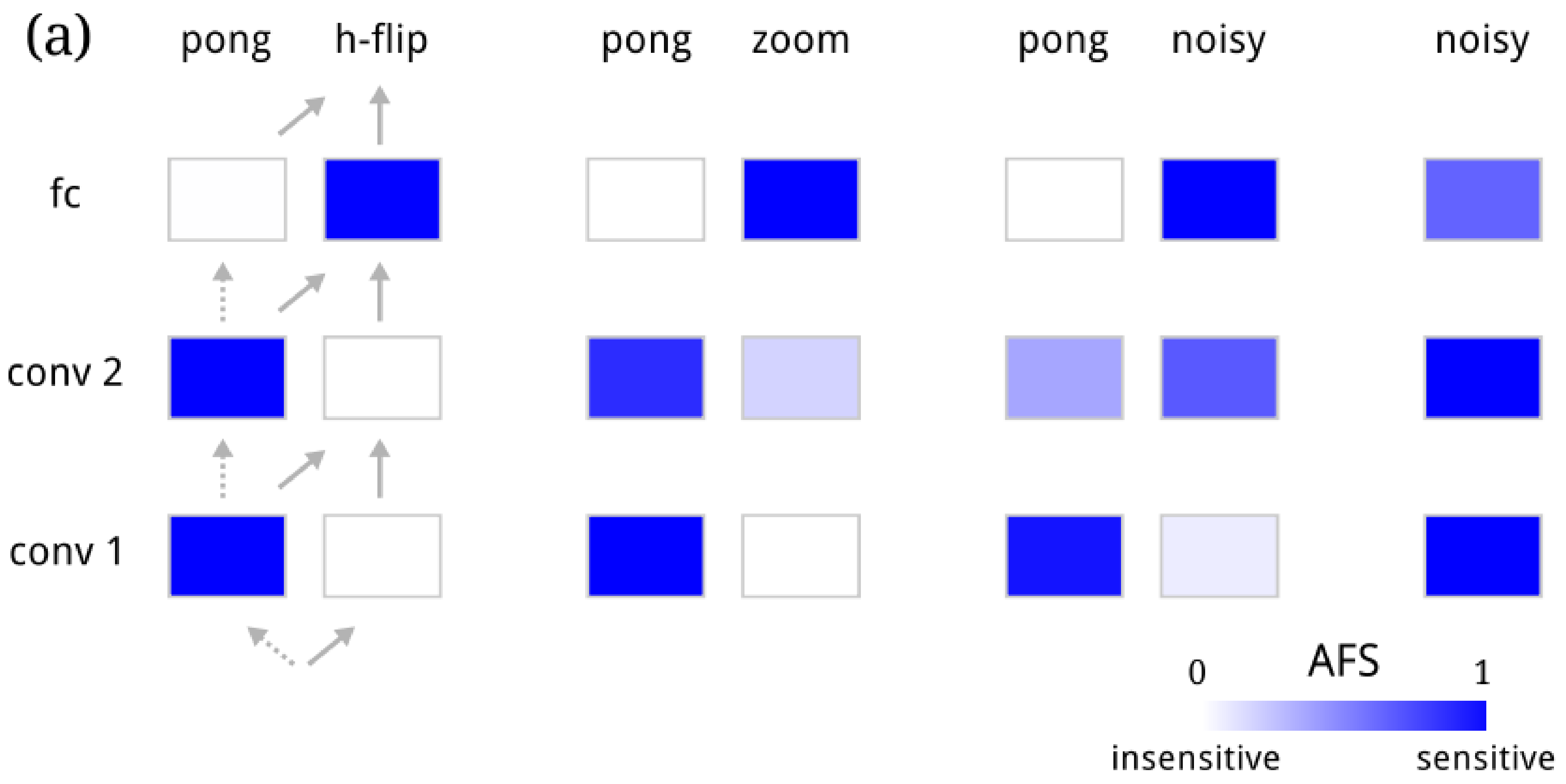
(b)


Figure 4: (a) Transfer matrix. Colours indicate transfer scores (clipped at 2). For progressive nets, the first column is trained on Pong, Noisy, or H-flip (table rows); the second column is trained on each of the other pong variants (table columns). (b) Example learning curves.

A. A. Rusu et al., "Progressive neural networks," 2016, arXiv:1606.04671.

PNN: Detailed Analysis



The network tends to reuse low-level features

A. A. Rusu et al., "Progressive neural networks," 2016, arXiv:1606.04671.

PNN: Limitations

1. The growth in the number of parameters with the number of tasks

2. Only a fraction of the new capacity is actually utilized

3. Require the task identity to choose network while testing

A. A. Rusu et al., "Progressive neural networks," 2016, arXiv:1606.04671.

Dynamically Expandable Representation (DER) — Dynamic Architecture

- A two-stage learning approach that utilizes a dynamically expandable representation
- Save part of previous data as the memory $\mathcal{M}_t$ for future training
- The model consists of two parts: feature extractor ϕ_t and classifier $\mathcal{H}_t$

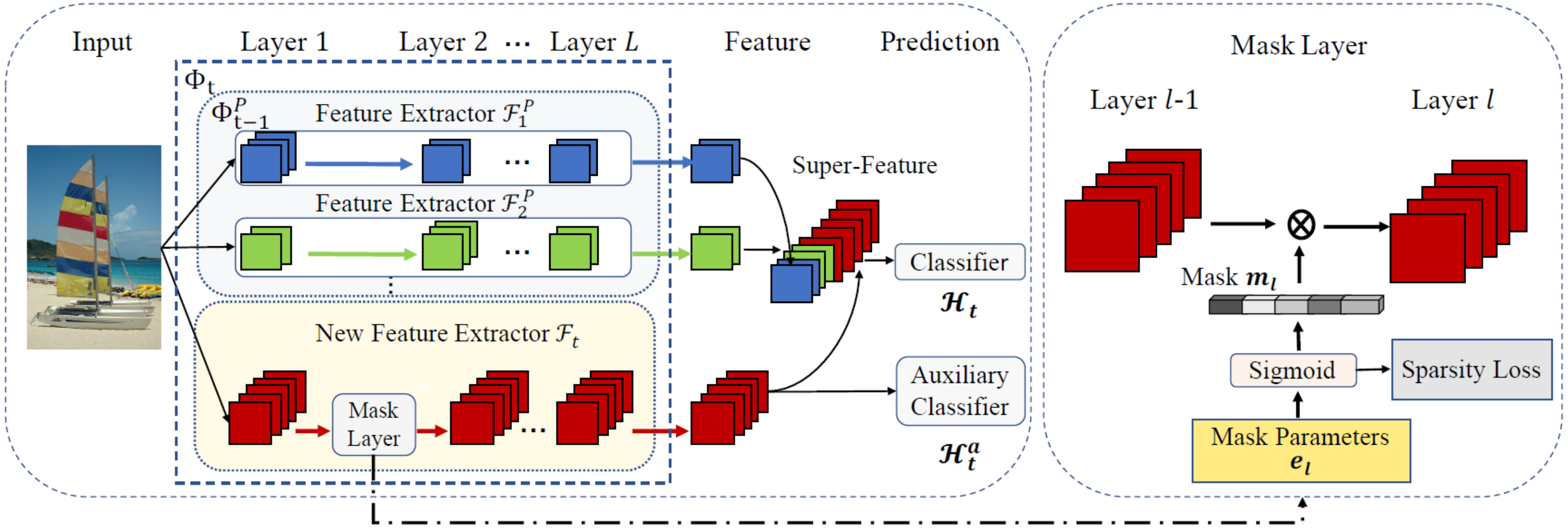


Figure 2: Dynamically Expandable Representation Learning. At step t, the model is composed of super-feature extractor Φ_t and classifier $\mathcal{H}_t$, where Φ_t is built by expanding the existing super-feature extractor Φ_{t-1}^P with new feature extractor $\mathcal{F}_t$. We also use an auxiliary classifier to regularize the model. Besides, the layer-wise channel-level mask is jointed learned with the representation, which is used to prune the network after the learning of model.

