Continual Learning and Catastrophic Forgetting

Renato Santos Aranha
Oct/2020

Main ref:
Lifelong Machine Learning, Second Edition
Zhiyuan Chen and Bing Liu
Morgan & Claypool 2018

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

1. **Catastrophic Forgetting**
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

CATASTROPHIC FORGETTING

Lifelong Learning + Deep Learning:
Continual Learning

Main issue:
catastrophic forgetting ☹
(learning later tasks degrade models learned from earlier tasks)

Goals:

- incremental learning of tasks
- relieve catastrophic forgetting problem

CATASTROPHIC FORGETTING

first recognized in 1989 [McCloskey et al, *Catastrophic interference in connectionist networks*]

Train on new tasks/class → NN tends to “forget” knowledge learned in previous trained tasks

<u>stability-plasticity dilemma</u>

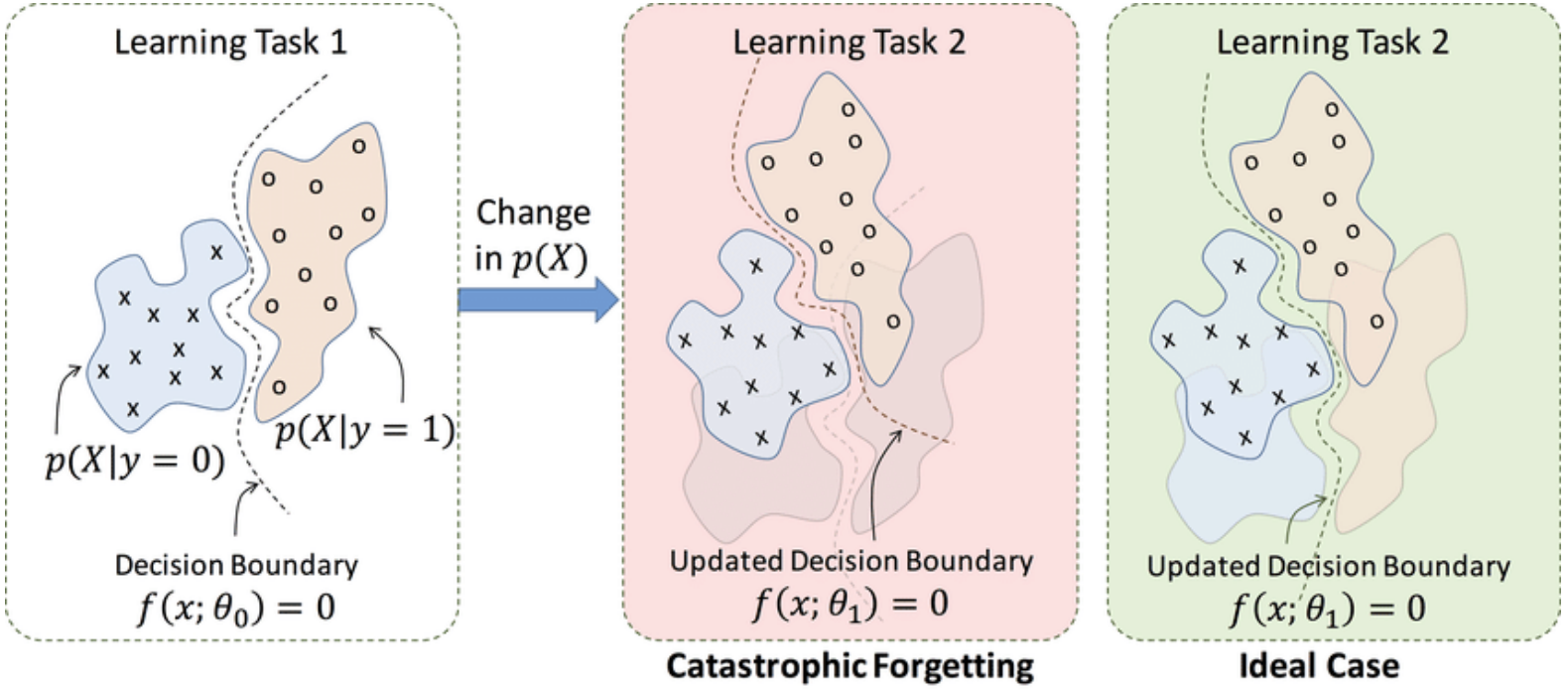
Model too stable → bad for future training data

Model too plastic → large weight changes ; forgets learned representations

CATASTROPHIC FORGETTING

first recognized in 1989 [McCloskey et al, *Catastrophic interference in connectionist networks*]

Train on new tasks/class → NN tends to "forget" knowledge learned in previous trained tasks



Kolouri et al. 2019

CATASTROPHIC FORGETTING

Example: transfer learning using a DNN

Source: labeled data
Target: little label data

Widely used approach: fine-tunning

Usual method:
backpropagation to adapt source model to target → output layers retrained given target data

Result:
catastrophic forgetting ☹
adaptation to target domain disrupts weights learned for source domain

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

Example:

- image classification with CNN classifiers

- θs → CNN shared parameters (e.g., five convolutional layers and two fully connected layers for AlexNet architecture)

- θo → task-specific parameters for previously learned tasks (e.g., the output layer for ImageNet classification)

- θn → randomly initialized task-specific parameters for new tasks (e.g., scene classifiers)

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

Approaches:

Feature extraction (e.g., [Donahue et al - ICML 2014):

- θs and θo unchanged
- θs and θo outputs used as features for new task in training θn.

Fine-tuning (e.g.,[Girshick et al - CVPR 2014]):

- θs and θn are optimized for new task
- θo fixed
- low learning rate typically used to prevent large drift in θs
- duplicate and fine-tuned for each new task (specialized networks)

Joint Training (e.g., [Caruana 1997):

- θs, θo, θn jointly optimized, for example by interleaving samples from each task.

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

Approaches:

Feature extraction (e.g., [Donahue et al - ICML 2014):
- θs and θo unchanged
- θs and θo outputs used as features for new task in training θn.

Fine-tuning (e.g.,[Girshick et al - CVPR 2014]):
- θs and θn are optimized for new task
- θo fixed
- low learning rate typically used to prevent large drift in θs
- duplicate and fine-tuned for each new task (specialized networks)

Joint Training (e.g., [Caruana 1997):
- θs, θo, θn jointly optimized, for example by interleaving samples from each task.

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

Approaches:

Feature extraction (e.g., [Donahue et al - ICML 2014):
- θs and θo unchanged
- θs and θo outputs used as features for new task in training θn.

Fine-tuning (e.g.,[Girshick et al - CVPR 2014]):
- θs and θn are optimized for new task
- θo fixed
- low learning rate typically used to prevent large drift in θs
- duplicate and fine-tuned for each new task (specialized networks)

Joint Training (e.g., [Caruana 1997]):
- θs, θo, θn jointly optimized, for example by interleaving samples from each task.

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

each of these has a drawback…

Category	Feature Extraction	Fine-Tuning	Duplicate and Fine-Tuning	Joint Training
New task performance	Medium	Good	Good	Good
Old task performance	Good	Bad	Good	Good
Training efficiency	Fast	Fast	Fast	Slow
Testing efficiency	Fast	Fast	Slow	Fast
Storage requirement	Medium	Medium	Large	Large
Require previous task data	No	No	No	Yes

CATASTROPHIC FORGETTING

Li and Hoiem [2016] LWF paper → overview on methods for dealing with catastrophic forgetting: sets of parameters and typical approaches

each of these has a drawback…

Category	Feature Extraction	Fine-Tuning	Duplicate and Fine-Tuning	Joint Training
New task performance	Medium	Good	Good	Good
Old task performance	Good	Bad	Good	Good
Training efficiency	Fast	Fast	Fast	Slow
Testing efficiency	Fast	Fast	Slow	Fast
Storage requirement	Medium	Medium	Large	Large
Require previous task data	No	No	No	Yes

page 56

In light of these, Li and Hoiem [2016] proposed “Learning without Forgetting” algorithm

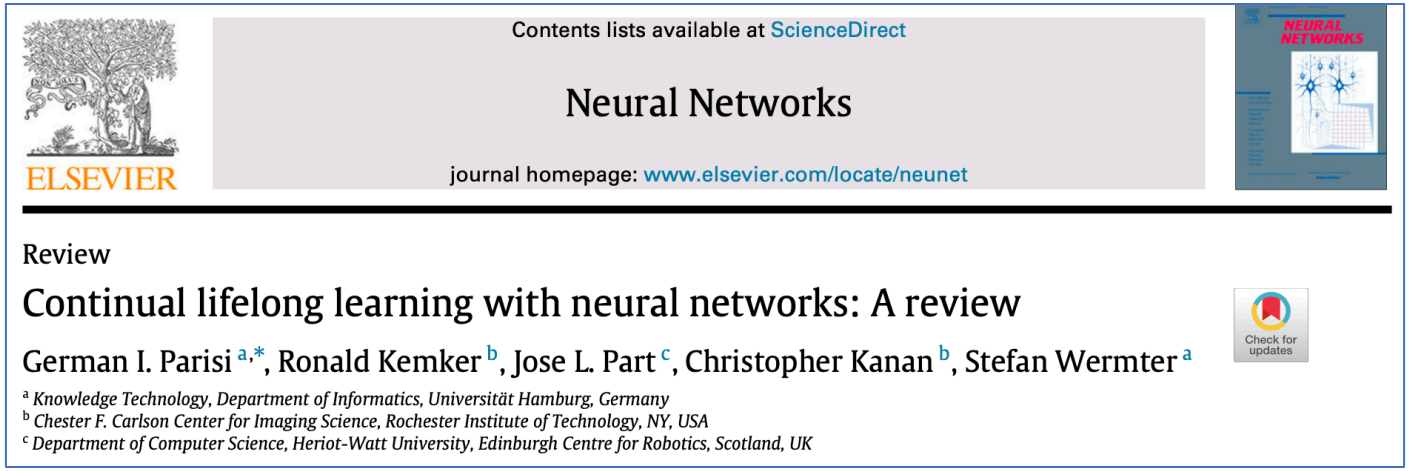
1. Catastrophic Forgetting
2. **Continual Learning in Neural Networks**
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…

comprehensive survey on CL → Parisi et al. 2018



Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Review

Continual lifelong learning with neural networks: A review

German I. Parisi [a,*], Ronald Kemker [b], Jose L. Part [c], Christopher Kanan [b], Stefan Wermter [a]

[a] Knowledge Technology, Department of Informatics, Universität Hamburg, Germany
[b] Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, NY, USA
[c] Department of Computer Science, Heriot-Watt University, Edinburgh Centre for Robotics, Scotland, UK

