Continual Prototype Evolution: Learning Online from Non-Stationary Data Streams

Matthias De Lange
KU Leuven
matthias.delange@kuleuven.be

Tinne Tuytelaars
KU Leuven
tinne.tuytelaars@kuleuven.be

https://arxiv.org/pdf/2009.00919.pdf

Continual learning and why you should care

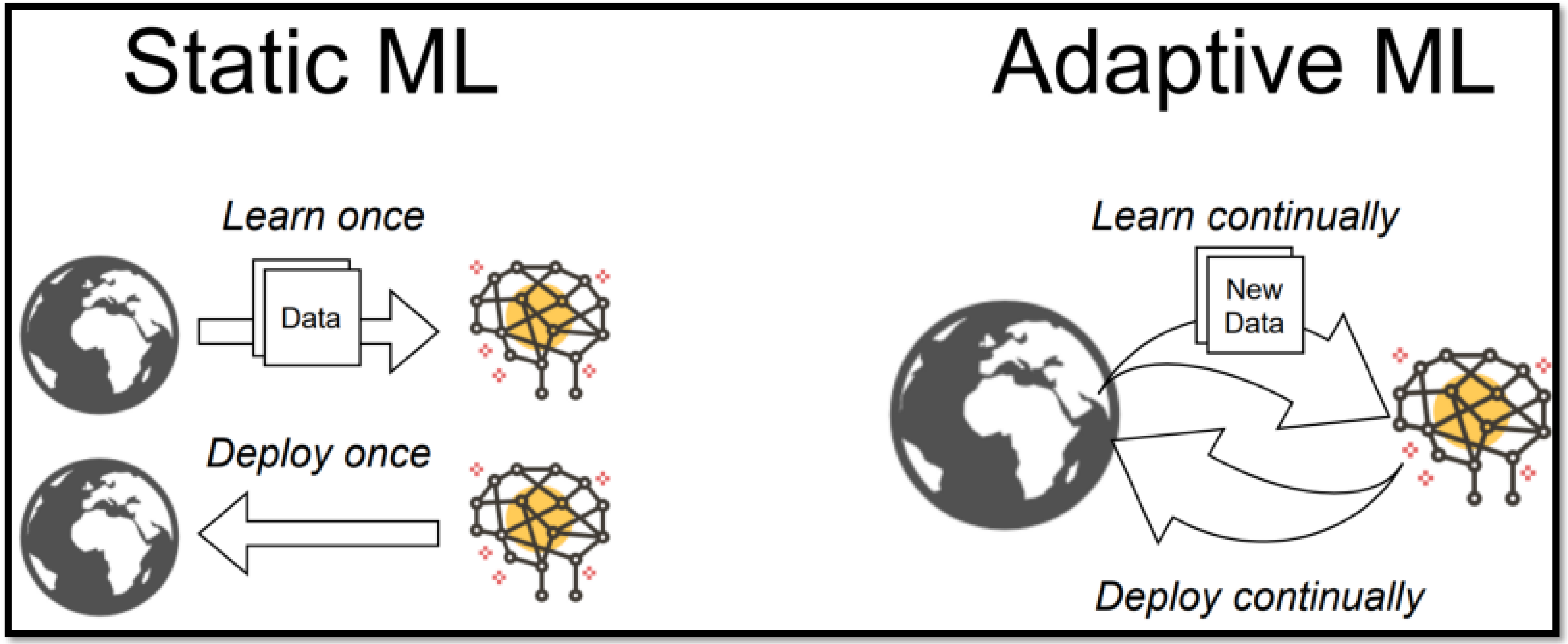
Continual learning and why you should care

KU LEUVEN

Continual learning and why you should care

Continual learning and why you should care

Continual learning and why you should care



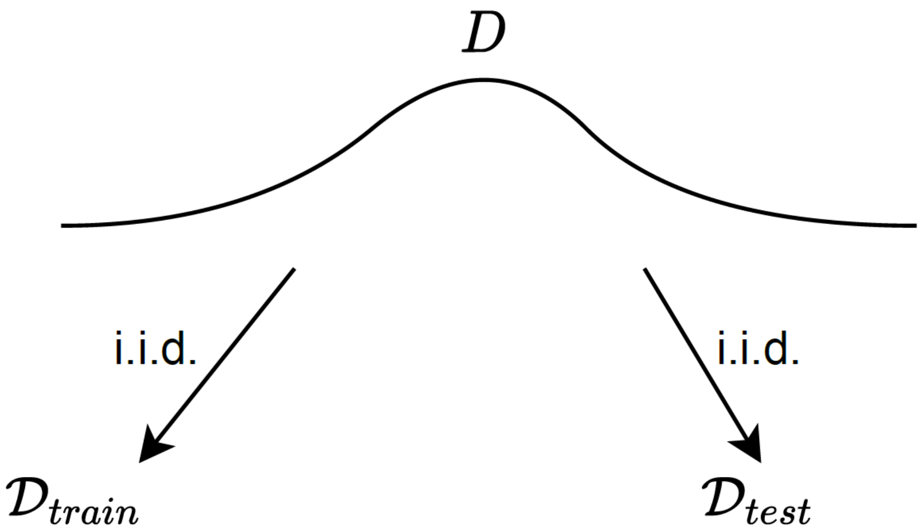
Roadmap

- What is Continual Learning?
- How to learn from data streams?
- Why representation learning?
- Continual Prototype Evolution (CoPE)
  - Evolving prototypes
  - PPP-loss
  - Balanced replay

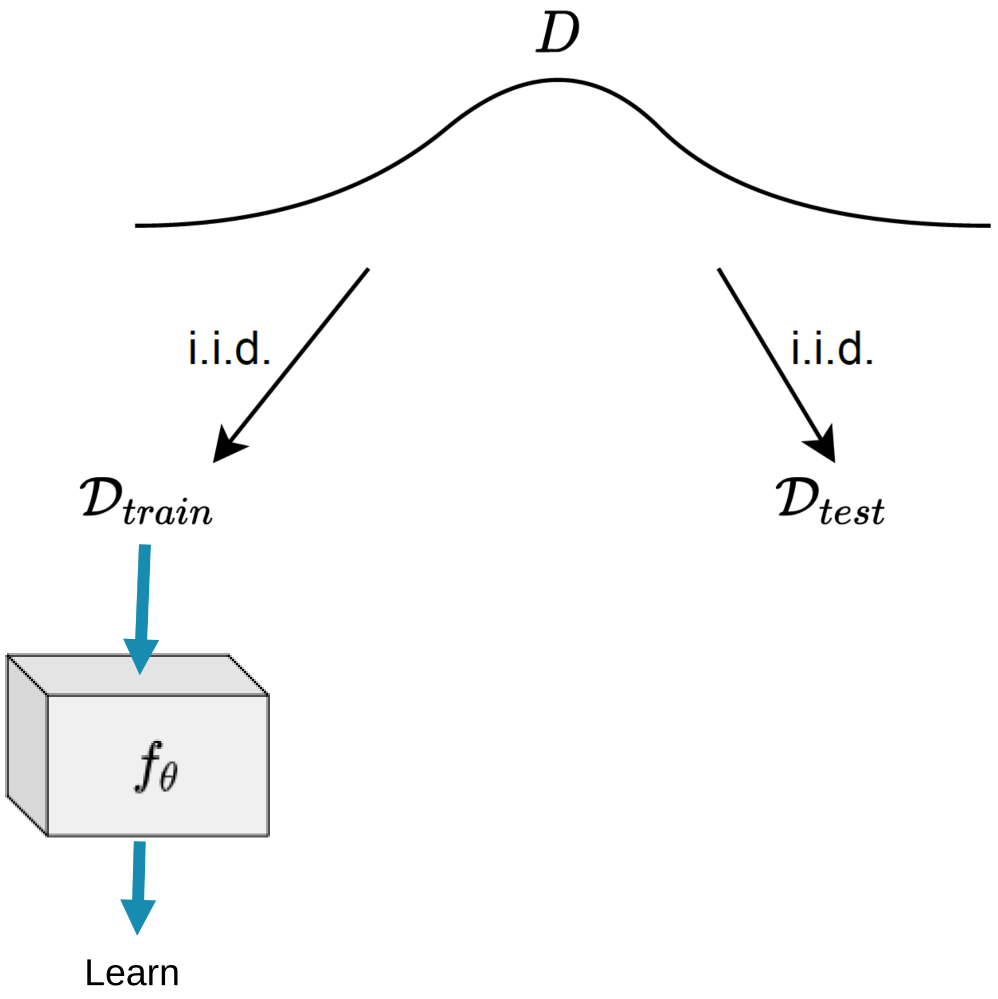
Follow along:

