Continual Learning

Eduardo Morales, Hugo Jair Escalante

INAOE

Outline

Introducción

- Los sistemas de aprendizaje han mostrado muy buenos resultados, inclusive mejores que humanos, en algunas tareas específicas
- Sin embargo, son sistemas estáticos, incapaces de adaptarse a cambios en el tiempo
- Volver a entrenar sistemas desde cero, cada vez que queremos incorporar una nueva clase o cambian las condiciones, es poco práctico, sobretodo para sistemas basados en aprendizaje profundo

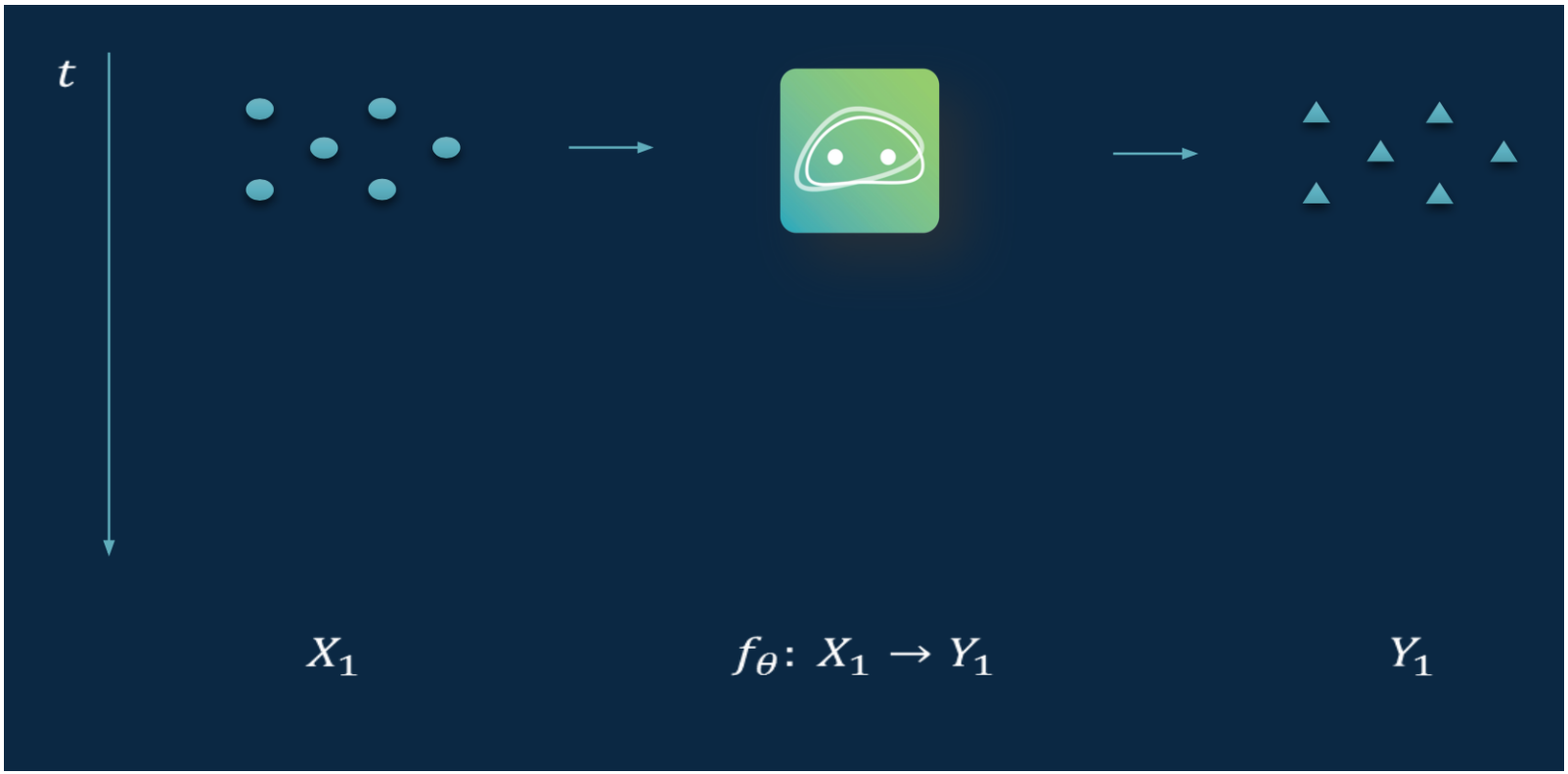
Introducción

- El aprendizaje incremental busca desarrollar sistemas que puedan aprender contínuamente para lidear con nuevas tareas, preservando lo aprendido en tareas previas
- Aunque presente, desde los inicios del área, ya que los sistemas de ML:
  - Se enfocaban a una sóla tarea simple con aprendizaje estático
  - Se tenían pocos datos y poco poder de cómputo
  - Con atributos creados a mano y principalmente en aprendizaje supervisado
- El uso de DL ha resaltado aún más la necesidad de desarrollar el aprendizaje contínuo

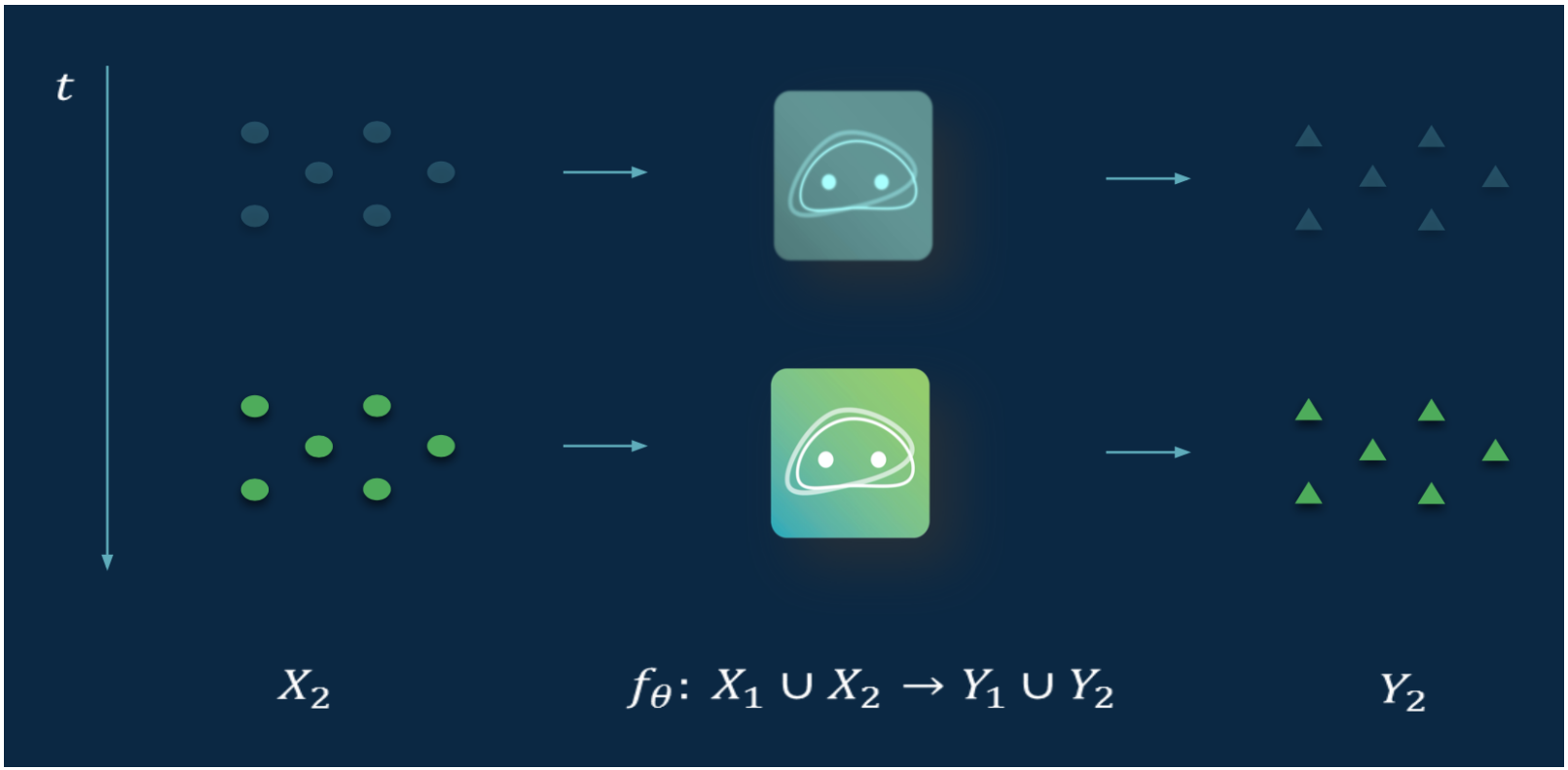
Aprendizaje Contínuo

- Se acerca más al aprendizaje humano que puede aprender e integrar nuevo conocimiento cuando se le presentan nuevas tareas
- Para la robótica, parece ofrecer la respuesta a *developmental robotics* (robótica del desarrollo) que toma inspiración en el desarrollo de los niños
- En este sentido se aprende en un esquema abierto y de forma contínua
- Esto requiere, entre otras habilidades, el poder generar automáticamente metas, explorar el ambiente, incorporar motivación intrínseca y curiosidad, y aprender de forma contínua