Parisi et al. 2018

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…

comprehensive survey on CL → Parisi et al. 2018

Contents

Parisi et al. 2018

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…

comprehensive survey on CL → Parisi et al. 2018

Contents

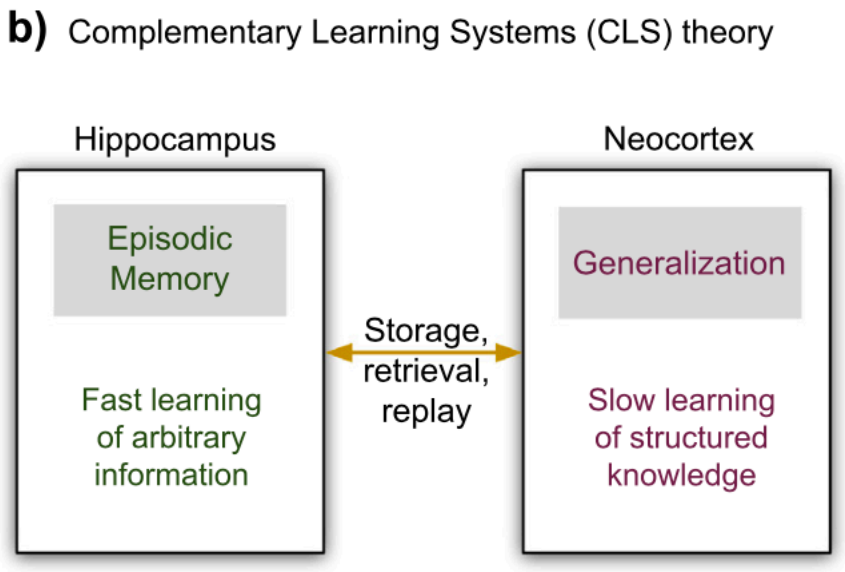
Parisi et al. 2018

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…

comprehensive survey on CL → Parisi et al. 2018



Parisi et al. 2018

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…

comprehensive survey on CL → Parisi et al. 2018

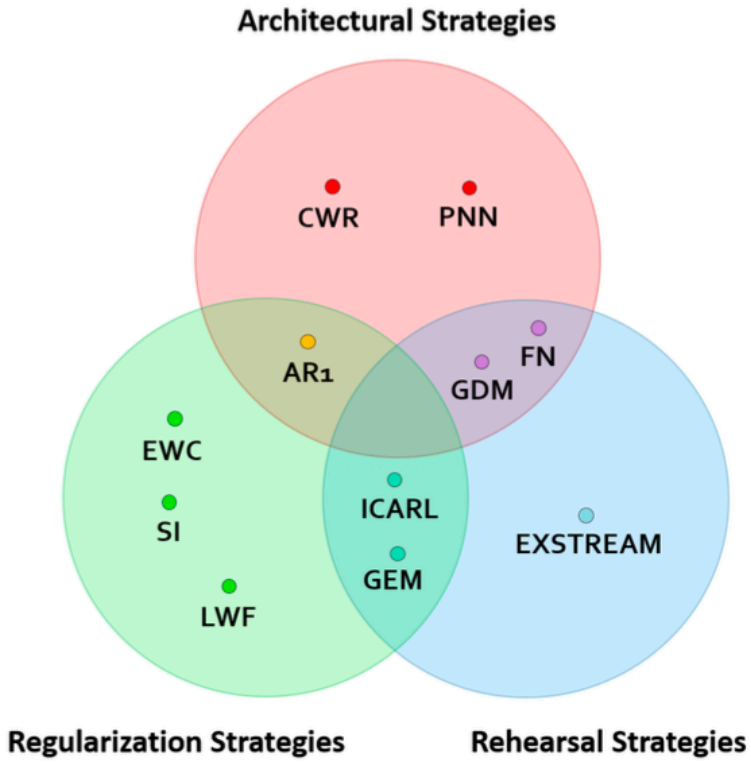
Contents

Parisi et al. 2018

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…

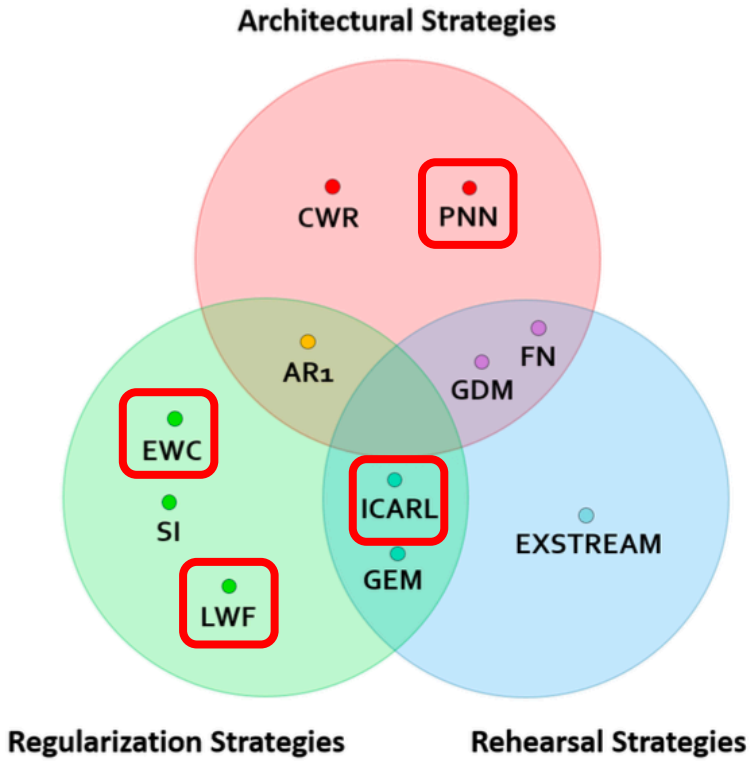


Maltoni, Lomonaco 2019

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

recently!
2016, 2017, 2018…



Maltoni, Lomonaco 2019

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

much of the existing work → supervised learning

Rusu et al. 2016
PNN - progressive neural network that retains models and learns lateral connections

Kirkpatrick et al. 2017
EWC - Elastic Weight Consolidation. Quantifies importance of weights to previous tasks, and selectively adjusts the plasticity of weights.

Rebuffi et al. 2017
Retain an exemplar set that best approximates previous tasks.

Aljundi et al. 2016
Expert gate - Measure task relatedness.

Rannen Ep Triki et al. 2017
Autoencoder to extend LWF [Li and Hoiem, 2016].

Shin et al. 2017
Generative Adversarial Networks (GANs). Generators for previous tasks, and then learn parameters that fit a mixed set of real data of the new task and replayed data of previous tasks.

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

much of the existing work → supervised learning

Jung et al. [2016]
"less forgetful" learning that regularizes some final hidden activations.

Rosenfeld and Tsotsos. [2017]
proposed controller modules to optimize loss on the new task with representations learned from previous tasks. They found that they could achieve satisfactory performance while only requiring about 22% of parameters of the fine-tuning method

Ans et al. [2004], Jin and Sendhoff [2006]
Dual-network architecture to generate pseudo-items which are used to self-refresh the previous tasks

Nguyen et al. [2017]
slight variation on EWC [Kirkpatrick et al. 2017]

Zenke et al. [2017]
Measures synapse consolidation strength in an online fashion and uses it as regularization in neural networks

CONTINUAL LEARNING IN NEURAL NETWORKS

many CL approaches have been proposed to deal with catastrophic forgetting

- dual-memory-based learning systems
- inspired by the complementary learning systems (CLS) theory [Kumaran et al., 2016, McClelland et al., 1995]
- interplay of hippocampus (short-term memory) and neocortex (long-term memory)

Gepperth and Karaoguz [2016]
Modified self-organizing map (SOM) as the long-term memory (LTM) and a short-term memory (STM) to store novel examples. During "sleep" phase, STM content is replayed to the system (intrinsic replay or pseudo-rehearsal [Robins, 1995]).
The replayed samples prevents the network from forgetting.

Kemker and Kanan [2018] - FearNet
Dual-memory system.
"Hippocampal" network for STM
"medial prefrontal cortex (mPFC)" network for LTM
third neural network to determine which memory to use for prediction.

Shin 2017
Deep Generative Replay

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. **Learning Without Forgetting**
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

The idea:
- optimize θs and θn on new task
- constraint: predictions on new task's examples using θs and θo do not shift much

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

The idea:
- optimize θs and θn on new task
- constraint: predictions on new task's examples using θs and θo do not shift much

This constraint helps the model "remember" its old parameters and maintain nice performance on previous tasks.

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

The idea:
- optimize θs and θn on new task
- constraint: predictions on new task's examples using θs and θo do not shift much

This constraint helps the model "remember" its old parameters and maintain nice performance on previous tasks.

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

Algorithm 4.1 Learning without Forgetting

Input: shared parameters θ_s, task-specific parameters for old tasks θ_o, training data X_n, Y_n for the new task.
Output: updated parameters θ_s^*, θ_o^*, θ_n^*.

1: // Initialization phase.
2: $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$
3: $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$
4: // Training phase.
5: Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$
6: Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$
7: $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \text{argmin}_{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n} \left(\mathcal{L}_{new}(\hat{Y}_n, Y_n) + \lambda_o \mathcal{L}_{old}(\hat{Y}_o, Y_o) + \mathcal{R}(\theta_s, \theta_o, \theta_n) \right)$

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

The idea:
- optimize θs and θn on new task
- constraint: predictions on new task's examples using θs and θo do not shift much

This constraint helps the model "remember" its old parameters and maintain nice performance on previous tasks.

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

page 55

Algorithm 4.1 Learning without Forgetting

Input: shared parameters θ_s, task-specific parameters for old tasks θ_o, training data X_n, Y_n for the new task.
Output: updated parameters θ_s^*, θ_o^*, θ_n^*.

1: // Initialization phase.
2: $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$
3: $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$
4: // Training phase.
5: Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$
6: Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$
7: $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \text{argmin}_{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n} \left(\mathcal{L}_{new}(\hat{Y}_n, Y_n) + \lambda_o \mathcal{L}_{old}(\hat{Y}_o, Y_o) + \mathcal{R}(\theta_s, \theta_o, \theta_n) \right)$

- $\mathcal{L}_{new}(\hat{Y}_n, Y_n)$: minimize the difference between the predicted values $\hat{Y}_n$ and the groundtruth Y_n. $\hat{Y}_n$ is the predicted value using the current parameters $\hat{\theta}_s$ and $\hat{\theta}_n$ (Line 5). In Li and Hoiem [2016], the multinomial logistic loss is used:

$$\mathcal{L}_{new}(\hat{Y}_n, Y_n) = -Y_n \cdot \log \hat{Y}_n .$$

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

The idea:
- optimize θs and θn on new task
- constraint: predictions on new task's examples using θs and θo do not shift much

This constraint helps the model "remember" its old parameters and maintain nice performance on previous tasks.

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

Algorithm 4.1 Learning without Forgetting

Input: shared parameters θ_s, task-specific parameters for old tasks θ_o, training data X_n, Y_n for the new task.
Output: updated parameters θ_s^*, θ_o^*, θ_n^*.

