

# Vision AI

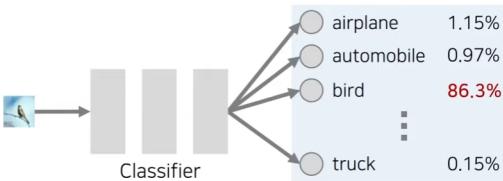
# Continual Learning

---

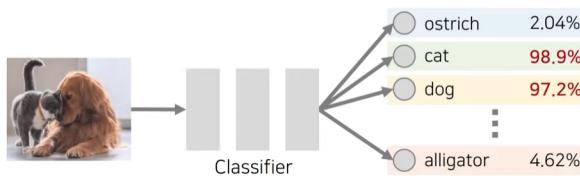
2022-04-13 | JiHyun Lee

# Continual Learning 개념

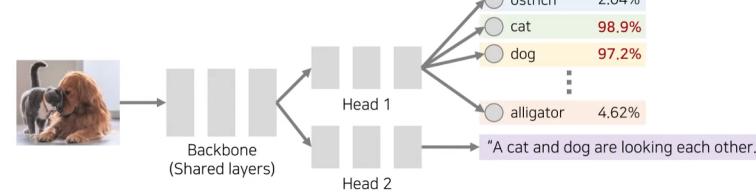
- Multi-class classification



- Multi-label classification



- Multi-Task Learning (MTL)



- Class 가 여러개.
- 하나의 입력은 여러 개의 class 중 하나의 output
- 가정) 하나의 입력은 한 개의 class 에만 속한다.
- 각각의 class 가 서로 배타적. 하나의 이미지가 하나의 클래스로 분류.
- 각각의 클래스 확률값을 모두 더했을 때 1
- 마지막 layer 에 softmax

- 하나의 이미지가 주어졌을 때, 각각의 클래스마다 존재 여부 확인.
- 일반적으로 각각의 레이블마다 확률 존재.
- 마지막 layer 에 binary classification

- task 가 여러개인 상황
- 하나의 이미지에 어떠한 사물이 포함되어 있는지 + 이러한 상황을 문장으로 표현.
- 해결 방법: shared layers, backbone network 사용. 다양한 task에서 공유될 수 있는 feature 학습. 즉, 하나의 이미지가 주어졌을 때, 이미지를 대표하는 shared feature 추출. 각각의 head에 넣어 각각 task마다 결과를 내보냄.
- 하나의 입력이 주어졌을 때, 다양한 task를 동시에 수행. 하나의 입력 데이터에 따라 task마다 정답 label이 존재해야 함.

## Continual Learning

- 하나의 네트워크를 학습 할 때 Task 가 지속적으로 바뀌는 상황.
- 연속적인 학습 과정 : Task 1에 대하여 모델을 학습한 후에 ( $\theta_1$ ), 추후에 task 2에 대해서 다시 학습하여 ( $\theta_2$ )로 업데이트. 하나의 네트워크임!
  - 하지만 task 2에 대해서 학습 할 때는, task 1의 학습 데이터셋에 접근하지 못함.
  - 이 때, task 2에 대해서 학습한 뒤에도, task 1의 테스트 데이터셋에 대해서 여전히 좋은 성능이 나오는 것이 목표.

### • Task 1



Task Transition

### • Task 2



사람이 어렸을 때, 중학교 수학을 공부한 뒤에 -> 고등학교, 대학교 수학을 공부할 때,

- 중학교 수학을 기억해야 함.
- 즉, 시간이 지나더라도 이전에 학습한 내용을 기억해야 함.

## Catastrophic forgetting

- 새로운 정보를 학습함에 따라서 이전에 배웠던 지식을 잊는 것.
- 뉴럴 네트워크의 경우, 과도하게 plasticity 한 경향이 있어 이전의 지식을 잊는 경향이 높다. 즉, 기본적으로 뉴럴 네트워크는 가소성이 높은 성향. 이전에 배웠던 지식은 잊는 경향이 있음.
- **Plasticity-stability dilemma**
  - **Plasticity** : 가소성이 있는 (plastic) 네트워크. 새로운 지식을 습득할 수 있다.
  - **Stability** : 안정성이 있는 (stable) 네트워크. 이전에 배웠던 지식을 잊지 않을 수 있다.