Yan S P, Xie J W, He X M. DER: Dynamically expandable representation for class incremental learning. CVPR 2021.

Dynamically Expandable Representation (DER)

1. **Expandable Representation Learning**: The current ϕ_t is build by expanding the feature extractor ϕ_{t-1} , the feature is then fed into the classifier $\mathcal{H}_t$ for prediction
2. **Dynamical Expansion**: Remove the model redundancy and maintain a compact representation

Yan S P, Xie J W, He X M. DER: Dynamically expandable representation for class incremental learning. CVPR 2021.

DER

$$\mathcal{L}_{\text{DER}} = \mathcal{L}_{\mathcal{H}_t} + \lambda_a \mathcal{L}_{\mathcal{H}_t^a} + \lambda_s \mathcal{L}_S$$

- **Classification loss**: Retrain the classifier $\mathcal{H}_t$ with currently available data $\widetilde{D}_t = D_t \cup \mathcal{M}_t$

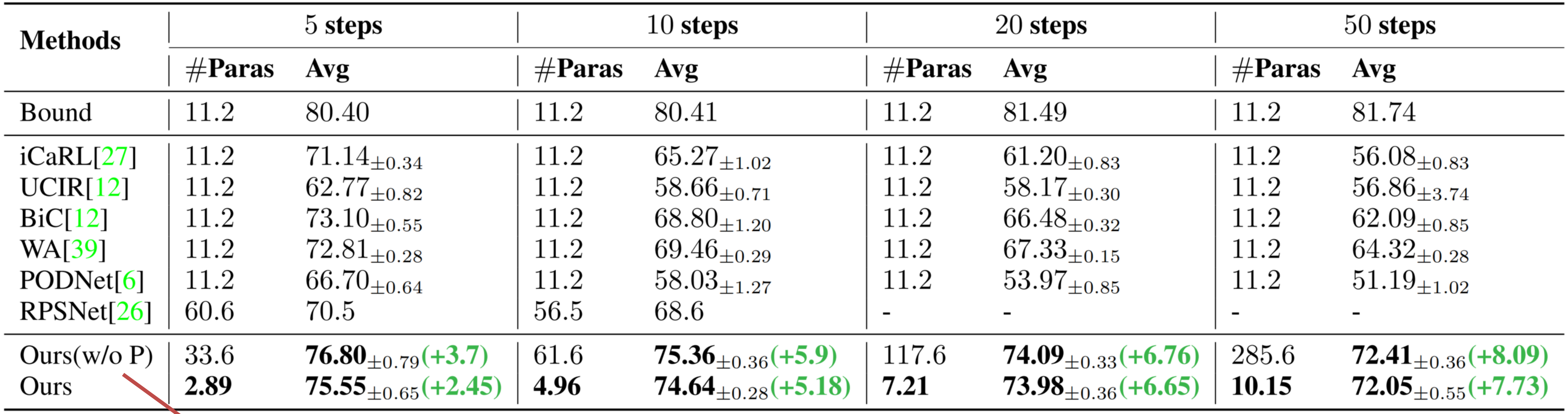
$$\mathcal{L}_{\mathcal{H}_t} = -\frac{1}{|\tilde{\mathcal{D}}_t|} \sum_{i=1}^{|\tilde{\mathcal{D}}_t|} \log(p_{\mathcal{H}_t}(y = y_i | \boldsymbol{x_i})))$$

- **Auxiliary classifier** $\mathcal{H}_t^a$: Encourage the model to discriminate old and new concepts

- **Sparsity loss**: $$\mathcal{L}_S = \frac{\sum_{l=1}^{L} K_l \|\boldsymbol{m_{l-1}}\|_1 \|\boldsymbol{m_l}\|_1}{\sum_{l=1}^{L} K_l c_{l-1} c_l}$$

DER: Experiment Results

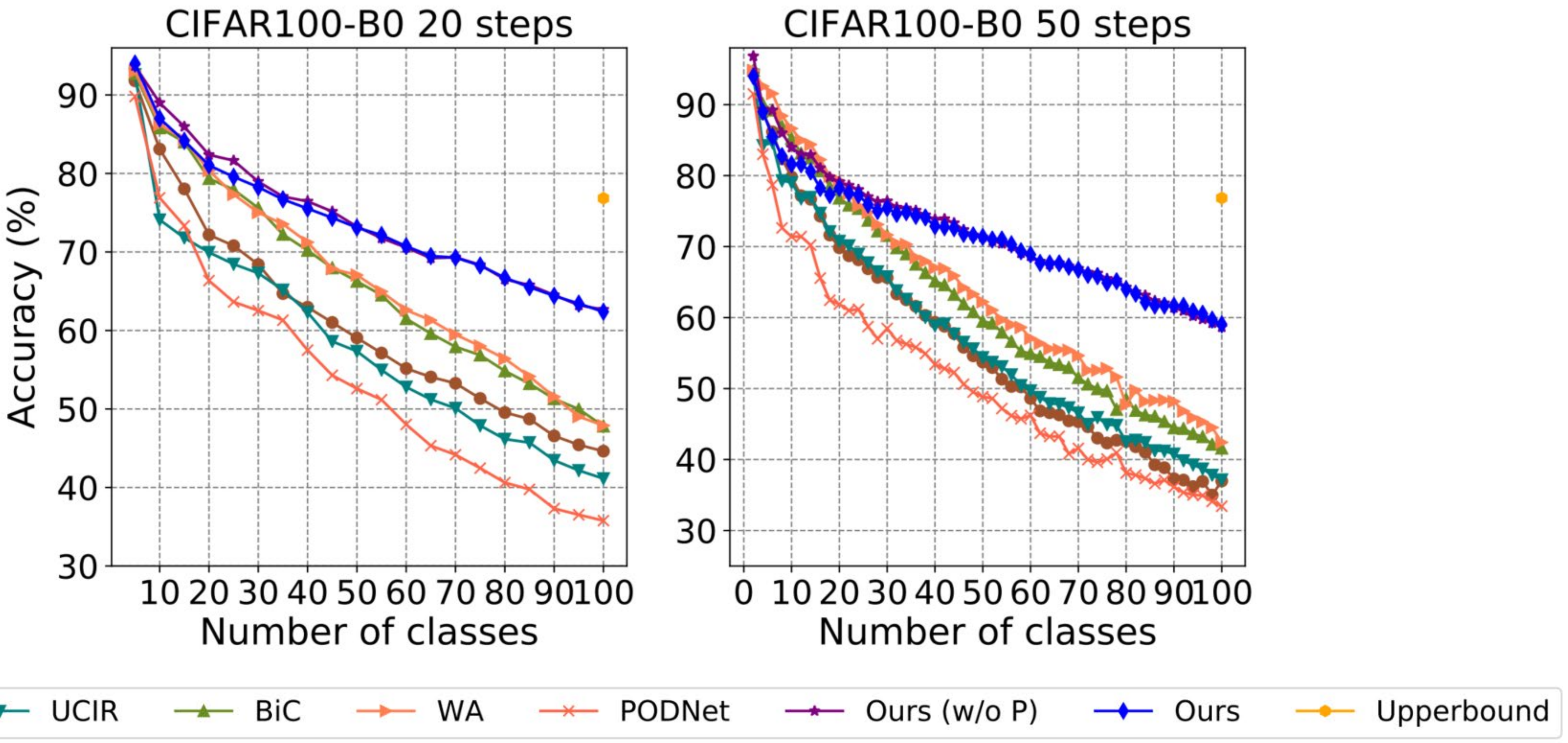
CIFAR-100: Train all 100 classes in several splits including 5/10/20/50 incremental steps, with fixed memory size of 2,000 exemplars over batches