1: // Initialization phase.
2: $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$
3: $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$
4: // Training phase.
5: Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$
6: Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$
7: $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \text{argmin}_{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n} \left(\mathcal{L}_{new}(\hat{Y}_n, Y_n) + \lambda_o \mathcal{L}_{old}(\hat{Y}_o, Y_o) + \mathcal{R}(\theta_s, \theta_o, \theta_n) \right)$

- $\mathcal{L}_{old}(\hat{Y}_o, Y_o)$: minimize the difference between the predicted values $\hat{Y}_o$ and the recorded values Y_o (Line 2), where $\hat{Y}_o$ comes from the current parameters $\hat{\theta}_s$ and $\hat{\theta}_o$ (Line 6). Li and Hoiem [2016] used knowledge distillation loss [Hinton et al., 2015] to encourage the outputs of one network to approximate the outputs of another. The distillation loss is defined as modified cross-entropy loss:

$$\mathcal{L}_{old}(\hat{Y}_o, Y_o) = -H(\hat{Y}_o', Y_o') = -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)},$$

LEARNING WITHOUT FORGETTING

Li and Hoiem [2016]

The idea:
- optimize θs and θn on new task
- constraint: predictions on new task's examples using θs and θo do not shift much

This constraint helps the model "remember" its old parameters and maintain nice performance on previous tasks.

- θ_s: set of parameters shared across all tasks;
- θ_o: set of parameters learned specifically for previous tasks; and
- θ_n: randomly initialized task-specific parameters for new tasks.

Algorithm 4.1 Learning without Forgetting

Input: shared parameters θ_s, task-specific parameters for old tasks θ_o, training data X_n, Y_n for the new task.
Output: updated parameters θ_s^*, θ_o^*, θ_n^*.

1: // Initialization phase.
2: $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$
3: $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$
4: // Training phase.
5: Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$
6: Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$
7: $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \text{argmin}_{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n} \left(\mathcal{L}_{new}(\hat{Y}_n, Y_n) + \lambda_o \mathcal{L}_{old}(\hat{Y}_o, Y_o) + \mathcal{R}(\theta_s, \theta_o, \theta_n) \right)$

- $\mathcal{R}(\theta_s, \theta_o, \theta_n)$: regularization term to avoid overfitting.

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. **Progressive Neural Networks**
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

PROGRESSIVE NEURAL NETWORKS

Rusu et al. [2016] (Google)

The idea:

- keep a pool of models as “knowledge”
- use lateral connections between them to adapt to the new task.

PROGRESSIVE NEURAL NETWORKS

Rusu et al. [2016] (Google)

The idea:

- keep a pool of models as “knowledge”
- use lateral connections between them to adapt to the new task.

N past tasks: T_1, T_2, …,T_N → PNN maintain N neural networks

when new task T_{N+1} is created → neural network is expanded with lateral connections

PROGRESSIVE NEURAL NETWORKS

Rusu et al. [2016] (Google)

The idea:

- keep a pool of models as “knowledge”
- use lateral connections between them to adapt to the new task.

N past tasks: T_1, T_2, …,T_N → PNN maintain N neural networks

when new task T_{N+1} is created → neural network is expanded with lateral connections

$$h_i^{(N+1)} = \max\left(0, W_i^{(N+1)} h_{i-1}^{(N+1)} + \sum_{n<N+1} U_i^{(n:N+1)} h_{i-1}^{(n)}\right)$$

PROGRESSIVE NEURAL NETWORKS

Rusu et al. [2016] (Google)

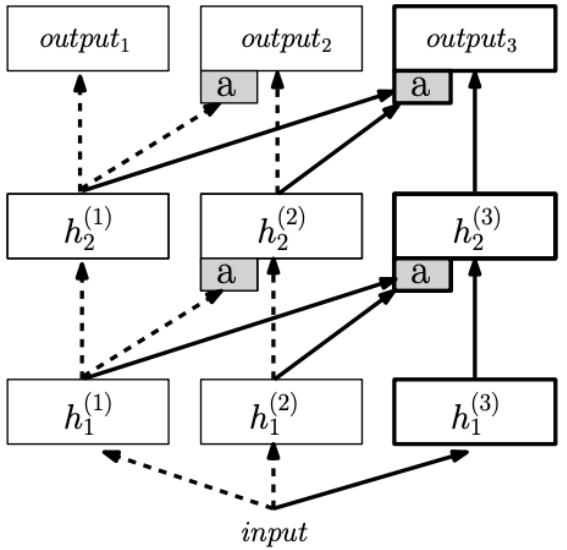
The idea:

- keep a pool of models as “knowledge”
- use lateral connections between them to adapt to the new task.

N past tasks: T_1, T_2, …,T_N → PNN maintain N neural networks

when new task T_{N+1} is created → neural network is expanded with lateral connections

$$h_i^{(N+1)} = \max\left(0, W_i^{(N+1)} h_{i-1}^{(N+1)} + \sum_{n<N+1} U_i^{(n:N+1)} h_{i-1}^{(n)}\right)$$



1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. **Elastic Weight Consolidation**
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

ELASTIC WEIGHT CONSOLIDATION

Kirkpatrick et al. [2017]

The idea:
inspired by human brain in which synaptic consolidation enables continual learning by reducing the plasticity of synapses related to previous learned tasks

plasticity of weights closely related to previous tasks → more prone to cause catastrophic forgetting

ELASTIC WEIGHT CONSOLIDATION

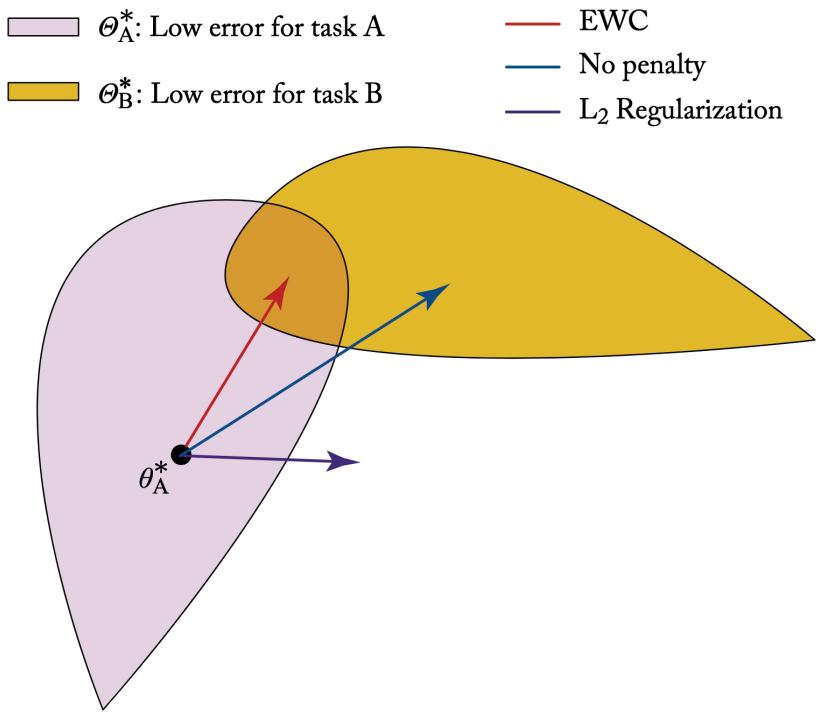
Kirkpatrick et al. [2017]

The idea:
inspired by human brain in which synaptic consolidation enables continual learning by reducing the plasticity of synapses related to previous learned tasks

plasticity of weights closely related to previous tasks → more prone to cause catastrophic forgetting

EWC:

- quantify weights importance in terms of impact on previous tasks
- constraint parameters to low-error task regions
- selectively decrease the plasticity of those important weights to previous tasks



ELASTIC WEIGHT CONSOLIDATION

Kirkpatrick et al. [2017]

The idea:
inspired by human brain in which synaptic consolidation enables continual learning by reducing the plasticity of synapses related to previous learned tasks

plasticity of weights closely related to previous tasks → more prone to cause catastrophic forgetting

EWC:

- quantify weights importance in terms of impact on previous tasks
- constraint parameters to low-error task regions
- selectively decrease the plasticity of those important weights to previous tasks

importance modeled as posterior distr. P(θ|D)
(prob. of parameter θ given a task's training data D)

Using Bayes rule:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D})$$

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}_{\boldsymbol{B}}|\theta) + \log p(\theta|\mathcal{D}_{\boldsymbol{A}}) - \log p(\mathcal{D}_{\boldsymbol{B}})$$

ELASTIC WEIGHT CONSOLIDATION

Kirkpatrick et al. [2017]

The idea:
inspired by human brain in which synaptic consolidation enables continual learning by reducing the plasticity of synapses related to previous learned tasks

plasticity of weights closely related to previous tasks → more prone to cause catastrophic forgetting

EWC:

- quantify weights importance in terms of impact on previous tasks
- constraint parameters to low-error task regions
- selectively decrease the plasticity of those important weights to previous tasks

importance modeled as posterior distr. P(θ|D)
(prob. of parameter θ given a task's training data D)

Using Bayes rule:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D})$$

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B)$$

all the information related to task A
intractable…

ELASTIC WEIGHT CONSOLIDATION

Kirkpatrick et al. [2017]

The idea:
inspired by human brain in which synaptic consolidation enables continual learning by reducing the plasticity of synapses related to previous learned tasks

plasticity of weights closely related to previous tasks → more prone to cause catastrophic forgetting

EWC:

- quantify weights importance in terms of impact on previous tasks
- constraint parameters to low-error task regions
- selectively decrease the plasticity of those important weights to previous tasks

importance modeled as posterior distr. P(θ|D)
(prob. of parameter θ given a task's training data D)

Using Bayes rule:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D})$$

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B)$$

all the information related to task A
intractable…

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. **iCaRL: Incremental Classifier and Representation Learning**
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

“child visiting the zoo will learn about many new animals without forgetting the pet it has at home”

Rebuffi et al. [2017]

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

3 properties of a class-incremental algorithm:

1. it should be trainable from a stream of data in which examples of different classes occur at different times
2. it should at any time provide a competitive multi-class classifier for the classes observed so far
3. its computational requirements and memory footprint should remain bounded (or grow slowly) w.r.t. the number of classes seen so far

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

3 properties of a class-incremental algorithm:

1. it should be trainable from a stream of data in which examples of different classes occur at different times

2. it should at any time provide a competitive multi-class classifier for the classes observed so far

3. its computational requirements and memory footprint should remain bounded (or grow slowly) w.r.t. the number of classes seen so far

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

3 properties of a class-incremental algorithm:

1. it should be trainable from a stream of data in which examples of different classes occur at different times

2. it should at any time provide a competitive multi-class classifier for the classes observed so far

3. its computational requirements and memory footprint should remain bounded (or grow slowly) w.r.t. the number of classes seen so far

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

3 properties of a class-incremental algorithm:

1. it should be trainable from a stream of data in which examples of different classes occur at different times

2. it should at any time provide a competitive multi-class classifier for the classes observed so far

essence of class incremental learning

3. its computational requirements and memory footprint should remain bounded (or grow slowly) w.r.t. the number of classes seen so far

prevents trivial solutions as storing all training examples and full retraining

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

- few existing techniques fulfill 3 properties…
- they are limited to situations with a fixed data representation
- iCaRL: learn classifiers and feature representations at the same time

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

- few existing techniques fulfill 3 properties…
- they are limited to situations with a fixed data representation
- iCaRL: learn classifiers and feature representations at the same time
  - maintains a set of exemplar examples for each observed class
  - for each class, an exemplar set is a subset of all examples of the class (most representative)
  - new example classified by choosing class whose exemplars are the most similar to it
  - new class found → exemplar sets of the previous classes trimmed (number of exemplars is fixed)

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

$\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$
$m \leftarrow K/t$
for $y = 1$ **to** $s - 1$ **do**
    $P_y \leftarrow P_y[1 : m]$
end for
for $y = s$ **to** t **do**
    $P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$
end for
$\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

1: $\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$ → Update representation model
2: $m \leftarrow K/t$
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $\quad P_y \leftarrow P_y[1 : m]$
5: **end for**
6: **for** $y = s$ **to** t **do**
7: $\quad P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$
8: **end for**
9: $\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

1: $\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$ → Update representation model
2: $m \leftarrow K/t$ → For each existing class, reduce number of exemplars per class to m
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $\quad P_y \leftarrow P_y[1 : m]$
5: **end for**
6: **for** $y = s$ **to** t **do**
7: $\quad P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$
8: **end for**
9: $\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

1: $\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$ → Update representation model
2: $m \leftarrow K/t$ → For each existing class, reduce number of exemplars per class to m
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $\quad P_y \leftarrow P_y[1 : m]$ → keep first m exemplars for each class
5: **end for**
6: **for** $y = s$ **to** t **do**
7: $\quad P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$
8: **end for**
9: $\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

1: $\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$ → Update representation model
2: $m \leftarrow K/t$ → For each existing class, reduce number of exemplars per class to m
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $\quad P_y \leftarrow P_y[1 : m]$ → keep first m exemplars for each class
5: **end for**
6: **for** $y = s$ **to** t **do**
7: $\quad P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$ → build exemplar set for each new class
8: **end for**
9: $\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

1: $\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$ → Update representation model
2: $m \leftarrow K/t$ → For each existing class, reduce number of exemplars per class to m
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $\quad P_y \leftarrow P_y[1 : m]$ → keep first m exemplars for each class
5: **end for**
6: **for** $y = s$ **to** t **do**
7: $\quad P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$ → build exemplar set for each new class
8: **end for**
9: $\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.3 iCaRL UpdateRepresentation

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ.

$\mathcal{D}^{exemplar} \leftarrow \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P_y\}$
$\mathcal{D}^{new} \leftarrow \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\}$
for $y = 1$ **to** $s - 1$ **do**
  $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}^{exemplar}$
end for
$\mathcal{D}^{train} \leftarrow \mathcal{D}^{exemplar} \cup \mathcal{D}^{new}$
Run network training (e.g., Backpropagation) with loss function that contains *classification* and *distillation* terms:

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

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.3 iCaRL UpdateRepresentation

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ.

1: $\mathcal{D}^{exemplar} \leftarrow \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P_y\}$ → dataset with all existing exemplars (original feature space, not learned representation)
2: $\mathcal{D}^{new} \leftarrow \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\}$ → dataset with new examples of new classes (original feature space, not learned representation)
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}^{exemplar}$
5: **end for**
6: $\mathcal{D}^{train} \leftarrow \mathcal{D}^{exemplar} \cup \mathcal{D}^{new}$
7: Run network training (e.g., Backpropagation) with loss function that contains *classification* and *distillation* terms:

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

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.3 iCaRL UpdateRepresentation

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ.

1: $\mathcal{D}^{exemplar} \leftarrow \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P_y\}$ → dataset with all existing exemplars (original feature space, not learned representation)
2: $\mathcal{D}^{new} \leftarrow \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\}$ → dataset with new examples of new classes (original feature space, not learned representation)
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}^{exemplar}$ → prediction output of each existing exemplar with current model
5: **end for**
6: $\mathcal{D}^{train} \leftarrow \mathcal{D}^{exemplar} \cup \mathcal{D}^{new}$
7: Run network training (e.g., Backpropagation) with loss function that contains *classification* and *distillation* terms:

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

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.3 iCaRL UpdateRepresentation

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ.

1: $\mathcal{D}^{exemplar} \leftarrow \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P_y\}$ → dataset with all existing exemplars (original feature space) not learned representation)
2: $\mathcal{D}^{new} \leftarrow \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\}$ → dataset with new examples of new classes (original feature space, not learned representation)
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}^{exemplar}$ → prediction output of each existing exemplar with current model
5: **end for**
6: $\mathcal{D}^{train} \leftarrow \mathcal{D}^{exemplar} \cup \mathcal{D}^{new}$
7: Run network training (e.g., Backpropagation) with loss function that contains *classification* and *distillation* terms:

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

Rebuffi et al. [2017]

CNN as feature extractor:

$$\varphi : \mathcal{X} \rightarrow \mathbb{R}^d$$

sigmoid classification layer:

$$g_y(x) = \frac{1}{1 + exp(-a_y(x))}$$

with $a_y(x) = w_y^{\mathrm{T}} \varphi(x)$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.3 iCaRL UpdateRepresentation

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ.

1: $\mathcal{D}^{exemplar} \leftarrow \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P_y\}$ → dataset with all existing exemplars (original feature space) not learned representation)
2: $\mathcal{D}^{new} \leftarrow \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\}$ → dataset with new examples of new classes (original feature space, not learned representation)
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}^{exemplar}$ → prediction output of each existing exemplar with current model
5: **end for**
6: $\mathcal{D}^{train} \leftarrow \mathcal{D}^{exemplar} \cup \mathcal{D}^{new}$
7: Run network training (e.g., Backpropagation) with loss function that contains *classification* and *distillation* terms:

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

Rebuffi et al. [2017]

CNN as feature extractor:

$$\varphi : \mathcal{X} \rightarrow \mathbb{R}^d$$

sigmoid classification layer:

$$g_y(x) = \frac{1}{1 + exp(-a_y(x))}$$

with $a_y(x) = w_y^{\mathrm{T}} \varphi(x)$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.2 iCaRL IncrementalTraining

Input: new training examples $X^s, \ldots, X^t$ of new classes $s, \ldots, t$, current model parameters Θ, current exemplar sets $\mathcal{P} = \{P_1, \ldots, P_{s-1}\}$, memory size K.
Output: updated model parameters Θ, updated exemplar sets $\mathcal{P}$.

1: $\Theta \leftarrow \text{UpdateReresentation}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$ → Update representation model
2: $m \leftarrow K/t$ → For each existing class, reduce number of exemplars per class to m
3: **for** $y = 1$ **to** $s - 1$ **do**
4: $\quad P_y \leftarrow P_y[1 : m]$ → keep first m exemplars for each class
5: **end for**
6: **for** $y = s$ **to** t **do**
7: $\quad P_y \leftarrow \text{ConstructExemplarSet}(X^y, m, \Theta)$ → build exemplar set for each new class
8: **end for**
9: $\mathcal{P} \leftarrow \{P_1, \ldots, P_t\}$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.4 iCaRL ConstructExemplarSet

Input: examples $X = \{x_1, \ldots, x_n\}$ of class t, the target number of exemplars m, current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$.
Output: exemplar set P for class y.

1: $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$
2: **for** $k = 1$ **to** m **do**
3: $\quad p_k \leftarrow \text{argmin}_{x \in X \text{ and } x \notin \{p_1, \ldots, p_{k-1}\}} \left|\left| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right|\right|$
4: **end for**
5: $P \leftarrow (p_1, \ldots, p_m)$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.4 iCaRL ConstructExemplarSet

Input: examples $X = \{x_1, \ldots, x_n\}$ of class t, the target number of exemplars m, current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$.
Output: exemplar set P for class y.

1: $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ → Average feature vector of all training examples of new class t
2: **for** $k = 1$ **to** m **do**
3: $p_k \leftarrow \text{argmin}_{x \in X \text{ and } x \notin \{p_1, \ldots, p_{k-1}\}} \left|\left| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right|\right|$
4: **end for**
5: $P \leftarrow (p_1, \ldots, p_m)$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.4 iCaRL ConstructExemplarSet

Input: examples $X = \{x_1, \ldots, x_n\}$ of class t, the target number of exemplars m, current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$.
Output: exemplar set P for class y.

1: $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ → Average feature vector of all training examples of new class t
2: **for** $k = 1$ **to** m **do**
3: $p_k \leftarrow \text{argmin}_{x \in X \text{ and } x \notin \{p_1, \ldots, p_{k-1}\}} \left|\left| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right|\right|$ → m exemplars selected whose average feature vector is the closest to /mu
4: **end for**
5: $P \leftarrow (p_1, \ldots, p_m)$

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.4 iCaRL ConstructExemplarSet

Input: examples $X = \{x_1, \ldots, x_n\}$ of class t, the target number of exemplars m, current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$.
Output: exemplar set P for class y.

1: $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ → Average feature vector of all training examples of new class t
2: **for** $k = 1$ **to** m **do**
3: $p_k \leftarrow \text{argmin}_{x \in X \text{ and } x \notin \{p_1, \ldots, p_{k-1}\}} \left|\left| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right|\right|$ → m exemplars selected whose average feature vector is the closest to /mu ?
4: **end for**
5: $P \leftarrow (p_1, \ldots, p_m)$

Here is the intuition of how the selection of exemplars works: the average feature vector over all exemplars should be close to the average feature vector over all examples of the class. As

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.4 iCaRL ConstructExemplarSet

Input: examples $X = \{x_1, \ldots, x_n\}$ of class t, the target number of exemplars m, current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$.
Output: exemplar set P for class y.

1: $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ → Average feature vector of all training examples of new class t
2: **for** $k = 1$ **to** m **do**
3: $p_k \leftarrow \text{argmin}_{x \in X \text{ and } x \notin \{p_1, \ldots, p_{k-1}\}} \left|\left| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right|\right|$ → m exemplars selected whose average feature vector is the closest to /mu
4: **end for**
5: $P \leftarrow (p_1, \ldots, p_m)$ → exemplar set

ICARL: INCREMENTAL CLASSIFIER AND REPRESENTATION LEARNING

Rebuffi et al. [2017]

The idea: training strategy that allows class incremental learning: only training data for few classes needed, and new classes can be added progressively.

"child visiting the zoo will learn about many new animals without forgetting the pet it has at home"

Rebuffi et al. [2017]

Algorithm 4.5 iCaRL Classify in iCaRL

Input: a test example x to be classified, sets of exemplars $\mathcal{P} = \{P_1, \ldots, P_t\}$, current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$.
Output: predicted class label y^* of x.