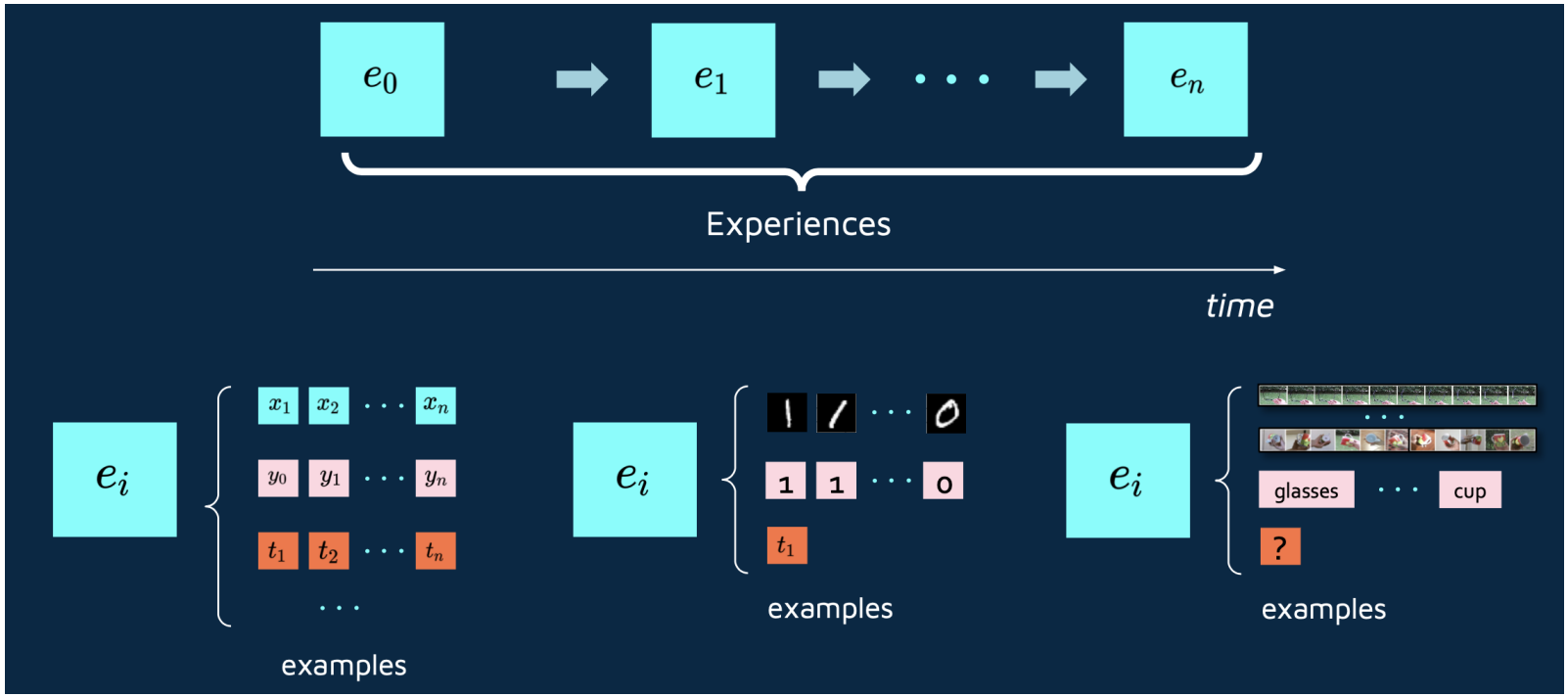
Continual Learning



Continual Learning



Continual Learning



Retos

- Balance *estabilidad-plasticidad*: Donde *estabilidad* se refiere a mantener el conocimiento previo, mientras la *plasticidad* se refiere a la habilidad de integrar nuevo conocimiento
- *Catastrophic forgetting*: ¿Cómo aprender una nueva tarea (posiblemente con sólo datos de esa nueva tarea) sin afectar lo aprendido previamente?
- Manejo de memoria: ¿Qué información almacenar para transferir de forma efectiva conocimiento y habilidades?
- ¿Cómo detectar cambios de distribución/nuevas clases?

Aprendizaje Contínuo (CL)

- El enfoque tradicional de *fine tuning* usado en *transfer learning* para nuevas tareas, no aplica al no tener, necesariamente, datos de las tareas previas y, por lo mismo, se puede degradar su desempeño en estas
- En la mayoría de los sistemas de CL las tareas se presentan de forma secuencial en sesiones, donde en cada sesión sólo se presentan los datos de una nueva tarea
- Se espera que después de cada sesión se desempeñe correctamente en todas las tareas previas usando datos de prueba nuevos

Temas Relacionados

Existen varios temas relacionados que a veces se usan indistintamente:

1. *Incremental Learning*: Se presentan las tareas en bloques y generalmente se usa en clasificación
2. *Continual Learning*: Se aprende de forma contínua y no está restringido a clasificación (e.g., se aplica también para RL)
3. *Life-Long Learning*: En principio se busca que sea por tiempos más prolongados y con más autonomía

¿Porqué es deseable tener un aprendizaje incremental?

1. Restricciones de memoria: Sistemas que no pueden tener acceso a (o guardar) todos los datos para todas las tareas (e.g., robótica)
2. Seguridad/privacidad de datos: Sistemas en donde no queremos almacenar los datos por restricciones de seguridad
3. Sustentabilidad: Los modelos de DL requieren muchas horas y mucho cómputo para crear un nuevo modelo, un esquema incremental puede actualizar un modelo con muchos menos recursos

Aprendizaje contínuo

Se puede dar en las formas tradicionales de aprendizaje:

- Aprendizaje contínuo supervisado: En donde los datos etiquetados no se tienen todos al mismo tiempo (el enfoque principal en esta clase)
- Aprendizaje contínuo no supervisado: Podemos pensar en robots que van aprendiendo de manera autónoma información de su entorno
- Aprendizaje por refuerzo contínuo: De alguna forma RL tiene ciertas similitudes con CL, aunque en este caso se busca aprender incrementalmente tareas diferentes

Algunos puntos a considerar

- Disponibilidad de datos
- Restricciones de memoria y cómputo
- Cantidad y tipo de supervisión
- Desempeño esperado

Aprendizaje incremental de clases (*Class-IL*)

- Class-IL aprende de una distribución de datos no estacionaria
- Idealmente nos gustaría:
  1. Explotar conocimiento de tareas previas para mejorar el aprendizaje de nuevas (*forward transfer*)
  2. Explotar los nuevos datos para mejorar el desempeño de tareas previas (*backward transfer*)

Condiciones de aprendizaje

La mayoría de los sistemas de aprendizaje incremental de clases suponen el siguiente esquema:

- Se aprende una serie de tareas
- Cada tarea consiste de un conjunto finito de clases disjuntas con otras tareas (pasadas y futuras)
- Durante el entrenamiento el agente sólo tiene acceso a los datos de una sola tarea
- Opcionalmente, se pueden almacenar algunos *examplars* de tareas pasadas

Problema de aprendizaje incremental

- Dado un conjunto de datos $(\mathcal{X}^t, \mathcal{Y}^t)$ para la tarea t, queremos minimizar el valor esperado de la función de pérdida sobre todas las tareas previas (aunque no tengamos o se tenga acceso limitado a sus datos)

$$\sum_{t=1}^{\tau} \mathbb{E}_{\mathcal{X}^t, \mathcal{Y}^t} \mathcal{L}(f_t(\mathcal{X}^t; \theta), \mathcal{Y}^t)$$

- Lo cual, para la tarea actual (t) se aproxima con el riesgo empírico con:

$$\frac{1}{N_t} \sum_{t=1}^{N_t} \mathcal{L}(f(x_i^t; \theta), y_i^t)$$

Subdivisiones

Con $P(\mathcal{X}^t) \neq P(\mathcal{X}^{t+1})$:

- Aprendizaje incremental de clases: $\{\mathcal{Y}^t\} \subset \{\mathcal{Y}^{t+1}\}$ con $P(\mathcal{Y}^t) \neq P(\mathcal{Y}^{t+1})$
- Aprendizaje incremental de tareas: $\{\mathcal{Y}^t\} \neq \{\mathcal{Y}^{t+1}\}$ con las etiquetas conocidas sólo para la tarea actual
- Aprendizaje incremental de dominios: $\{\mathcal{Y}^t\} = \{\mathcal{Y}^{t+1}\}$ con $P(\mathcal{Y}^t) = P(\mathcal{Y}^{t+1})$
- Aprendizaje incremental de datos: General para cualquier flujo de datos sin nociones de clases, tareas o dominios

Problema de aprendizaje incremental

Nos vamos a concentrar en métodos que siguen el siguiente esquema:

- Problema:

$$\tau = \{(C^1, D^1), (C^2, D^2), \ldots, (C^n, D^n)\}$$

- Para cada tarea t se tiene un conjunto de clases:

$$C^t = \{c_1^t, c_2^t, \ldots, c_m^t\}$$

- Denotamos como N^t al número total de clases vistas hasta el momento incluyendo las de la tarea actual:

$$N^t = \sum_{i=1}^{t} |C^i|$$

- Y $D^t = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ con $y \in \{0, 1\}^{N^t}$, con $C^i \cap C^j = \emptyset$ si $i \neq j$

Suposiciones

- Se aprende con redes neuronales profundas parametrizadas por sus pesos θ (en lo que se ha centrado últimamente el área, pero en general aplica para otras técnicas de ML)
- La salida de la red es $o(x) = h(x;\theta)$ y su predicción es $\hat{y} = \sigma(h(x;\theta))$ donde σ es una función *softmax*

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$K =$ número de clases y $\mathbf{z} = (z_1, z_2, \ldots, z_K)$

- Después de entrenar para la tarea t evaluamos a la red sobre todas las clases hasta el momento

Función de pérdida

La mayoría usa entropía cruzada, la cual se puede hacer:

- Considerando todas las clases hasta el momento

$$\mathcal{L}_c(\mathbf{x}, \mathbf{y}; \theta^t) = \sum_{k=1}^{N^t} y_k \log \frac{\exp(\mathbf{o}_k)}{\sum_{i=1}^{N^t} \exp(\mathbf{o}_i)}$$

  Aquí los errores se propagan para todas las clases actuales y pasadas

- Considerando sólo las clases de la tarea

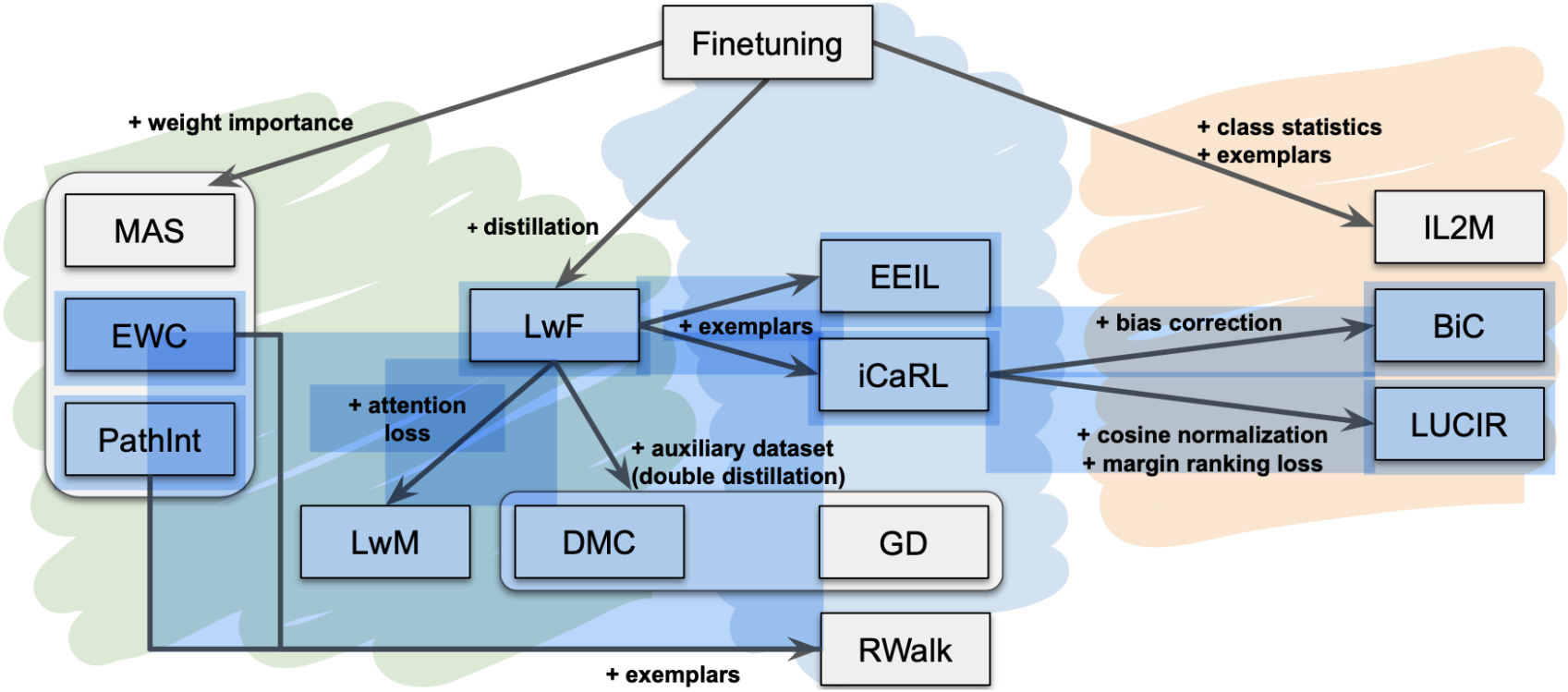
$$\mathcal{L}_{c*}(\mathbf{x}, \mathbf{y}; \theta^t) = \sum_{k=1}^{|C^t|} y_{N^{t-1}+k} \log \frac{\exp(\mathbf{o}_{N^{t-1}+k})}{\sum_{i=1}^{|C^t|} \exp(\mathbf{o}_{N^{t-1}+i})}$$