https://arxiv.org/pdf/2009.00919.pdf

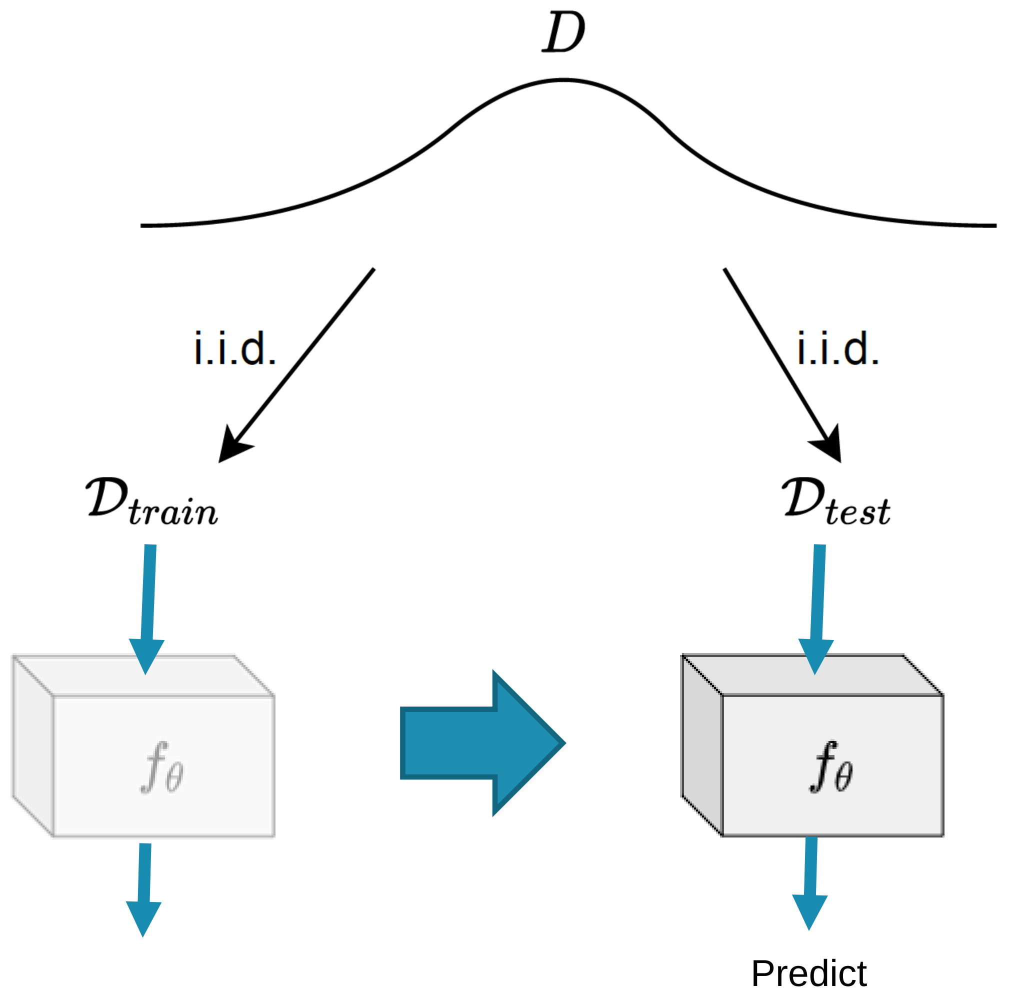
Standard Machine Learning



Standard Machine Learning

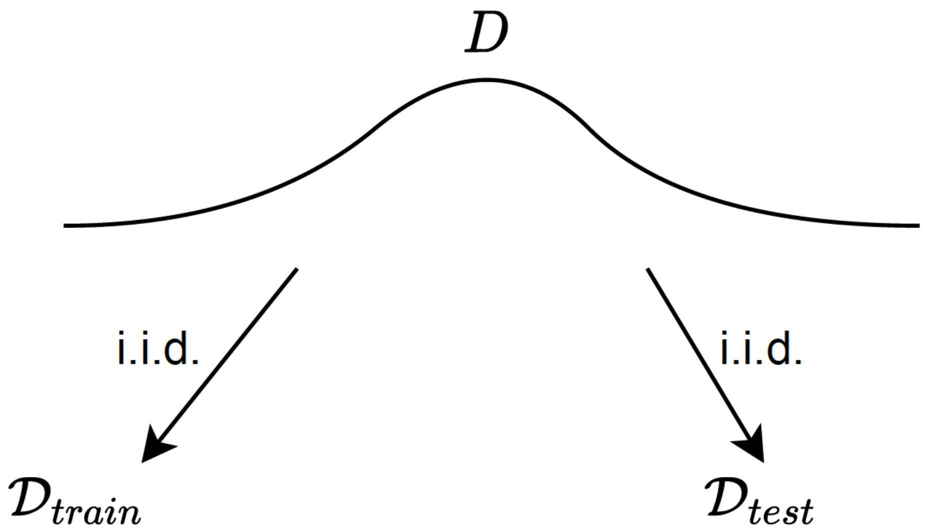


Standard Machine Learning



Standard Machine Learning

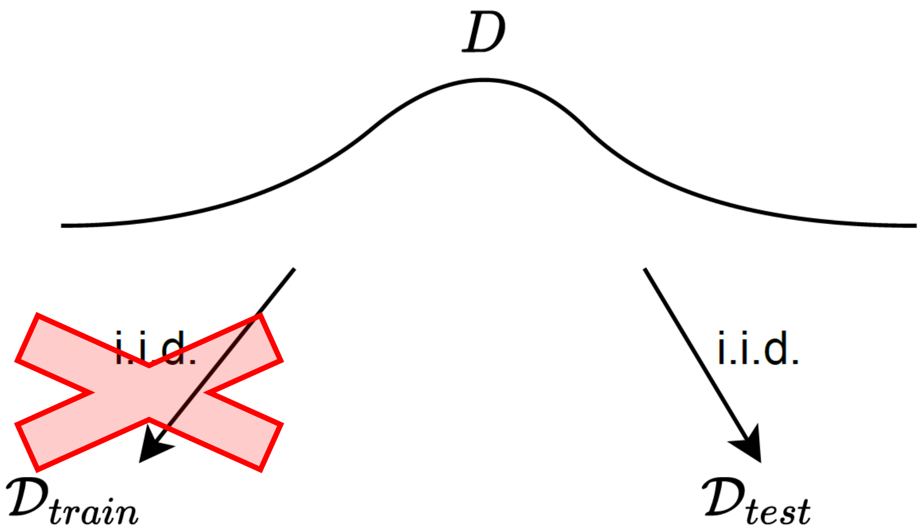
- Once trained, once deployed
- Deployed system is static over time
- Any changes? Retrain & Redeploy