1: **for** $y = 1$ **to** t **do**
2: $\quad \mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$ → Average feature vector of every exemplar set
3: **end for**
4: $y^* \leftarrow \underset{y=1,\ldots,t}{\operatorname{argmin}} ||\varphi(x) - \mu_y||$ → pick the class whose exemplar set average is the closest to x

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. **Expert Gate**
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model
trained given a specific task (i.e. good at this particular task)

system need to know what model to load when making a prediction on a test example…

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)

system need to know what model to load when making a prediction on a test example…

Expert Gate algorithm determines the relevance of tasks, and only load the most relevant model in memory during inference

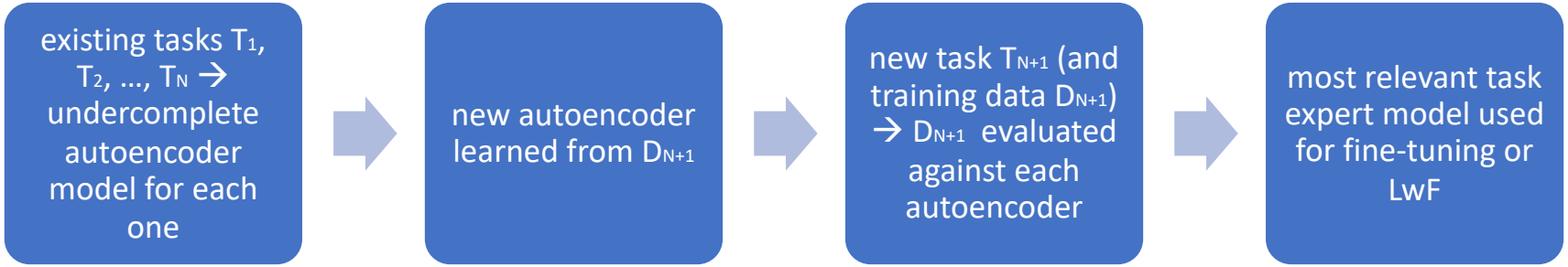
EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)

system need to know what model to load when making a prediction on a test example…

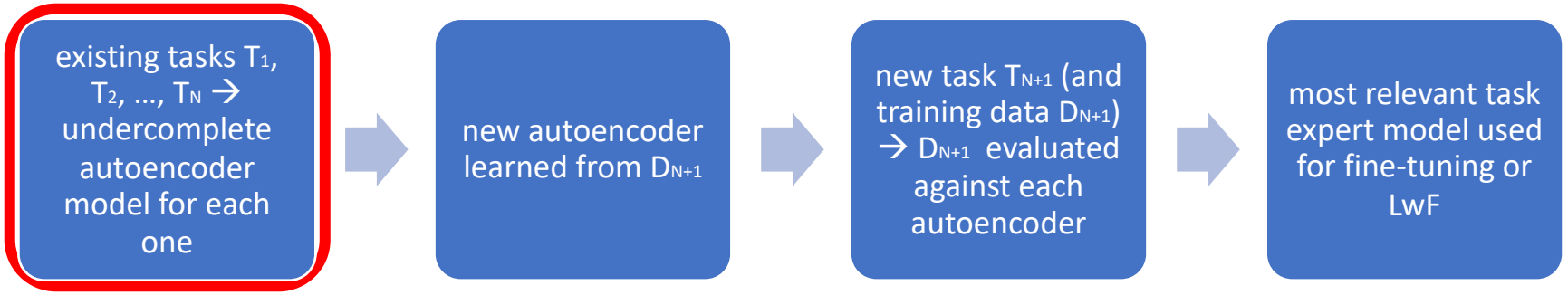
Expert Gate algorithm determines the relevance of tasks, and only load the <u>most relevant model</u> in memory during inference



EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



intuition: undercomplete autoencoder learn lower-dimensional feature representation (describes the data in a compact way)

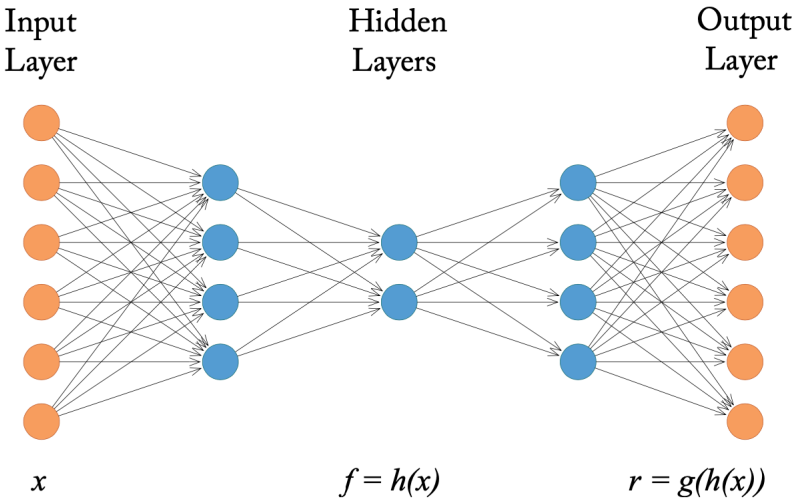
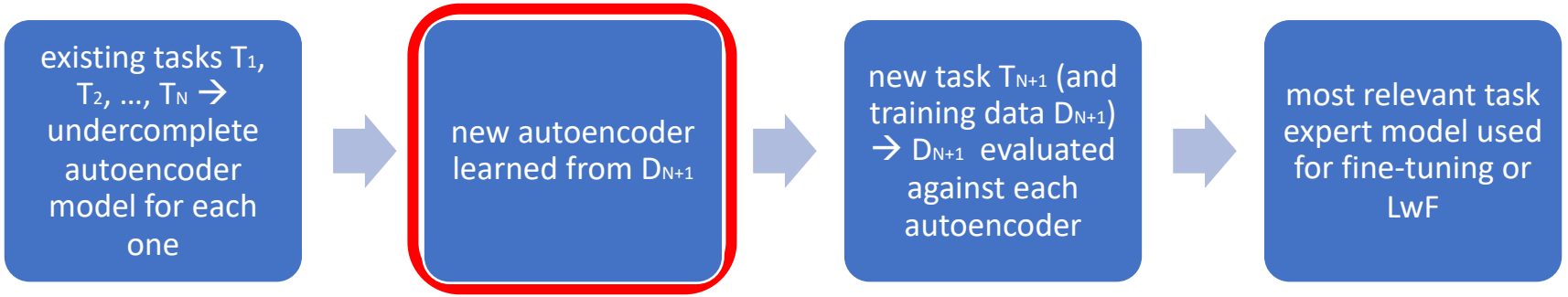


Figure 4.2: An example of undercomplete autoencoder model.

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



intuition: undercomplete autoencoder learn lower-dimensional feature representation (describes the data in a compact way)

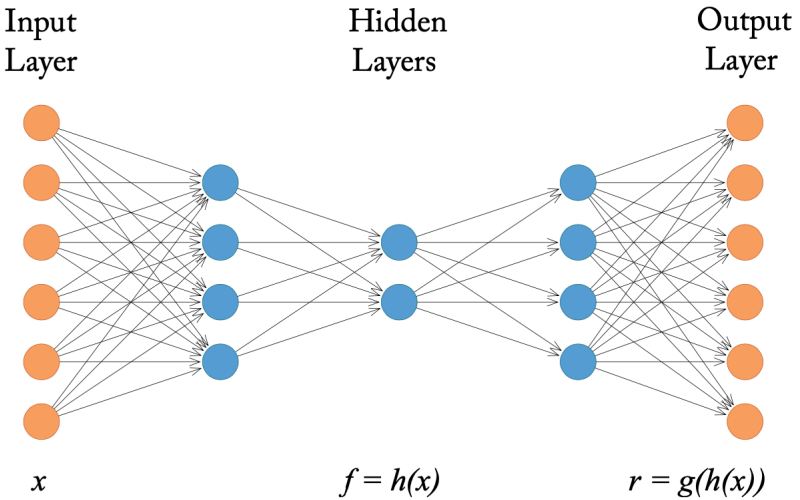
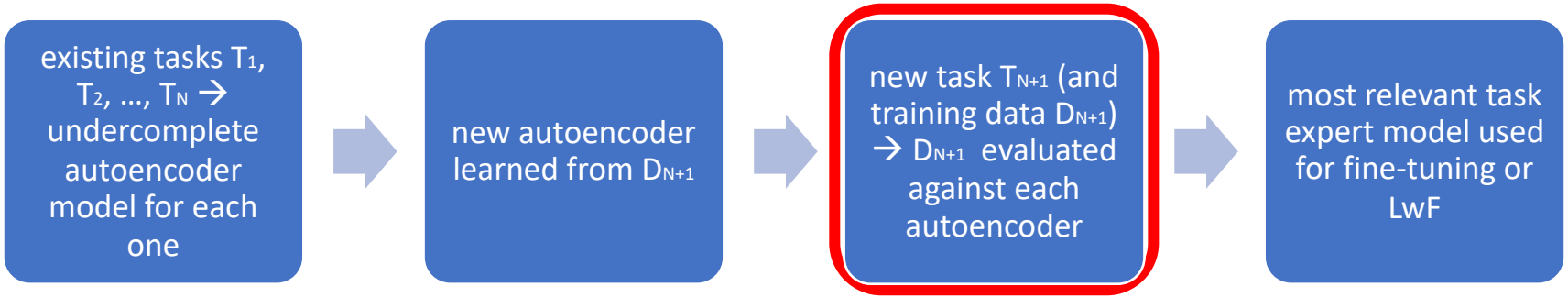


Figure 4.2: An example of undercomplete autoencoder model.