  Los errores se propagan sólo de las probabilidades relacionadas con las clases de la tarea t

Retos

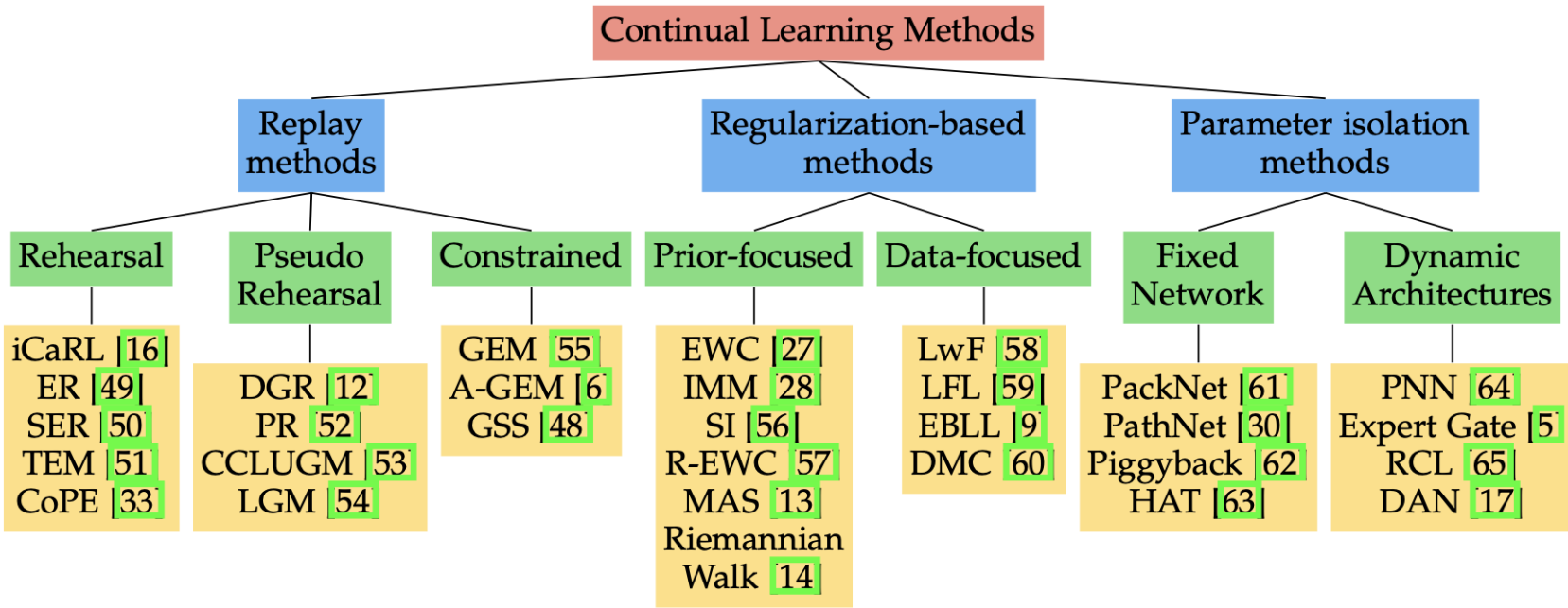
- Cómo balancear el retener conocimiento de tareas previas con la adquisición de nuevo conocimiento
- *Catastrophic forgetting*, algunas razones:
  1. *Weight drift*: Al aprender nuevas tareas, los pesos de las tareas anteriores se modifican para minimizar el error en las nuevas tareas
  2. *Activation drift*: Se puede enfocar en mantener las activaciones (salidas de la red), aunque se cambien los pesos, lo que mantiene los mismos problemas
  3. *Inter-task confusion*: Lo que se busca es discriminar entre todas las clases, pero como no se aprenden al mismo tiempo, los pesos no pueden discriminarlas a todas
  4. *Task-recency bias*: Las tareas separadas pueden tener salidas no comparables y normalmente el sesgo se va hacia las tareas más recientes.

Métodos[1]



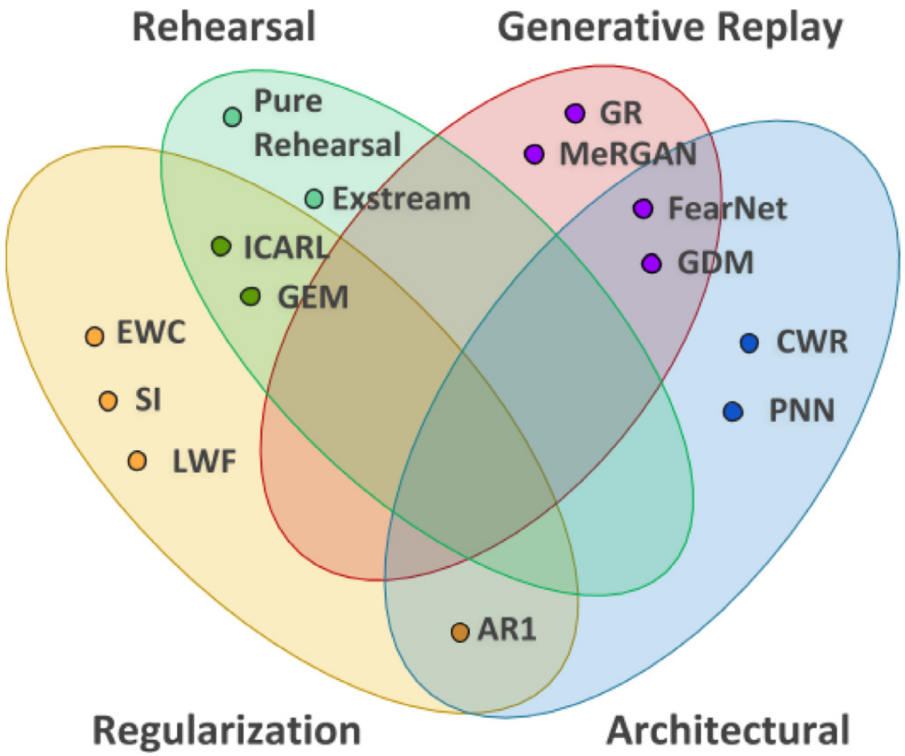
[1] M. Masana, X. Liu, B. Twardowski, M. Menta, A.D. Bagdanov, J. van de Weijer (2021). Class-incremental learning: survey and performance evaluation on image classification.

Métodos[2]



[2]M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, T. Tuytelaars (2021). A continual learning survey: Defying forgetting in classification tasks.

Métodos[3]



[3]T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, N. Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. Information Fusion 58 (2020) 52-68.

Casos Base

- Naïve: Continúa aprendiendo con las nuevas clases
- Completo: Entrena con todos los datos
- Ensamble: Aprende un modelo diferente para cada tarea
- Acumulado: Para cada tarea, entrena desde cero (para la última tarea se vuelve equivalente al enfoque Completo)

Enfoques

Existen diferentes enfoques que se pueden agrupar en métodos basados en:

1. *Replay/Rehearsal based methods*: Rehearsal/psuedo-rehearsal/contrained
2. *Regularization-based methods*: Prior focused/data focused
3. *Dynamic Architecture/Parameter isolation/bias correction methods*: Fixed networks/Dynamic architectures
4. Híbridos

Replay/Rehearsal based methods

- La idea es: Guardar (algunos) ejemplos, usar ejemplos prototípicos, generar ejemplos o mantener descripciones abstractas de las clases para evitar el olvido catastrófico
- *Rehearsal* se refiere a volver a visitar ejemplos pasados para entrenamientos futuros
- *Pseudo-rehearsal* es cuando se usa un modelo generativo para crear los ejemplos
- En *rehearsal*, en general se tiene una memoria fija que se distribuye entre todas las tareas anteriores, por lo que al sacar el gradiente para la tarea actual se toman datos tanto de la memoria (guardados o generados) como de la tarea actual

Replay

- Existen dos restricciones: El tamaño de la memoria, que debe de aproximar la distribución de las clases (*storage policy*), y el tamaño del *mini batch* para el gradiente estocástico (*retrieval policy*)
- Posibles problemas: Sobreajuste a los datos de *rehearsal* lo cual afecta su generaización

iCaRL

- *Incremental Classifier and Representation Learning* (iCaRL) fue uno de los primeros
- Selecciona *examplars* de cada clase, el total está restringido a un tamaño fijo K (se reduce el número de *examplars* por clase al aumentar el número de clases)
- La selección de *examplars* se hace añadiendo el que cause que el promedio actual de los *examplars* seleccionados se acerque más al promedio global de todos los *examplars*
- Clasifica usando el promedio más cercano de los atributos de los *examplars*
- Usa una CNN como extractor de atributos fijos y un número variable de pesos que dependen del número de clases
- Entrena la red con los nuevos ejemplos (*classification*) y los *examplars* (*distillaton*)

iCaRL

Algorithm 1 iCaRL CLASSIFY

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

Algorithm 3 iCaRL UPDATEREPRESENTATION

input $X^s, \ldots, X^t$ // training images of classes $s, \ldots, t$
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // exemplar sets
require Θ // current model parameters
  // form combined training set:

$$\mathcal{D} \leftarrow \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\} \cup \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P^y\}$$

  // store network outputs with pre-update parameters:
for $y = 1, \ldots, s-1$ **do**
  $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}$
end for
run network training (*e.g.* BackProp) with loss function

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

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

GEM (*Gradient Episodic Memory*)

- Guarda un conjunto de ejemplos de cada clase (*episodic memory*)
- Restringe el gradiente de nuevos ejemplos de tal forma que no aumente la pérdida en los datos de la memoria
- GEM es computacionalmente caro ya que resuelve un problema de optimización para sesgar los gradientes

