

# Open-World Machine Learning

Eduardo Morales, Hugo Jair Escalante

INAOE

# Outline

- 1 Introducción
- 2 Identificación de desconocidos
- 3 Aprendizaje Continuo
- 4 Métodos
- 5 Evaluación
- 6 Open-World Learning
- 7 Trabajo Actual y Futuro

# 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 de mundo abierto busca desarrollar sistemas que puedan aprender continuamente para lidiar con nuevas tareas, preservando lo aprendido en tareas previas
- Aunque presente, desde los inicios del área, el área ha tomado relevancia con los desarrollos de DL por las siguientes razones:
  - Son muy costosos de entrenar como para pensar en su actualización
  - Parece que su desarrollo se está estancando y el buscar nuevas líneas de investigación resulta atractivo

# Aprendizaje de Mundo Abierto

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

# Algunos Retos

- ¿Cómo identificar información o situaciones desconocidas?
- ¿Cómo fomentar la curiosidad?
- ¿Cómo fomar hipótesis y qué estrategias seguir para probarlas?
- ¿Cómo mantener el aprendizaje a lo largo de la vida útil del sistema sin perder desempeño?
- ¿Cómo aprender sin/con poca interacción del usuario?

# Algunos Retos para Clasificación

- ¿Cómo identificar clases desconocidas?
- ¿Cómo representar las clases para facilitar su clasificación y discriminación con clases nuevas?
- ¿Cómo lidiar con muy pocos ejemplos por clase y de todos modos aprender un concepto robusto?
- ¿Cómo mantener el aprendizaje de clases a lo largo de la vida útil del sistema sin perder desempeño?
- ¿Cómo aprender sin/con poca interacción del usuario?

# Áreas Relacionadas

- Continual Learning
- Incremental Learning
- Contrastive Learning
- OOD y Open Set Recognition
- Novel Class Discovery
- Life-Long Learning
- Semi-Supervised Open-World Learning
- Open-World Learning

# Otras Áreas Relacionadas

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

- Multi-task Learning
- Transfer Learning
- Concept Drift/Domain Adaptation
- Learning to Learn (Meta Learning)
- On-line Learning
- Curriculum Learning
- Active Learning

# Algunas Aplicaciones

- Vehículos autónomos
- Diagnóstico médico
- LLMs
- Robótica

# Rechazo de desconocidos

- En lugar de dar la clasificación más probable, se busca identificar que se trata de una clase desconocida
- Se ha abordado con diferentes nombres: *anomaly detection*, *out-of-distribution* (OOD) y *open-set recognition* (OSR)
- En OOD la distancia semántica entre ejemplos OOD y ejemplos conocidos (*in-distribution* - ID) tiende a ser grande (no relacionados), mientras que en OSR un subconjunto de clases se toma como ID y otro subconjunto de clases como OOD
- ODD se basa en detección y OSR también en desempeño de ID

# OOD

- Se basan en una medida de desempeño ( $S(x)$ ) y un umbral, si la muestra tiene una medida arriba del umbral se considera ID y si no, se considera OOD
- Inicialmente se uso *softmax*:

$$p(c|x) = \frac{e^{z_c(x)}}{\sum_{j=1}^K e^{z_j(x)}}$$

donde  $z_c(x)$  es el *logit* ( $\log(p/(1 - p))$ ) de  $x$  y entonces  $S(x) = \max_c p(c|x)$

- Sin embargo, las NNs pueden producir niveles de confianza muy altos para datos alejados de los datos de entrenamiento
- La distribución porterior del *softmax* puede tener un etiquetado de salida con sobreajuste

# Funciones de Energía

- Buscando reducir esto, se propuso usar una función de energía, que mapea una entrada  $x$  a un escalar llamado energía
- Un conjunto de valores de energía se pueden transformar en una densidad de probabilidad usando la distribución de Gibbs:

$$p(y|x) = \frac{e^{-E(x,y)/T}}{\int_{y'} e^{-E(x,y')/T}}$$

- El denominador se llama función de partición
- La función de energía libre de Helmholtz  $E(x)$  se puede expresar con el negativo del logaritmo de la función de partición:

$$E(x) = -T \ln \int_{y'} e^{-E(x,y')/T}$$

# Función de Energía

- Si consideramos a  $E(x, y) = -f_y(x)$ , donde  $f_y(x)$  es el *logit* de la clase  $y$  en una distribución de  $K$  clases:

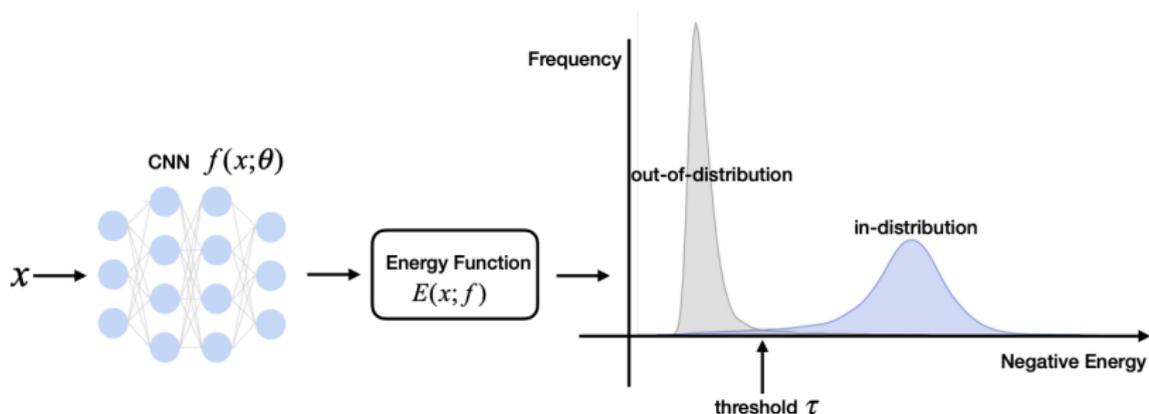
$$E(x; f) = -T \ln \sum_i^K e^{f_i(x)/T}$$

- Para detectar OOD:

$$g(x; \tau, f) = \begin{cases} 0 & \text{if } -E(x; f) \leq \tau \\ 1 & \text{if } -E(x; f) > \tau \end{cases}$$

- Se usa el negativo para que valores altos correspondan a ejemplos de la clase y valores bajo a OOD

# Energy-Based OOD



Open-World  
Machine  
Learning

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

Aprendizaje  
Continuo

Métodos

Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro

## ¿Por qué softmax sobreajusta?

- Si buscamos la clase con más probabilidad:

$$\max_y p(c|x) = \max_y \frac{e^{f_y(x)}}{\sum_i e^{f_i(x)}} = \frac{e^{f^{\max}(x)}}{\sum_i e^{f_i(x)}}$$

- Si sacamos el logaritmo y suponemos que  $T = 1$ :

$$\ln \max_y p(c|x) = \underbrace{E(x; f(x))}_{\downarrow \text{ for ID } x} + \underbrace{f^{\max}(x)}_{\uparrow \text{ for ID } x}$$

- Entonces *softmax* es un caso especial de la medida basada en energía con  $T = 1$  donde los *logits* son sesgados por el *logit* más alto

# Open-Set Recognition

Existen tres tipos de modelos:

- Discriminativos: Incluyen una clase extra de OOD, e.g., OpenMax, evalúa la probabilidad de pertenecer a esa clase (clase 0) y usa una distribución de Weibull para recalibrar pesos
- Generativos: Generan ejemplos conocidos y desconocidos o generan puros desconocidos usando GANs
- Híbridos: Combinan los dos