## 해결 방법 (3가지)

1. 과거의 (모든) 데이터와 새로운 데이터를 이용해 처음부터 다시 학습
  - a. 가장 기본적인 해결 방식
  - b. 하지만, task 가 바뀔 때마다 네트워크를 완전히 새롭게 학습하므로 연산 자원의 낭비가 심함.
  - c. 현실에서는 과거의 학습 데이터에 접근하지 못하는 경우도 있음.
1. 과거의 학습 데이터를 다시 사용 (replay)
  - a. 과거의 학습 데이터를 일부 저장하고 있어야 함. -> 어떤 데이터를 저장해야 하지?
    - i. 과거의 데이터를 대표할 수 있는 데이터.
  - b. Task 가 연속적으로 변경됨에 따라서 저장 비용 (storage cost) 이 증가하는 경향이 있음.
1. Task가 바뀔 때마다 모델 변경
  - a. Task 가 변경될 때마다 기존의 가중치를 수정하거나 새로운 레이어 추가
    - i. 모델 아키텍처, 학습 가중치 변경 (일부 혹은 전부 수정)
    - ii. 학습 가능한 새로운 layer 추가.
    - iii. 네트워크 구조가 계속 변경된다는 문제. 또한 네트워크 가중치를 효율적으로 증가시켜야 함.

## 해결 방법 (3가지)

1. 과거의 (모든) 데이터와 새로운 데이터를 이용해 처음부터 다시 학습
2. 과거의 학습 데이터를 다시 사용 (replay)
3. Task가 바뀔 때마다 모델 변경



아직까지 training regimes 가 catastrophic forgetting에 미치는 영향에 대해 분석한 논문은 적음.

### Stable SGD: Understanding the Role of Training Regimes in Continual Learning (NIPS 2020)

- 본 논문은 과거의 학습 데이터를 다시 사용하지 않으며, 아키텍처를 바꾸지 않는 상황에서 쓸 수 있는 간단한 방법 제안.
- 학습할 때, 1) dropout regularization, 2) batch size, 3) learning rate 조절과 같이 잘 알려진 학습 테크닉을 적절히 활용하는 것만으로도 catastrophic forgetting 문제를 개선할 수 있음.

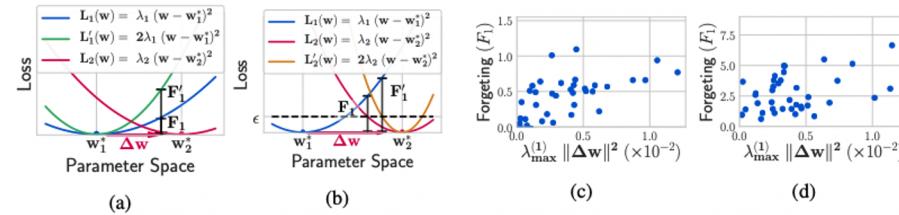
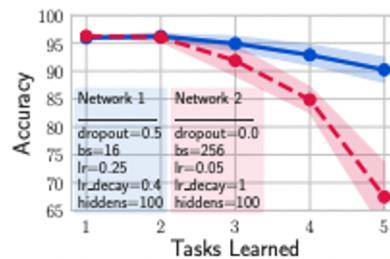


Figure 2: (a): For a fixed  $\Delta w$ , the wider the curvature of the first task, the less the forgetting. (b): The wider the curvature of the second task, the smaller  $\|\Delta w\|$ . (c) and (d): Empirical verification of (5) for Rotated MNIST and Permuted MNIST, respectively.

## Continual Learning and Catastrophic forgetting

- pdf 자료:  
[https://khoury.northeastern.edu/home/hand/teaching/cs7150-summer-2020/Continual\\_Learning\\_and\\_Catastrophic\\_Forgetting.pdf](https://khoury.northeastern.edu/home/hand/teaching/cs7150-summer-2020/Continual_Learning_and_Catastrophic_Forgetting.pdf)
- 블로그 링크 : 대부분 한번씩 보셨을 것 같기는 합니다 ㅎㅎ  
<https://winnerus.medium.com/ai-%EB%85%BC%EB%AC%B8%EB%A6%AC%EB%B7%B0-continual-learning-on-deep-learning-16969792acc7>

# Continual Learning 개념

## Continual Learning SoTA Model

- Leaderboards -> 참여 challenge에 유사한 모델 (+paper) 먼저 보면 될 듯.