GEM

Algorithm 1 Training a GEM over an *ordered* continuum

procedure TRAIN(f_θ, Continuum$_{\text{train}}$, Continuum$_{\text{test}}$)
  $\mathcal{M}_t \leftarrow \{\}$ for all $t = 1, \ldots, T$.
  $R \leftarrow 0 \in \mathbb{R}^{T \times T}$.
  for $t = 1, \ldots, T$ **do:**
    for (x, y) in Continuum$_{\text{train}}(t)$ **do**
      $\mathcal{M}_t \leftarrow \mathcal{M}_t \cup (x, y)$
      $g \leftarrow \nabla_\theta \ell(f_\theta(x, t), y)$
      $g_k \leftarrow \nabla_\theta \ell(f_\theta, \mathcal{M}_k)$ for all $k < t$
      $\tilde{g} \leftarrow$ PROJECT$(g, g_1, \ldots, g_{t-1})$, see (11).
      $\theta \leftarrow \theta - \alpha\tilde{g}$.
    end for
    $R_{t,:} \leftarrow$ EVALUATE$(f_\theta$, Continuum$_{\text{test}})$
  end for
  return f_θ, R
end procedure

MIR (*Maximally Interfered Retrieval*)

- Guarda un conjunto de ejemplos de cada clase (*replay buffer*) o los genera usando un modelo generativo
- Para el *buffer* selecciona los k ejemplos del *buffer* que producen el mayor aumento en la función de pérdida con respecto a la actualización virtual de los ejemplos

$$\theta^v = \theta - \alpha \nabla \mathcal{L}(f_\theta(\mathbf{X}_t), \mathbf{Y}_t)$$

$$s_{MI-1}(x) = l(f_{\theta^v}(x), y) - l(f_\theta(x), y)$$

- Usa una *variational auto-encoder* (VAE), en donde la idea es generar ejemplos de otras clases que se parezcan más a los ejemplos de la clase actual (de máxima interferencia)

MIR

Algorithm 1: Experience MIR (ER-MIR)

Input: Learning rate α, Subset size C; Budget $\mathcal{B}$

Initialize: Memory $\mathcal{M}$; θ
for $t \in 1..T$ **do**
  for $B_n \sim D_t$ **do**
    %%Virtual Update
    $\theta^v \leftarrow \texttt{SGD}(B_n, \alpha)$
    %Select C samples
    $B_C \sim \mathcal{M}$
    %Select based on score
    $S \leftarrow sort(s_{MI}(B_C))$
    $B_{\mathcal{M}_C} \leftarrow \{S_i\}_{i=1}^{\mathcal{B}}$
    $\theta \leftarrow SGD(B_n \cup B_{\mathcal{M}_C}, \alpha)$
    %Add samples to memory
    $\mathcal{M} \leftarrow UpdateMemory(B_n)$;
  end
end

Algorithm 2: Generative-MIR (GEN-MIR)

Input: Learning rate α

Initialize: Memory $\mathcal{M}$; θ, ϕ,γ
for $t \in 1..T$ **do**
  $\theta^{'}, \phi^{'}, \gamma^{'} \leftarrow \theta, \phi, \gamma$
  for $B_n \sim D_t$ **do**
    %Virtual Update
    $\theta^v \leftarrow \texttt{SGD}(B_n, \alpha)$
    $B_C \leftarrow$ Retrieve samples as per Eq (2)
    $B_G \leftarrow$ Retrieve samples as per Eq (3)
    %Update Classifier
    $\theta \leftarrow SGD(B_n \cup B_C, \alpha)$
    %Update Generative Model
    $\phi, \gamma \leftarrow SGD(B_n \cup B_G, \alpha)$
  end
end

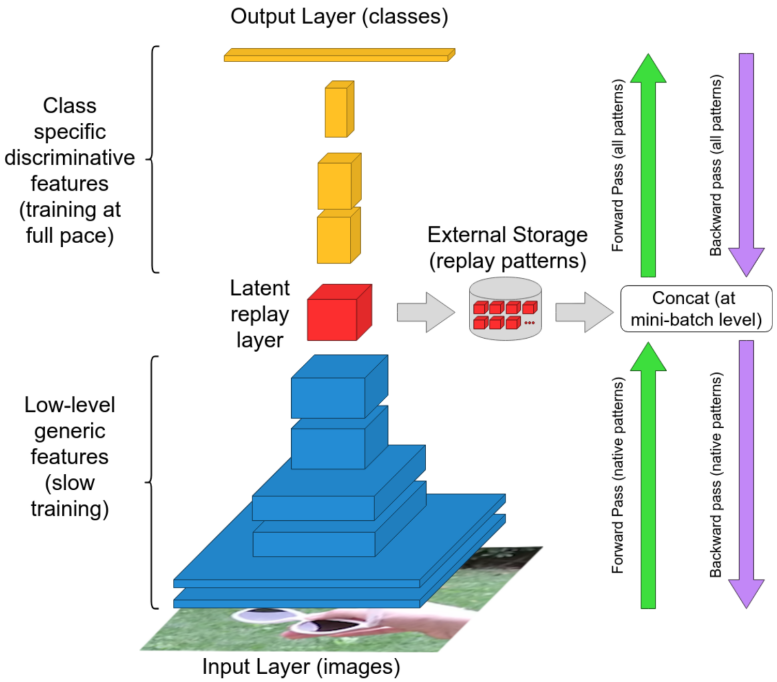
Latent Replay

- En lugar de guardar ejemplos de las clases pasadas, se guarda información de capas intermedias, lo cual reduce las necesidades de memoria
- Reducen el aprendizaje en las capas debajo de la de *latent replay* y dejan que se entrenen normalmente las capas superiores
- Mantienen una memoria externa fija

Latent Replay

Algorithm 1 Pseudocode describing how the external memory RM is populated across the training batches. Note that the amount h of patterns to add progressively decreases to maintain a nearly balanced contribution from the different training batches, but no constraints are enforced to achieve a class-balancing.

$RM = \varnothing$
RM_{size} = number of patterns to be stored in RM
for each training batch B_i:
    train the model on shuffled $B_i \cup RM$
    $h = \frac{RM_{size}}{i}$
    R_{add} = random sampling h patterns from B_i
    $R_{replace} = \begin{cases} \varnothing & \text{if } i == 1 \\ \text{random sample } h \text{ patterns from } RM & \text{otherwise} \end{cases}$
    $RM = (RM - R_{replace}) \cup R_{add}$



GDumb

- Esquema muy sencillo que mostró ser muy competitivo con varios de los algoritmos de CL (ECCV-2020)

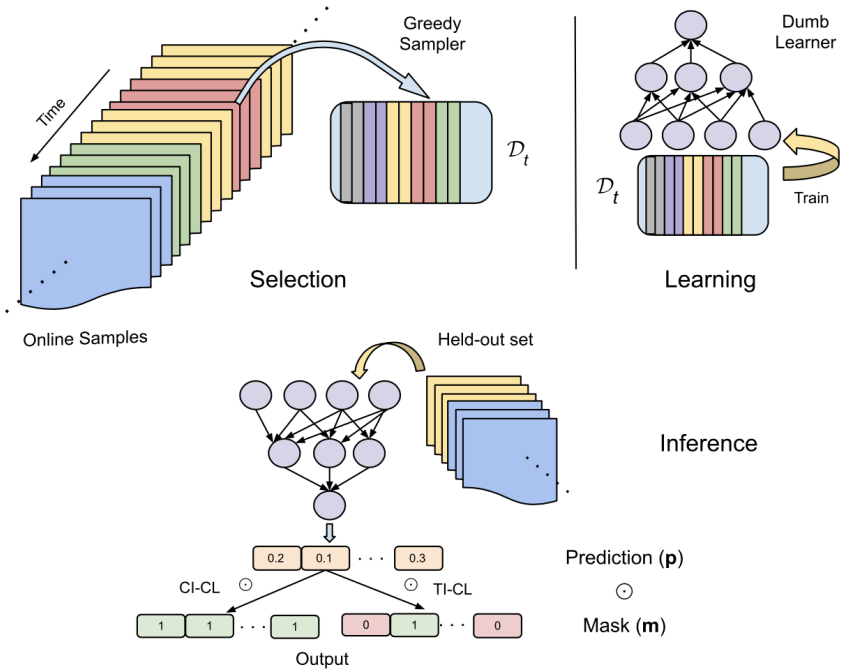


Fig. 1. Our approach (GDumb): The sampler greedily stores samples while balancing the classes. When asked, the learner trains a network from scratch on memory $\mathcal{D}_t$ provided by the sampler. If a mask $\mathbf{m}$ is given at inference, GDumb classifies on the subset of labels provided by the mask. Depending on the mask, GDumb's inference can vary between two extremes: CI (class-incremental) and TI (task-incremental) formulations.

GDumb - Resultados

Method	MNIST	
k	(300)	(500)
MLP-100		
FSS-Clust [37]	75.8 ± 1.7	83.4 ± 2.6
GSS-Clust [37]	75.7 ± 2.2	83.9 ± 1.6
GSS-IQP [37]	75.9 ± 2.5	84.1 ± 2.4
GSS-Greedy [37]	82.6 ± 2.9	84.8 ± 1.8
GDumb (Ours)	**88.9 ± 0.6**	**90.0 ± 0.4**
MLP-400		
GEN [43]	-	75.5 ± 1.3
GEN-MIR [11]	-	81.6 ± 0.9
ER [44]	-	82.1 ± 1.5
GEM [7]	-	86.3 ± 1.4
ER-MIR [11]	-	87.6 ± 0.7
GDumb (Ours)	-	**91.9 ± 0.5**

(A1)

Method	CIFAR10		
k	(200)	(500)	(1000)
GEM [7]	16.8 ± 1.1	17.1 ± 1.0	17.5 ± 1.6
iCARL [4]	28.6 ± 1.2	33.7 ± 1.6	32.4 ± 2.1
ER [44]	27.5 ± 1.2	33.1 ± 1.7	41.3 ± 1.9
ER-MIR [11]	29.8 ± 1.1	40.0 ± 1.1	47.6 ± 1.1
ER5 [11]	-	-	42.4 ± 1.1
ER-MIR5 [11]	-	-	49.3 ± 0.1
GDumb (Ours)	**35.0 ± 0.6**	**45.8 ± 0.9**	**61.3 ± 1.7**

(A1)

Method	MNIST	CIFAR-10
k	(500)	(500)
Fine tuning	18.8 ± 0.6	18.5 ± 0.2
AGEM [36]	29.0 ± 5.3	18.5 ± 0.6
BGD [48]	13.5 ± 5.1	18.2 ± 0.5
GEM [7]	87.2 ± 1.3	20.1 ± 1.4
GSS-Greedy [37]	84.2 ± 2.6	28.0 ± 1.3
HAL [35]	77.9 ± 4.2	32.1 ± 1.5
ER [44]	81.0 ± 2.3	33.3 ± 1.5
MIR [11]	84.9 ± 1.7	34.5 ± 2.0
GMED (ER) [12]	82.7 ± 2.1	35.0 ± 1.5
GMED (MIR) [12]	87.9 ± 1.1	35.5 ± 1.9
GDumb (Ours)	**91.9 ± 0.5**	**45.8 ± 0.9**

(A2)