EVER

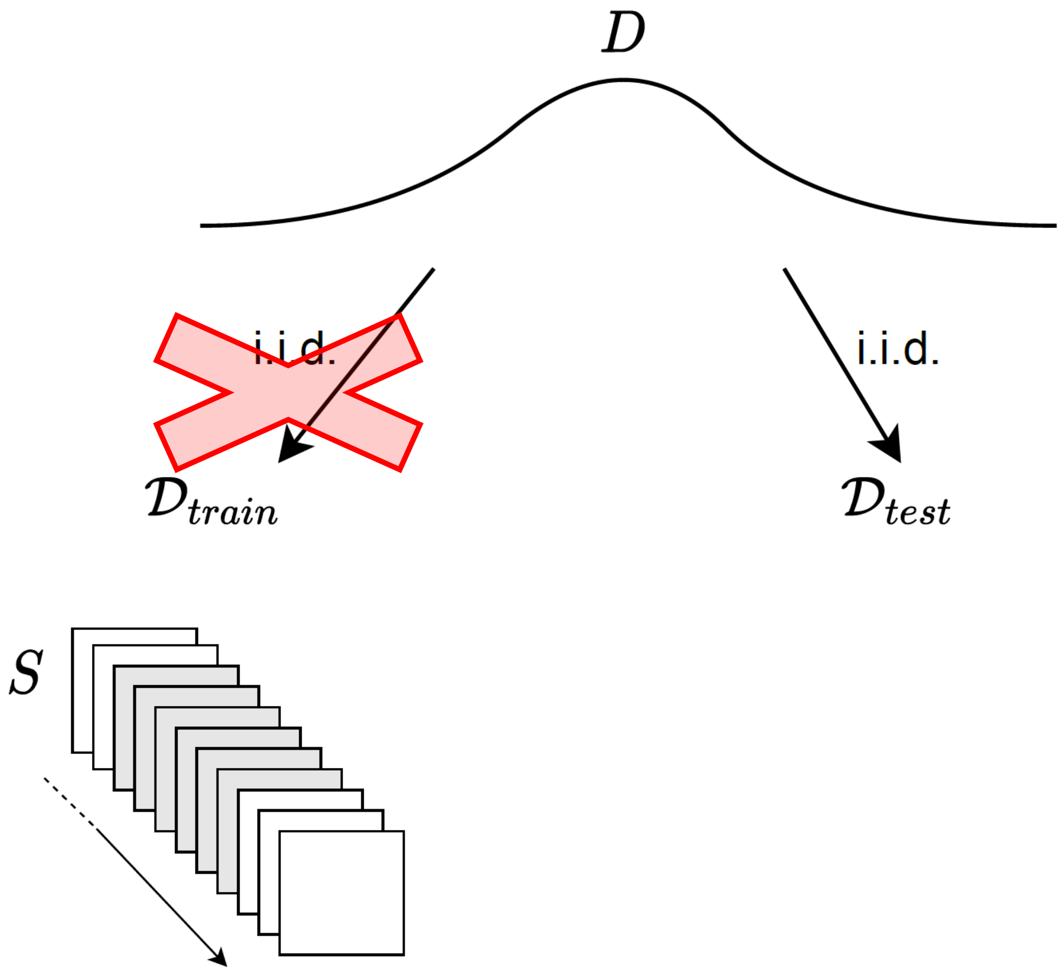

EVER
CHANGING

In Continual Learning



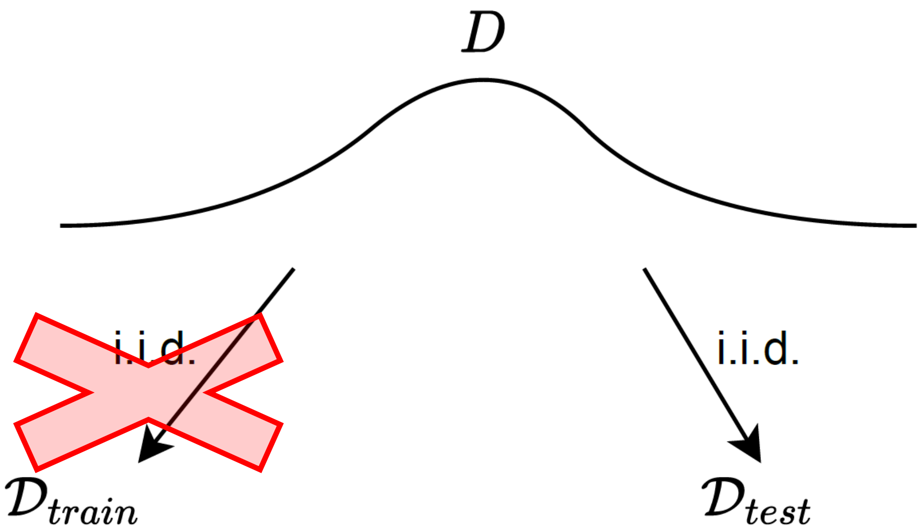
KU LEUVEN

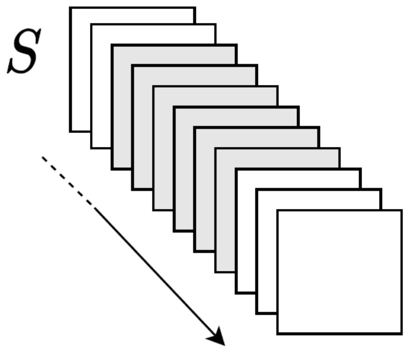
In Continual Learning



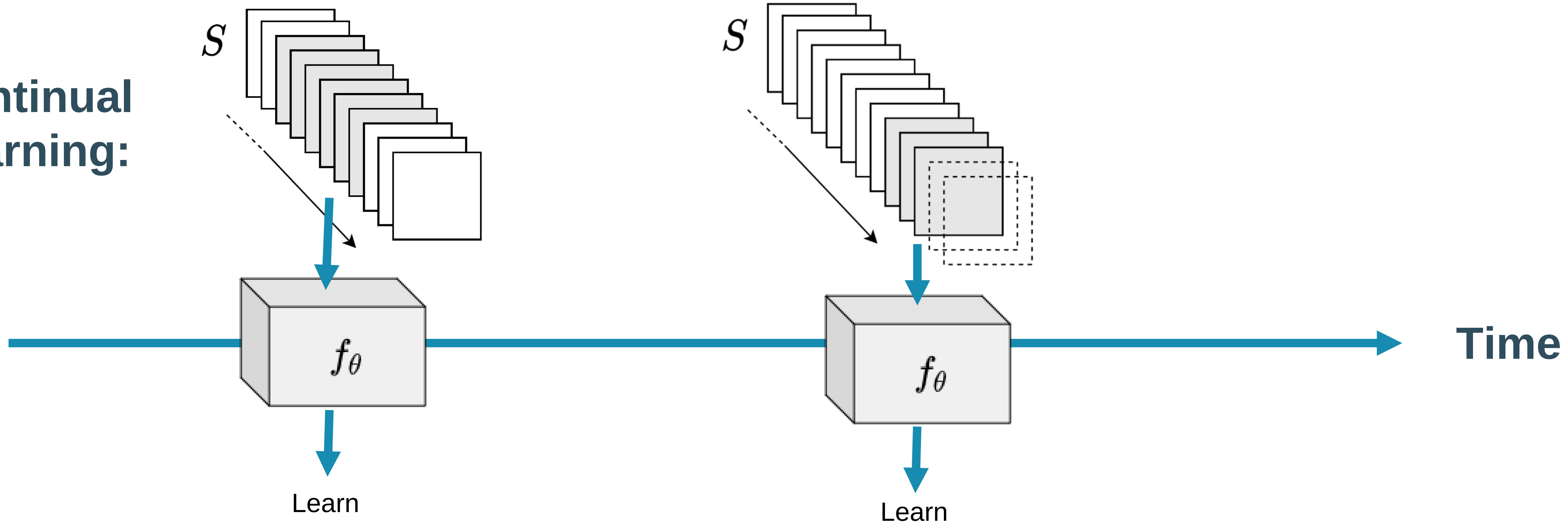
In Continual Learning

- Available training data changes over time
- Assumptions on D determined by $\mathcal{D}_{test}$
  - What should the system learn?

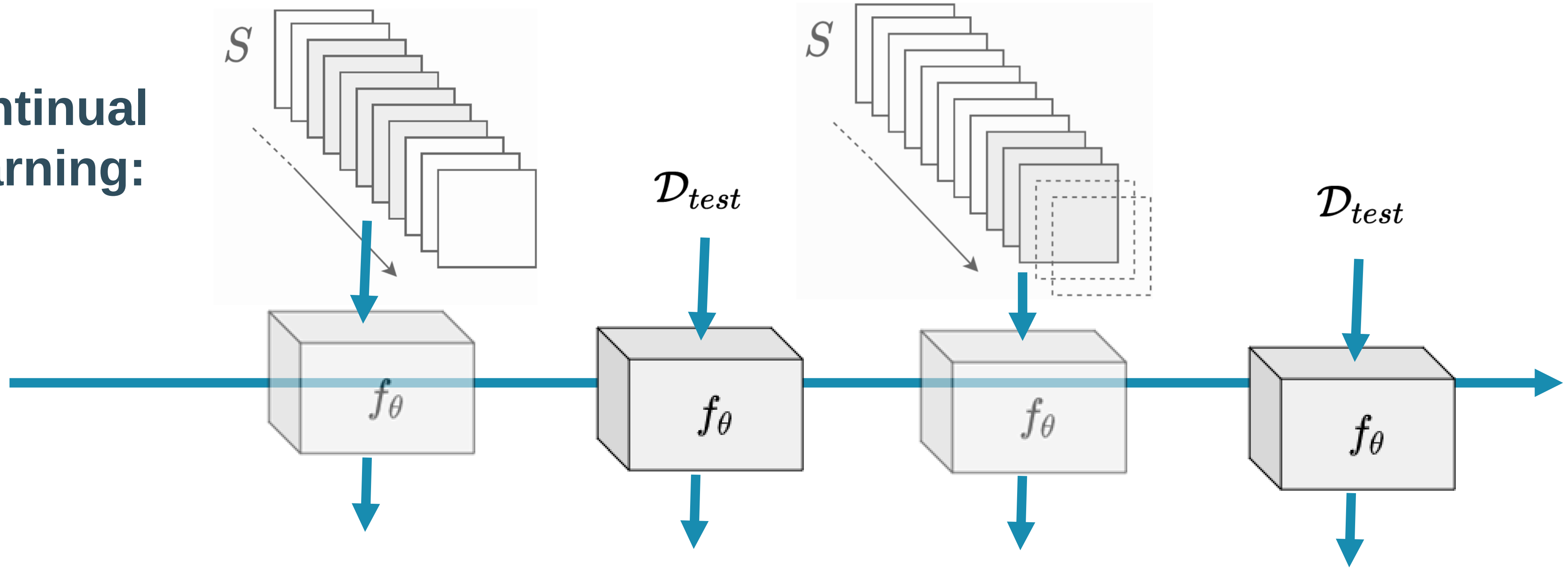




Continual Learning:



Continual Learning:



- We use our static test set to measure performance of our system
- Over different points in time

Used Metrics

- With $\mathcal{D}_{test}$ static, iid over seen/all classes

- Average Accuracy (Avg over tasks in $\mathcal{D}_{test}$)

$$A_T = \frac{1}{T}\sum_{i=1}^{T} a_{T,i}$$

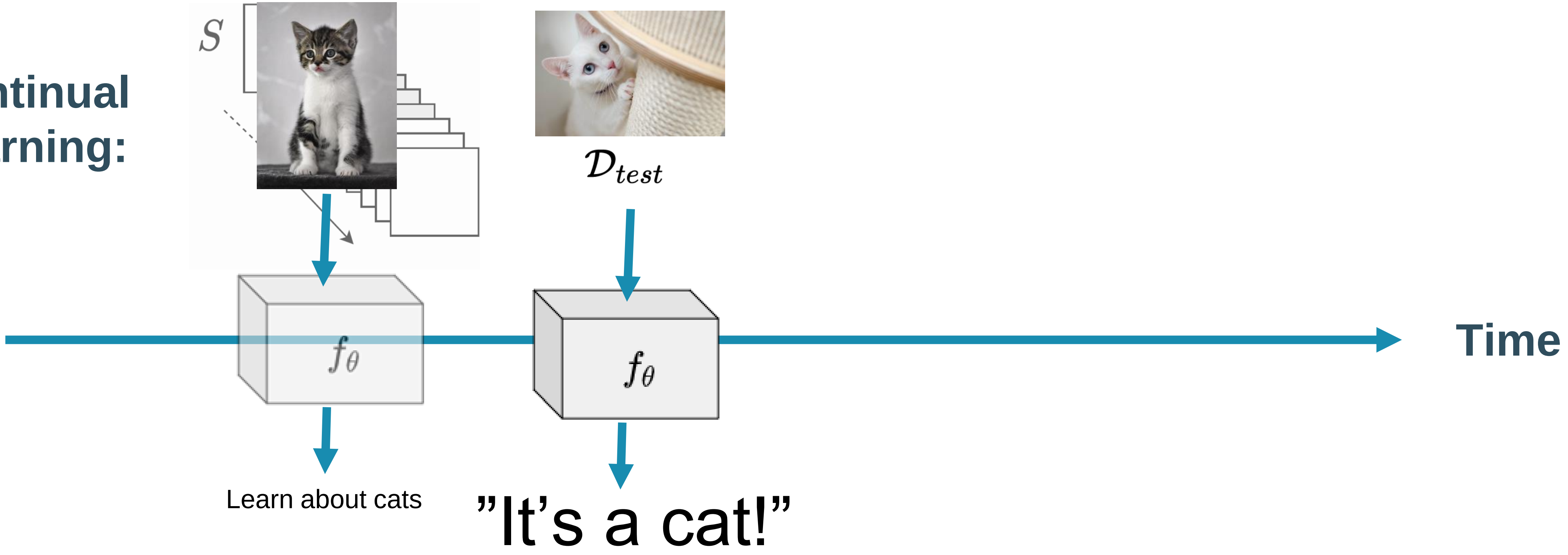
$a_{\text{learned until}X,\text{ eval on}X}$

- Average Forgetting (Avg over tasks in $\mathcal{D}_{test}$)