<https://paperswithcode.com/task/continual-learning>

- 2021년 1위

○ VARMS - <https://github.com/YehLi/OpenWorldVision/tree/master/moco-sup>

■ youtube (7:21:02~) : <https://www.youtube.com/watch?v=TknWpbXEKeA>

		Phase: Test Phase, Split: Test set												
		Order by metric												
Rank	Participant	macro top1	macro top5	micro top1	micro top5	macro top1	macro top5	top1	top5	macro accuracy	macro known	macro novel	accuracy known	novel augment
1	VARMS	0.84	0.94	0.56	0.68	0.91	0.98	0.46	0.74	0.77				
2	HRLHOW	0.81	0.93	0.55	0.71	0.90	0.98	0.44	0.72	0.75				
3	CognitiveVision	0.80	0.92	0.52	0.64	0.89	0.97	0.39	0.68	0.75				

MoCo 기반



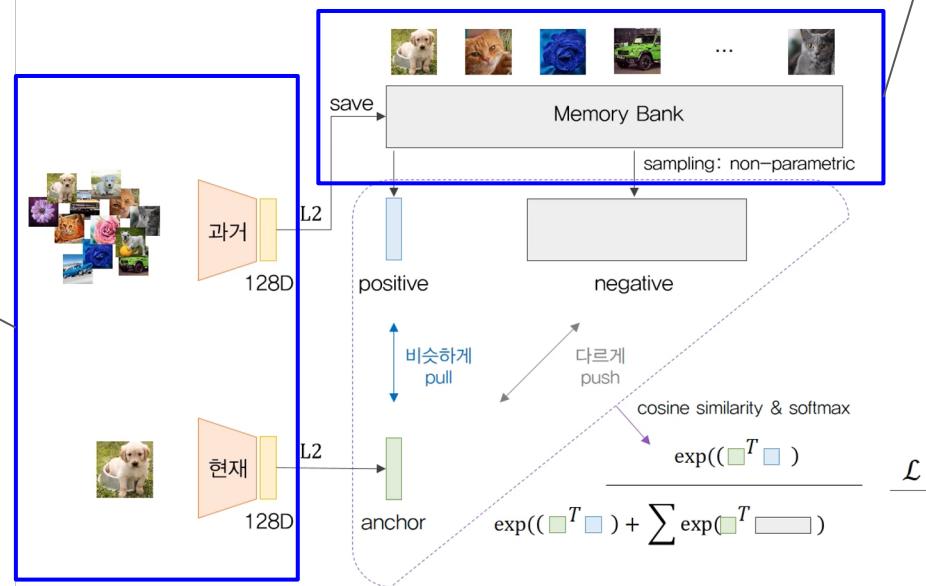
## Contrastive Learning

- 레이블 정보 없이 관측치 레벨에서 학습, 공유하는 representation, semantic structure 가 있을 것.
- 관측치 사이의 유사성을 기반으로, 관측치끼리 구분하도록 학습을 한다면, 레이블 정보 없이도 좋은 representation 을 얻을 수 있지 않을까?
- 수많은 관측치들을 어떻게 구분하도록 만들 수 있을까?
  - ImageNet 데이터는 약 1200만개 이상의 관측치를 갖고 있어, softmax 를 단순히 사용하기에는 어렵다.
  - 분모가 커짐으로 인해 확률값이 너무 작아져서 학습이 어려움.