Method	MNIST		CIFAR10	
	Memory	Accuracy	Memory	Accuracy
Finetune	0	18.8 ± 0.5	0	15.0 ± 3.1
GEN [28]	4.58	79.3 ± 0.6	34.5	15.3 ± 0.5
GEN-MIR [11]	4.31	82.1 ± 0.3	38.0	15.3 ± 1.2
LwF [3]	1.91	33.3 ± 2.5	4.38	19.2 ± 0.3
ADI [47]	1.91	55.4 ± 2.6	4.38	24.8 ± 0.9
ARM [41]	1.91	56.2 ± 3.5	4.38	26.4 ± 1.2
ER [44]	0.39	83.2 ± 1.9	3.07	41.3 ± 1.9
ER-MIR [11]	0.39	85.6 ± 2.0	3.07	47.6 ± 1.1
iCarl [4] (5 iter)	-	-	3.07	32.4 ± 2.1
GEM [7]	0.39	86.3 ± 0.1	3.07	17.5 ± 1.6
GDumb (ours)	**0.39**	**91.9 ± 0.5**	**3.07**	**61.3 ± 1.7**

(A3)

Method	MNIST	SVHN
No memory		
MAS [32]	19.5 ± 0.3	17.3
SI [5]	19.7 ± 0.1	17.3
EWC [6]	19.8 ± 0.1	18.2
Online EWC [29]	19.8 ± 0.04	18.5
LwF [3]	24.2 ± 0.3	-
k=4400		
DGR [28]	91.2 ± 0.3	-
DGR+Distill	91.8 ± 0.3	-
GEM [7]	92.2 ± 0.1	75.6
RtF [50]	92.6 ± 0.2	-
RPS-Net [23]	96.2	88.9
OvA-INN [49]	96.4	-
iTAML [24]	97.9	**94.0**
GDumb (Ours)	**97.8 ± 0.2**	93.4 ± 0.4

(B1)

Method	CIFAR100	
	Acc. (Avg)	Acc. (last)
No memory		
Finetune	17.8 ± 0.72	5.9 ± 0.15
SI [5]	23.6 ± 1.90	13.3 ± 1.14
MAS [32]	24.7 ± 1.76	10.4 ± 0.80
EWC [6]	25.4 ± 1.99	9.5 ± 0.83
RWALK [8]	25.6 ± 1.92	11.1 ± 2.14
LwF [3]	32.3 ± 1.92	14.1 ± 0.87
DMC [14]	45.0 ± 1.96	23.8 ± 1.90
k=2000		
GDumb (Ours)	45.2 ± 1.70	24.1 ± 0.97
DMC++ [14]	56.8 ± 0.86	-
iCARL [4]	58.8 ± 1.90	42.9 ± 0.79
EEIL [18]	63.4 ± 1.6	-
BiC [16]	63.8	46.9

(B2)

Method	CIFAR100	ImageNet100
iCaRL [4]	44.2 ± 1.0	54.97
BiC [16]	47.1 ± 1.5	46.49
UCIR (NME) [17]	48.6 ± 0.4	55.44
UCIR (CNN) [17]	49.3 ± 0.3	57.25
PODNet (CNN) [21]	58.0 ± 0.5	62.08
GDumb (CNN)	58.4 ± 0.8	**62.86**
PODNet (NME) [21]	**61.4 ± 0.7**	-

(B3)

Method	MNIST
(k)	(4400)
GEM [7]	98.42 ± 0.10
EWC [6]	98.64 ± 0.22
SI [5]	99.09 ± 0.15
Online EWC [29]	99.12 ± 0.11
MAS [32]	99.22 ± 0.21
DGR [28]	99.50 ± 0.03
LwF [3]	99.60 ± 0.03
DGR+Distil [28]	99.61 ± 0.02
RtF	99.66 ± 0.03
GDumb	**99.77 ± 0.03**

(C1)

Method	Parameters	Regularization	Accuracy
No stored samples			
mean-IMM [33]	3.5M	none	32.42
mode-IMM [33]	9.0M	dropout	42.41
SI [5]	3.5M/9.0M	L2/dropout	43.74
HAT [51]	3.5M/9.0M	L2	44.19
EWC [6]	613K	none	45.13
LwF [3]	9.0M	L2	48.11
EBLL [52]	9.0M	L2	48.17
MAS [32]	3.5M/9.0M	none	48.98
PackNet [53]	613K/3.5M	L2/dropout	55.96
k=4500			
GEM [7]	613K/3.5M	none/dropout	44.23
GDumb	834K	cutmix	45.50
iCARL [4]	613K/3.5M	dropout	48.55
k=9000			
GEM [7]	613K/3.5M	none/dropout	45.27
iCARL [4]	613K/3.5M	dropout	49.94
GDumb	834K	cutmix	**57.27**

(C2)

Method	CIFAR100
(k)	(1105)
RWalk [8]	40.9 ± 3.97
EWC [6]	42.4 ± 3.02
Base	42.9 ± 2.07
MAS [32]	44.2 ± 2.39
SI [5]	47.1 ± 4.41
iCARL [4]	50.1
S-GEM [36]	56.2
PNN [26]	59.2 ± 0.85
GEM [7]	61.2 ± 0.78
A-GEM [36]	63.1 ± 1.24
TinyER [34]	68.5 ± 0.65
GDumb	60.3 ± 0.85

(D)

Method	MNIST	CIFAR10
Reservoir [43]	69.12	-
GSS-Clust [37]	-	25.0
FSS-Clust [37]	-	26.0
GSS-IQP [37]	76.49	29.6
GSS-Greedy [37]	77.96	29.6
GDumb (Ours)	**88.93**	**45.8**

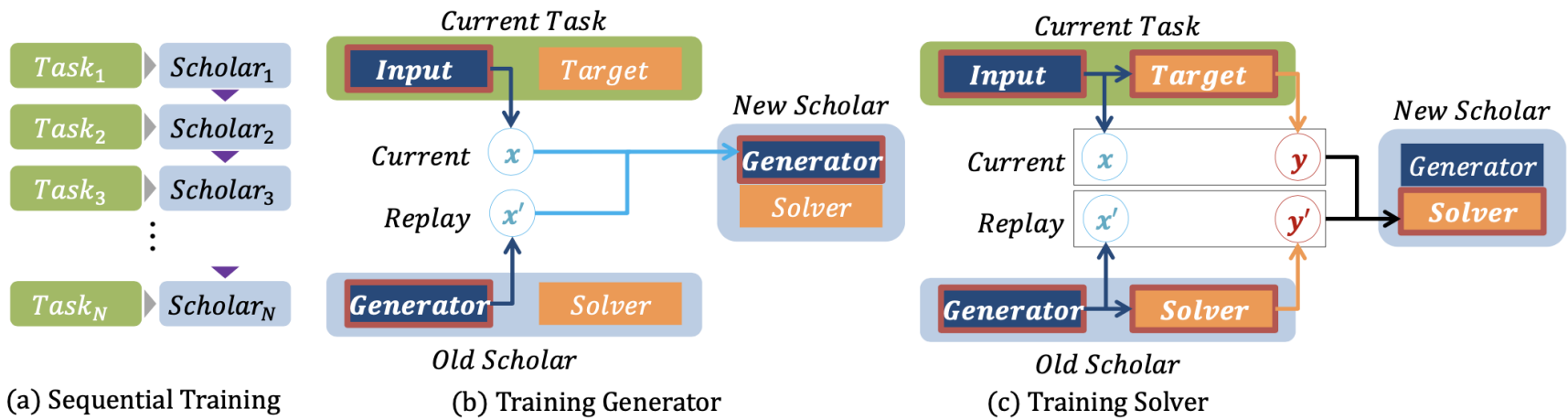
(E)

Method	Train time	Memory (Train)	Memory (Test)
Base	105s	P + B*H	P + B*H
EWC	250s	4*P + B*H	P + B*H
PNN	409s	2*P*T + B*H*T	2*P*T + B*H*T
GEM	5238s	P*T + (B+M)*H	P + B*H
A-GEM	449s	2*P + (B+M)*H	P + B*H
GDumb	**60s**	**P + M*H**	**P + B*H**

(Resources)

Deep Generative Replay

- En lugar de guardar ejemplos, se crea un modelo generativo
- Se entrena un nuevo generador para imitar la distribución de los datos de la tarea actual y de las tareas anteriores
- Se aprende un nuevo generador con los ejemplos de la tarea nueva y los generados considerando el generador anterior



(a) Sequential Training (b) Training Generator (c) Training Solver

Aspectos a considerar

- Esquemas de memoria: Incremental, fija, ...
- Estrategias de muestreo: Aleatoria, creación de ejemplos prototípicos, selección usando heurísticas (entropía, distancia a las fronteras de decisión, ...)
- Los datos de las otras tareas se pueden usar como regularizadores
- Balance de tareas: Aunque se usen ejemplos de las tareas anteriores, existe un desbalance con el número de ejemplos de las tareas nuevas
- Se han propuesto combinar este enfoque con esquemas de regularización, pero no se han visto claros beneficios

Métodos basados en regularización

- Añaden términos a la función de pérdida para reducir el "olvido catastrófico"
- Los métodos basados en regularización en los pesos buscan evitar el *weight drift*
- Determinan la importancia de los parámetros y la usan para penalizar cambios en los más importantes
- Por ejemplo, añadir en la función de pérdida:

$$\mathcal{L}_{reg}(\theta^t) = \frac{1}{2} \sum_{i=1}^{|\theta^{t-1}|} \Omega_i (\theta_i^{t-1} - \theta_i^t)^2$$

donde $|\theta^{t-1}|$ es el número de pesos de la red y Ω tiene los valores de importancia de cada peso de la red

Elastic Weight Consolidation (EWC)

- En el cerebro la consolidación sináptica reduce la platicidad en las sinapsis que son vitales para tareas previamente aprendidas
- Desde un punto de vista probabilista, queremos encontrar los valores de los parámetros de la red más probables dados los datos $p(\theta|D)$
- Usando Bayes y sacando el logaritmo:

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

Elastic Weight Consolidation

Si tenemos una secuencia de datos independientes $D = \{D_A, D_B\}$ (D_B apareciendo después de D_A)

$$\log p(\theta|D) = \log p(D_B|D_A, \theta) + \log p(\theta|D_A) - \log p(D_B|D_A)$$

$$\log p(\theta|D) = \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B)$$

Donde:

- $\log p(D_B|\theta)$ es el negativo de la función de pérdida sobre la tarea actual
- $\log p(\theta|D_A)$ es la información que se tiene de la tarea anterior (en donde debe de estar información sobre qué parámetros en θ son los importantes)
- O sea que la posterior $p(\theta|D_A)$ para la tarea A se vuelve ahora la *prior* de B

Elastic Weight Consolidation

- Como no podemos calcular $p(\theta|D_A)$, la aproximamos con una distribución gaussiana, cuya media son los parámetros θ^*_A y cuya covarianza es el inverso del negativo de la matriz de información de Fisher (F).