Methods	5 steps		10 steps		20 steps		50 steps	
	#Paras	**Avg**	**#Paras**	**Avg**	**#Paras**	**Avg**	**#Paras**	**Avg**
Bound	11.2	80.40	11.2	80.41	11.2	81.49	11.2	81.74
iCaRL[27]	11.2	$71.14_{\pm0.34}$	11.2	$65.27_{\pm1.02}$	11.2	$61.20_{\pm0.83}$	11.2	$56.08_{\pm0.83}$
UCIR[12]	11.2	$62.77_{\pm0.82}$	11.2	$58.66_{\pm0.71}$	11.2	$58.17_{\pm0.30}$	11.2	$56.86_{\pm3.74}$
BiC[12]	11.2	$73.10_{\pm0.55}$	11.2	$68.80_{\pm1.20}$	11.2	$66.48_{\pm0.32}$	11.2	$62.09_{\pm0.85}$
WA[39]	11.2	$72.81_{\pm0.28}$	11.2	$69.46_{\pm0.29}$	11.2	$67.33_{\pm0.15}$	11.2	$64.32_{\pm0.28}$
PODNet[6]	11.2	$66.70_{\pm0.64}$	11.2	$58.03_{\pm1.27}$	11.2	$53.97_{\pm0.85}$	11.2	$51.19_{\pm1.02}$
RPSNet[26]	60.6	70.5	56.5	68.6	-	-	-	-
Ours(w/o P)	33.6	**76.80**$_{\pm0.79}$(+3.7)	61.6	**75.36**$_{\pm0.36}$(+5.9)	117.6	**74.09**$_{\pm0.33}$(+6.76)	285.6	**72.41**$_{\pm0.36}$(+8.09)
Ours	**2.89**	**75.55**$_{\pm0.65}$(+2.45)	**4.96**	**74.64**$_{\pm0.28}$(+5.18)	**7.21**	**73.98**$_{\pm0.36}$(+6.65)	**10.15**	**72.05**$_{\pm0.55}$(+7.73)

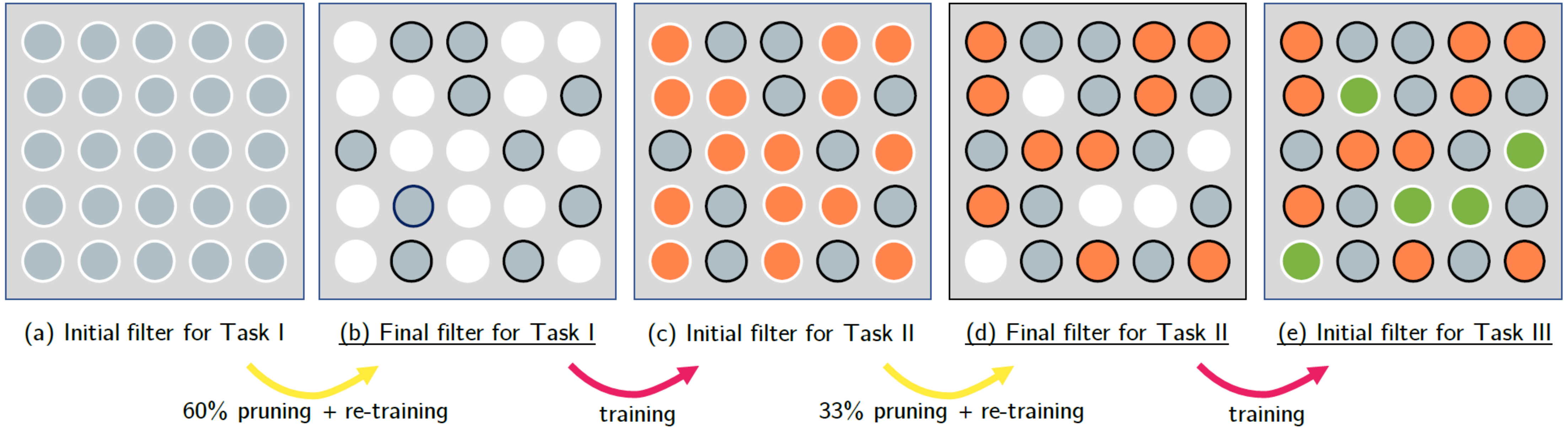
Without pruning (sparsity loss)

DER: Experimental Results



PackNet — Static Network

Use network pruning techniques to create free parameters that can then be employed for learning new tasks, without adding extra network capacity



(a) Initial filter for Task I
(b) Final filter for Task I
(c) Initial filter for Task II
(d) Final filter for Task II
(e) Initial filter for Task III
60% pruning + re-training
training
33% pruning + re-training
training

Figure 1: Illustration of the evolution of a 5×5 filter with steps of training. Initial training of the network for Task I learns a dense filter as illustrated in (a). After pruning by 60% (15/25) and re-training, we obtain a sparse filter for Task I, as depicted in (b), where white circles denote 0 valued weights. Weights retained for Task I are kept fixed for the remainder of the method, and are not eligible for further pruning. We allow the pruned weights to be updated for Task II, leading to filter (c), which shares weights learned for Task I. Another round of pruning by 33% (5/15) and re-training leads to filter (d), which is the filter used for evaluating on task II (Note that weights for Task I, in gray, are not considered for pruning). Hereafter, weights for Task II, depicted in orange, are kept fixed. This process is completed until desired, or we run out of pruned weights, as shown in filter (e). The final filter (e) for task III shares weights learned for tasks I and II. At test time, appropriate masks are applied depending on the selected task so as to replicate filters learned for the respective tasks.

A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," CVPR 2018.

PackNet: Pruning Procedure

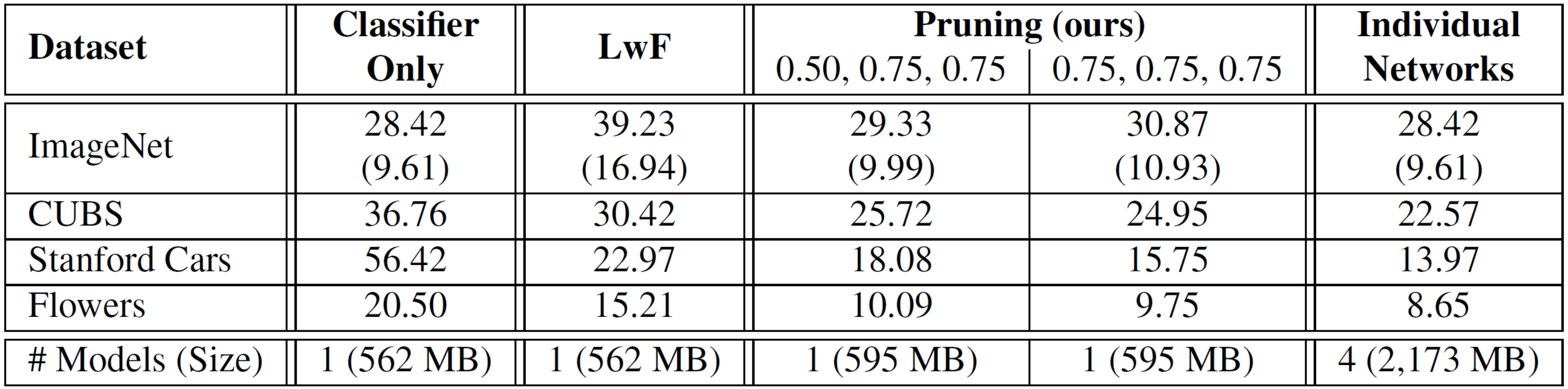
1. The weights in a layer are sorted by their absolute magnitude, and the lowest 50% or 75% are selected for removal