# MetaMax

- Obten todas los vectores de activación de los ejemplos de cada clase (antes de *softmax*)
- Con todos los que son *no-match* (i.e., los vectores de las otras clases para los ejemplos de la clase), construye una distribución de Weibull ( $f(x; \lambda, k)$ )
- Supone que existe una clase desconocida
- Usa el (meta) modelo para decidir si aceptar o no el resultado del *softmax*
- Inicialmente  $m_i = 1$  (ajuste) para todas las clases,  $\beta =$  número de las clases con valores más altos a revisar

# MetaMax

- Si el vector de activación del ejemplo  $a_{b_i}$  es mayor que el de la clase  $a_j$ , no hagas nada
- Si no, calcula el ajuste:

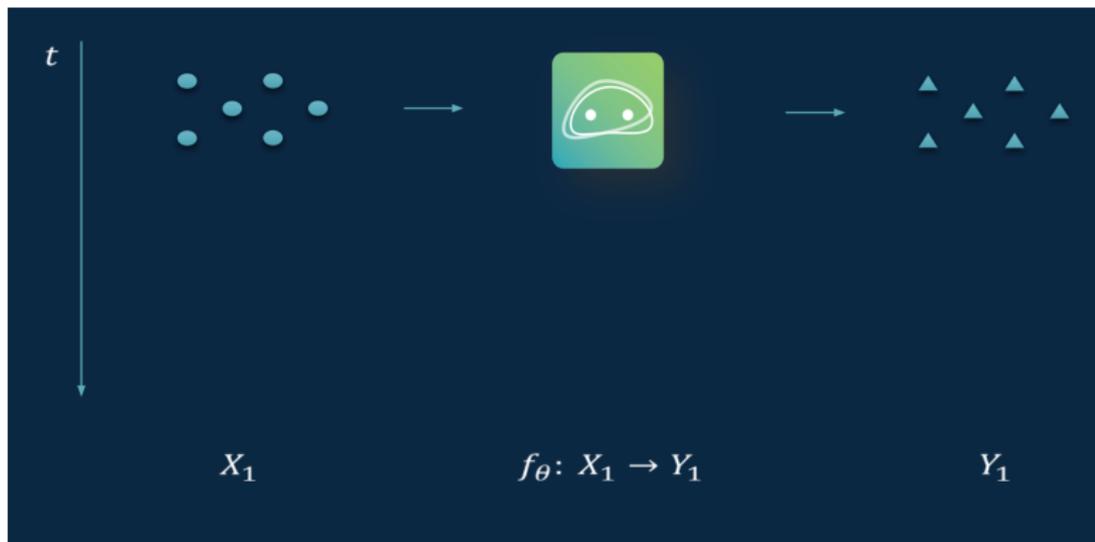
$$m_{b_i} = 1 - \frac{\beta - i}{\beta} e^{-\left(\frac{a_{b_i}}{\lambda_{b_i}}\right)^{K_{b_i}}}$$

- Actualiza el vector de activación:  $\hat{\mathbf{a}} = \mathbf{a} \cdot \mathbf{m}$
- Define el vector de activación de la clase desconocida (si  $m_i = 1$  no se hace nada):  $a_K = \sum_i (a_i - a_i \cdot m_i)$
- Calcula el *softmax* ajustado:

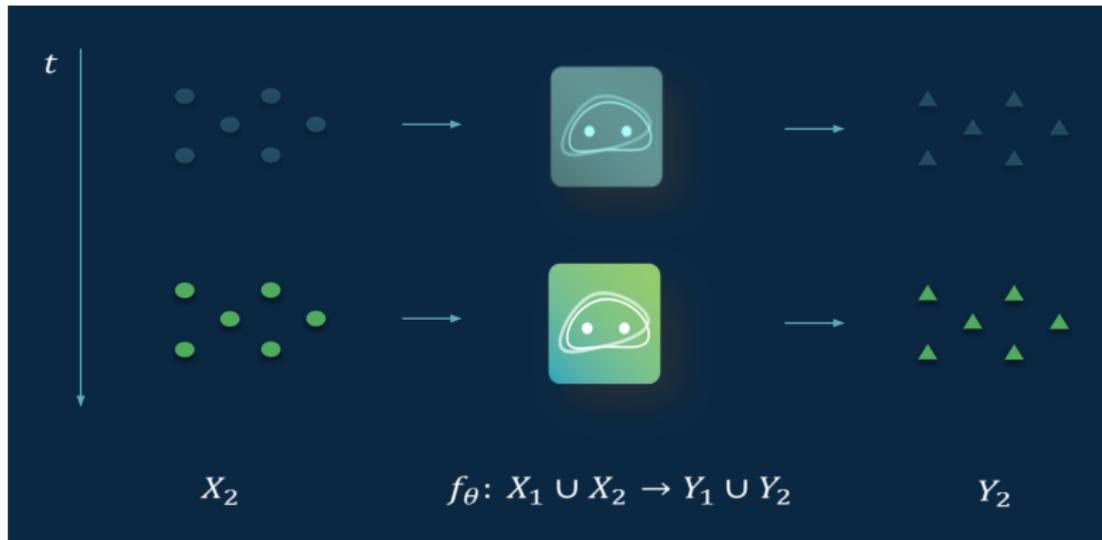
$$\hat{P}(y = j|x) = \frac{e^{a_j}}{\sum_{i=0}^K e^{a_i}}$$

Si  $\hat{y} = \arg \max \hat{P}(y = j|x) = K$ , el ejemplo es desconocido

# Continual Learning



# Continual Learning



Open-World  
Machine  
Learning

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

Aprendizaje  
Continuo

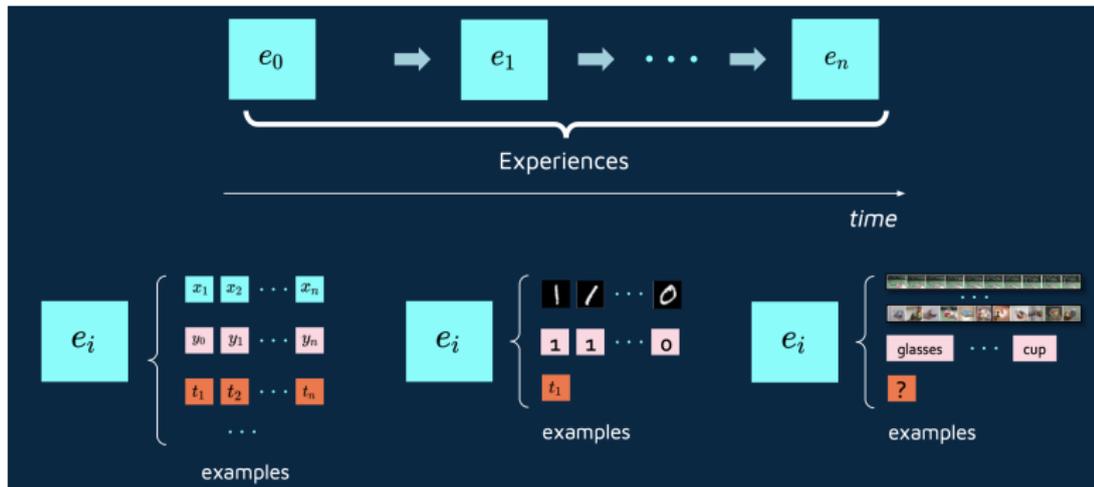
Métodos

Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro

# Continual Learning



Open-World  
Machine  
Learning

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

Aprendizaje  
Continuo

Métodos

Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro

# 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 Continuo (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

# ¿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 continuo

Se puede dar en las formas tradicionales de aprendizaje:

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

# Algunos puntos a considerar

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

- 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 *exemplars* 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:

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

# Problema de aprendizaje incremental

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

- Problema:

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

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

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

- 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  $D^t = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$  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  $\theta$  (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  $\sigma$  es una función *softmax*