## 01 | Intuitive Understanding

좀 더 좋은 품질의 embedding

- 네트워크 디자인
- Data Augmentation

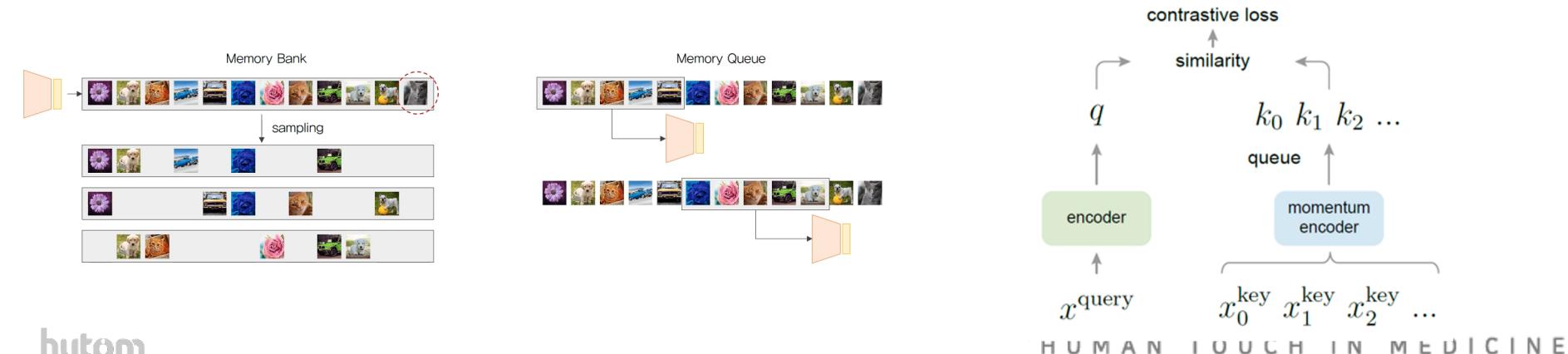


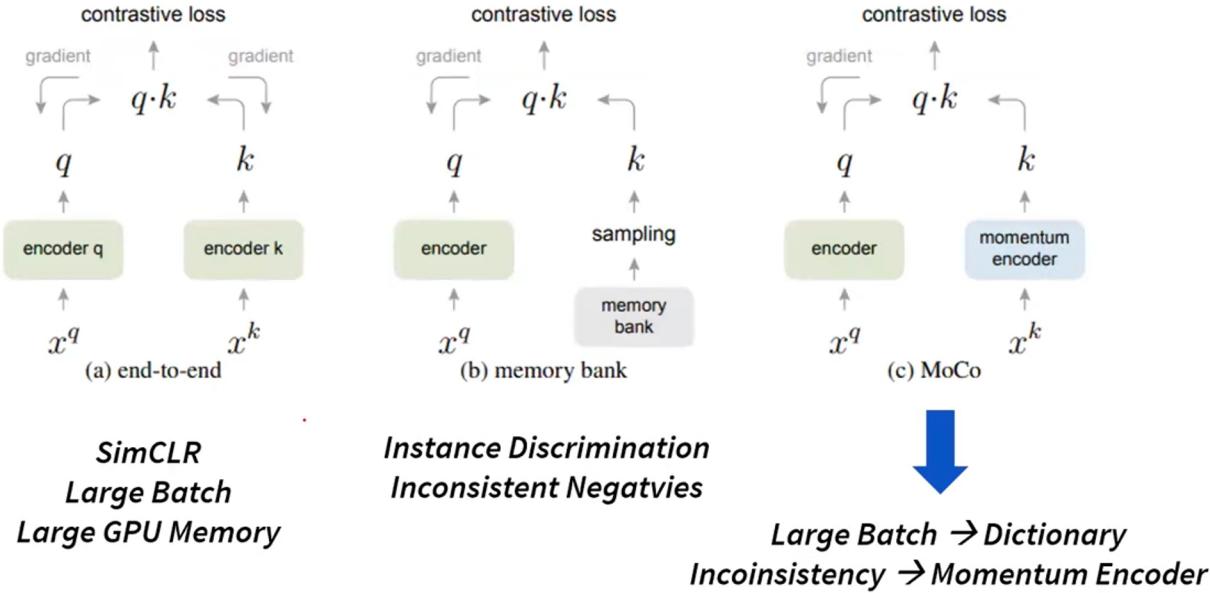
- Define example (example 을 잘 정의하기)
- Sampling strategy (단순히 random sampling 이 아닌 좀 더 nice 한 전략)

- Instance discrimination을 하기 위해 모든 데이터의 임베딩 정보를 저장한다. 이때 임베딩 크기를 맞추기 위해 L2-normalization을 수행한다.
- 현재 타겟 이미지의 과거 임베딩 벡터 & 나머지 이미지들의 일부 임베딩 벡터를 샘플링
- 비슷한 정도를 측정하기 위해 cosine 유사도 측정 (L2 Euclidean – Cosine)
- 나 자신에 대한 유사도는 높이며, 나머지는 내리고 싶다. Cosine 유사도를 logit으로 사용하고 확률로 표현하기 위해 softmax 적용 (NCE)  $\rightarrow P(i|\nu)$
- Loss: 앞에서 얻은 cosine 유사도를 logit으로 사용하고, 타겟 클래스 레이블을 모두 0으로 cross entropy. (minimize negative log-likelihood)