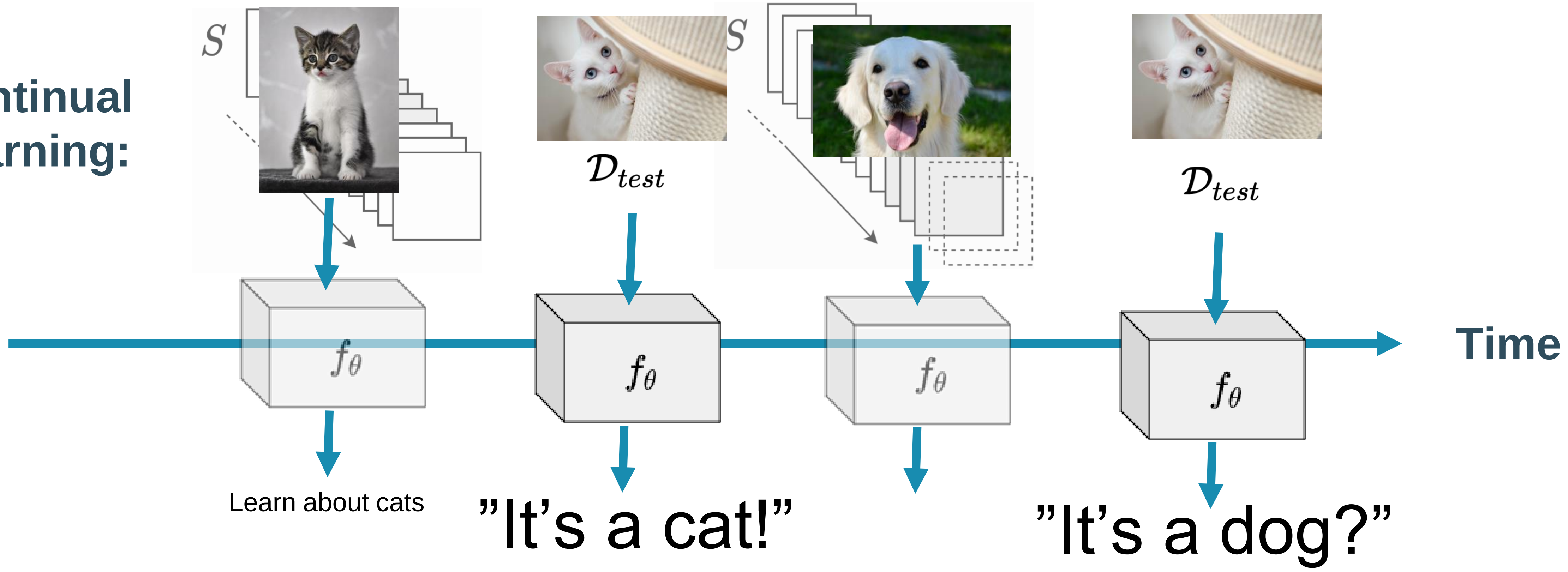
$$F_T = \frac{1}{T-1}\sum_{i=1}^{T-1} a_{i,i} - a_{T,i}$$

$a_{\text{learned until}X,\text{ eval on}X}$

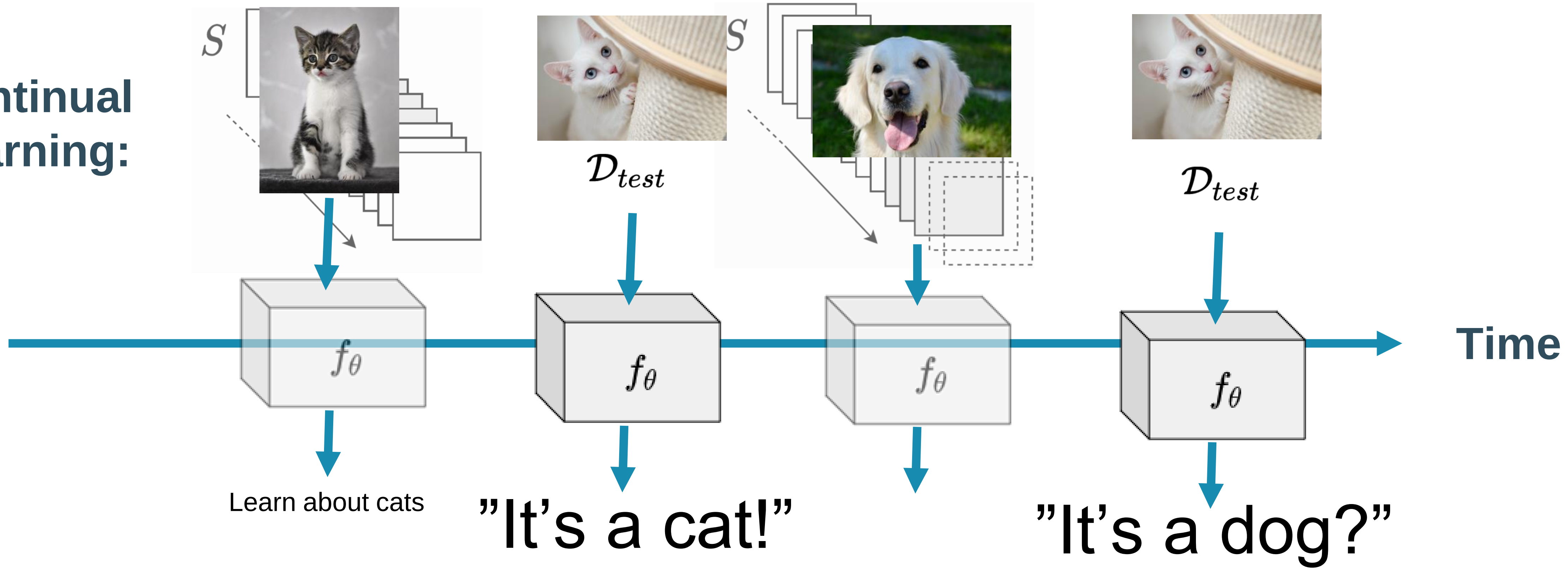
Continual Learning:



Continual Learning:



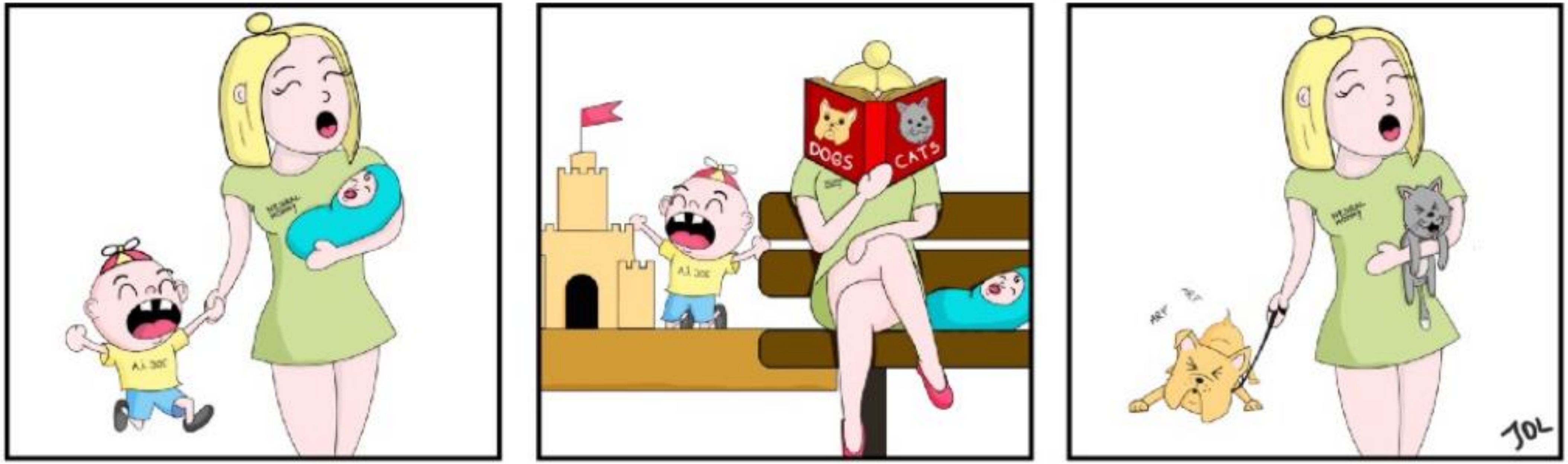
Continual Learning:



Continual Learning is hard in Neural Networks! → Very high Forgetting:

"Catastrophic Forgetting"

Why neural networks make terrible mothers...



An illustration of catastrophic forgetting in neural networks. Cartoon credits @Jasper De Lange.

Roadmap

- What is Continual Learning?
- **How to learn from data streams?**
- Why representation learning?
- Continual Prototype Evolution (CoPE)
    - Evolving prototypes
    - PPP-loss
    - Balanced replay

Follow along:

https://arxiv.org/pdf/2009.00919.pdf

How to learn from data streams?

- Current continual learning paradigms

[1] continual learning	Eval	Train	
task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

How to learn from data streams?

- Current continual learning paradigms

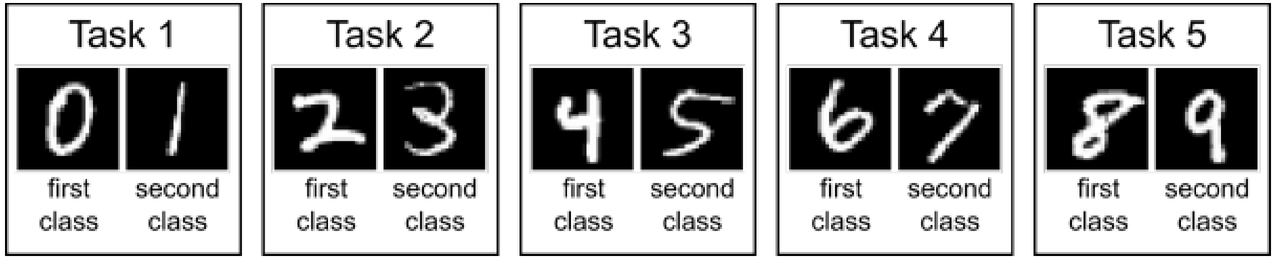
[1]	**continual learning**	Eval	Train	
	task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
	class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
	domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

What exactly is a task?

- Grouped data subset by designer → Explicit bias by design



[1]

- Algorithmically, e.g. every K new classes → Implicit bias by design

How to learn from data streams?

- Current continual learning paradigms

[1]

continual learning	Eval	Train	
task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

How to define task-free settings?

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

Questions? matthias.delange@kuleuven.be

How to learn from data streams?

- Current continual learning paradigms

	continual learning	Eval	Train	
[1]	task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
	class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
	domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

How to define task-free settings?

What resources are available?

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

Questions? matthias.delange@kuleuven.be

How to learn from data streams?

- Current continual learning paradigms

[1] continual learning	Eval	Train	
task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

How to define task-free settings?

What resources are available?

What information is available when?

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

Questions? matthias.delange@kuleuven.be

How to learn from data streams?

- Current continual learning paradigms

[1] continual learning	Eval	Train	
task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

How to define task-free settings?

What resources are available?

What information is available when?

How are training & testing interacting?

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

How to learn from data streams?

- Current continual learning paradigms

	continual learning	Eval	Train	
[1]	task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
	class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
	domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

How to define task-free settings?

What resources are available?