$$\log(P(\theta|D_A)) = \frac{1}{2}(\theta-\theta^*_{D_A})^T \left(\frac{\partial^2(\log(P(\theta|D_A)))}{\partial^2\theta}\right)(\theta-\theta^*_{D_A})+\Delta$$

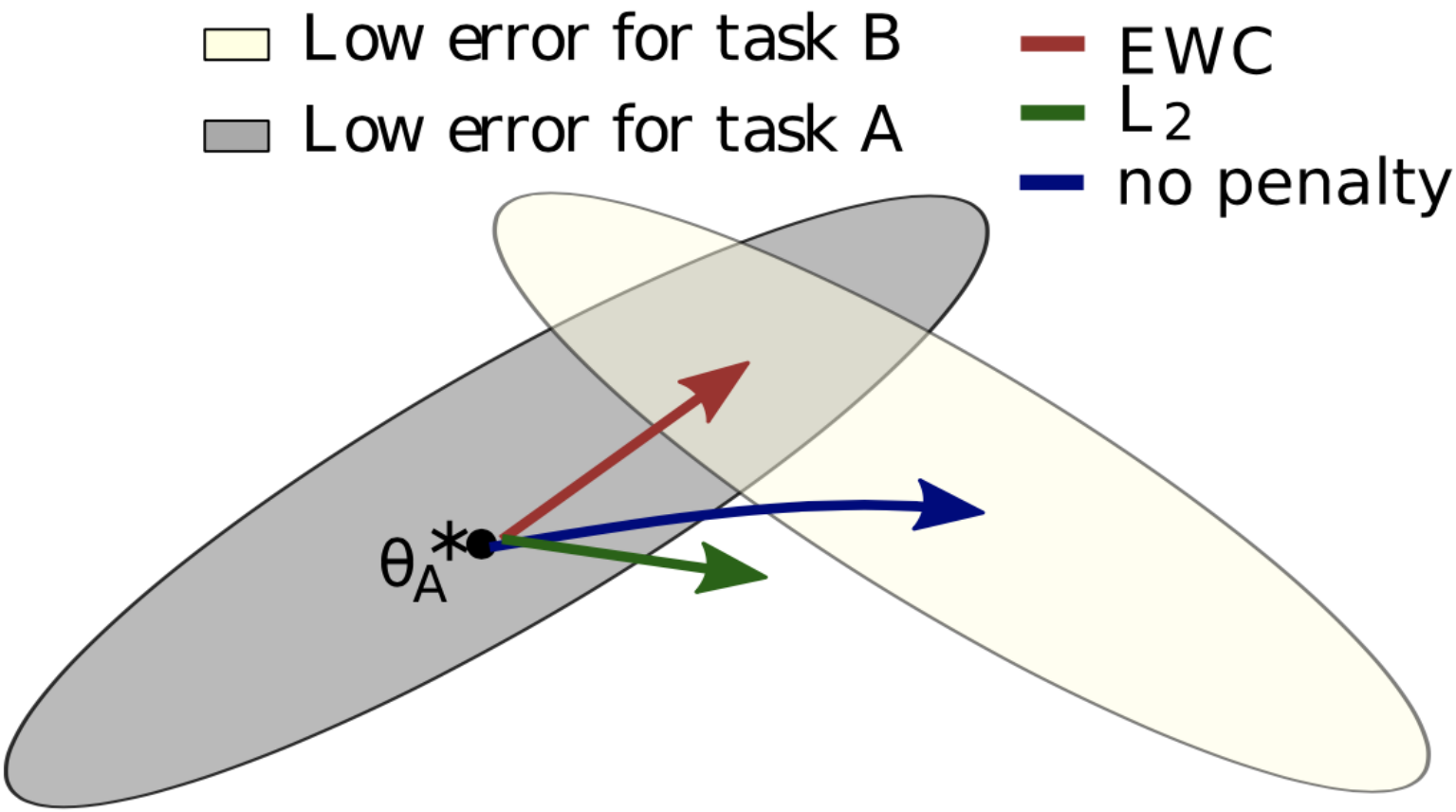
donde $\Delta = \log(P(\theta^*_{D_A}|D_A))$

- Lo que se minimiza en EWC para la tarea K es:

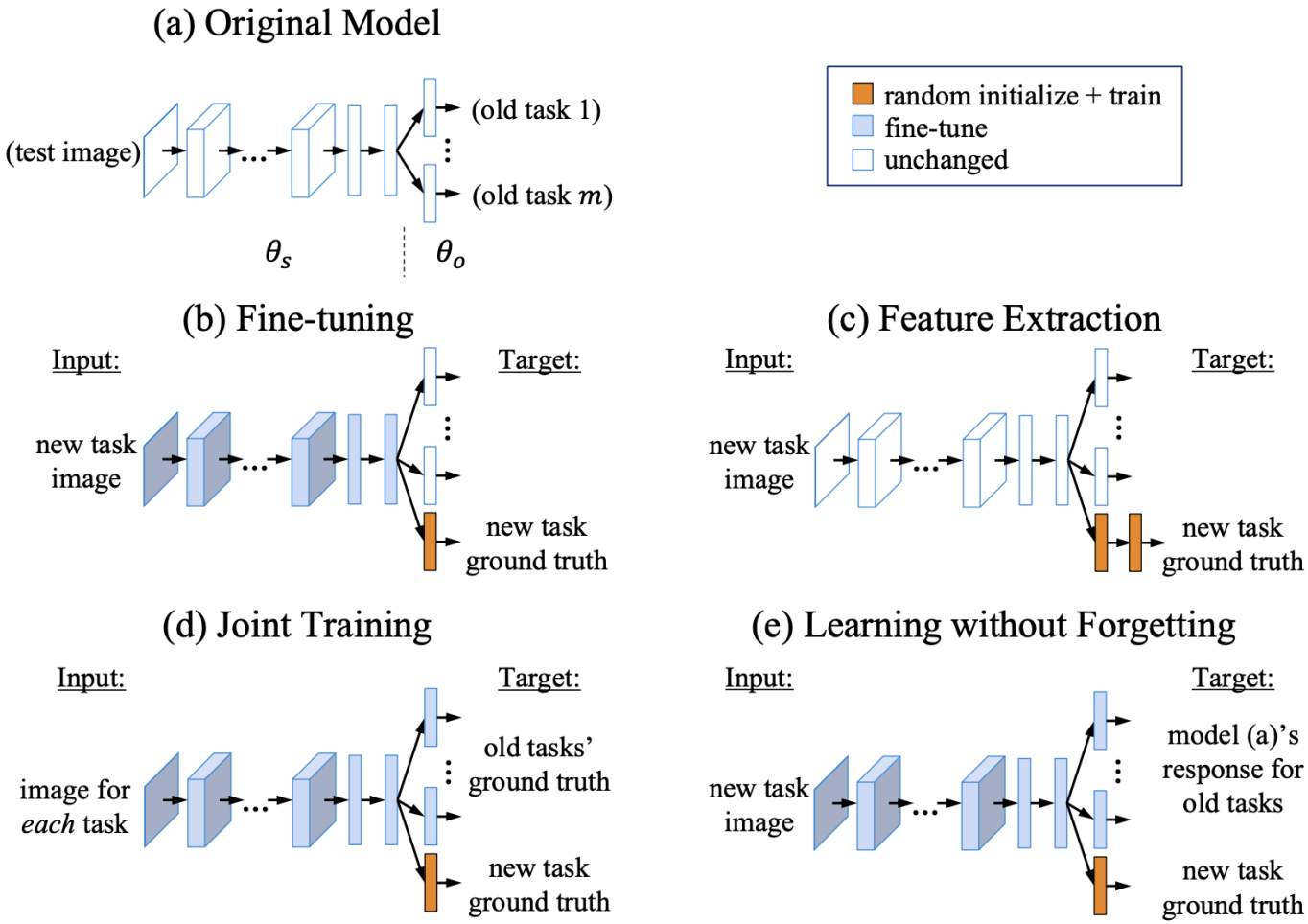
$$\mathcal{L}(\theta) = \mathcal{L}_K(\theta) + \sum_{k=1}^{K-1}\frac{\lambda}{2}\sum_{i=1}^{N_{params}} F^k_{ii}(\theta_i - \hat{\theta}_{k,i})^2$$

donde $\hat{\theta}_{k,i}$ en el i-ésimo elemento de θ_k, que es el vector de parámetros después de entrenar para la tarea k y F^k_{ii} en la diagonal de la matriz de información de Fisher

EWC



Learning without Forgetting (LwF)



Learning without Forgetting (LwF)

- Usa sólo ejemplos de las clases nuevas
- Dada una red con parámetros compartidos (θ_s) y parámetros específicos para una(s) tarea(s) (θ_o), el objetivo es aprender parámetros para una nueva tarea (θ_n) sin perjudicar las tareas anteriores y sin los datos de las tareas anteriores
- Primero se obtienen las salidas y_o para la red anterior en las nuevas tareas
- Se añaden nodos para cada una de las nuevas clases
- Se entrena la red para minimizar la función de pérdida en todas las clases junto con un regularizador
- Primero se congelan los pesos de θ_s y θ_o (*warm-up*) y se entrena θ_n hasta converger y después de entrena toda la red

Learning without Forgetting (LwF)

LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

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

Initialize:

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

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

Train:

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

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

$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

Synaptic Intelligence

- Idea: Darle una medida de importancia a los pesos
- Se busca encontrar una función de pérdida útil para todas las tareas
- Podemos calcular el cambio en la función de pérdida (L) cuando hacemos una actualización de los parámetros ($\delta(t)$) en el tiempo t, usando el gradiente de L ($g = \frac{\partial L}{\partial \theta}$):

$$L(\theta(t) + \delta(t)) - L(\theta(t)) \approx \sum_k g_k(t)\delta_k(t)$$

donde $\delta_k(t) = \theta'_k(t)$

Synaptic Intelligence

- Si lo hacemos para todo el entrenamiento:

$$\int g(\theta(t))d\theta = \int_{t^{\mu-1}}^{t^{\mu}} g(\theta(t)) \cdot \theta'(t)dt = -\sum_{k} w_k^{\mu}$$

donde se puede ver como la suma de las contribuciones de todos los pesos w_k^{μ} (el signo menos es porque estamos interesador en decrementar la función de pérdida)

Synaptic Intelligence

- La importancia de un parámetro θ_k para una tarea está determinado por: (i) cuánto contribuye en la caída de la función de pérdida durante el entrenamiento (w_k^ν) y (ii) qué tanto se movió ($\delta_k^\mu \equiv \theta_k(t^\nu) - \theta_k(t^{\nu-1})$)
- Para evitar cambios en parámetros importantes se usa una función surrogada:

$$\tilde{L}_\mu = L_\mu + c \sum_k \Omega_k^\mu (\tilde{\theta}_k - \theta_k)^2$$

donde los pesos de referencia son de los parámetros en la tarea anterior $\tilde{\theta}_k = \theta_k(t^{\mu-1})$

- Y la fuerza de cada parámetro es:

$$\Omega_k^\mu = \sum_{\nu<\mu} \frac{w_k^\nu}{(\delta_k^\nu)^2 + \xi}$$

Variantes ...