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



reconstruction error of database D using an autoencoder A_k:

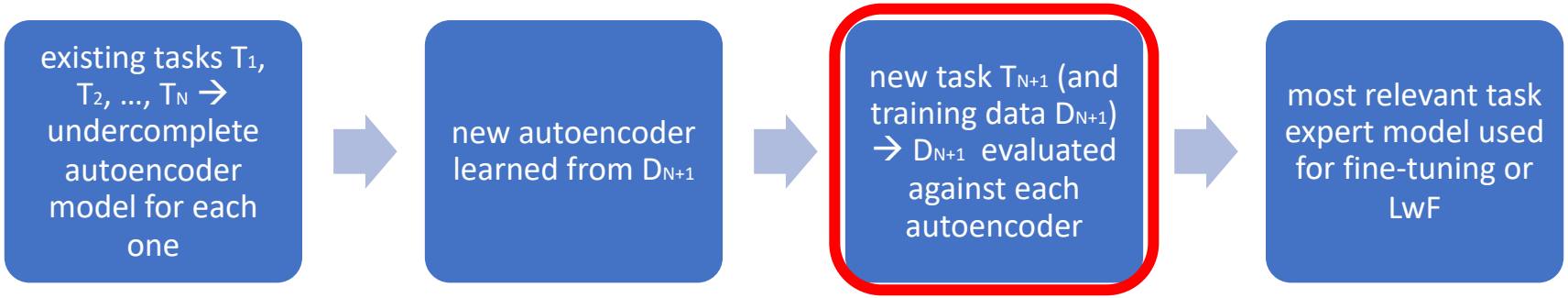
$$Er_k = \frac{\sum_{x \in \mathcal{D}} er_x^k}{|\mathcal{D}|}$$

er_x^k is the reconstruction error of applying x to the autoencoder A_k

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



Task relatedness:

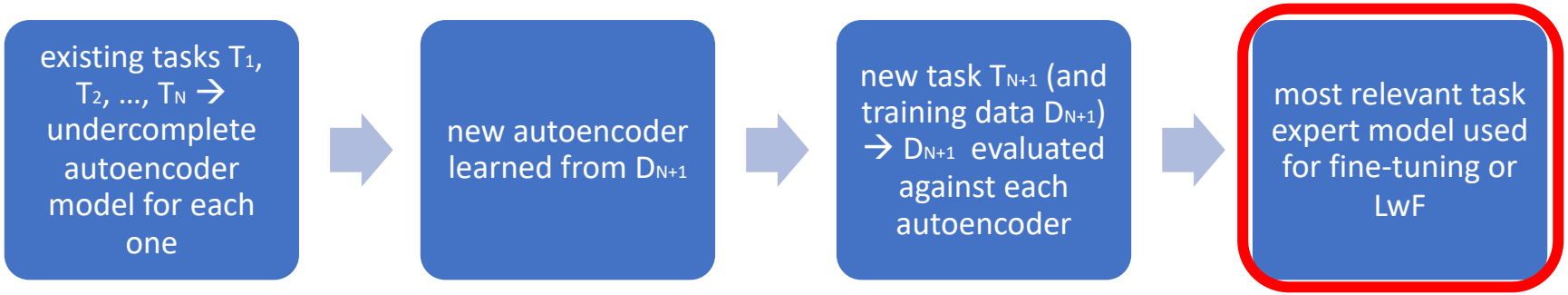
$$Relatedness(\mathcal{T}_{N+1}, \mathcal{T}_k) = 1 - \frac{Er_{N+1} - Er_k}{Er_k}$$

data of existing tasks are discarded → only D_{N+1} is used to evaluate relatedness

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



Task relatedness:

$$Relatedness(\mathcal{T}_{N+1}, \mathcal{T}_k) = 1 - \frac{Er_{N+1} - Er_k}{Er_k}$$

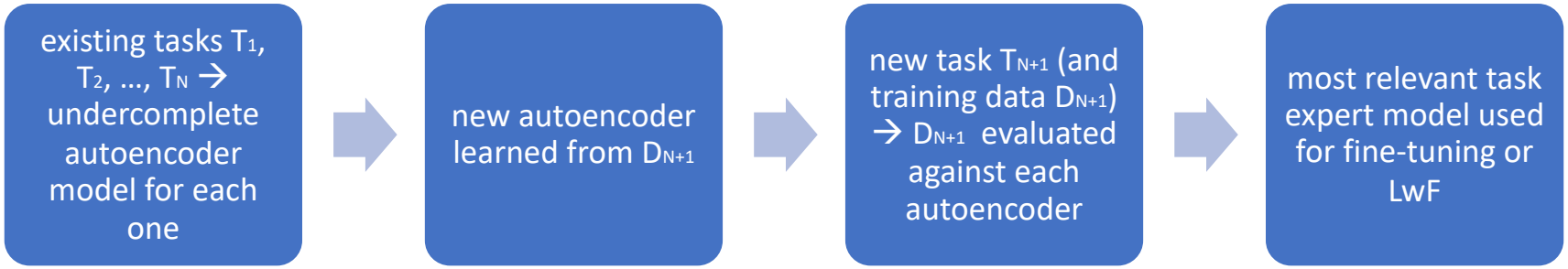
data of existing tasks are discarded → only D_{N+1} is used to evaluate relatedness

two tasks are sufficiently related → LwF is applied;
otherwise, fine-tuning

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



test phase

- If X_t yields a small reconstruction error in autoencoder A_k, → X_t similar to data used to train A_k
- specialized expert E_k should hence be used to make predictions on X_t

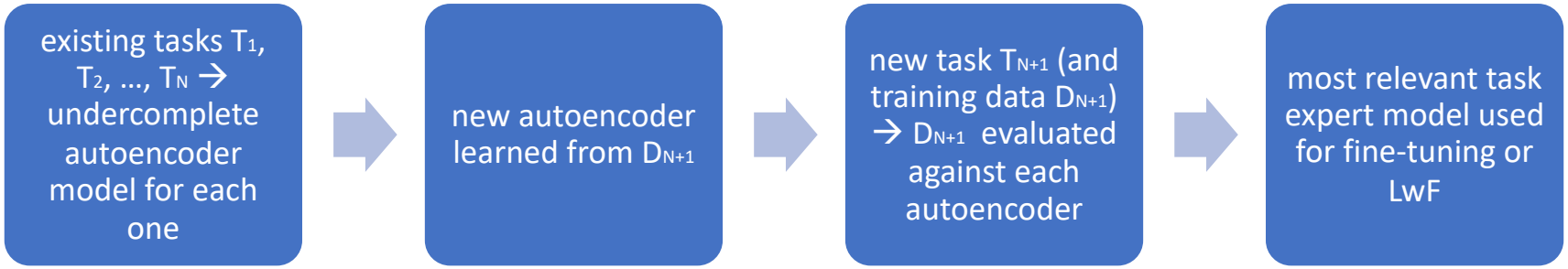
the probability p_k of X_t being relevant to an expert E_k:

$$p_k = \frac{exp(-er_{x_t}^k/\tau)}{\sum_j exp(-er_{x_t}^j/\tau)}$$

EXPERT GATE

Aljundi et al. [2016]

The idea: Network of Experts where each expert is a model trained given a specific task (i.e. good at this particular task)



Algorithm 1 Expert Gate

Training Phase input: expert-models $(E_1, ., E_j)$, tasks-autoencoders $(A_1, ., A_j)$, new task (T_k), data (D_k) ; output: E_k

1: A_k =train-task-autoencoder (D_k)
2: (*rel*,*rel-val*)=select-most-related-task(D_k,A_k,{A})
3: **if** *rel-val* >*rel-th* **then**
4: E_k=LwF(E_{rel}, D_k)
5: **else**
6: E_k=fine-tune(E_{rel}, D_k)
7: **end if**

Test Phase input: x ; output: prediction

8: i=select-expert$(\{A\}, x)$
9: prediction = activate-expert(E_i, x)

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. **Continual Learning with Generating Replay**
9. Evaluating Catastrophic Forgetting
10. Evaluation Datasets

CONTINUAL LEARNING WITH GENERATIVE REPLAY

Shin et al. [2017]

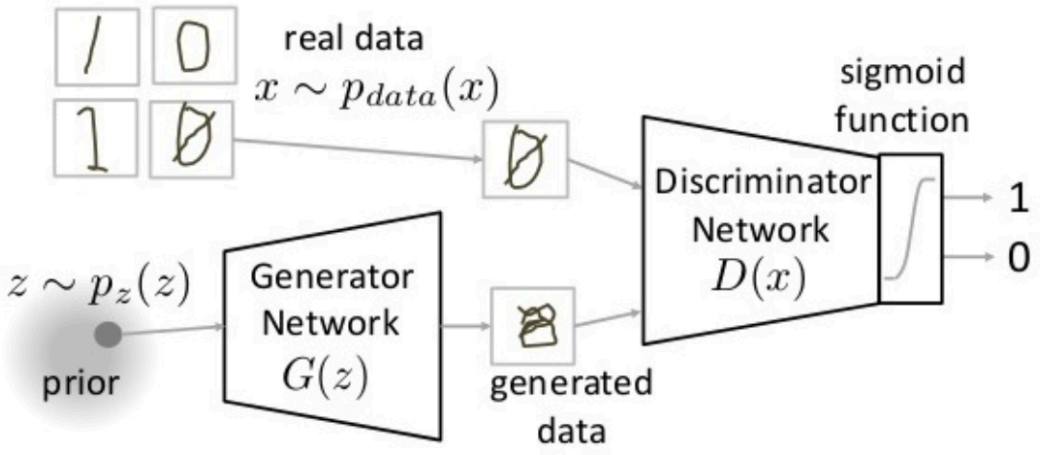
The idea: replay examples from a generative model
without referring to the actual data of past tasks

CONTINUAL LEARNING WITH GENERATIVE REPLAY

Shin et al. [2017]

The idea: replay examples from a generative model
without referring to the actual data of past tasks

Generative Adversarial Networks (GANs) - Overview



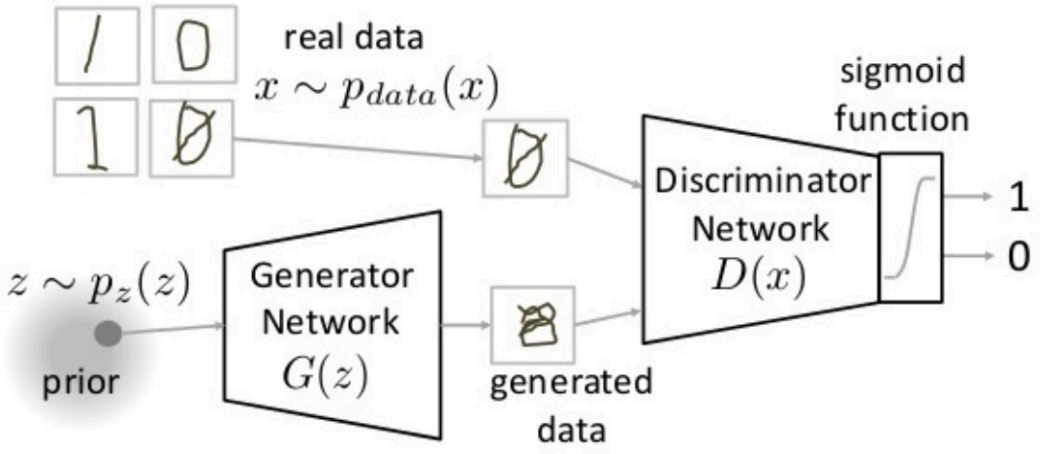
https://robotronblog.com/2017/09/05/gans/

CONTINUAL LEARNING WITH GENERATIVE REPLAY

Shin et al. [2017]

The idea: replay examples from a generative model
without referring to the actual data of past tasks

Generative Adversarial Networks (GANs) - Overview



https://robotronblog.com/2017/09/05/gans/

cost function for D:

$$J = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

minimax game (game theory) yields objective function for D and G:

$$\begin{aligned}\mathcal{L}(D, G) &= \min_G \max_D V(D, G) \\ &= \min_G \max_D -J \\ &= \min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]\end{aligned}$$