What information is available when?

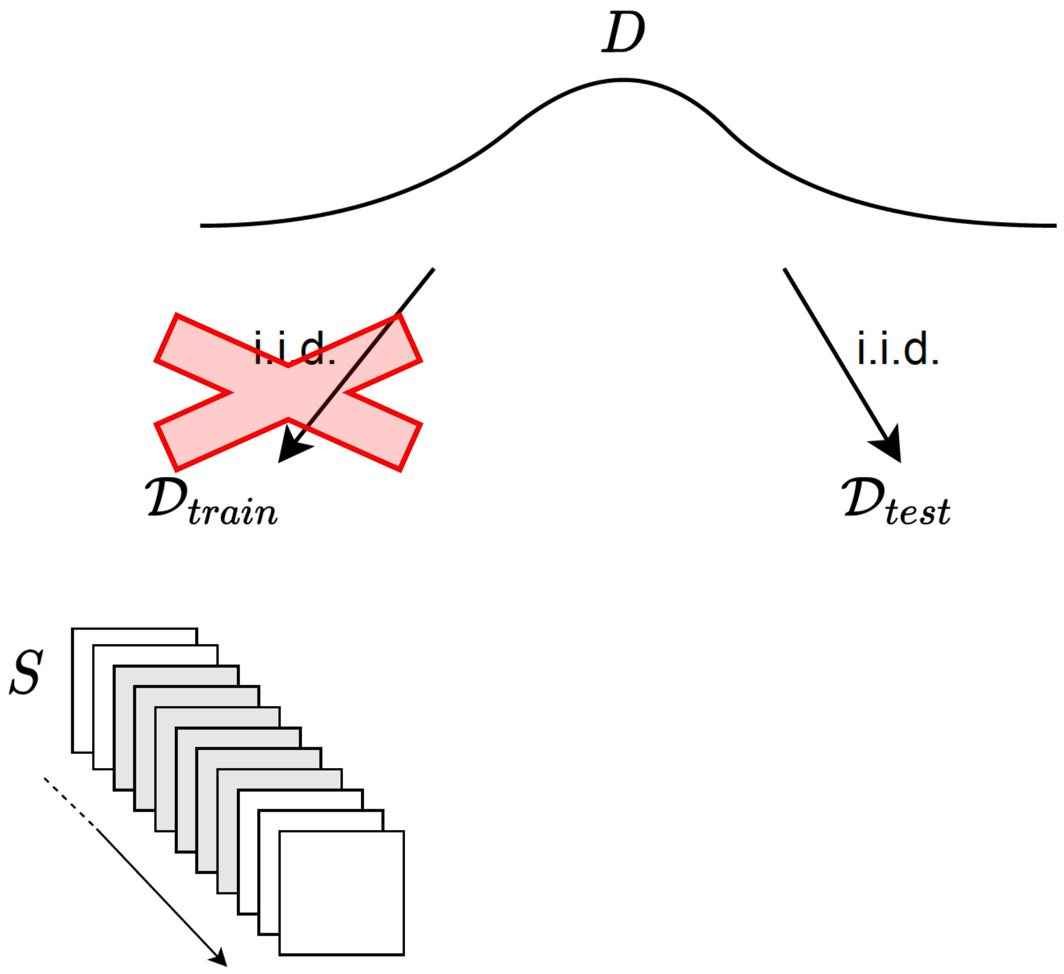
How are training & testing interacting?

What about drifting concepts?

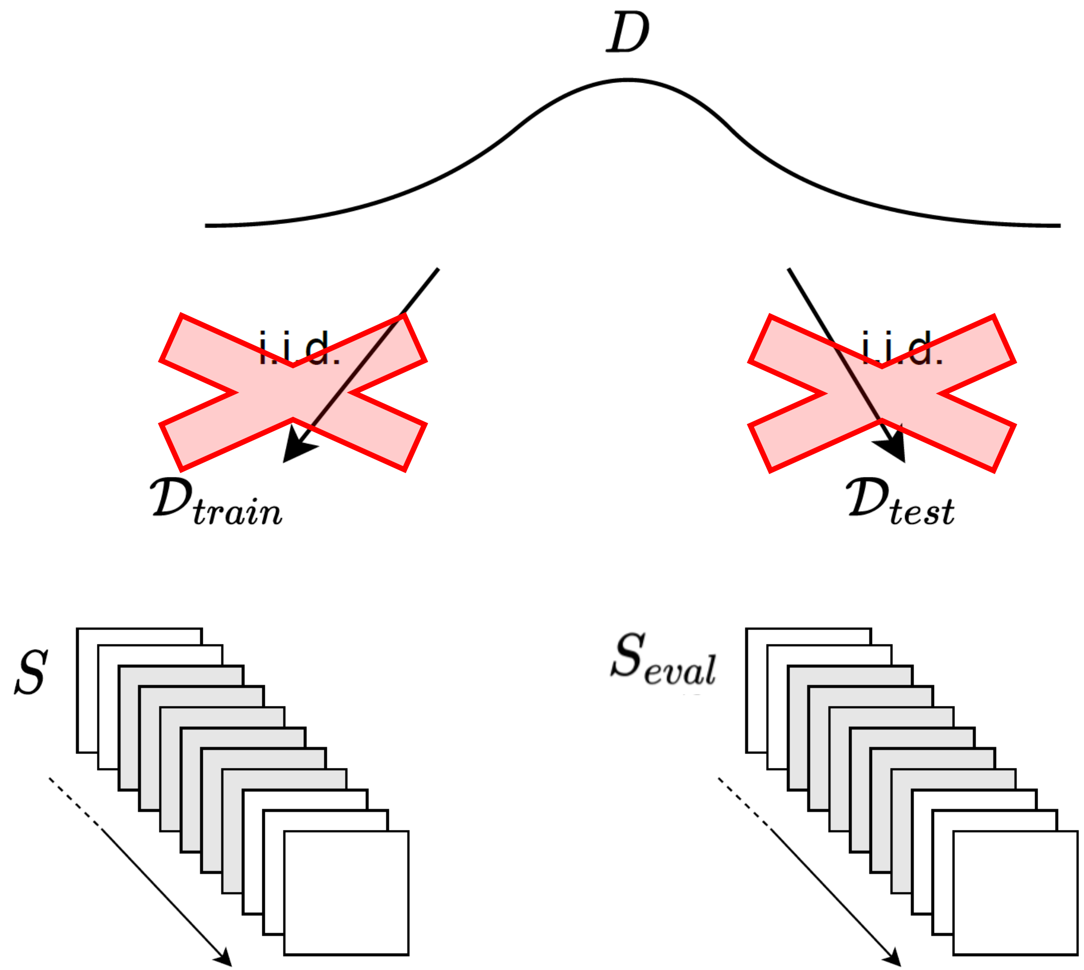
[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

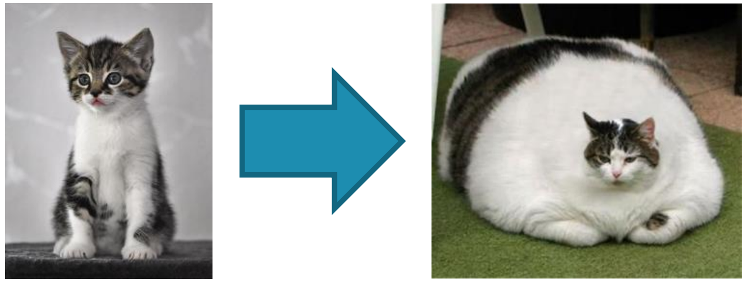
Questions? matthias.delange@kuleuven.be

Sidenote: Concept Drift



Sidenote: Concept Drift





How to learn from data streams?

- Current continual learning paradigms

	continual learning	Eval	Train	
[1]	task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Task transitions
	class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Class-subset transitions
	domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	→ Domain transitions

What exactly is a task?

How to define task-free settings?

What resources are available?

What information is available when?

How are training & testing interacting?

What about drifting concepts?

[1] van de Ven, Gido M., and Andreas S. Tolias. "Three scenarios for continual learning." *arXiv preprint 1904.07734* (2019).

Questions? matthias.delange@kuleuven.be

Learner-Evaluator Framework

- Operate independently

How are training & testing interacting?

- Generalizable to any data stream
  - No notion of task required

What exactly is a task?
How to define task-free settings?

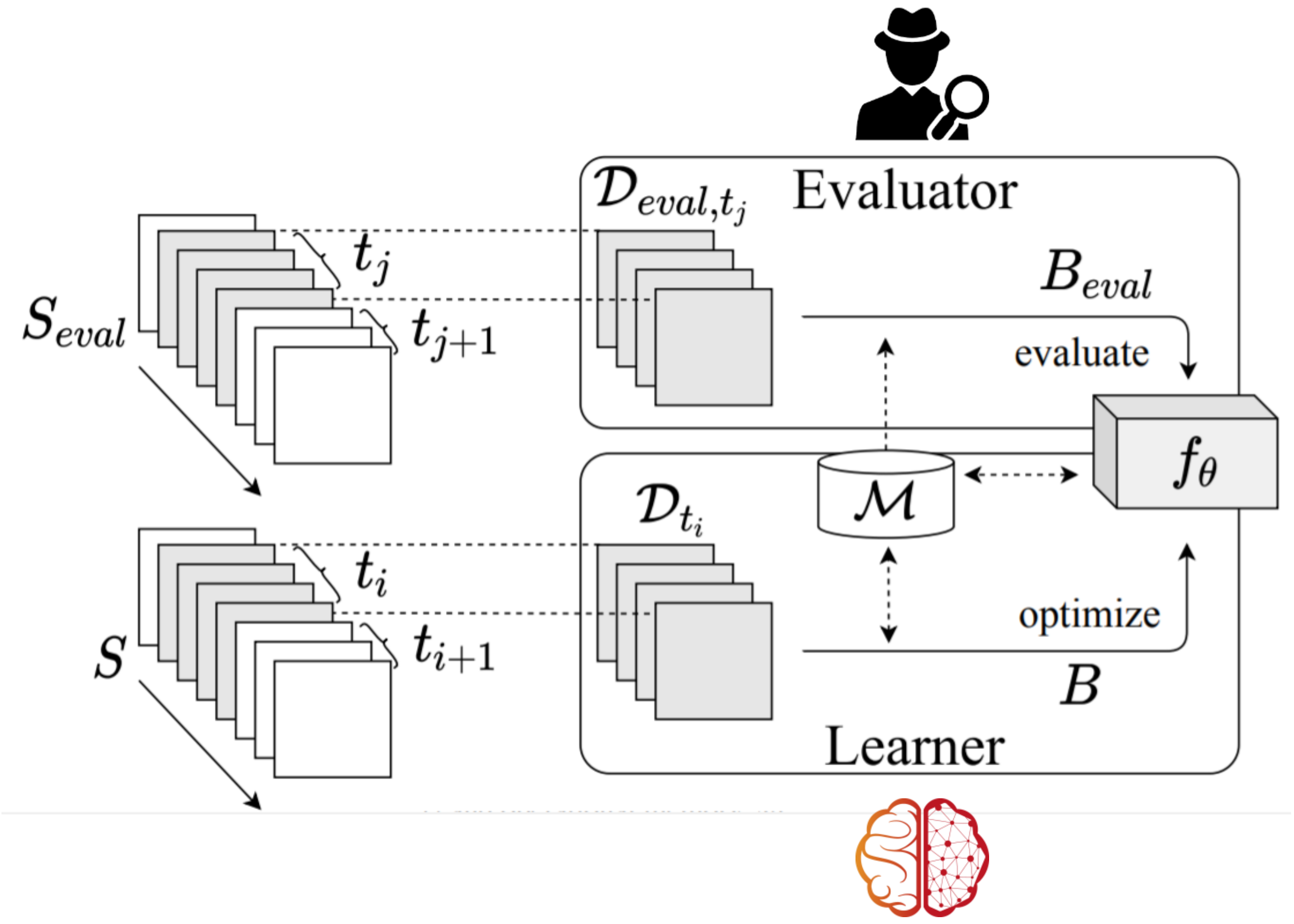
- Generalizable to any evaluation
  - Concept drift ❤️ Continual Learning