**Moco**

- Memory Bank의 단점
  - 관측치 전체에 대한 embedding 정보를 계속 저장하고 있어야 함 - 메모리 이슈
  - 랜덤 샘플링을 통해 negative example 구성 - 데이터 샘플별로 학습에 기여하는 정도가 다름.
- **Memory Queue 사용**
  - First In First Out (FIFO)
  - 모든 샘플들이 동등한 횟수로 학습에 사용됨.
  - embedding 정보를 계속 저장하는 것이 아니라, 그 때 그 때 embedding vector를 산출하기 때문에 메모리 이슈 해결.
- **Momentum Encoding**
  - Queue에 들어있는 이미지 벡터에 대한 이미지 임베딩을 산출하기 위해, 사용되는 네트워크.
  - 과거 모델 weight들의 가중평균으로 업데이트
  - 과거의 나 자신들이 'teacher'가 되는 mean teacher 방식과 동일 (semi-supervised model)



**SimCLR**

- negative 를 많이 얻기 위해서, batch 사이즈를 키워야 함.

**Instance Discrimination**

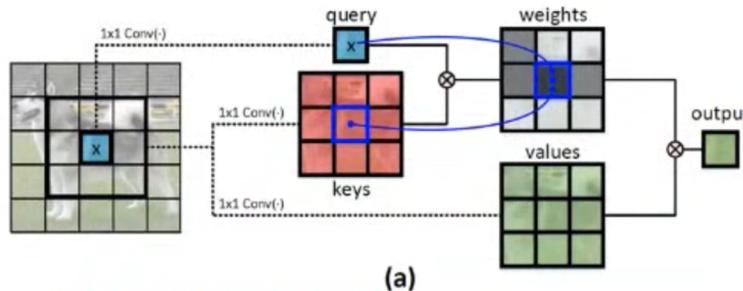
- 특정 n 개의 데이터를 뽑아서, negative sample 로 사용함. -> 인코더는 계속 학습되는데, negative sample 은 계속 동일한 데이터가 사용될 수 있음.

**MoCo**

- negative sample 을 지속적으로 관리.  
간접적으로 많은 양의 negative sample 을 볼 수 있다.

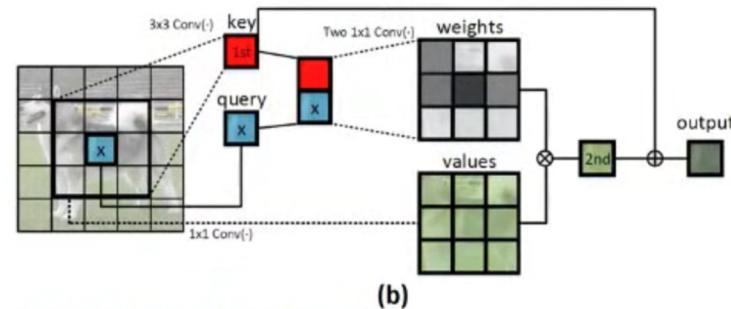
## • Contextual Transformer Networks (CoTNet)

- Conventional self-attention solely exploits the isolated query-key pairs to measure attention matrix, but leaves rich contexts among keys underexploited.
- CoT block first captures the first-order context among neighbor keys, which is further leveraged to trigger self-attention that mines the second-order context.



**Existing Vision Transformers**

- Capitalize on pairs of isolated queries and keys



**Contextual Transformer**

- Exploit rich contextual information among input keys

- Contextual Transformer Networks (CoTNet)

Table 1: Effect of utilizing different replacement settings on the four stages (**res2**→**res3**→**res4**→**res5**) in the basic backbone of ResNet-50 and two widely adopted architecture changes, ResNet-D [2] and Squeeze-and-Excitation [3] (**D-SE**). ✓ denotes the stage is replaced with our CoT blocks. \* denotes the use of architecture changes (D-SE). We adopt the default setup for training on ImageNet.