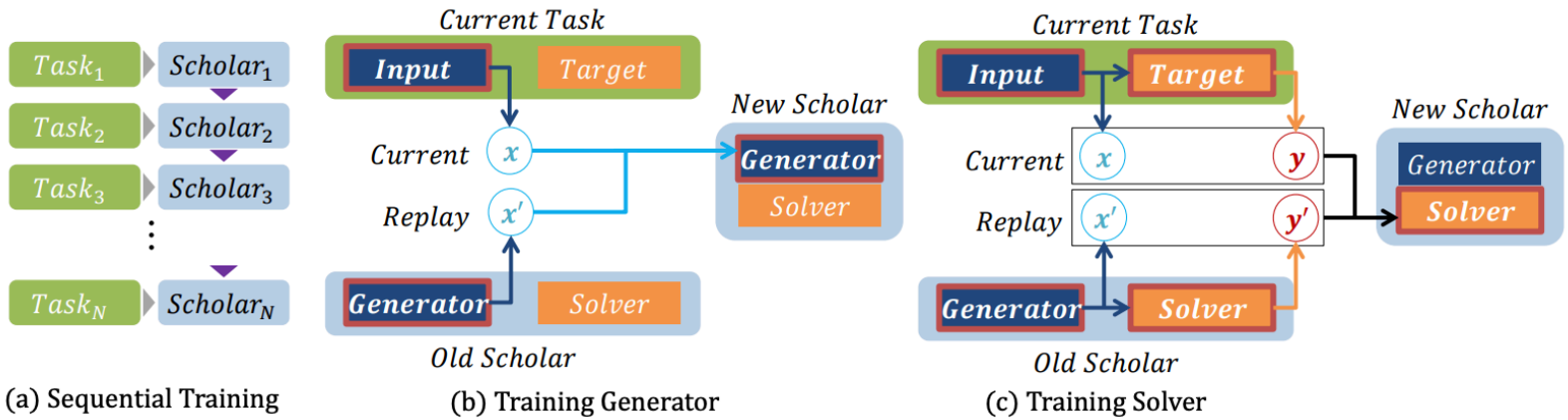
CONTINUAL LEARNING WITH GENERATIVE REPLAY

Shin et al. [2017]

The idea: replay examples from a generative model
without referring to the actual data of past tasks

Generative Replay

Definition 2 *A scholar H is a tuple $\langle G, S \rangle$, where a generator G is a generative model that produces real-like samples and a solver S is a task solving model parameterized by θ.*



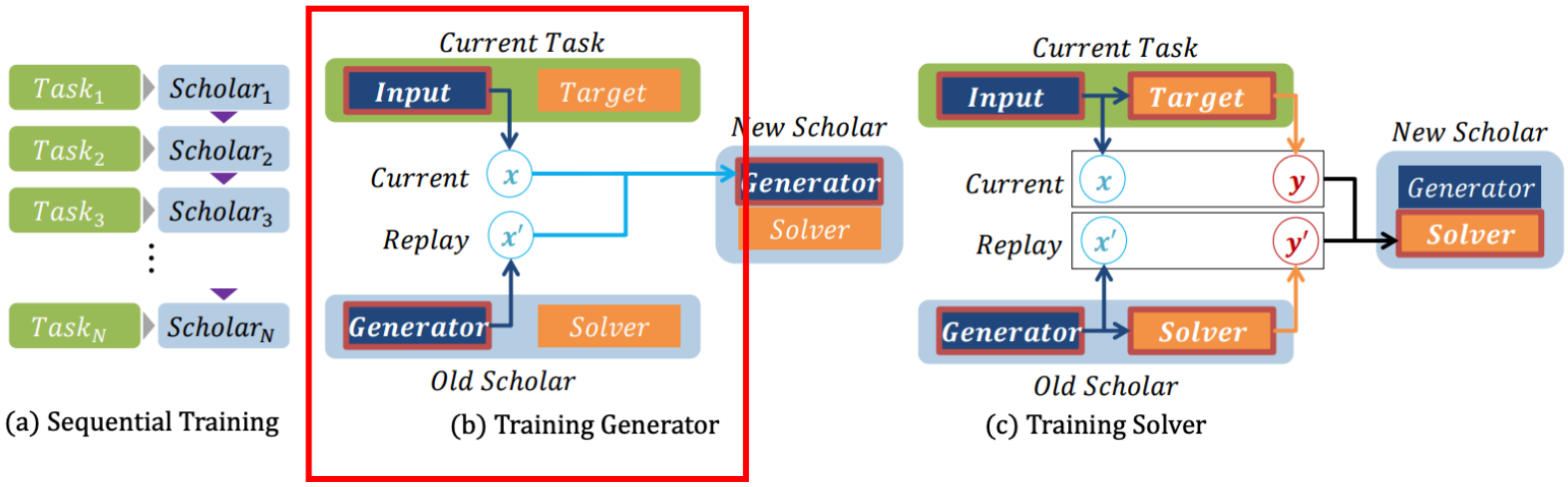
CONTINUAL LEARNING WITH GENERATIVE REPLAY

Shin et al. [2017]

The idea: replay examples from a generative model
without referring to the actual data of past tasks

Generative Replay

Definition 2 *A scholar H is a tuple $\langle G, S \rangle$, where a generator G is a generative model that produces real-like samples and a solver S is a task solving model parameterized by θ.*



intrinsic replay
(or pseudo-rehearsal)

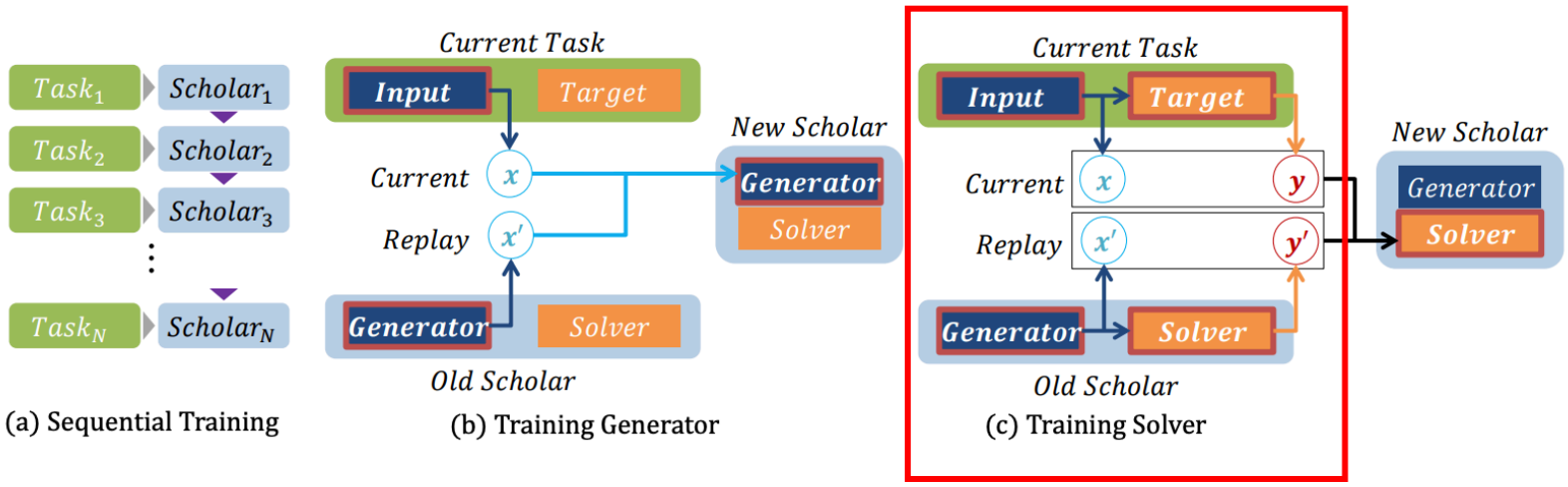
CONTINUAL LEARNING WITH GENERATIVE REPLAY

Shin et al. [2017]

The idea: replay examples from a generative model
without referring to the actual data of past tasks

Generative Replay

Definition 2 *A scholar H is a tuple $\langle G, S \rangle$, where a generator G is a generative model that produces real-like samples and a solver S is a task solving model parameterized by θ.*



$$\mathcal{L}_{train}(\theta_{N+1}) = r\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_{N+1}}[L(S(\boldsymbol{x};\theta_{N+1}),\boldsymbol{y})] + (1-r)\mathbb{E}_{\boldsymbol{x}'\sim G_N}[L(S(\boldsymbol{x}';\theta_{N+1}),S(\boldsymbol{x}';\theta_N))]$$

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. **Evaluating Catastrophic Forgetting**
10. Evaluation Datasets

EVALUATING CATASTROPHIC FORGETTING

two main papers [Goodfellow et al., 2013a, Kemker et al., 2018] in literature that evaluate ideas addressing catastrophic forgetting in neural networks

An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, Yoshua Bengio

Catastrophic forgetting is a problem faced by many machine learning models and algorithms. When trained on one task, then trained on a second task, many machine learning models "forget" how to perform the first task. This is widely believed to be a serious problem for neural networks. Here, we investigate the extent to which the catastrophic forgetting problem occurs for modern neural networks, comparing both established and recent gradient-based training algorithms and activation functions. We also examine the effect of the relationship between the first task and the second task on catastrophic forgetting. We find that it is always best to train using the dropout algorithm--the dropout algorithm is consistently best at adapting to the new task, remembering the old task, and has the best tradeoff curve between these two extremes. We find that different tasks and relationships between tasks result in very different rankings of activation function performance. This suggests the choice of activation function should always be cross-validated.

Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE)
Cite as: arXiv:1312.6211 **[stat.ML]**
(or arXiv:1312.6211v3 **[stat.ML]** for this version)

Measuring Catastrophic Forgetting in Neural Networks

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, Christopher Kanan

Deep neural networks are used in many state-of-the-art systems for machine perception. Once a network is trained to do a specific task, e.g., bird classification, it cannot easily be trained to do new tasks, e.g., incrementally learning to recognize additional bird species or learning an entirely different task such as flower recognition. When new tasks are added, typical deep neural networks are prone to catastrophically forgetting previous tasks. Networks that are capable of assimilating new information incrementally, much like how humans form new memories over time, will be more efficient than re-training the model from scratch each time a new task needs to be learned. There have been multiple attempts to develop schemes that mitigate catastrophic forgetting, but these methods have not been directly compared, the tests used to evaluate them vary considerably, and these methods have only been evaluated on small-scale problems (e.g., MNIST). In this paper, we introduce new metrics and benchmarks for directly comparing five different mechanisms designed to mitigate catastrophic forgetting in neural networks: regularization, ensembling, rehearsal, dual-memory, and sparse-coding. Our experiments on real-world images and sounds show that the mechanism(s) that are critical for optimal performance vary based on the incremental training paradigm and type of data being used, but they all demonstrate that the catastrophic forgetting problem has yet to be solved.