2. Only prune weights belonging to the current task, but not modify weights that belong to a prior task

3. Overhead: Storage of a sparsity mask indicating which parameters are active for a particular task

PackNet: Experiment Results

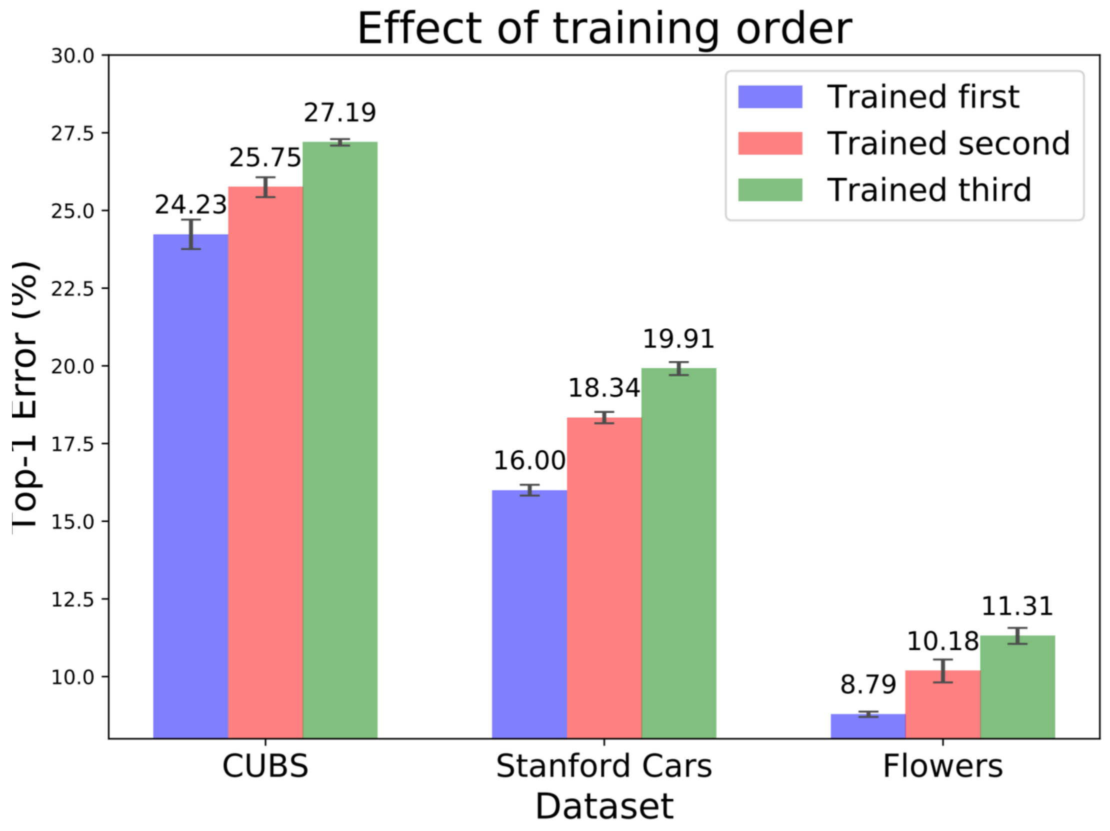
Continual Learning



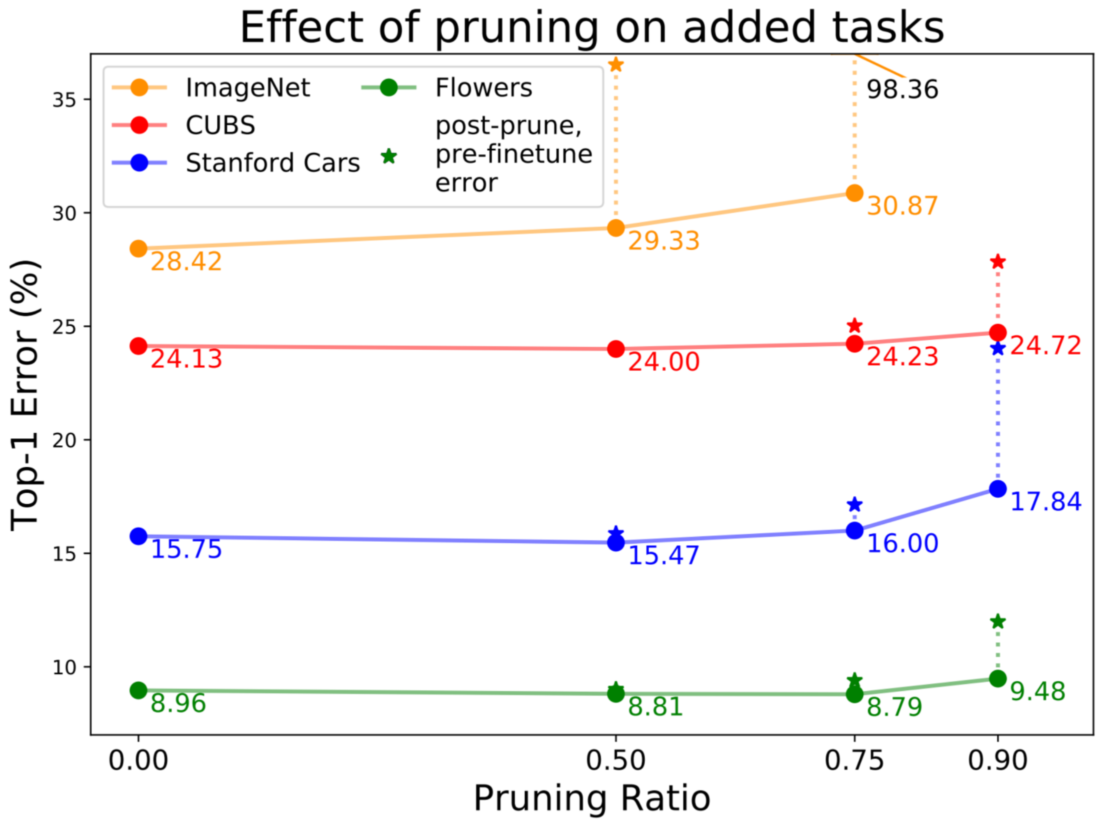
Dataset	Classifier Only	LwF	Pruning (ours) 0.50, 0.75, 0.75	Pruning (ours) 0.75, 0.75, 0.75	Individual Networks
ImageNet	28.42 (9.61)	39.23 (16.94)	29.33 (9.99)	30.87 (10.93)	28.42 (9.61)
CUBS	36.76	30.42	25.72	24.95	22.57
Stanford Cars	56.42	22.97	18.08	15.75	13.97
Flowers	20.50	15.21	10.09	9.75	8.65
# Models (Size)	1 (562 MB)	1 (562 MB)	1 (595 MB)	1 (595 MB)	4 (2,173 MB)

Table 2: Errors on fine-grained tasks. Values in parentheses are top-5 errors, while all others are top-1 errors. The numbers at the top of the Pruning columns indicate the ratios by which the network is pruned after each successive task. For example, 0.50, 0.75, 0.75 indicates that the initial ImageNet-trained network is pruned by 50%, and after each task is added, 75% of the parameters belonging to that task are set to 0.