	res2	res3	res4	res5	D-SE	Params	GFLOPs	Infer	Top-1 Acc.	Top-5 Acc.
ResNet-50						25.5M	4.1	508 ex/s	77.3	93.6
CoTNet-50		✓				23.5M	4.0	491 ex/s	78.5	94.1
		✓	✓			22.4M	3.7	443 ex/s	79.0	94.3
	✓	✓	✓			22.3M	3.4	390 ex/s	79.0	94.4
	✓	✓	✓	✓		22.2M	3.3	331 ex/s	79.2	94.5
SE-ResNetD-50				*		35.7M	4.4	444 ex/s	79.1	94.5
SE-CoTNetD-50	✓	✓	*			23.1M	4.1	414 ex/s	79.8	94.7

- Comparison of different backbones on validation set

Model	Top-1 Accuracy						
	Eval 4.1	Eval 4.2	Eval 5.1	Eval 5.2	Eval 6.1	Eval 6.2	Average
ResNet-RS-50	82.74	71.99	82.19	74.96	83.08	77.31	78.71
ResNet-RS-50 + MoCo v2	83.74	73.47	83.25	76.78	84.35	78.39	80.00
ResNet-RS-50 + Supervised MoCo v2	83.45	74.02	84.34	76.08	84.17	79.38	80.24
ResNet-RS-50 + Pseudo label	84.69	75.81	85.19	77.26	84.14	79.59	81.11
CoTNet-50	84.10	73.75	83.21	75.96	83.41	79.12	79.93
CoTNet-50 + 250 Epoch	86.80	76.21	85.41	78.69	86.04	80.88	82.34
CoTNet-101 + 250 Epoch	86.65	79.62	85.65	79.75	87.55	82.38	83.60
<b>CoTNet-101 + 250 Epoch + TenCrop Average</b>	<b>88.02</b>	<b>80.71</b>	<b>87.13</b>	<b>81.44</b>	<b>88.63</b>	<b>83.65</b>	<b>84.93</b>

- 2021년 1위
  - VARMS - <https://github.com/YehLi/OpenWorldVision/tree/master/moco-sup>

The screenshot shows a GitHub issue page with a dark theme. The title of the issue is "guide on code #1". A green button labeled "Open" is visible. Below it, the user "zou-yiqi" is listed as having opened the issue on 21 Feb with 0 comments. A comment from "zou-yiqi" dated 21 Feb is shown, expressing gratitude for the work and asking for a guide on how to use the code, as well as information on how to get data for the OWV Competition.

guide on code #1

Open zou-yiqi opened this issue on 21 Feb · 0 comments

zou-yiqi commented on 21 Feb

Thanks for your excellent work!  
I want to reproduce your result in OpenWorldVision Competition. Could you please write a guide on how to use your code ?  
Besides, how can I get the data of OWV Competition? I fill the google form in official website but get no responses.

- git star 841
  - <https://arxiv.org/pdf/1904.07734.pdf> (Three scenarios for continual learning, NIPS workshop 2018)
  - <https://github.com/GMvandeVen/continual-learning>
- Incremental / Continual Learning 의 접근 방식 (2가지)
  - Class-Incremental Learning
    - 여러 task 를 순서대로 학습한 뒤, 모든 task의 클래스에 대한 시험을 한 번에 보는 방식
  - Task-Incremental Learning
    - 여러 task 를 순서대로 학습한 뒤, 시험을 볼 때 이전 어떤 task 에서 학습한 문제라는 걸 알려준 뒤 class 를 맞추도록 시험을 보는 방식.

## Three scenarios for continual learning

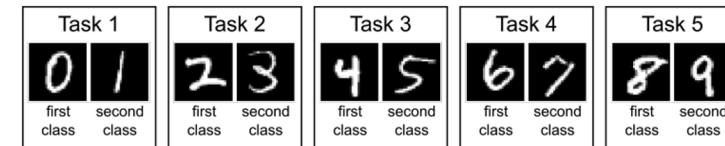


Figure 1: Schematic of split MNIST task protocol.

Table 2: Split MNIST according to each scenario.