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

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

- 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

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

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

- Considerando sólo las clases de la tarea

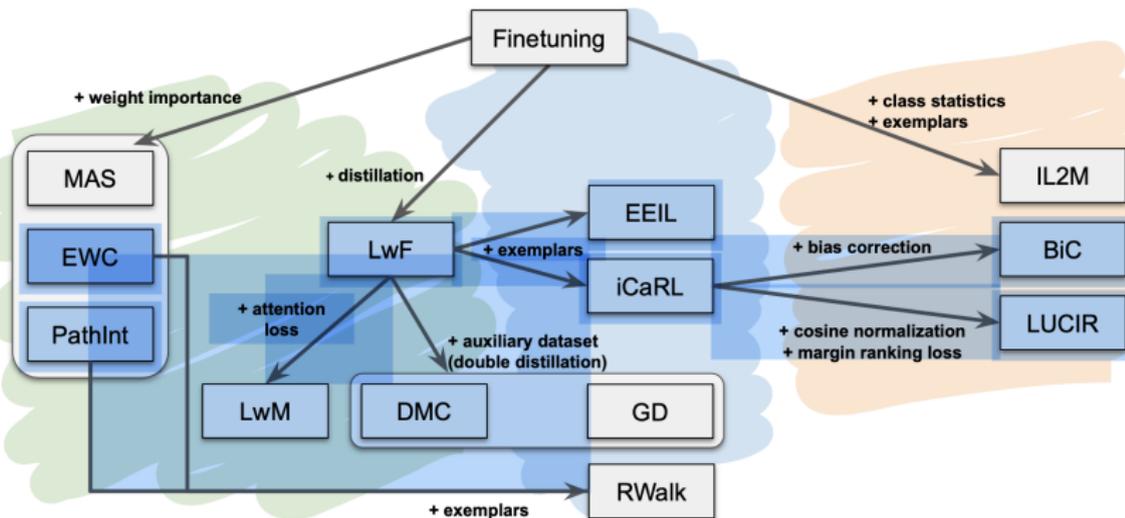
$$\mathcal{L}_{C^*}(\mathbf{x}, \mathbf{y}; \theta^t) = \sum_{k=1}^{|C^t|} y_{N^{t-1}+k} \log \frac{e^{\mathbf{o}_{N^{t-1}+k}}}{\sum_{i=1}^{|C^t|} e^{\mathbf{o}_{N^{t-1}+i}}}$$

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

# Retos

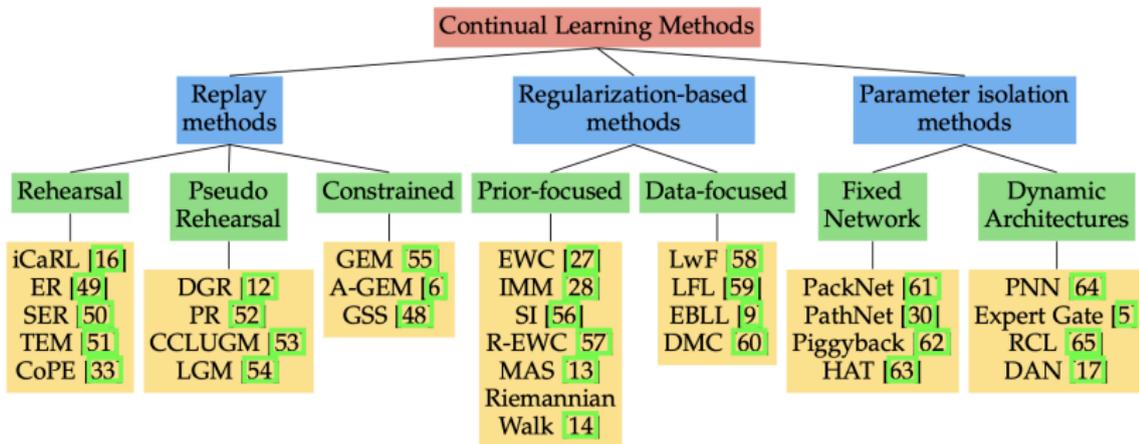
- Cómo balancear el retener conocimiento de tareas previas con la adquisición de nuevo conocimiento
- *Catastrophic forgetting*, algunas razones:
  - 1 *Weight drift*: Al aprender nuevas tareas, los pesos de las tareas anteriores se modifican para minimizar el error en las nuevas tareas
  - 2 *Activation 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<sup>1</sup>



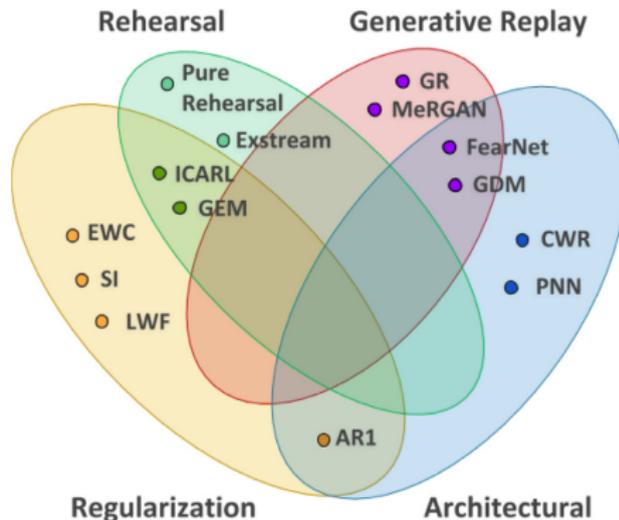
<sup>1</sup>M. Masana, X. Liu, B. Twardowski, M. Menta, A.D. Bagdanov, J. van de Weijer (2021). Class-incremental learning: survey and performance evaluation on image classification.

# Métodos<sup>2</sup>



<sup>2</sup>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<sup>3</sup>



<sup>3</sup>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/pseudo-rehearsal/constrained
- 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 generalización

# iCaRL

- *Incremental Classifier and Representation Learning* (iCaRL) fue uno de los primeros
- Selecciona *exemplars* de cada clase, el total está restringido a un tamaño fijo  $K$  (se reduce el número de *exemplars* por clase al aumentar el número de clases)
- La selección de *exemplars* se hace añadiendo el que cause que el promedio actual de los *exemplars* seleccionados se acerque más al promedio global de todos los *exemplars*
- Clasifica usando el promedio más cercano de los atributos de los *exemplars*
- 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 *exemplars* (*distillation*)

## iCaRL

Open-World  
Machine  
LearningEduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

---

**Algorithm 1** iCaRL CLASSIFY

---

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

---



---

**Algorithm 3** iCaRL UPDATEREPRESENTATION

---

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

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

// store network outputs with pre-update parameters:

**for**  $y = 1, \dots, s - 1$  **do**

$$q_i^y \leftarrow g_y(x_i) \quad \text{for all } (x_i, \cdot) \in \mathcal{D}$$

**end for**

run network training (e.g. BackProp) with loss function

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

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

---

# GEM (*Gradient Episodic Memory*)

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

- Guarda un conjunto de ejemplos de cada clase (*episodic memory*) de tamaño fijo
- Restringe el gradiente de nuevos ejemplos de tal forma que no aumente la pérdida en los datos de la memoria de las clases anteriores
- 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( $f_\theta$ , Continuum<sub>train</sub>, Continuum<sub>test</sub>) $\mathcal{M}_t \leftarrow \{\}$  for all  $t = 1, \dots, T$ . $R \leftarrow 0 \in \mathbb{R}^{T \times T}$ .**for**  $t = 1, \dots, T$  **do**:  **for**  $(x, y)$  in Continuum<sub>train</sub>( $t$ ) **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  $k < t$      $\tilde{g} \leftarrow \text{PROJECT}(g, g_1, \dots, g_{t-1})$ , see (11).     $\theta \leftarrow \theta - \alpha \tilde{g}$ .  **end for**   $R_{t,:} \leftarrow \text{EVALUATE}(f_\theta, \text{Continuum}_{\text{test}})$ **end for****return**  $f_\theta, R$ **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  $\alpha$ , Subset size  $C$ ; Budget  $\mathcal{B}$

```

1 Initialize: Memory  $\mathcal{M}$ ;  $\theta$ 
2 for  $t \in 1..T$  do
3   for  $B_n \sim D_t$  do
4     %Virtual Update
5      $\theta^v \leftarrow \text{SGD}(B_n, \alpha)$ 
6     %Select C samples
7      $B_C \sim \mathcal{M}$ 
8     %Select based on score
9      $S \leftarrow \text{sort}(s_{MI}(B_C))$ 
10     $B_{\mathcal{M}C} \leftarrow \{S_i\}_{i=1}^{\mathcal{B}}$ 
11     $\theta \leftarrow \text{SGD}(B_n \cup B_{\mathcal{M}C}, \alpha)$ 
12    %Add samples to memory
13     $\mathcal{M} \leftarrow \text{UpdateMemory}(B_n)$ ;
14  end
15 end