What about drifting concepts?

- Horizon $\mathcal{D}_t$
  Operational memory $\mathcal{M}$

What information is available when?
What resources are available?

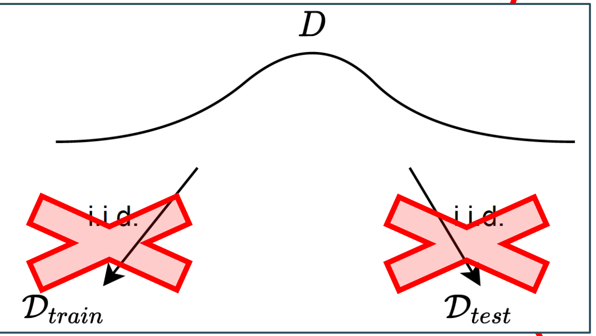
Learner-evaluator framework

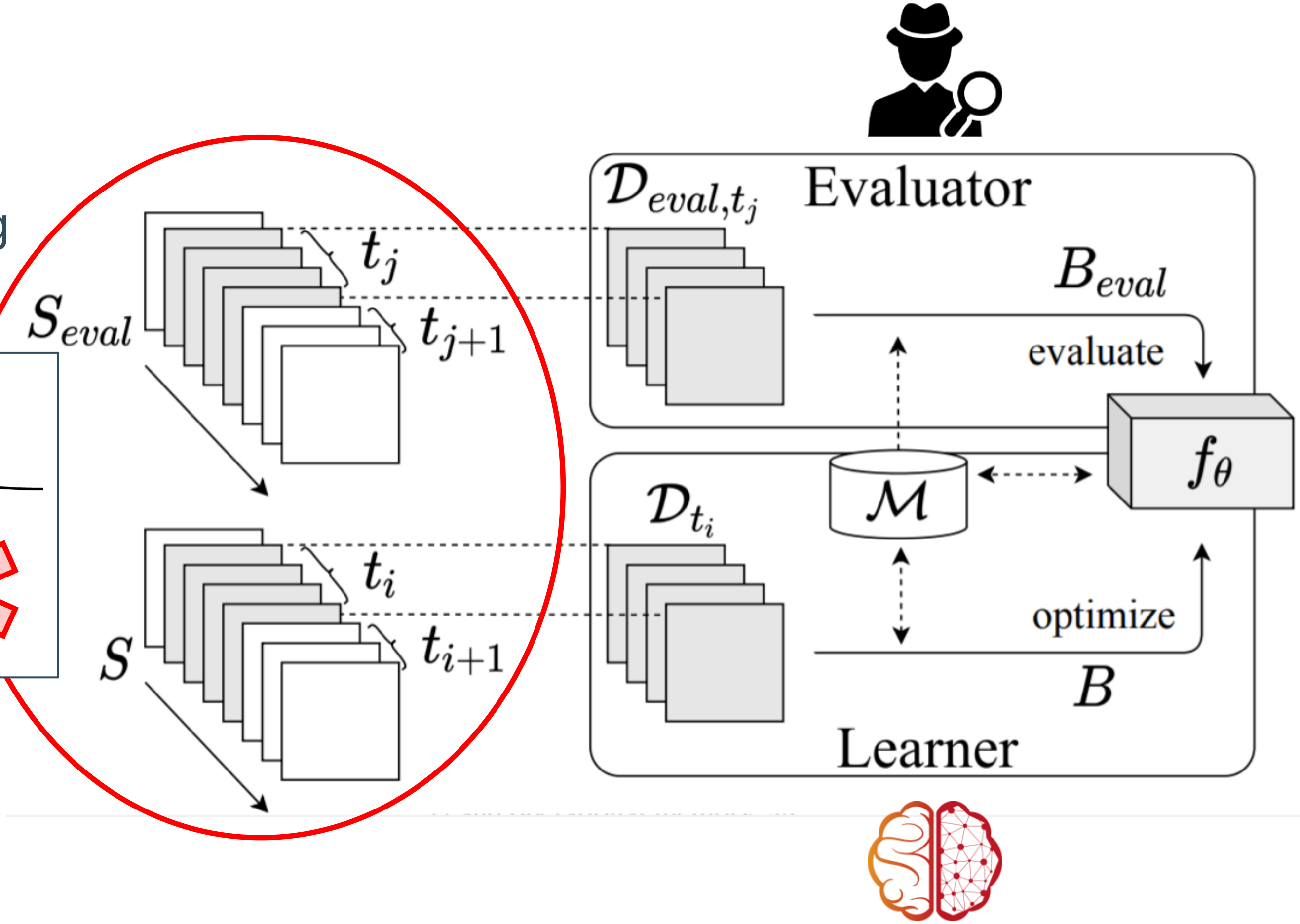


Learner-evaluator framework

Streams model both

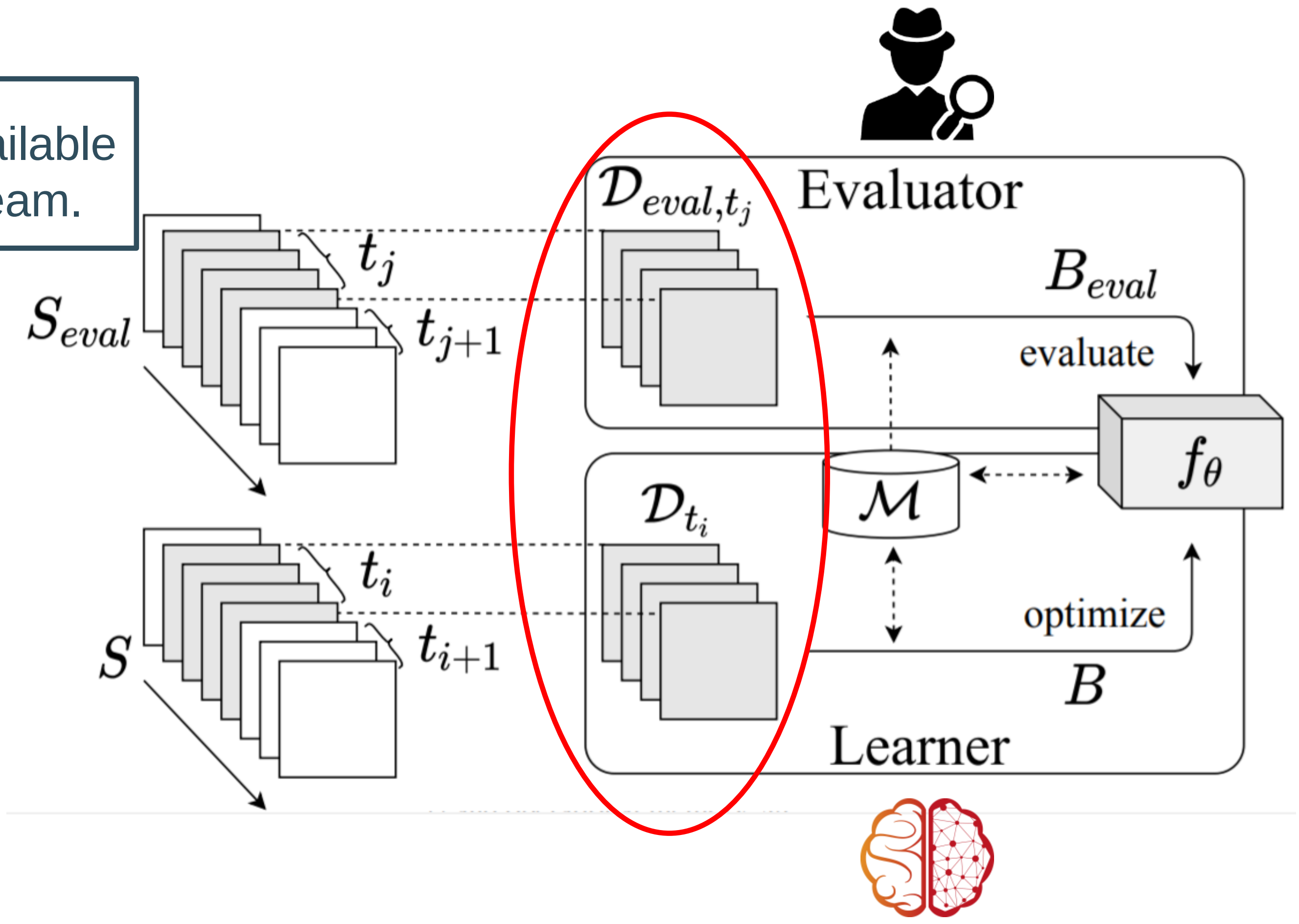
1. Continual Learning
2. Concept Drift





Learner-evaluator framework

Horizon $\mathcal{D}$ is the available subset of the data stream.