<b>Task-IL</b>	With task given, is it the 1 <sup>st</sup> or 2 <sup>nd</sup> class? (e.g., 0 or 1)
<b>Domain-IL</b>	With task unknown, is it a 1 <sup>st</sup> or 2 <sup>nd</sup> class? (e.g., in [0, 2, 4, 6, 8] or in [1, 3, 5, 7, 9])
<b>Class-IL</b>	With task unknown, which digit is it? (i.e., choice from 0 to 9)

<sup>1</sup> Center for Neuroscience and Artificial Intelligence, Baylor College of Medicine, Houston

<sup>2</sup> Computational and Biological Learning Lab, University of Cambridge, Cambridge

<sup>3</sup> Department of Electrical and Computer Engineering, Rice University, Houston

{ven, astolias}@bcm.edu

Table 4: Average test accuracy (over all tasks) on the split MNIST task protocol. Each experiment was performed 20 times with different random seeds, reported is the mean ( $\pm$  SEM) over these runs.

Approach	Method	Task-IL	Domain-IL	Class-IL
<i>Baselines</i>	<i>None – lower bound</i>	87.19 ( $\pm$ 0.94)	59.21 ( $\pm$ 2.04)	19.90 ( $\pm$ 0.02)
	<i>Offline – upper bound</i>	99.66 ( $\pm$ 0.02)	98.42 ( $\pm$ 0.06)	97.94 ( $\pm$ 0.03)
Task-specific	XdG	99.10 ( $\pm$ 0.08)	-	-
Regularization	EWC	98.64 ( $\pm$ 0.22)	63.95 ( $\pm$ 1.90)	20.01 ( $\pm$ 0.06)
	Online EWC	99.12 ( $\pm$ 0.11)	64.32 ( $\pm$ 1.90)	19.96 ( $\pm$ 0.07)
	SI	99.09 ( $\pm$ 0.15)	65.36 ( $\pm$ 1.57)	19.99 ( $\pm$ 0.06)
Replay	LwF	99.57 ( $\pm$ 0.02)	71.50 ( $\pm$ 1.63)	23.85 ( $\pm$ 0.44)
	DGR	99.50 ( $\pm$ 0.03)	95.72 ( $\pm$ 0.25)	90.79 ( $\pm$ 0.41)
	DGR+distill	99.61 ( $\pm$ 0.02)	96.83 ( $\pm$ 0.20)	91.79 ( $\pm$ 0.32)
Replay + Exemplars	iCaRL (budget = 2000)	-	-	94.57 ( $\pm$ 0.11)

Table 5: Idem as Table 4, except on the permuted MNIST task protocol.

Approach	Method	Task-IL	Domain-IL	Class-IL
<i>Baselines</i>	<i>None – lower bound</i>	81.79 ( $\pm$ 0.48)	78.51 ( $\pm$ 0.24)	17.26 ( $\pm$ 0.19)
	<i>Offline – upper bound</i>	97.68 ( $\pm$ 0.01)	97.59 ( $\pm$ 0.01)	97.59 ( $\pm$ 0.02)
Task-specific	XdG	91.40 ( $\pm$ 0.23)	-	-
Regularization	EWC	94.74 ( $\pm$ 0.05)	94.31 ( $\pm$ 0.11)	25.04 ( $\pm$ 0.50)
	Online EWC	95.96 ( $\pm$ 0.06)	94.42 ( $\pm$ 0.13)	33.88 ( $\pm$ 0.49)
	SI	94.75 ( $\pm$ 0.14)	95.33 ( $\pm$ 0.11)	29.31 ( $\pm$ 0.62)
Replay	LwF	69.84 ( $\pm$ 0.46)	72.64 ( $\pm$ 0.52)	22.64 ( $\pm$ 0.23)
	DGR	92.52 ( $\pm$ 0.08)	95.09 ( $\pm$ 0.04)	92.19 ( $\pm$ 0.09)
	DGR+distill	97.51 ( $\pm$ 0.01)	97.35 ( $\pm$ 0.02)	96.38 ( $\pm$ 0.03)
Replay + Exemplars	iCaRL (budget = 2000)	-	-	94.85 ( $\pm$ 0.03)

### 3.1 Task-specific Components

A simple explanation for catastrophic forgetting is that after a neural network is trained on a new task, its parameters are optimized for the new task and no longer for the previous one(s). This suggests that not optimizing the entire network on each task could be one strategy for alleviating catastrophic forgetting. A straightforward way to do this is to explicitly define a different sub-network per task. Several recent papers use this strategy, with different approaches for selecting the parts of the network for each task. A simple approach is to randomly assign which nodes participate in each task (*Context-dependent Gating* [XdG; 4]). Other approaches use evolutionary algorithms [17] or gradient descent [18] to learn which units to employ for each task. By design, these approaches are limited to the Task-IL scenario, as task identity is required to select the correct task-specific components.