PackNet: Detailed Analysis



The most challenging or unrelated task should be added first

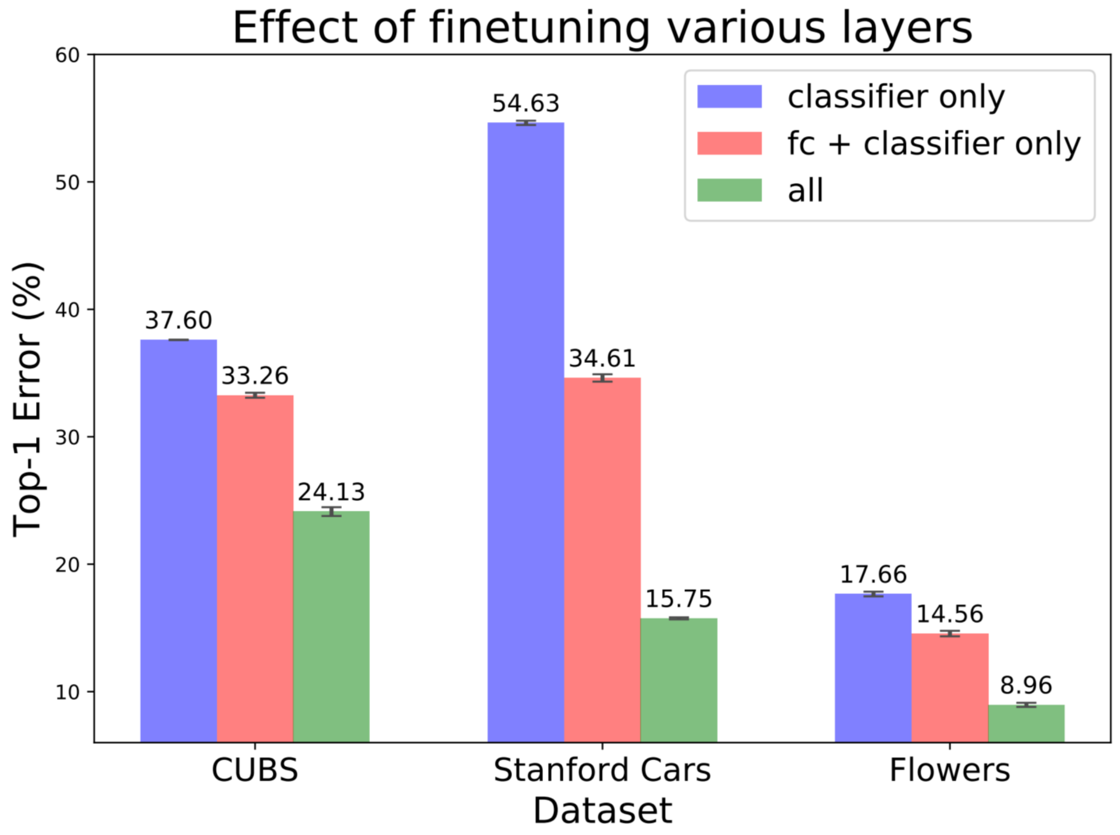


Effective transfer learning as very few parameter modifications are enough to obtain good accuracies

PackNet: Detailed Analysis

Dataset	Pruning 0.50, 0.75, 0.75	
	Separate Bias	Shared Bias
CUBS	25.62	25.72
Stanford Cars	18.17	18.08
Flowers	10.11	10.09

Sharing biases reduces the storage overhead, as each convolutional, fully-connected, or batch-normalization layer can contain an associated bias term.



Better to finetune all layers

Hard Attention to the Task (HAT) — Static Network

- **Motivation:**
  - ✓ The network may learn a new set of features, some of which not overlap much with previous tasks'

- **Basic idea:**
  - ✓ Learn to use the task identifier to condition every layer, and then exploit this learned condition to prevent forgetting previous tasks

J. Serra, D. Surıs, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," ICLR 2018.

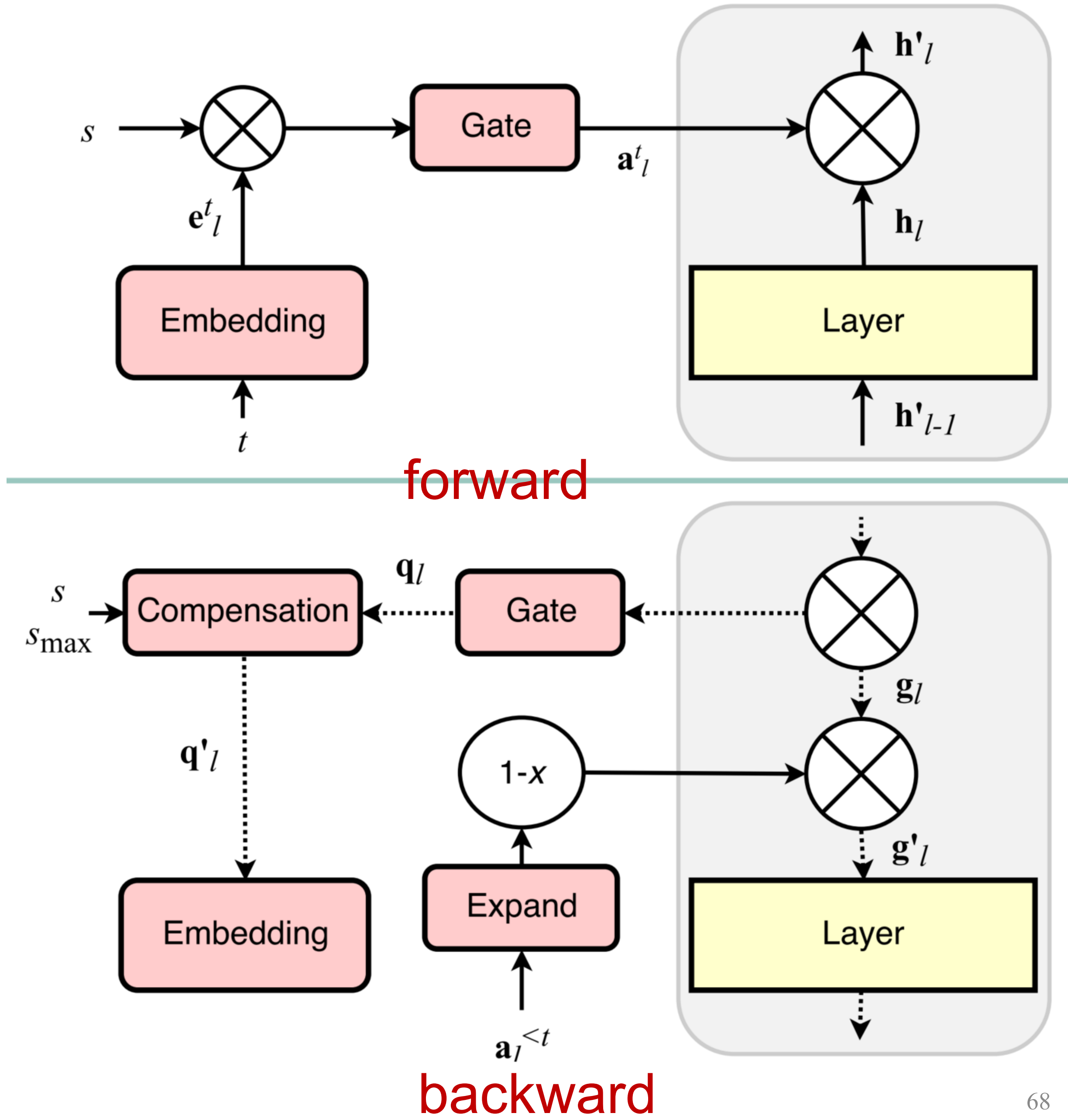
HAT: Architecture

The output of each layer

$$\mathbf{h}'_l = \mathbf{a}^t_l \odot \mathbf{h}_l$$

where $\boldsymbol{a}^t_l$ is a gated version of a single-layer task embedding $\boldsymbol{e}^t_l$

$$\mathbf{a}^t_l = \underbrace{\sigma}_{\text{Sigmoid gate function}} \left(\underbrace{s}_{\text{Positive scaling parameter}} \mathbf{e}^t_l \right) ,$$



HAT: Network Training

- Condition the gradients according to the cumulative attention from all previous tasks

$$\mathbf{a}_l^{\leq t} = \max\left(\mathbf{a}_l^t, \mathbf{a}_l^{\leq t-1}\right),$$

- Modify the gradient with the reverse of the minimum of the cumulative attention in the current and previous layers

$$g'_{l,ij} = \left[1 - \min\left(a_{l,i}^{\leq t}, a_{l-1,j}^{\leq t}\right)\right] g_{l,ij},$$