```

---

**Algorithm 2: Generative-MIR (GEN-MIR)**


---

**Input:** Learning rate  $\alpha$

```

1 Initialize: Memory  $\mathcal{M}$ ;  $\theta, \phi, \gamma$ 
2 for  $t \in 1..T$  do
3    $\theta', \phi', \gamma' \leftarrow \theta, \phi, \gamma$ 
4   for  $B_n \sim D_t$  do
5     %Virtual Update
6      $\theta^v \leftarrow \text{SGD}(B_n, \alpha)$ 
7      $B_C \leftarrow$  Retrieve samples as per Eq (2)
8      $B_G \leftarrow$  Retrieve samples as per Eq (3)
9     %Update Classifier
10     $\theta \leftarrow \text{SGD}(B_n \cup B_C, \alpha)$ 
11    %Update Generative Model
12     $\phi, \gamma \leftarrow \text{SGD}(B_n \cup B_G, \alpha)$ 
13  end
14 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

Open-World  
Machine  
Learning

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

Aprendizaje  
Continuo

Métodos

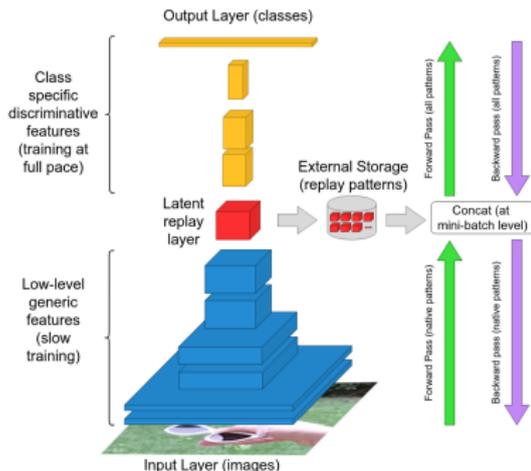
Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro

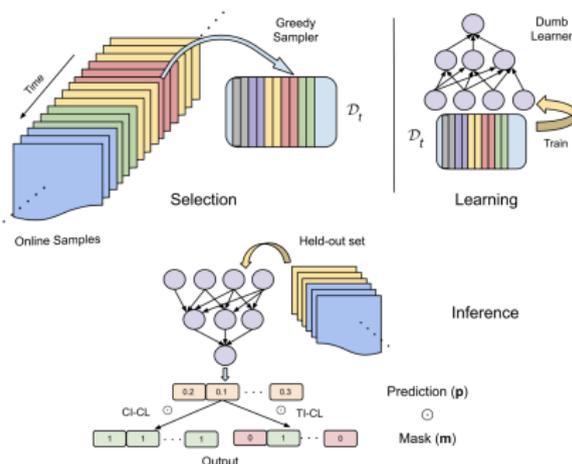
**Algorithm 1** Pseudocode describing how the external memory  $RM$  is populated across the training batches. Note that the amount  $h$  of patterns to add progressively decreases to maintain a nearly balanced contribution from the different training batches, but no constraints are enforced to achieve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

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



# 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 k	MNIST	
	(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)	<b>88.6 ± 0.6</b>	<b>90.0 ± 0.4</b>
MLP-600		
GEN [43]	-	75.5 ± 1.3
GEN-MIR [11]	-	81.0 ± 0.9
ER [14]	-	82.1 ± 1.5
GEM [7]	-	86.3 ± 1.4
ER-MIR [11]	-	87.6 ± 0.7
GDumb (Ours)	-	<b>91.9 ± 0.5</b>

(A1)

Method	MNIST	SVHN	Method	CIFAR100	
				Acc. (Avg)	Acc. (last)
No memory					
MAS [10]	19.3 ± 0.3	17.3	Finetune	17.8 ± 0.72	5.9 ± 0.15
SI [1]	19.7 ± 0.1	17.3	SI [1]	23.6 ± 1.90	13.3 ± 1.14
EWC [14]	19.8 ± 0.1	18.2	MAS [10]	24.7 ± 1.76	10.4 ± 0.80
Online EWC [20]	19.8 ± 0.04	18.5	MAS [10]	25.4 ± 1.99	9.5 ± 0.83
LwF [3]	24.2 ± 0.3	-	RWALK [3]	25.6 ± 1.92	11.1 ± 2.14
k=1000					
DGR [28]	91.2 ± 0.3	-	LwF [1]	32.3 ± 1.92	14.1 ± 0.87
DGR+Distill	91.8 ± 0.3	-	DMC [14]	45.0 ± 1.96	23.8 ± 1.90
GEM [7]	92.2 ± 0.1	75.6	k=2000		
RIF [24]	92.6 ± 0.2	-	GDumb (Ours)	45.2 ± 1.70	24.1 ± 0.97
RFS-Net [23]	98.2	88.9	DMC+ [14]	56.8 ± 0.86	-
OvA-INN [40]	96.4	-	ICARL [1]	38.8 ± 1.90	42.9 ± 0.79
TAML [24]	97.9	94.0	HEEL [14]	63.4 ± 1.6	-
GDumb (Ours)	<b>97.8 ± 0.2</b>	<b>93.4 ± 0.4</b>	BIC [10]	63.8	46.9

(B1)

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)	<b>88.93</b>	<b>45.8</b>

(E)

Method k	MNIST (500)	CIFAR-10 (500)	Method	MNIST		CIFAR10	