### 3.2 Regularized Optimization

When task identity information is not available at test time, an alternative strategy is to still preferentially train a different part of the network for each task, but to always use the entire network for execution. One way to do this is by differently regularizing the network’s parameters during training on each new task, which is the approach of *Elastic Weight Consolidation* [EWC; 1] and *Synaptic Intelligence* [SI; 7]. Both methods estimate for all parameters of the network how important they are for the previously learned tasks and penalize future changes to them accordingly (i.e., learning is slowed down for parts of the network important for previous tasks).

### 3.3 Modifying Training Data

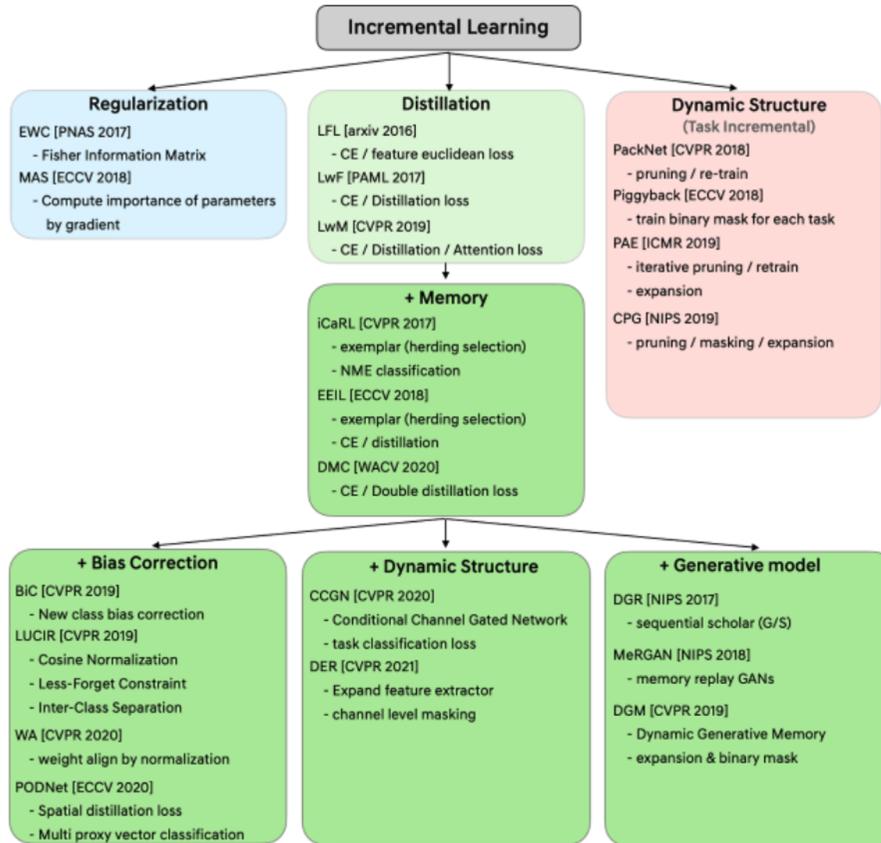
An alternative strategy for alleviating catastrophic forgetting is to complement the training data for each new task to be learned with “pseudo-data” representative of the previous tasks. This strategy is referred to as replay.

### 3.4 Using Exemplars

If it is possible to store data from previous tasks, another strategy for alleviating catastrophic forgetting is to use stored data as “exemplars” during execution. A recent method that successfully used this strategy is *iCaRL* [2]. This method uses a neural network for feature extraction and performs classification based on a nearest-class-mean rule [23] in that feature space, whereby the class means are calculated from the stored data. To protect the feature extractor network from becoming unsuitable for previously learned tasks, iCaRL also replays the stored data—as well as the current task inputs with a special form of distillation—during training of the feature extractor.

# continual-learning

다음은 Incremental / Continual Learning의 굵직한 논문들을 흐름대로 살펴보겠습니다.



연구흐름

# End of the Document