Masks prevent large updates to weights important to previous tasks

PackNet vs HAT

- **Similarity:**

  Employ masks to constrain the network

- **Differences:**

  1. HAT’s constraint is based on network weights, which allows for a potentially better use of the network’s capacity; PackNet is based on heuristic weight pruning.
  2. HAT's mask is learnable; PackNet uses pre-assigned pruning ratios.
  3. HAT's mask is not always binary, it can be between 0 and 1

HAT: Experimental Results

- **Forgetting ratio:**

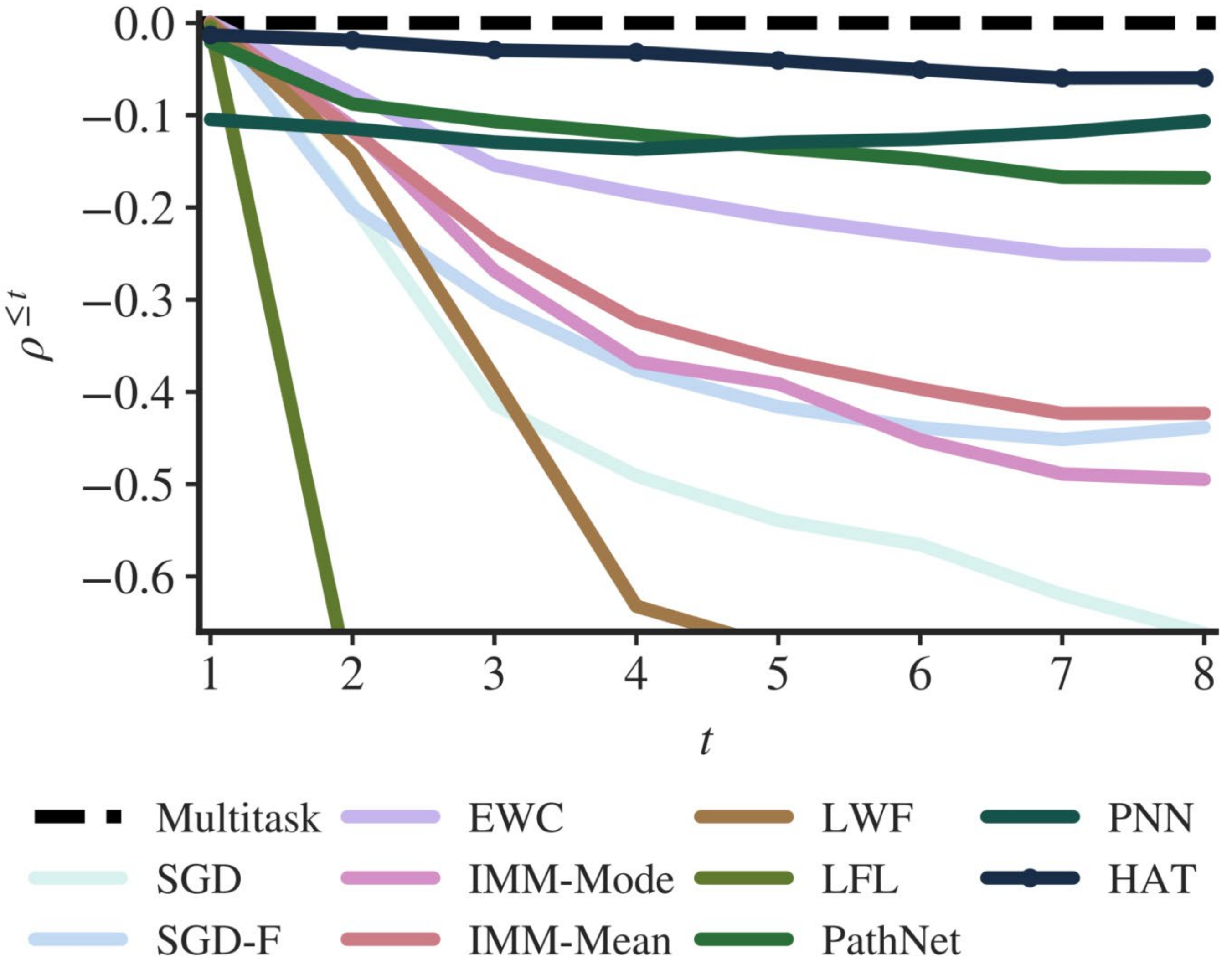
The accuracy measured on task τ after sequentially learning task t

$$\rho^{\tau\leq t} = \frac{A^{\tau\leq t} - A^{\tau}_{\mathrm{R}}}{A^{\tau\leq t}_{\mathrm{J}} - A^{\tau}_{\mathrm{R}}} - 1,$$

Joint training

Random accuracy

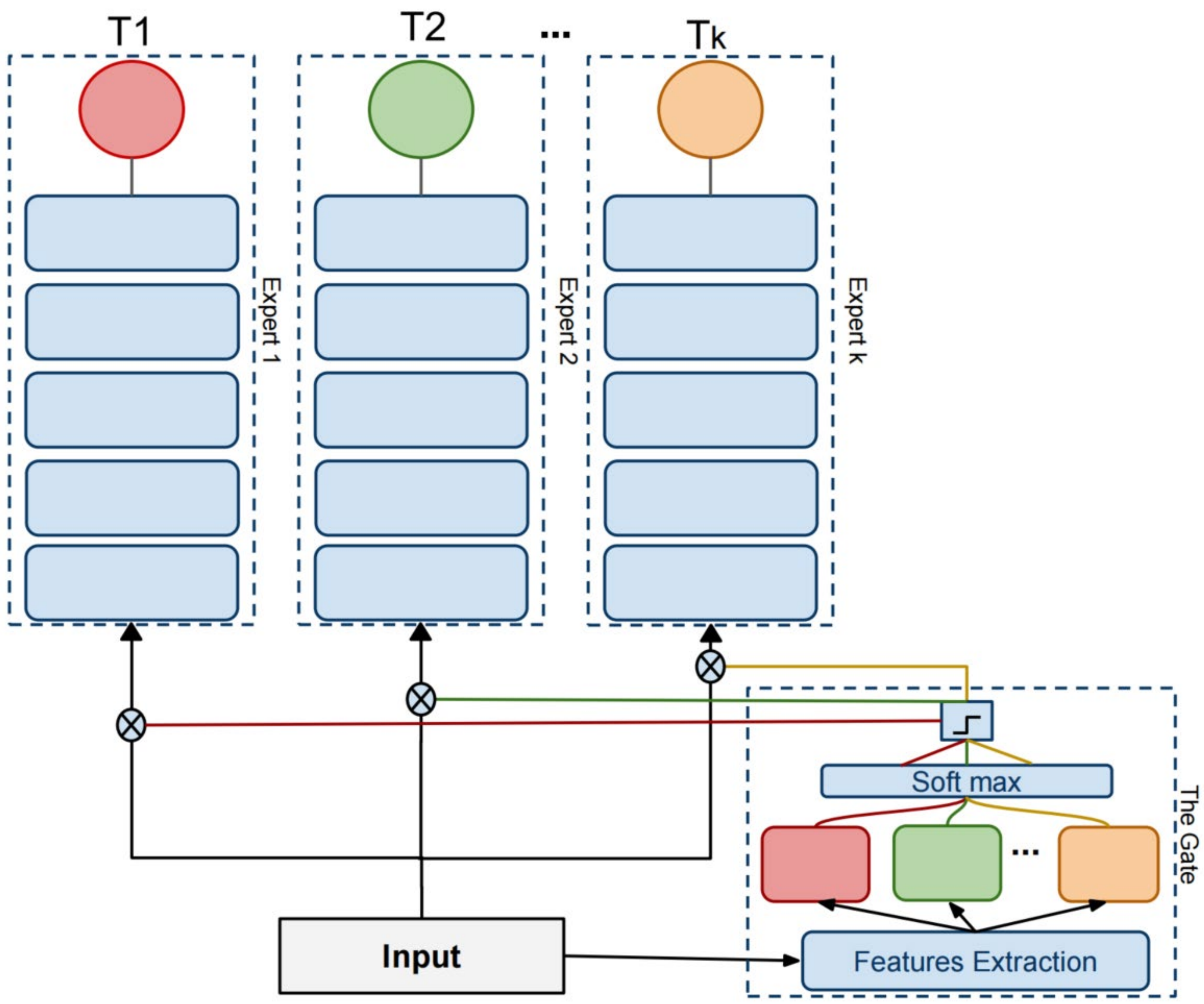
$$\rho^{\leq t} = \frac{1}{t}\sum_{\tau=1}^{t} \rho^{\tau\leq t}.$$