Offline (std ML) $\mathcal{D} = S$

Online $\mathcal{D} = B$

Models all CL-paradigms based on transition $\mathcal{D}_{t_i \rightarrow t_{i+1}}$

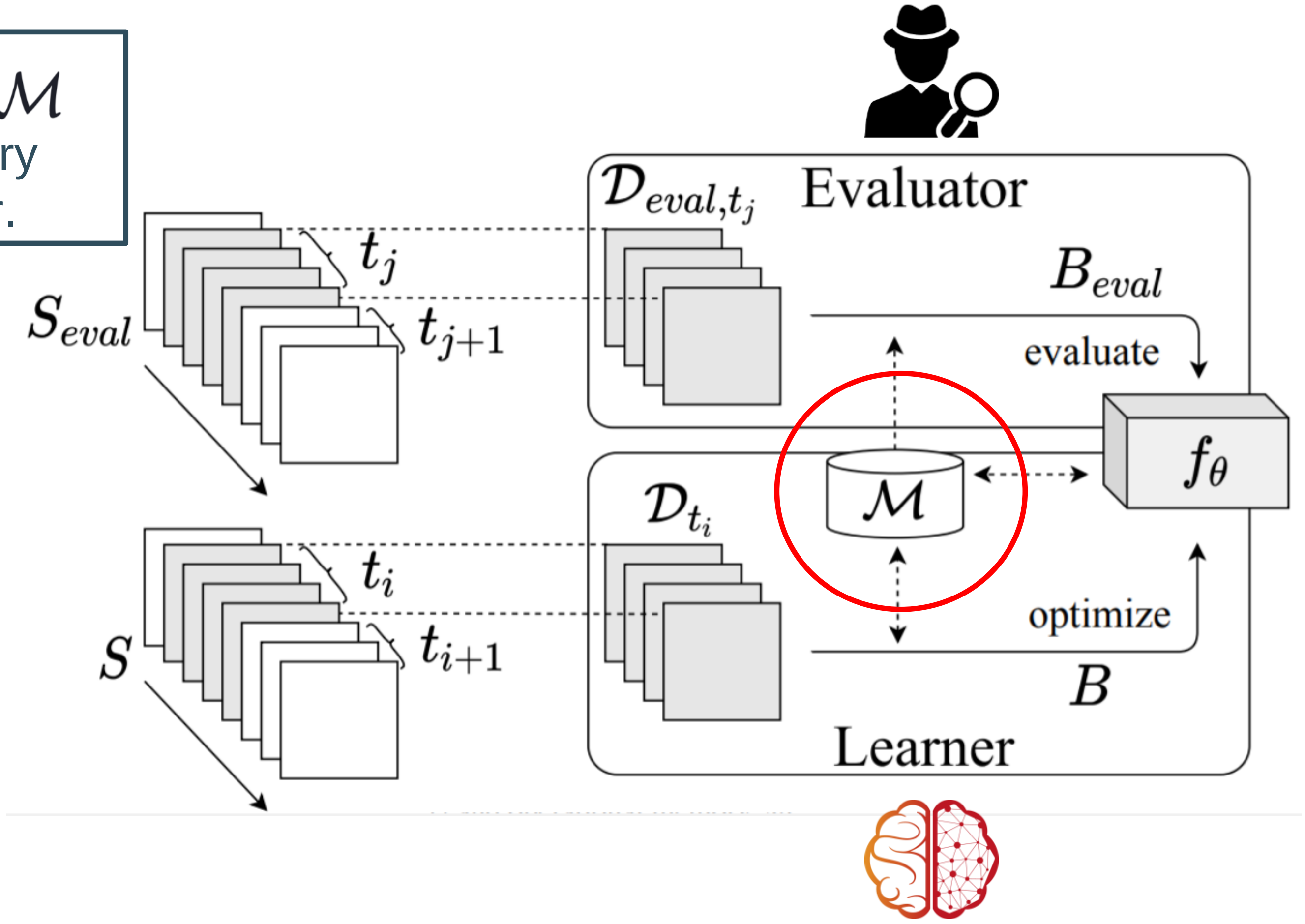
→ Data incremental learning

- Learning from data streams? **Data incremental learning!**
  = *Task-free learning/streaming learning/task-agnostic learning*

	evaluator	learner			
	sample	*sample*	*horizon* $\mathcal{D}$	*iid*	
online learning	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i)$	batch ($\mathcal{D} = B$)	✓	
continual learning					
task incr.	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	task ($\mathcal{D}_{t=t_i}$)	✗	→ Task transitions
class incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	class subset ($\mathcal{D}_{t=t_i}$)	✗	→ Class-subset transitions
domain incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i, t_i)$	domain ($\mathcal{D}_{t=t_i}$)	✗	→ Domain transitions
data incr.	$(\mathbf{x}_i, \mathbf{y}_i)$	$(\mathbf{x}_i, \mathbf{y}_i)$	any subset ($B \leq \mathcal{D} < S$)	✗	**→ Data stream subsets, no assumptions**

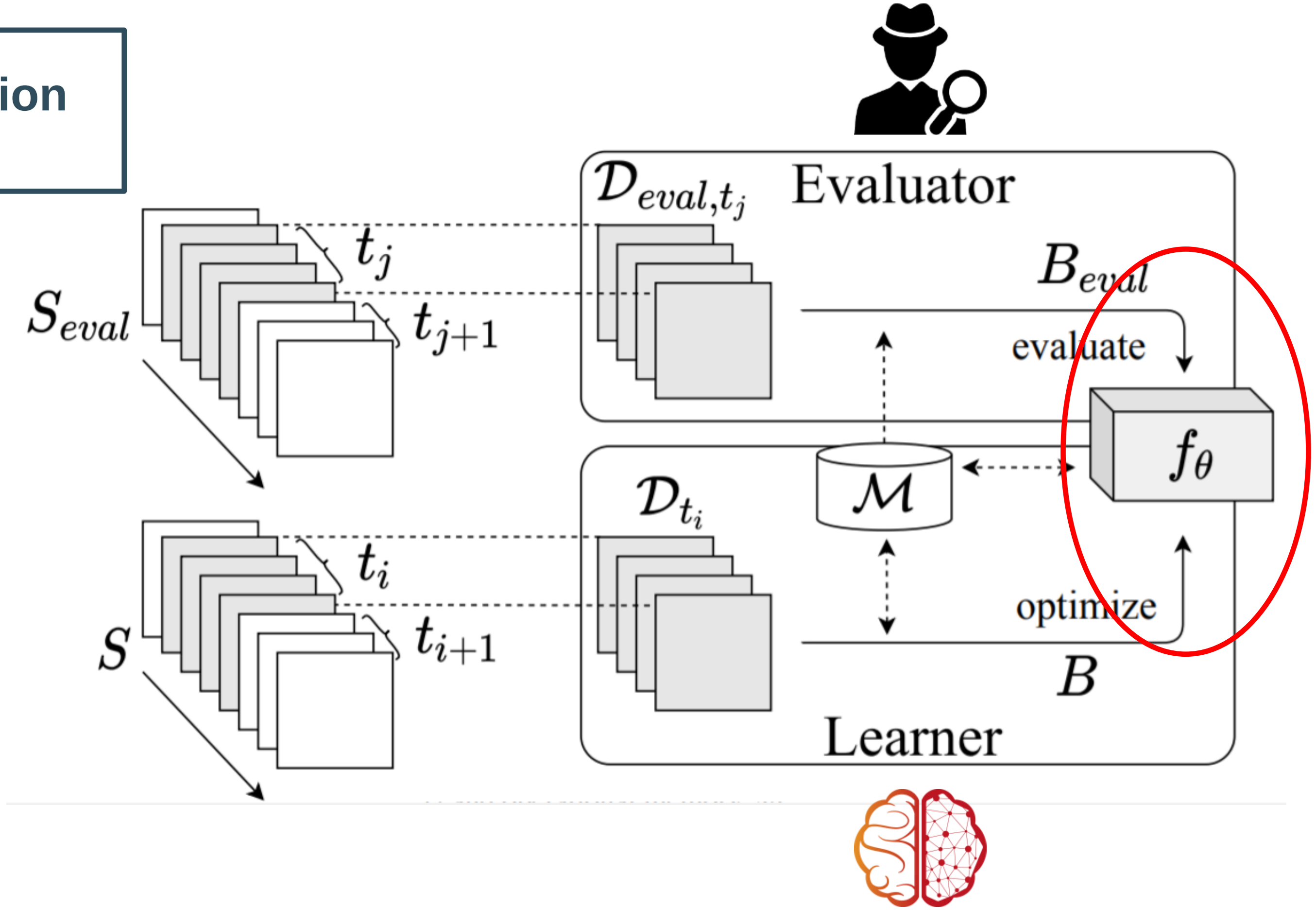
Learner-evaluator framework

Operational Memory $\mathcal{M}$ is the additional memory used by the CL learner.



Learner-evaluator framework

Learning and evaluation act independent.