Comments: To appear in AAAI 2018
Subjects: **Artificial Intelligence (cs.AI)**; Computer Vision and Pattern Recognition (cs.CV); Machine Learning (cs.LG)
Cite as: arXiv:1708.02072 **[cs.AI]**
(or arXiv:1708.02072v4 **[cs.AI]** for this version)

EVALUATING CATASTROPHIC FORGETTING

two main papers [Goodfellow et al., 2013a, Kemker et al., 2018] in literature that evaluate ideas addressing catastrophic forgetting in neural networks

An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, Yoshua Bengio

Catastrophic forgetting is a problem faced by many machine learning models and algorithms. When trained on one task, then trained on a second task, many machine learning models "forget" how to perform the first task. This is widely believed to be a serious problem for neural networks. Here, we investigate the extent to which the catastrophic forgetting problem occurs for modern neural networks, comparing both established and recent gradient-based training algorithms and activation functions. We also examine the effect of the relationship between the first task and the second task on catastrophic forgetting. We find that it is always best to train using the dropout algorithm--the dropout algorithm is consistently best at adapting to the new task, remembering the old task, and has the best tradeoff curve between these two extremes. We find that different tasks and relationships between tasks result in very different rankings of activation function performance. This suggests the choice of activation function should always be cross-validated.

Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE)
Cite as: arXiv:1312.6211 **[stat.ML]**
(or arXiv:1312.6211v3 **[stat.ML]** for this version)

approaches to attempt to reduce catastrophic forgetting:

- dropout training [Hinton et al., 2012]
- logistic sigmoid activation function
- rectified linear [Jarrett et al., 2009]
- hard local winner take all (LWTA) [Srivastava et al., 2013]
- maxout [Goodfellow et al., 2013b].

experiments:

- only pairs of tasks (“old” and “new”)
- MNIST classification and Amazon reviews

results:

- dropout training is good to prevent forgetting
- activation function matter less than choice of algorithm

EVALUATING CATASTROPHIC FORGETTING

two main papers [Goodfellow et al., 2013a, Kemker et al., 2018] in literature that evaluate ideas addressing catastrophic forgetting in neural networks

evaluated several more recent CL algorithms:

- elastic weight consolidation (EWC) [Kirkpatrick et al., 2017]
- PathNet [Fernando et al., 2017]
- GeppNet [Gepperth and Karaoguz, 2016]
- Fixed expansion layer (FEL) [Coop et al., 2013]

three benchmark experiments:

- Data Permutation Experiment
- Incremental Class Learning
- Multi-modal Learning (image and audio classification)

three datasets:

- MNIST
- CUB200 [Welinder et al., 2010] (Caltech-UCSD birds 200)
- AudioSet [Gemmeke et al., 2017]

Measuring Catastrophic Forgetting in Neural Networks

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, Christopher Kanan

Deep neural networks are used in many state-of-the-art systems for machine perception. Once a network is trained to do a specific task, e.g., bird classification, it cannot easily be trained to do new tasks, e.g., incrementally learning to recognize additional bird species or learning an entirely different task such as flower recognition. When new tasks are added, typical deep neural networks are prone to catastrophically forgetting previous tasks. Networks that are capable of assimilating new information incrementally, much like how humans form new memories over time, will be more efficient than re-training the model from scratch each time a new task needs to be learned. There have been multiple attempts to develop schemes that mitigate catastrophic forgetting, but these methods have not been directly compared, the tests used to evaluate them vary considerably, and these methods have only been evaluated on small-scale problems (e.g., MNIST). In this paper, we introduce new metrics and benchmarks for directly comparing five different mechanisms designed to mitigate catastrophic forgetting in neural networks: regularization, ensembling, rehearsal, dual-memory, and sparse-coding. Our experiments on real-world images and sounds show that the mechanism(s) that are critical for optimal performance vary based on the incremental training paradigm and type of data being used, but they all demonstrate that the catastrophic forgetting problem has yet to be solved.

Comments: To appear in AAAI 2018
Subjects: **Artificial Intelligence (cs.AI)**; Computer Vision and Pattern Recognition (cs.CV); Machine Learning (cs.LG)
Cite as: arXiv:1708.02072 **[cs.AI]**
(or arXiv:1708.02072v4 **[cs.AI]** for this version)

EVALUATING CATASTROPHIC FORGETTING

two main papers [Goodfellow et al., 2013a, Kemker et al., 2018] in literature that evaluate ideas addressing catastrophic forgetting in neural networks

evaluated several more recent CL algorithms:

- elastic weight consolidation (EWC) [Kirkpatrick et al., 2017]
- PathNet [Fernando et al., 2017]
- GeppNet [Gepperth and Karaoguz, 2016]
- Fixed expansion layer (FEL) [Coop et al., 2013]

three benchmark experiments:

- Data Permutation Experiment
- Incremental Class Learning
- Multi-modal Learning (image and audio classification)

three datasets:

- MNIST
- CUB200 [Welinder et al., 2010] (Caltech-UCSD birds 200)
- AudioSet [Gemmeke et al., 2017]

Results (accuracy on the new task):

- PathNet performs best in data permutation
- GreppNet performs best in incremental class learning
- EWC has the best results in multi-modal learning.

Measuring Catastrophic Forgetting in Neural Networks

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, Christopher Kanan

Deep neural networks are used in many state-of-the-art systems for machine perception. Once a network is trained to do a specific task, e.g., bird classification, it cannot easily be trained to do new tasks, e.g., incrementally learning to recognize additional bird species or learning an entirely different task such as flower recognition. When new tasks are added, typical deep neural networks are prone to catastrophically forgetting previous tasks. Networks that are capable of assimilating new information incrementally, much like how humans form new memories over time, will be more efficient than re-training the model from scratch each time a new task needs to be learned. There have been multiple attempts to develop schemes that mitigate catastrophic forgetting, but these methods have not been directly compared, the tests used to evaluate them vary considerably, and these methods have only been evaluated on small-scale problems (e.g., MNIST). In this paper, we introduce new metrics and benchmarks for directly comparing five different mechanisms designed to mitigate catastrophic forgetting in neural networks: regularization, ensembling, rehearsal, dual-memory, and sparse-coding. Our experiments on real-world images and sounds show that the mechanism(s) that are critical for optimal performance vary based on the incremental training paradigm and type of data being used, but they all demonstrate that the catastrophic forgetting problem has yet to be solved.

Comments: To appear in AAAI 2018
Subjects: **Artificial Intelligence (cs.AI)**; Computer Vision and Pattern Recognition (cs.CV); Machine Learning (cs.LG)
Cite as: arXiv:1708.02072 **[cs.AI]**
(or arXiv:1708.02072v4 **[cs.AI]** for this version)

1. Catastrophic Forgetting
2. Continual Learning in Neural Networks
3. Learning Without Forgetting
4. Progressive Neural Networks
5. Elastic Weight Consolidation
6. iCaRL: Incremental Classifier and Representation Learning
7. Expert Gate
8. Continual Learning with Generating Replay
9. Evaluating Catastrophic Forgetting
10. **Evaluation Datasets**

EVALUATION DATASETS

regarding evaluation datasets → images are most commonly used for evaluating CL

- **MNIST** [LeCun et al., 1998][1] is perhaps the most commonly used dataset (used in more than half of the works introduced in this chapter). It consists of labeled examples of handwritten digits. There are 10 digit classes. One way to produce datasets for multiple tasks is to create the representations of the data by randomly permuting the elements of input feature vectors [Goodfellow et al., 2013a, Kemker et al., 2018, Kirkpatrick et al., 2017]. This paradigm ensures that the tasks are overlapping and have equal complexity.

- **CUB-200** (Caltech-UCSD Birds 200) [Welinder et al., 2010][2]) is another popular dataset for LL evaluation. It is an image dataset with photos of 200 bird species. It has been used in Aljundi et al. [2016, 2017], Kemker et al. [2018], Li and Hoiem [2016], Rannen Ep Triki et al. [2017], and Rosenfeld and Tsotsos [2017].

- **CIFAR-10** and **CIFAR-100** [Krizhevsky and Hinton, 2009][3] are also widely used. They contain images of 10 classes and 100 classes, respectively. They are used in Fernando et al. [2017], Jung et al. [2016], Lopez-Paz et al. [2017], Rebuffi et al. [2017], Venkatesan et al. [2017], Zenke et al. [2017], and Rosenfeld and Tsotsos [2017].

- **SVHN** (Google Street View House Numbers) [Netzer et al., 2011][4] is similar to MNIST, but contains an order of magnitude more labeled data. These images are from real-world problems and are harder to solve. It also has 10 digit classes. It is used in Aljundi et al. [2016, 2017], Fernando et al. [2017], Jung et al. [2016], Rosenfeld and Tsotsos [2017], Shin et al. [2017], Venkatesan et al. [2017], and Seff et al. [2017].

Other image datasets include **Caltech-256** [Griffin et al., 2007],[5] **GTSR** [Stallkamp et al., 2012],[6] **Human Sketch dataset** [Eitz et al., 2012],[7] **Daimler** (DPed) [Munder and Gavrila, 2006],[8] **MIT Scenes** [Quattoni and Torralba, 2009],[9] **Flower** [Nilsback and Zisserman, 2008],[10] **FGVC-Aircraft** [Maji et al., 2013],[11] **ImageNet ILSVRC2012** [Russakovsky et al., 2015],[12] and **Letters** (Chars74K) [de Campos et al., 2009].[13]

References

- **Michael McCloskey and Neal J. Cohen, (1989).** Catastrophic interference in connectionist networks: The sequential learning problem. In Psychology of Learning and Motivation, vol. 24, pages 109–165, Elsevier. DOI: 10.1016/s0079-7421(08)60536-8

- **German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter, (2018a).** Continual lifelong learning with neural networks: A review. ArXiv Preprint ArXiv:1802.07569

- **Zhizhong Li and Derek Hoiem, (2016)**. Learning without forgetting. In European Conference on Computer Vision

- **Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell, (2016)**. Progressive neural networks. ArXiv Preprint ArXiv:1606.04671

- **James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, An- drei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al., (2017)**. Overcoming catastrophic forgetting in neural networks. Proc. of the National Academy of Sciences, 114(13):3521–3526. DOI: 10.1073/pnas.1611835114. 72

- **Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert, (2017).** iCaRL: Incremental classifier and representation learning. In CVPR, pages 5533–5542. DOI: 10.1109/cvpr.2017.587

- **Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars, (2016)**. Expert gate: Lifelong learning with a network of experts. CoRR, abs/1611.06194, 2. DOI: 10.1109/cvpr.2017.753

- **Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim, (2017)**. Continual learning with deep generative replay. In NIPS, pages 2994–3003

Thanks!