				Memory	Accuracy	Memory	Accuracy
Fine tuning							
AGEM [30]	18.8 ± 0.6	18.5 ± 0.2	Finetune	0	18.8 ± 0.5	0	15.0 ± 3.1
BGD [48]	29.0 ± 5.3	18.5 ± 0.6	GEN [28]	4.08	79.3 ± 0.6	4.57	15.3 ± 0.5
GEM [7]	13.5 ± 5.1	18.2 ± 0.5	GEN-MIR [11]	4.31	82.1 ± 0.3	38.0	19.3 ± 1.2
GEM [7]	87.2 ± 1.3	20.1 ± 1.4	LwF [3]	1.91	33.3 ± 2.5	4.38	28.2 ± 0.3
GSS-Greedy [37]	84.2 ± 2.6	28.0 ± 1.3	ADI [47]	1.91	55.4 ± 2.6	4.38	24.8 ± 0.9
HAL [30]	77.9 ± 4.2	32.1 ± 1.5	ARM [41]	1.91	56.2 ± 3.5	4.38	26.4 ± 1.2
ER [14]	81.0 ± 2.3	33.3 ± 1.5	ER [14]	0.39	83.2 ± 1.9	3.07	41.3 ± 1.9
MIR [11]	84.9 ± 1.7	34.5 ± 2.0	ER-MIR [11]	0.39	85.6 ± 2.0	3.07	47.6 ± 1.1
GMED (ER) [12]	82.7 ± 2.1	35.0 ± 1.5	ICARL [1]	-	-	3.07	32.4 ± 2.1
GMED (MIR) [12]	87.9 ± 1.1	35.5 ± 1.9	GEM [7]	0.39	86.3 ± 0.1	3.07	17.5 ± 1.6
GDumb (Ours)	<b>91.9 ± 0.5</b>	<b>45.8 ± 0.9</b>	GDumb (ours)	<b>0.39</b>	<b>91.9 ± 0.5</b>	<b>3.07</b>	<b>61.3 ± 1.7</b>

(A2)

Method k	MNIST (500)	CIFAR-10 (500)	Method	MNIST		CIFAR10	
				Memory	Accuracy	Memory	Accuracy
No stored samples							
mas-dmm [13]	3.5M	none	32.42	Method	CIFAR100		
SS [1]	3.5M/9.0M	L2/dropout	42.41	(A)	(100)		
HAT [1]	3.5M/9.0M	L2	44.19	RWALK [1]	40.9 ± 3.97		
EWC [1]	0.1K	none	43.74	EWC [1]	42.4 ± 3.02		
LwF [1]	9.0M	L2	48.11	Base	42.9 ± 2.07		
EBL [14]	9.0M	L2	48.17	MAS [10]	44.2 ± 2.30		
MAS [10]	3.5M/9.0M	none	48.96	SI [1]	47.1 ± 4.41		
PadNet [13]	0.1K/3.5M	L2/dropout	50.96	ICARL [1]	50.1		
k=1000							
GEM [7]	61K/3.5M	none/dropout	44.23	S-GEM [14]	54.9		
GDumb	80K	cutmix	45.20	PSN [15]	58.2 ± 0.85		
ICARL [1]	61K/3.5M	dropout	48.15	GEM [7]	61.2 ± 0.78		
k=3000							
GEM [7]	61K/3.5M	none/dropout	45.27	A-GEM [14]	63.1 ± 1.24		
ICARL [1]	61K/3.5M	dropout	49.24	TrpER [14]	68.3 ± 0.65		
GDumb	60K	cutmix	97.27	GDumb	68.3 ± 0.65		

(A3)

Method	MNIST (k)	CIFAR100	ImageNet100
47.1 ± 1.5	46.49		
48.6 ± 0.4	55.44		
49.3 ± 0.3	57.25		
58.0 ± 0.5	62.08		
58.4 ± 0.8	<b>62.86</b>		
61.4 ± 0.7	-		

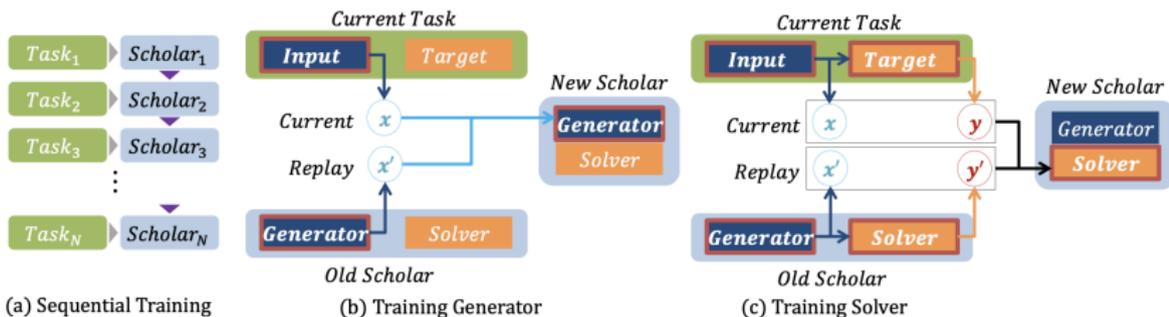
(B3)

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



# 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  $\Omega$  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  $\theta$  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  $\theta_A^*$  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:

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

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

## EWC

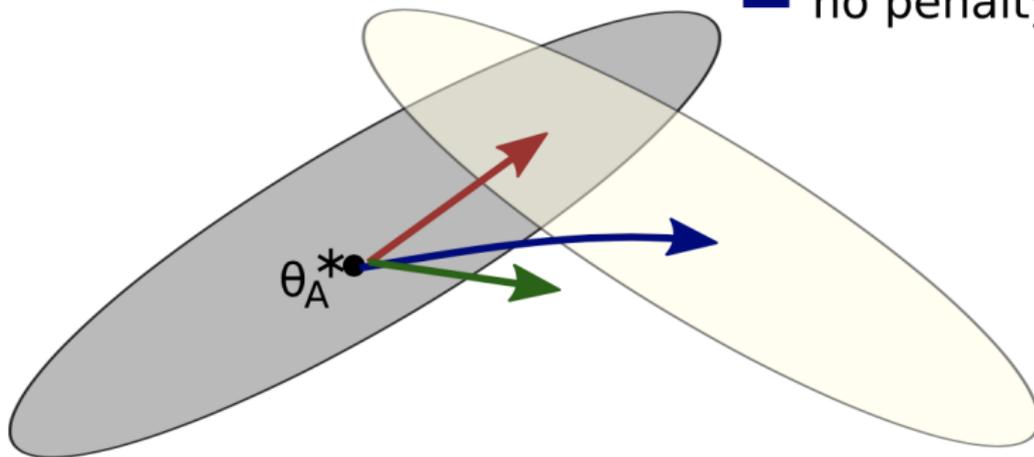
Low error for task B

Low error for task A

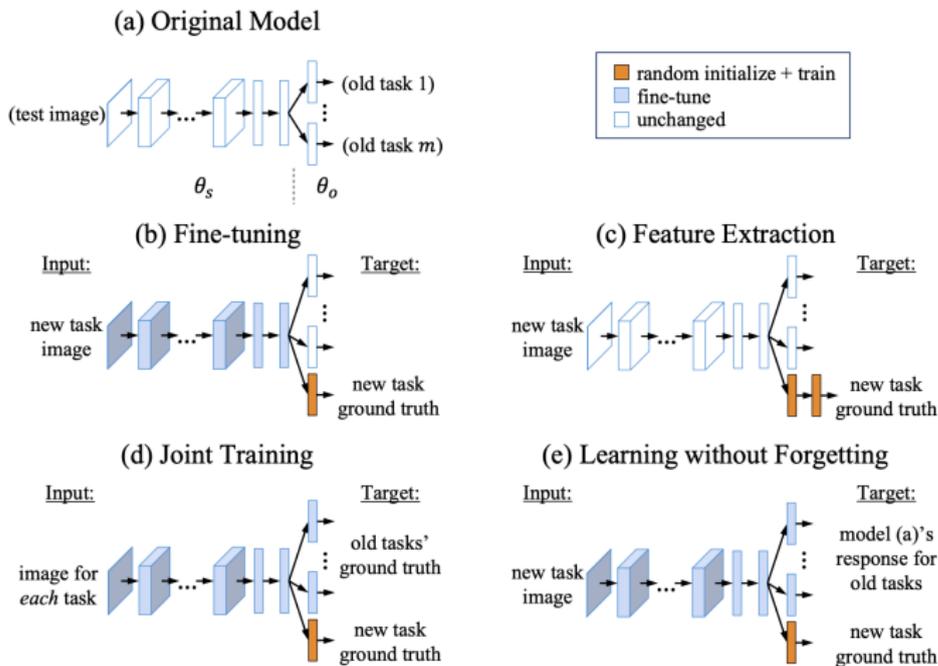
EWC

L<sub>2</sub>

no penalty



# Learning without Forgetting (LwF)



# Learning without Forgetting (LwF)

- Usa sólo ejemplos de las clases nuevas