- *Learning without Memorizing* (LwM): La idea es que la atención de la red entrenada con las tareas anteriores no debería de cambiar con las nuevas tareas
- La función de pérdida de atención es:

$$\mathcal{L}_{AD}(x;\theta^t) = \| \frac{Q^{t-1}(x)}{\|Q^{t-1}(x)\|_2} - \frac{Q^t(x)}{\|Q^t(x)\|_2} \|$$

- Donde el mapa de atención Q está dado por:

$$Q^t(x) = \text{Grad-CAM}(x, \theta^t, c)$$

generado por el algoritmo de Grad-CAM, el cual calcula el gradiente con respecto a la clase objetivo (c) para producir las regiones de la imagen que más contribuyeron a la predicción

- Lo que se usa son predicciones sobre la clase más probable de las anteriormente aprendidas

Hypernetworks

- Idea: Crear un modelo que aprenda a generar pesos a las redes

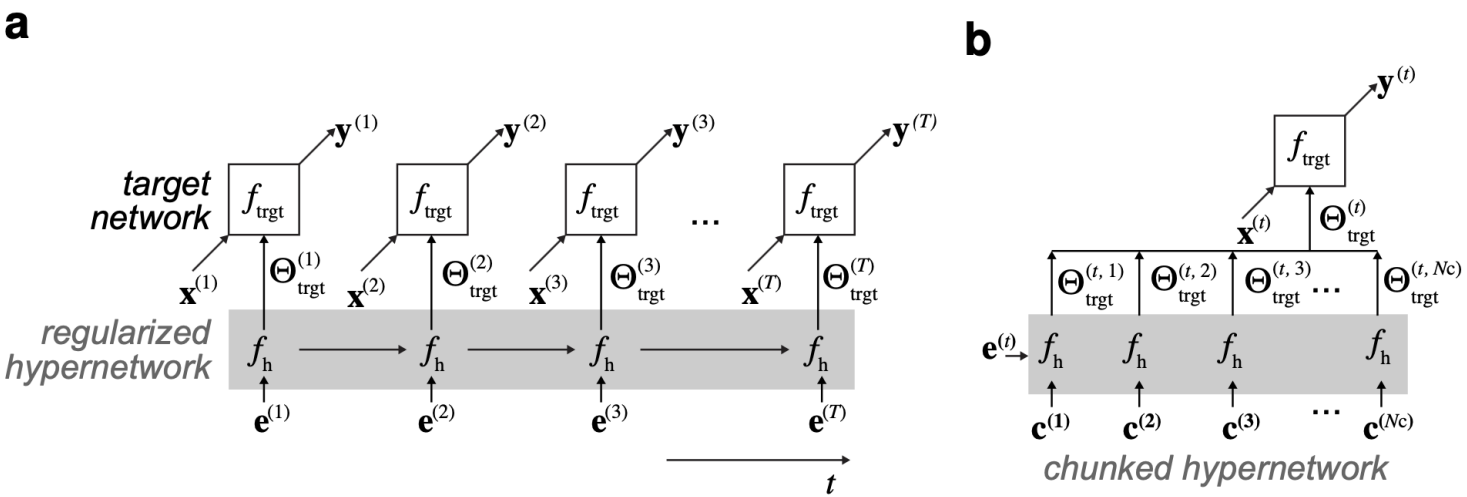


Figure 1: **Task-conditioned hypernetworks for continual learning.** **(a)** Commonly, the parameters of a neural network are directly adjusted from data to solve a task. Here, a weight generator termed *hypernetwork* is learned instead. Hypernetworks map embedding vectors to weights, which parameterize a target neural network. In a continual learning scenario, a set of task-specific embeddings is learned via backpropagation. Embedding vectors provide task-dependent context and bias the hypernetwork to particular solutions. **(b)** A smaller, chunked hypernetwork can be used iteratively, producing a chunk of target network weights at a time (e.g., one layer at a time). Chunked hypernetworks can achieve model compression: the effective number of trainable parameters can be smaller than the number of target network weights.

Hypernetworks

- En lugar de aprender a obtener los parámetros (θ_{trgt}) para una función particular (f_{trgt}), se aprenden los parámetros (θ_h) de un meta-modelo, cuya salida es θ_{trgt}
- En la práctica, en lugar de producir todos los pesos de la red, produce incrementalmente (en *chunks*) los de cada capa

Dynamic Architecture/Parameter isolation methods

- La arquitectura de puede cambiar explícita o implícitamente
- Explícita: Para cada nueva tarea se crea un nuevo modelo conectado con los anteriores
- Si no hay restricciones en cuanto al tamaño de la arquitectura, se pueden crear nuevas ramas para las nuevas tareas, congelando los parámetros anteriores

Dynamic Architecture/Parameter isolation methods

- Implícita: Usa diferentes parámetros para cada tarea sin cambiar la arquitectura
- Se pueden congelar algunos pesos, el problema es buscar congelar suficientes, para no olvidar, pero no tantos, como para no aprender cosas nuevas (e.g., PackNet)
- Alternativamente se pueden definir caminos dinámicos para usar un camino particular para diferentes tareas sin modificar pesos anteriores (e.g., PathNet)
- También se pueden definir los pesos importantes usando un mecanismo de atención (e.g., HAT)

PackNet

- Empieza con una red pre-entrenada en ImageNet, e.g., VGG-16
- Poda un porcentaje de pesos (los vuelve ceros), eliminando los que tienen pesos más bajos, y re-entrena la red para la tarea 1 por unas cuantas épocas, los nuevos pesos se fijan
- Entrena la red para la tarea 2, usando los pesos de la tarea 1 fijos, y sigue el proceso (poda, sólo los de la tarea actual, re-entrena y fija)
- Para la tarea N tiene pesos fijos para las $N-1$ tareas y algunos pesos “libres” que se usan para entrenar la tarea

PackNet

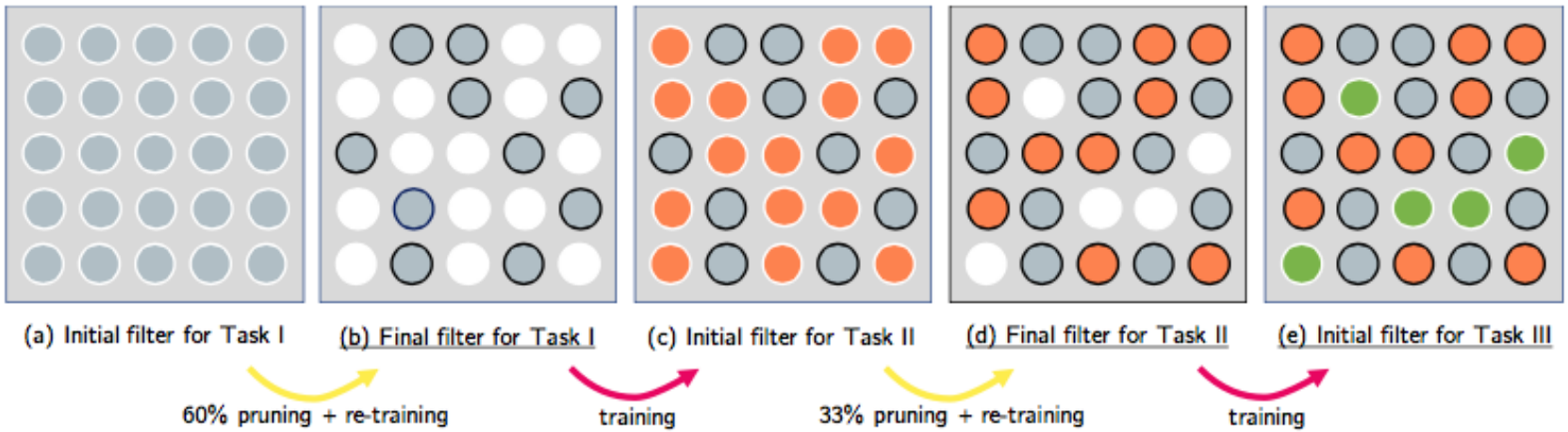
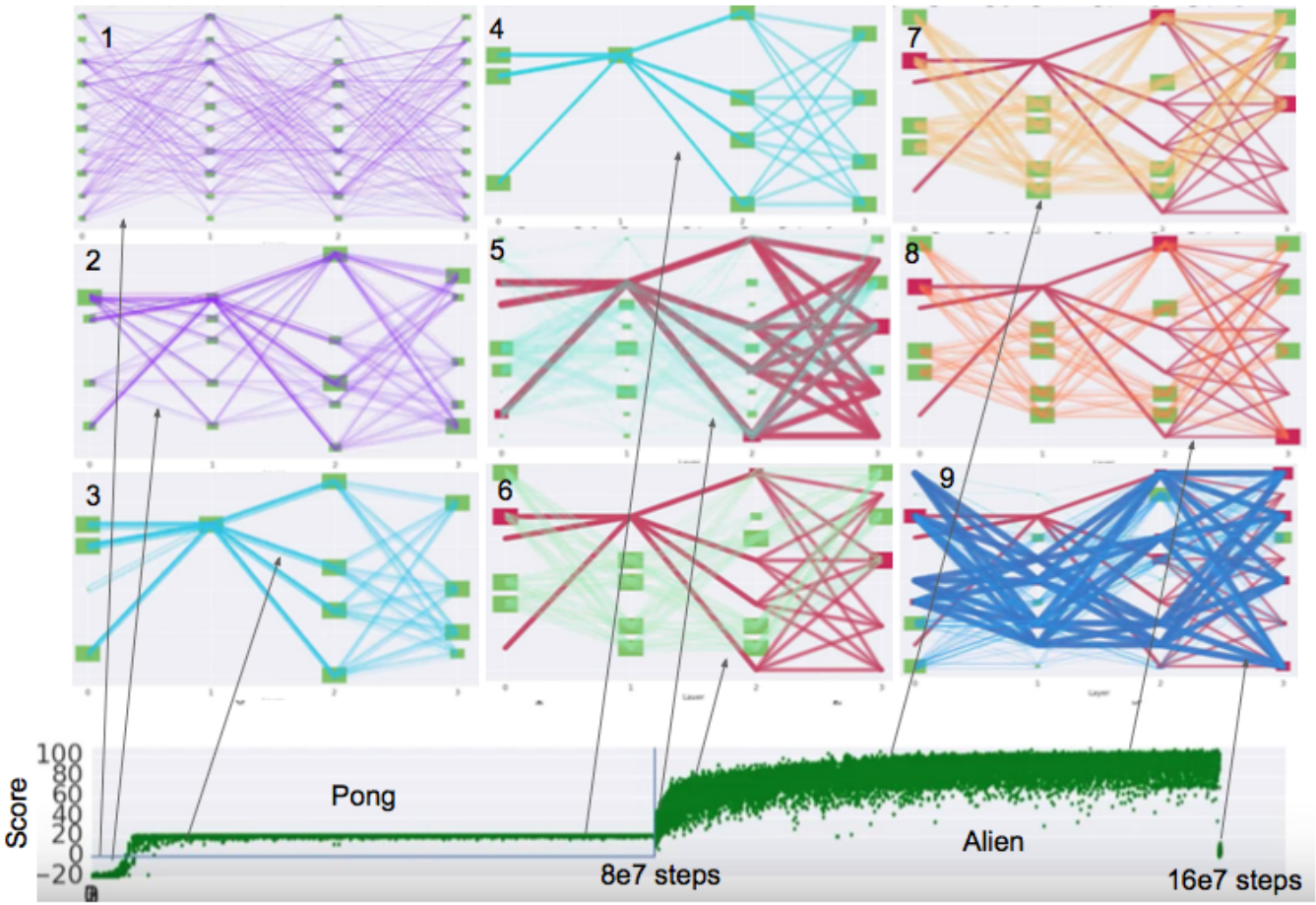