Expert Gate — Static Network

Basic idea:

- ✓ Add a new expert model whenever a new task arrives，and transfer knowledge from previous models
- ✓ Learn a gating mechanism that uses the test sample to decide which expert to activate



R. Aljundi, P. Chakravarty, and T. Tuytelaars, “Expert gate: Lifelong learning with a network of experts,” CVPR 2017.

Expert Gate

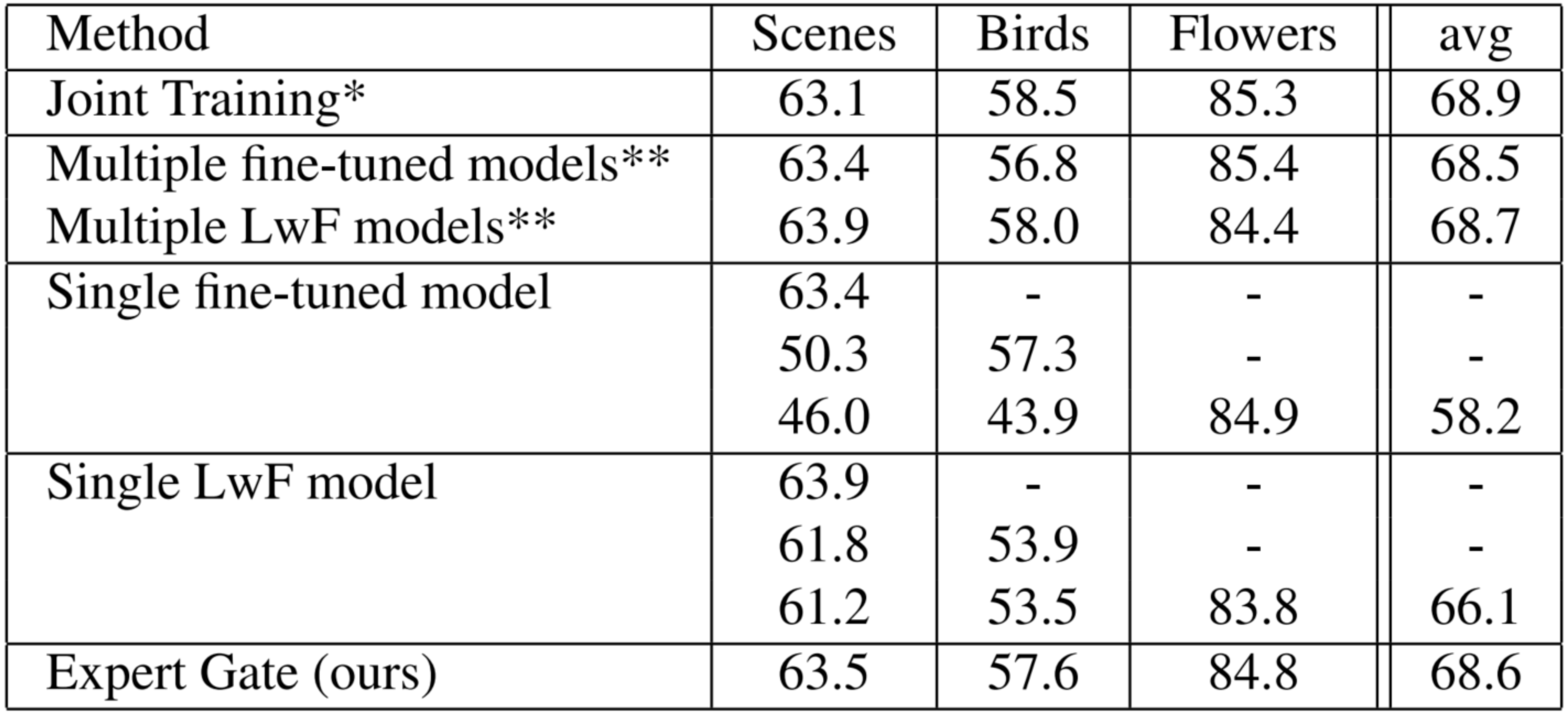
1. **An autoencoder gate is trained for each task.** The autoencoder of each domain/task should be better at reconstructing the data of that task than others.

2. **Select the most relevant expert.** First compute the reconstruction error er_i of the i-th autoencoder and feed it into an extra softmax layer:

$$p_i = \frac{exp(-er_i/\overset{\text{Temperature}}{t})}{\sum_j exp(-er_j/t)}$$

Then load the expert model associated with the most confident autoencoder.

Expert Gate: Experiment Results

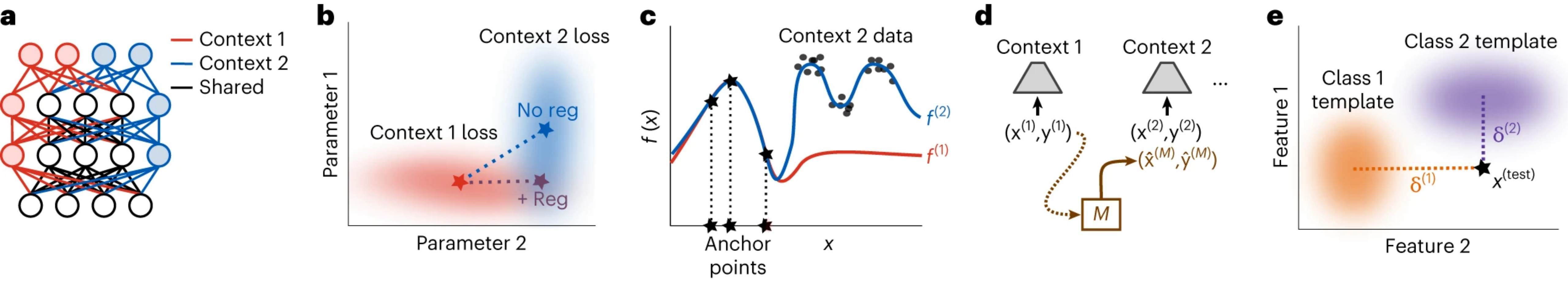


Method	Scenes	Birds	Flowers	avg
Joint Training*	63.1	58.5	85.3	68.9
Multiple fine-tuned models**	63.4	56.8	85.4	68.5
Multiple LwF models**	63.9	58.0	84.4	68.7
Single fine-tuned model	63.4	-	-	-
	50.3	57.3	-	-
	46.0	43.9	84.9	58.2
Single LwF model	63.9	-	-	-
	61.8	53.9	-	-
	61.2	53.5	83.8	66.1
Expert Gate (ours)	63.5	57.6	84.8	68.6

Comparison of the Three Methods

Solution	Advantage	Disadvantage
Rehearsal	• Simple and effective	• Require additional computation and storage of raw input samples • Fix memory deteriorates the ability of exemplar sets to represent the original distribution
Regularization	• Avoid storing raw inputs, protect privacy, and alleviate memory requirements	• As more tasks are added the weights gradually deviate from the optimal weights of previous tasks
Parameter Isolation	• Guarantee maximal stability by fixing the parameter subsets of previous tasks	• Require a task identity to activate corresponding masks or task branch during prediction