- Dada una red con parámetros compartidos ( $\theta_s$ ) y parámetros específicos para una(s) tarea(s) ( $\theta_o$ ), el objetivo es aprender parámetros para una nueva tarea ( $\theta_n$ ) 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  $\theta_s$  y  $\theta_o$  (*warm-up*) y se entrena  $\theta_n$  hasta converger y después de entrena toda la red

# Learning without Forgetting (LwF)

Eduardo

Morales, Hugo

Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

## LEARNING WITHOUT FORGETTING:

### Start with:

$\theta_s$ : shared parameters

$\theta_o$ : task specific parameters for each old task

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

### Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$  // 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

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

Aprendizaje  
Continuo

Métodos

Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro

- 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  $w_k^{\mu}$  (el signo menos es porque estamos interesado en decrementar la función de pérdida)

# Synaptic Intelligence

- La importancia de un parámetro  $\theta_k$  para una tarea está determinado por: (i) cuánto contribuye en la caída de la función de pérdida durante el entrenamiento ( $w_k^\nu$ ) 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:

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

- 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

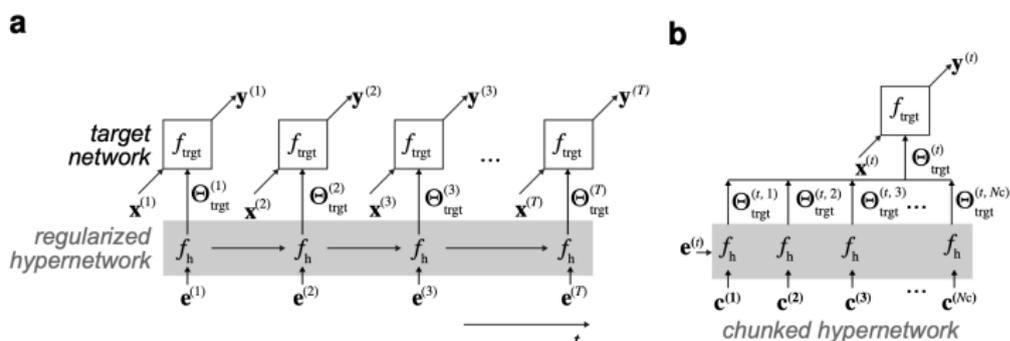


Figure 1: **Task-conditioned hypernetworks for continual learning.** (a) Commonly, the parameters of a neural network are directly adjusted from data to solve a task. Here, a weight generator termed *hypernetwork* is learned instead. Hypernetworks map embedding vectors to weights, which parameterize 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 ( $\theta_{trgt}$ ) para una función particular ( $f_{trgt}$ ), se aprenden los parámetros ( $\theta_h$ ) de un meta-modelo, cuya salida es  $\theta_{trgt}$
- 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

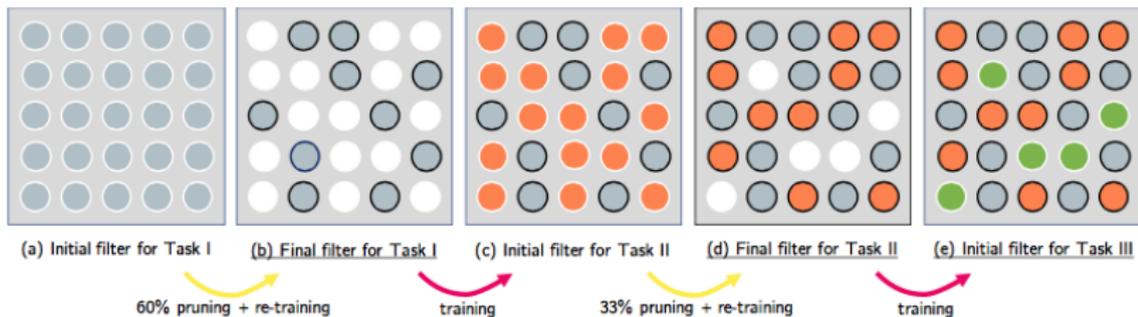


Figure 1: Illustration of the evolution of a  $5 \times 5$  filter with steps of training. Initial training of the network for Task I learns a dense filter as illustrated in (a). After pruning by 60% (15/25) and re-training, we obtain a sparse filter for Task I, as depicted in (b), where white circles 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

Open-World  
Machine  
Learning

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

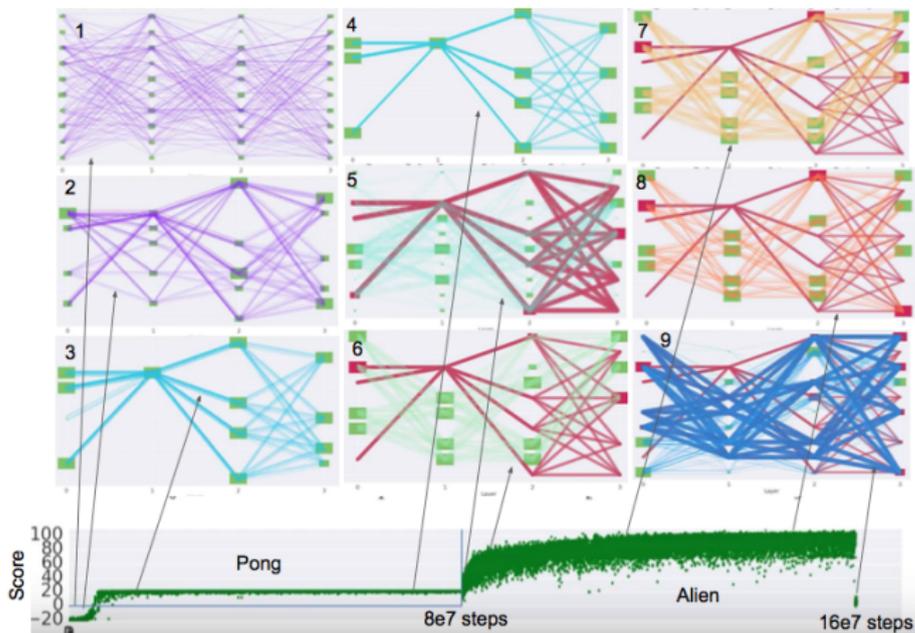