Figure 1: Illustration of the evolution of a 5×5 filter with steps of training. Initial training of the network for Task I learns a dense filter as illustrated in (a). After pruning by 60% (15/25) and re-training, we obtain a sparse filter for Task I, as depicted in (b), where white circles denote 0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

PathNet

- PathNet es una red neuronal multicapa y en cada capa hay M modulos, los cuales son redes neuronales
- Las salidas de los modulos de cada capa se suman antes de pasar a la siguiente capa
- Los modulos se activan para cada camino y se permiten entre 3 y 4 modulos por camino
- Se inicializan aleatoriamente N caminos, se entrena cada camino y se seleccionan por torneo (GA)
- Cuando converge, se fija ese camino, y se re-inicializan aleatoriamente los caminos restantes para la siguiente tarea

PathNet



HAT

- *Hard Attention to the Task* (HAT), incluye un mecanismo de atención para identificar las tareas
- Las salidas de cada capa l (h_l) se multiplican por un vector a_l^t ($h'_l = a_l^t \odot h_l$), donde a_l^t es un *embedding* de una sola capa ($a_l^t = \sigma(se_l^t)$) y σ es una sigmoide
- Todas las capas funcionan igual salvo la última donde a_l^t es un vector binario pre-determinado
- La idea del mecanismo de atención es crear filtros sobre qué unidades usar en cada capa dependiendo de la tarea

HAT

- Los gradientes se condicionan con la atención acumulada de las tareas pasadas:

$$a_l^{\leq t} = \max(a_l^t, a_l^{\leq t-1})$$

- y los gradientes ($g_{l,ij}$) de la capa l se modifican con el contrario del mínimo de la atención acumulada

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

donde i, j se refieren a la salida (l) y entrada ($l-1$) capas respectivamente

HAT

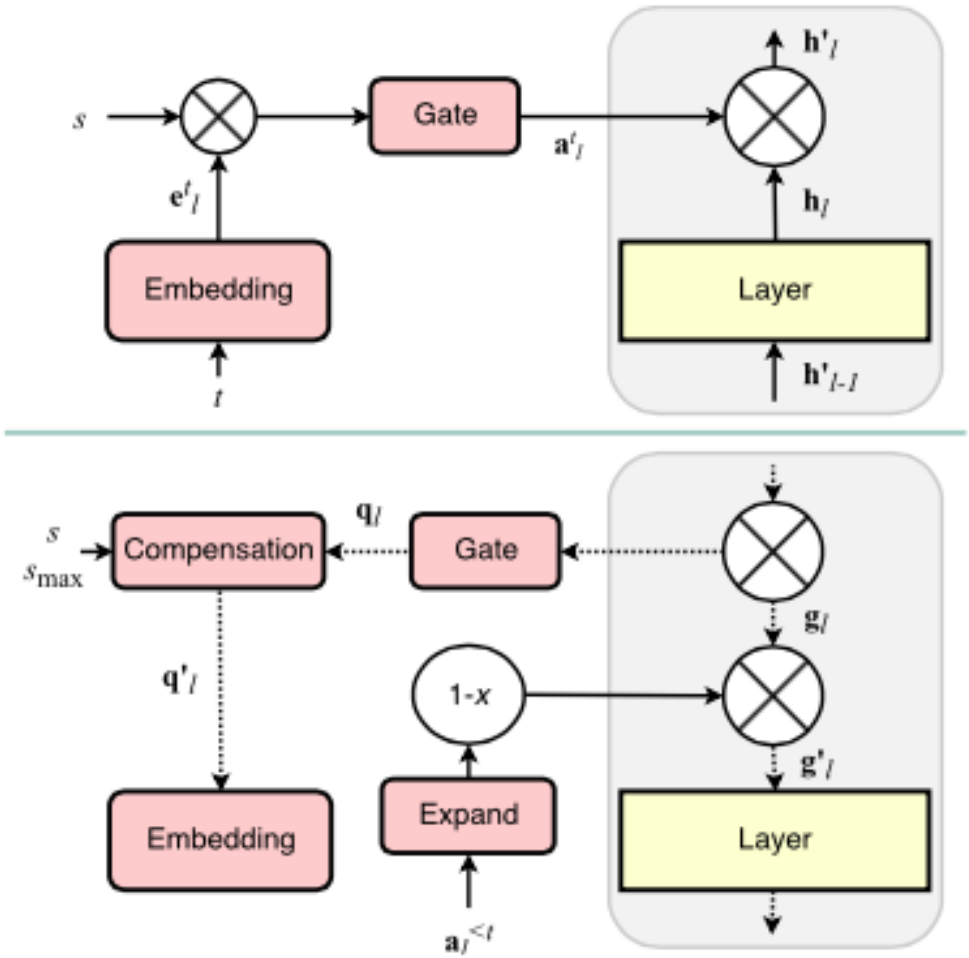


Figure 1. Schematic diagram of the proposed approach: forward (top) and backward (bottom) passes.

Arquitecturas Maestro-Estudiante

- Se aprenden dos arquitecturas: (i) Una modela la tarea actual y se espera que sea fácilmente adaptable y (ii) la otra se usa como memoria de las tareas anteriores
- En un esquema Maestro-Estudiante, el maestro puede tener una o varias GANs para diferentes tareas
- Si se tiene una nueva tarea se expande la GAN (se ajusta o se añade una nueva), si no, se ajusta con los nuevos datos
- El estudiante aprende del maestro que genera los ejemplos para todas las tareas incluyendo la nueva, haciendo la labor de *knowledge distillation*

Evaluación

Existen diferentes medidas de evaluación:

- Desempeño relativo entre un algoritmo entrenado incrementalmente y uno con todos los datos

$$\omega_{all} = \frac{1}{T} \sum_{t=1}^{T} \frac{\alpha_{all,t}}{\alpha_{offlline,t}}$$

- Considerando una razón de olvido:

$$\rho^{\leq t} = \frac{1}{t} \sum_{\tau=1}^{t} \frac{A^{\tau \leq t} - A_R^{\tau}}{A_J^{\tau \leq t} - A_R^{\tau}} - 1$$

donde $A^{\tau \leq t}$ es la *accuracy* de la tarea τ después de aprender en forma secuencial la tarea t, A_R^{τ} es la *accuracy* de un clasificador estratificado aleatorio usando información de la tarea τ y $A_J^{\tau \leq t}$ es la *accuracy* de la tarea τ después de aprender t tareas en conjunto

Evaluación

- Se puede medir el desempeño promedio, medirlo considerando *backward transfer* (de las tareas anteriores) o BWT y considerando *forward transfer* (de las tareas posteriores) o FWT

R	Te_1	Te_2	Te_3
Tr_1	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$
Tr_2	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$
Tr_3	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$

Evaluación

- También se puede considerar en cada paso el desempeño de la diagonal y de todos los elementos que están abajo

$$A = \frac{\sum_{i \geq j}^{N} R_{i,j}}{\frac{N(N+1)}{2}}$$

- BTW

$$BWT = \frac{\sum_{i=2}^{N} \sum_{j=1}^{i-1} (R_{i,j} - R_{j,j})}{\frac{N(N+1)}{2}}$$

- FWT

$$FWT = \frac{\sum_{i<j}^{N} R_{i,j}}{\frac{N(N+1)}{2}}$$

Evaluación

- Eficiencia en tamaño del modelo: Comparar el tamaño de memoria en términos de los parámetros (θ) usado en comparación con la memoria del primer modelo:

$$MS = \min(1, \frac{\sum_{i=1}^{N} \frac{Mem(\theta_1)}{Mem(\theta_i)}}{N})$$

- Eficiencia en ejemplos guardados:

$$SSS = 1 - \min(1, \frac{\sum_{i=1}^{N} \frac{Mem(M_1)}{Mem(D)}}{N})$$

- También se pueden hacer comparaciones en términos de eficiencia

CL en Robótica

- Un robot debe de interactuar con su medio ambiente
- Algunas restricciones: Memoria y capacidad de cómputo limitada, movilidad restringida, inestables y frágiles, si algo se aprende mal puede afectar tareas futuras
- Algunas ventajas son: Información multimodal, que permite obtener conocimiento de diferentes sensores; están en control de sus interacciones, lo cual puede ayudar a obtener información causal
- CL se puede aplicar en percepción, aprendizaje por refuerzo, aprendizaje basado en modelos

Áreas actuales

- Optimizar la memoria creando ejemplares más eficientes (y efectivos)
- Generar atributos (más que ejemplos) de las capas intermedias
- Auto-aprendizaje y aprendizaje no supervisado incremental
- Investigación en otras funciones de pérdida
- Meta-Aprendizaje para aprendizaje continuo
- Escenarios de aprendizaje más realistas en donde las tareas pueden estar traslapadas

Idealmente

- Memoria constante
- Sin fronteras entre tareas
- Aprendizaje en línea
- *Forward transfer* (poco trabajo, *zero-shot learning*)
- *Backward transfer* (lo más buscado para evitar CF)
- Aplicable no solo en clasifición
- Adaptativo
- No requiere de un oráculo para probarlo
- Puede volver a visitar tareas anteriores
- Olvido gradual (*graceful forgetting*)

Áreas Relacionadas

- *Multi-task learning*: Aprender varias tareas al mismo tiempo en donde se puede compartir parte de los modelos
- *Transfer Learning*: Explotar conocimiento aprendido de otra tarea para aprender mejor/más rápido la tarea actual
- *Concept Drift/Domain Adaptation*: Cambios no anticipados en la distribución de los datos que requieren modificar el modelo actual
- *Learning to learn (meta learning)*: Aprender qué hacer para aprender más rápido o adaptarse más rápido

Áreas Relacionadas

- *On-line learning*: Aprender a partir de un flujo de datos
- *Open world learning*: Detectar nuevas clases durante el periodo de pruebas e integrarlas
- *Curriculum learning*: Se da una secuencia de tareas en un orden pre-establecido para aprender una tarea compleja
- *Active Learning*: El algoritmo puede interactuar con el usuario para obtener información de las etiquetas para nuevos ejemplos

Algunas Referencias

- Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D. Bagdanov, Joost van de Weijer (2021). Class-incremental learning: survey and performance evaluation on image classification.
- Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, Tinne Tuytelaars. A continual learning surver: Defying forgetting in classification tasks.
- T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, N. Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. Information Fusion 58 (2020) 52-68.
- N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, D. Maltoni. Don't forget, there is more than forgetting: new metrics for Continual Learning.