Comparison of the Three Methods



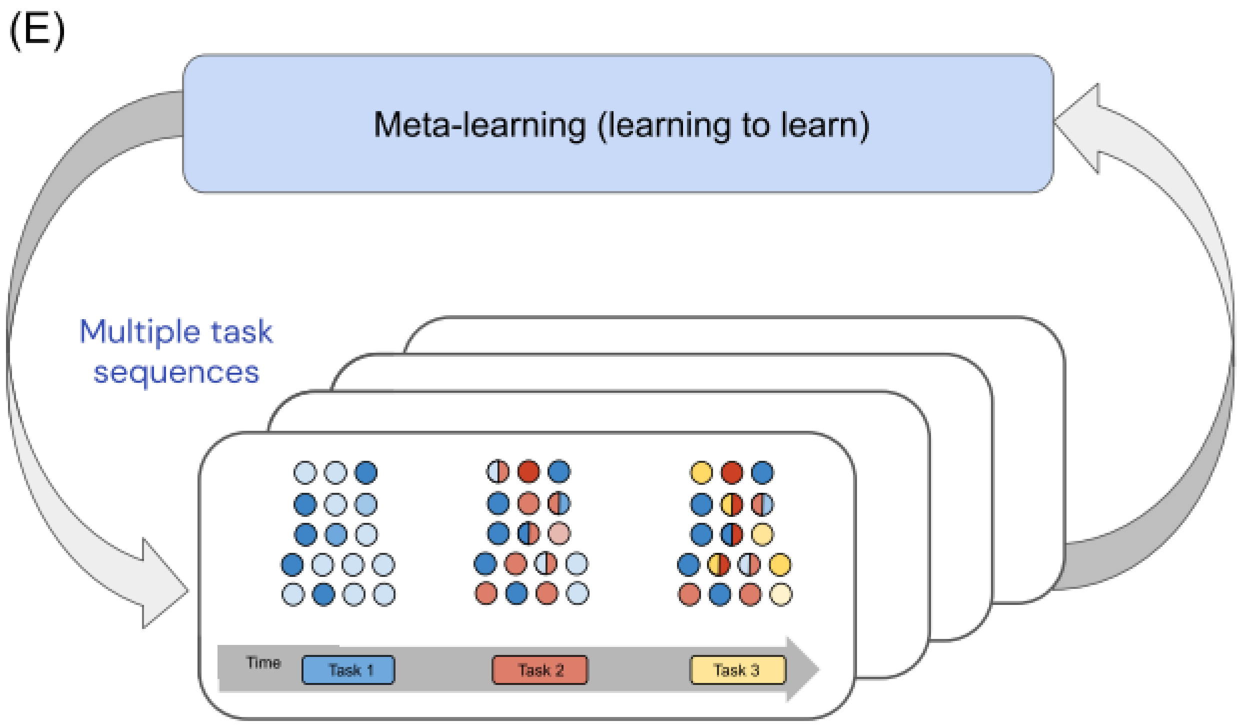
a. Context-specific components. Certain parts of the network are only used for specific contexts.
b. Parameter regularization. Parameters important for past contexts are encouraged not to change too much when learning new contexts.
c. Functional regularization. The input–output mapping learned previously is encouraged not to change too much at a particular set of inputs (the 'anchor points') when training on new contexts.
d. Replay. The training data of a new context is complemented with data representative of past context. The replayed data is sampled from *M*, which can be a memory buffer or a generative model.
e. Template-based classification. A 'template' is learned for each class (for example, a prototype, an energy value or a generative model), and classification is performed based on which template is most suitable for the sample to be classified.

Outline

- Basic Concepts
- Rehearsal Methods
- Regularization-Based Methods
- Parameter Isolation Methods
- **Discussion**

Meta-Learning & Continual Learning

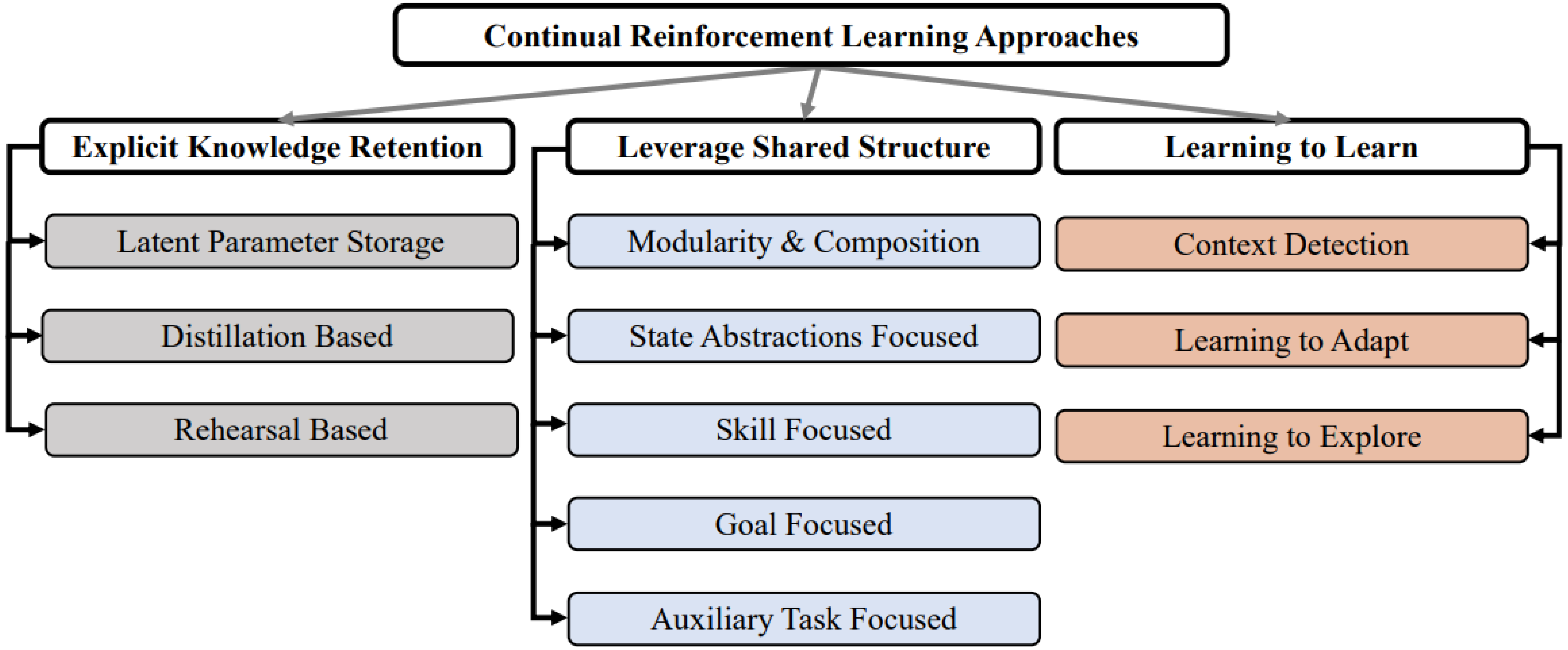
- Differences in focus：
  - ✓ Meta-learning: Learning to learn
  - ✓ Continual learning: Learn reusable representations from non-stationary data
- Two main categories:
  1. Meta continual-learning
  2. Continual meta-learning



Hadsell R, Rao D, Rusu A A, et al. Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences, 2020.

Continual Reinforcement Learning

- Reinforcement Learning: Learning from (sparse) rewards
- Continual Learning: Learn reusable representations non-stationary data
- Quite orthogonal objectives but some shared constraints (single agent view, non-stationary environments, sample bias, etc..)



Hadsell R, Rao D, Rusu A A, et al. Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences, 2020.

Continual Unsupervised Learning

- Ideal paradigm to combine with CL
  - ✓ No Continual Labeling
  - ✓ Less Bias
- Why this is still not the case?
  - ✓ Changing the paradigm: More Data, Less Supervision
  - ✓ Less impactful applications for now (tend to use supervised learning for impressive results)

Frontiers in Continual Learning - Continual Learning Course (continualai.org)

Thank you!