Aprendizaje  
Continuo

Métodos

Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro



# 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  $\sigma$  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

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

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

## HAT

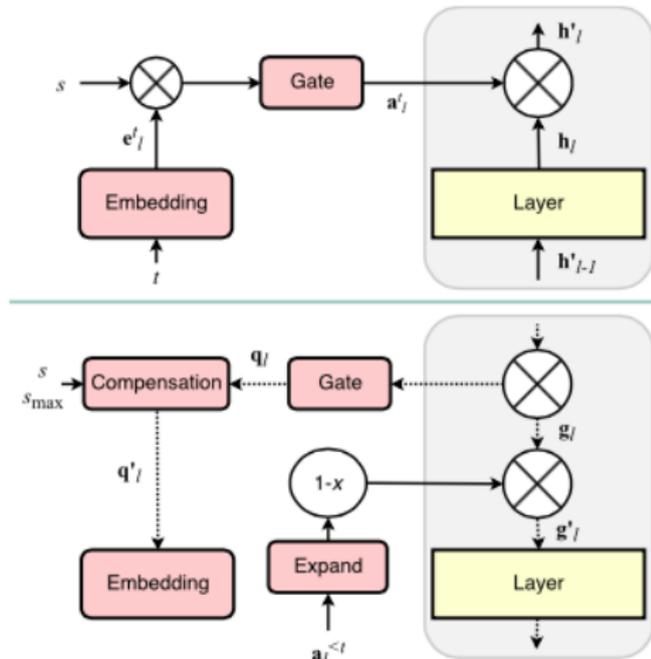


Figure 1. Schematic diagram of the proposed approach: forward (top) and backward (bottom) passes.

# Arquitecturas Maestro-Estudiante

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

- 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

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

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

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

- BWT

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

- 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 ( $\theta$ ) usado en comparación con la memoria del primer modelo:

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

- Eficiencia en ejemplos guardados:

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

- También se pueden hacer comparaciones en términos de eficiencia

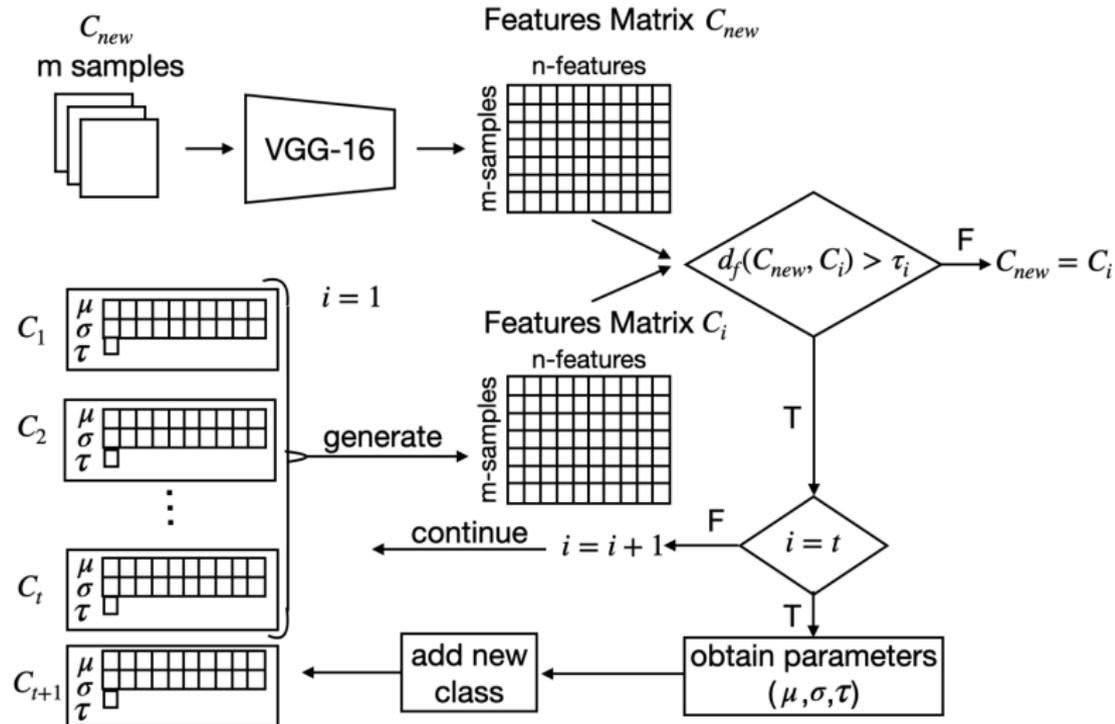
# Open-World Learning

- El área se acerca cada vez más a un aprendizaje de mundo abierto, pero todavía no existen sistemas completos
- El sistema tiene que aprender y adaptarse continuamente al ambiente
- Tiene que clasificar correctamente clases conocidas y clases desconocidas
- Aprender (y etiquetar) las clases nuevas e incorporarlas al clasificador actual
- Mantener desempeño a lo largo del tiempo

# OWL 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
- OWL se puede aplicar en percepción, aprendizaje por refuerzo, aprendizaje basado en modelos

## DISCO



# Distancia Fréchet

Open-World  
Machine  
Learning

Eduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidos

Aprendizaje  
Continuo

Métodos

Evaluación

Open-World  
Learning

Trabajo Actual  
y Futuro

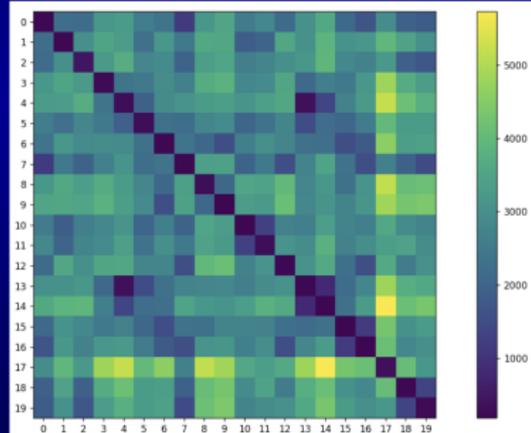
0 - CD\_player    1 - Crock\_Pot    2 - analog\_clock    3 - apron    4 - bassinet

5 - bathtub    6 - bookcase    7 - cellular\_telephone    8 - chiffonier    9 - chins\_cabinet

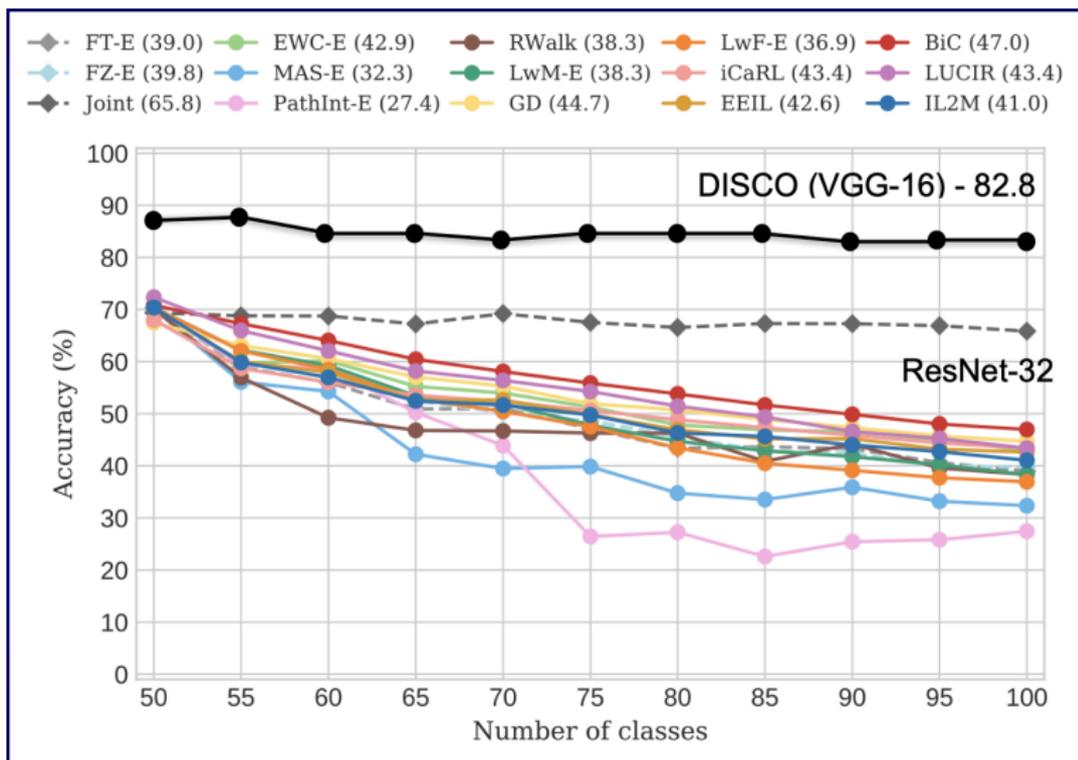
10 - coffee\_mug    11 - coffeepot    12 - computer\_keyboard    13 - cradle    14 - crib

15 - desk    16 - desktop\_computer    17 - dial\_telephone    18 - digital\_clock    19 - digital\_watch

- Coffee mug - Coffee pot
- Cradle - Crib - Bassinet
- Dial telephone - Digital clock
- Desk - Desktop computer



# Algunos Resultados



## DISCO 2

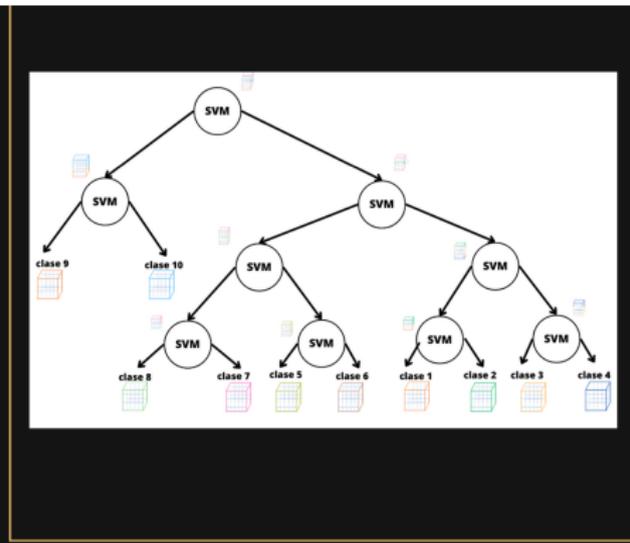
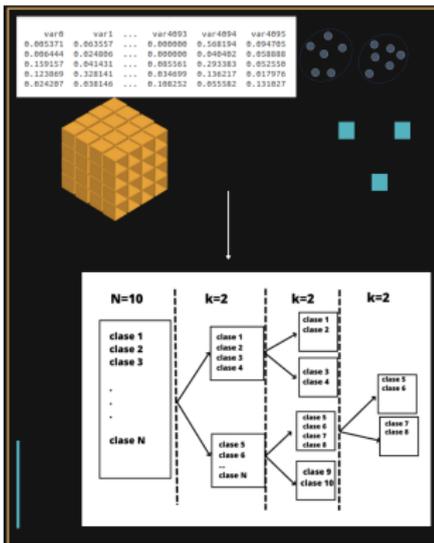
Open-World  
Machine  
LearningEduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

## DISCO 2

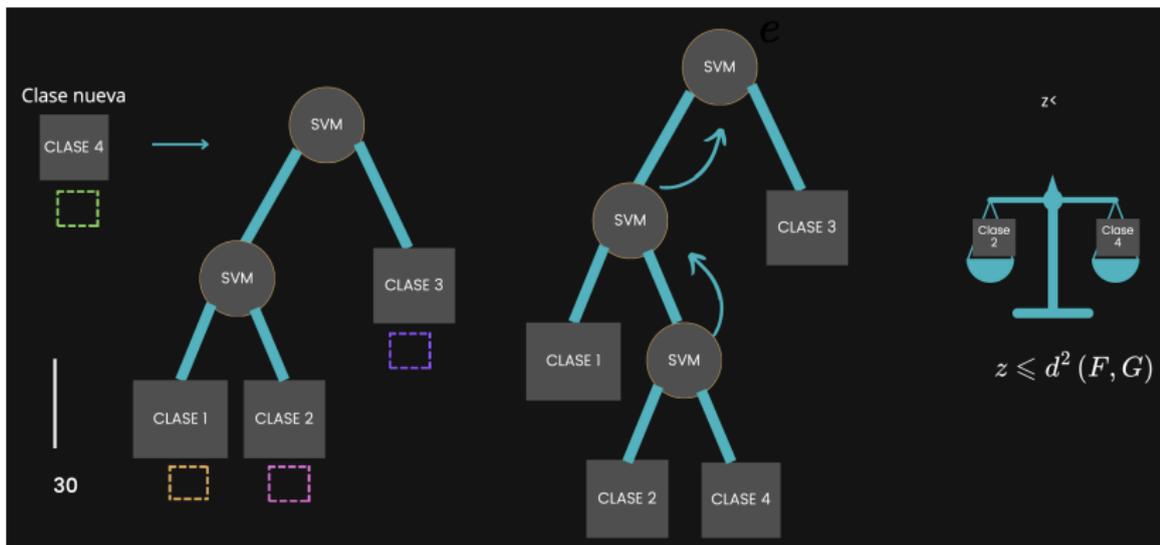
Open-World  
Machine  
LearningEduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

## DISCO 2

Open-World  
Machine  
LearningEduardo  
Morales, Hugo  
Jair Escalante

## Pruebas: Recorrido



Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

## DISCO 2

## Pruebas: Resultados del Recorrido

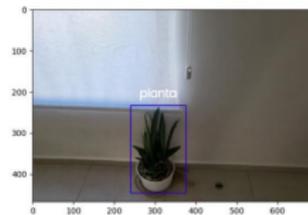
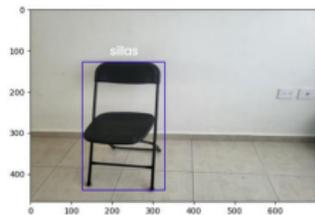
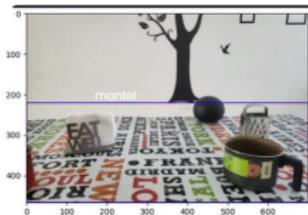
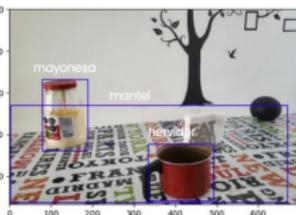
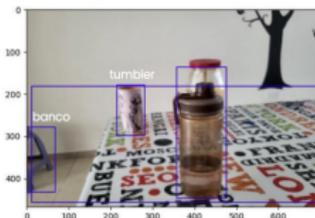
Open-World  
Machine  
LearningEduardo  
Morales, Hugo  
Jair Escalante

Introducción

Identificación  
de  
desconocidosAprendizaje  
Continuo

Métodos

Evaluación

Open-World  
LearningTrabajo Actual  
y Futuro

Clase
0 sillas
1 botella
2 hervidor
3 banco
4 tumbler
5 planta
6 gallon
7 mayonesa
8 mantel

# Open-World Unlearning

- En general los sistemas de aprendizaje, quieren aprender, sin embargo, también se podría pensar en desaprender información
- Algunos posibles ejemplos son:
  - Eliminar información sensible
  - Eliminar o al menos reducir sesgos
  - Realizar un olvido selectivo (e.g., información desactualizada)
- Área abierta de investigación

# Idealmente

- Memoria lineal o logaritmica o constante
- Sin fronteras entre tareas
- Aprendizaje en línea
- *Forward transfer (zero-shot learning)*
- *Backward transfer* (mejorar lo aprendido)
- Aplicable no solo en clasificación
- Adaptativo
- No requiere de un oráculo para probarlo
- Puede volver a visitar tareas anteriores
- Olvido gradual (*graceful forgetting*)

# Áreas actuales

- Optimizar los modelos y la memoria de la información almacenada
- Organizar el conocimiento
- Auto-aprendizaje y aprendizaje no supervisado incremental
- Escenarios de aprendizaje más realistas en donde las tareas pueden estar traslapadas
- Área abierta de investigación

## Algunas Referencias

- Marc Masana, Xialei Liu, Bartłomiej 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.

# Algunas Referencias

- A. Bendale, T.E. Boult (2015). Towards Open Set Deep Networks. arXiv: 1511.06233v1
- W. Liu, X. Wang, J.D. Owens, Y. Li (2021). Energy-based Out-of-distribution detection. arXiv:2019.03759v4.
- Z. Lyo, N.B. Gutierrez, W.J. Beksi (2022). MetaMax: Improved Open-Set Deep Neural Networks via Weibull Calibration. arXiv:2211.10872v1
- F. Zhu, S. Ma, Z. Cheng, X.Y. Z, Z. Zhang, C.L. Liu (2024). Open-world Machine Learning: A review and new outlooks. arXiv:2401:01759v2