Roadmap

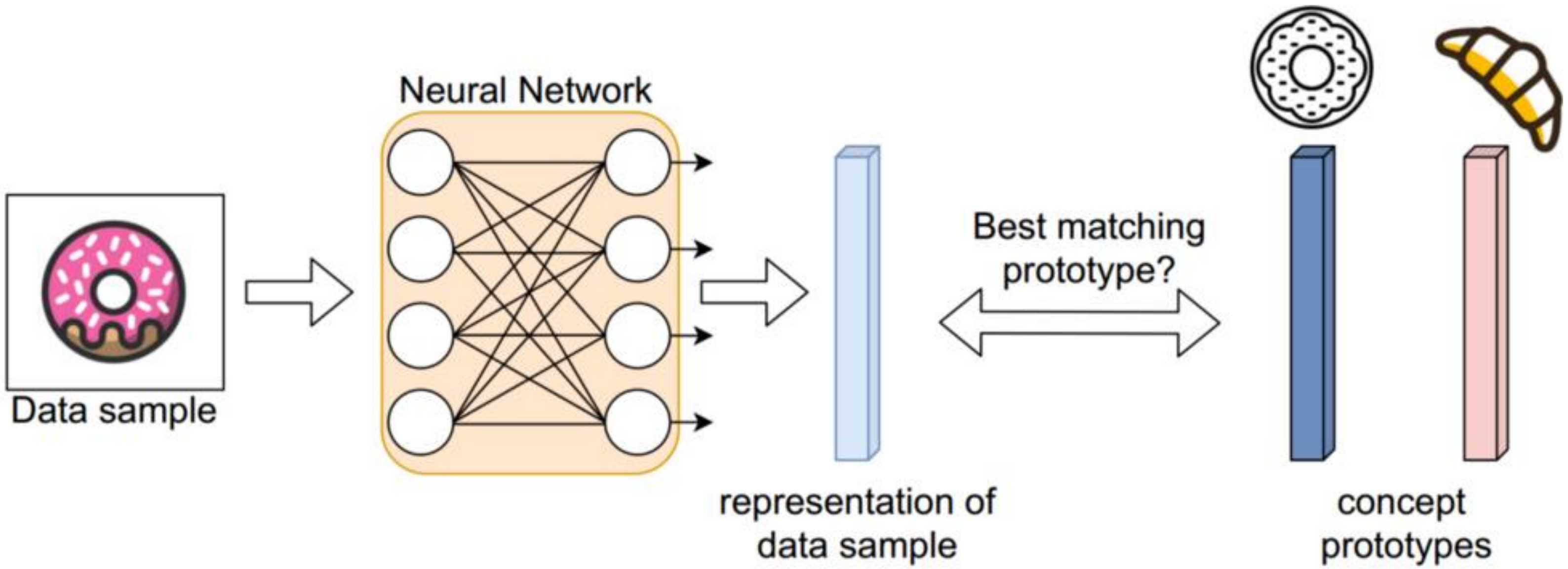
- What is Continual Learning?
- How to learn from data streams?
- **Why representation learning?**
- Continual Prototype Evolution (CoPE)
  - Evolving prototypes
  - Balanced replay
  - PPP-loss

Follow along:

https://arxiv.org/pdf/2009.00919.pdf

Why representation learning?

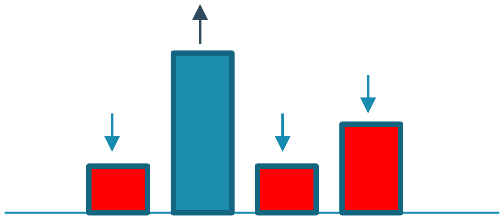
- Catastrophic forgetting
  → Optimization influences **entire parameter space**!
- Instead, learn a **low-dimensional embedding** with prototypes



Why representation learning?

- **Typical softmax classifier with CE-loss**
  - Push away other-class weight vectors
  - Long unseen classes = unpredictable

  → Contrastive losses: Pair/triplet wise interaction



Questions? matthias.delange@kuleuven.be

But How?

2 problems to maintain prototypes

1. Each update step → Representation space changes
   = Prototypes become stale

2. Non-iid data → Some classes never seen again
   = Prototypes become stale

Roadmap

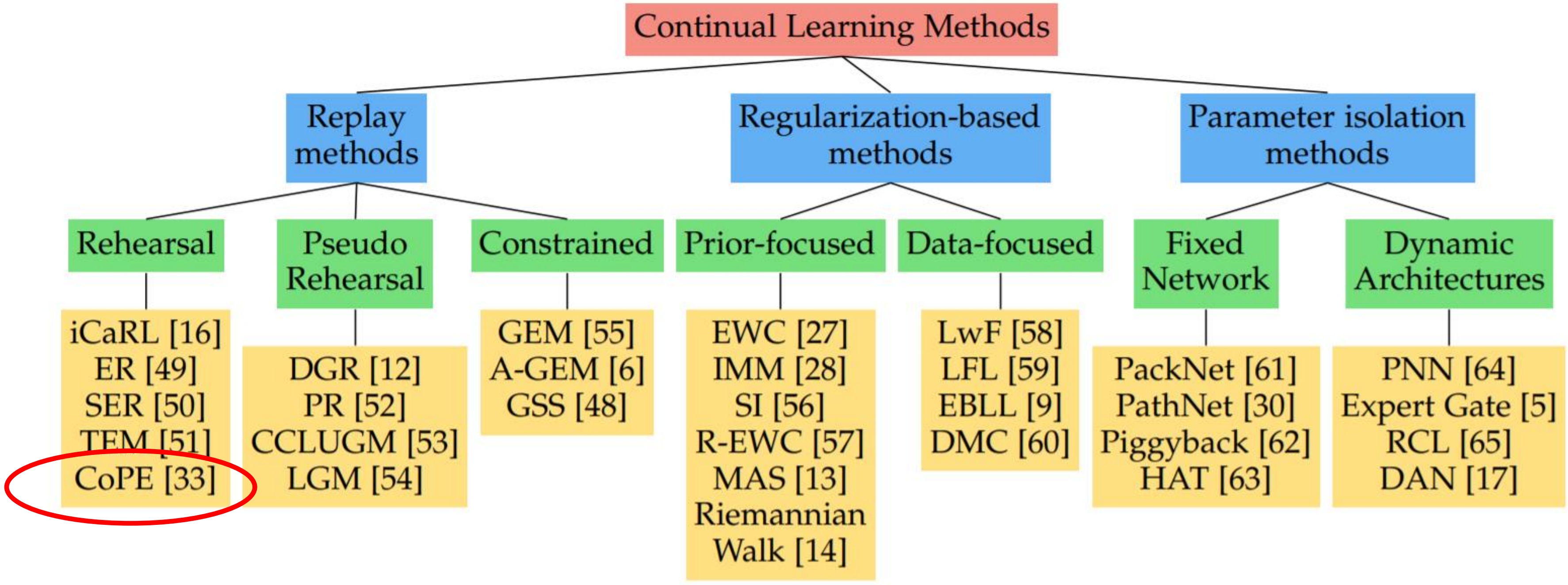
- What is Continual Learning?
- How to learn from data streams?
- Why representation learning?
- **Continual Prototype Evolution (CoPE)**
  - Evolving prototypes
  - PPP-loss
  - Balanced replay

Follow along:

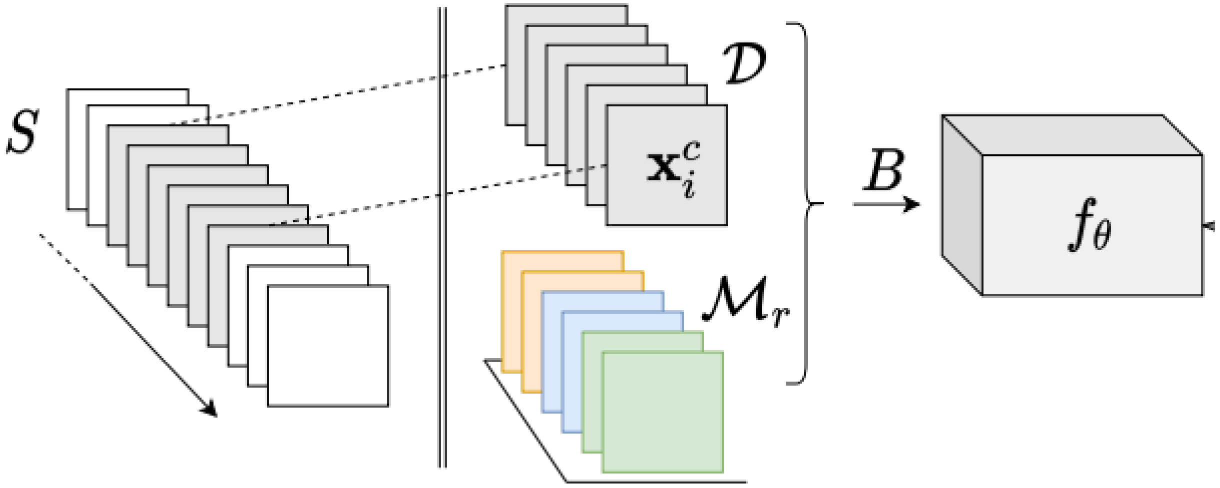
https://arxiv.org/pdf/2009.00919.pdf

Questions? matthias.delange@kuleuven.be

CoPE: Continual Prototype Evolution

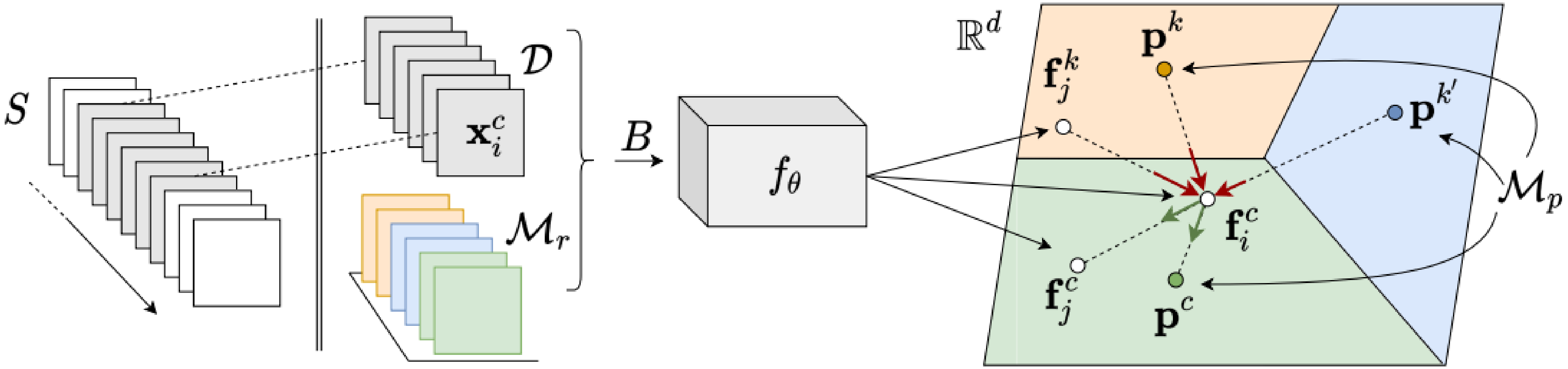


CoPE: Continual Prototype Evolution



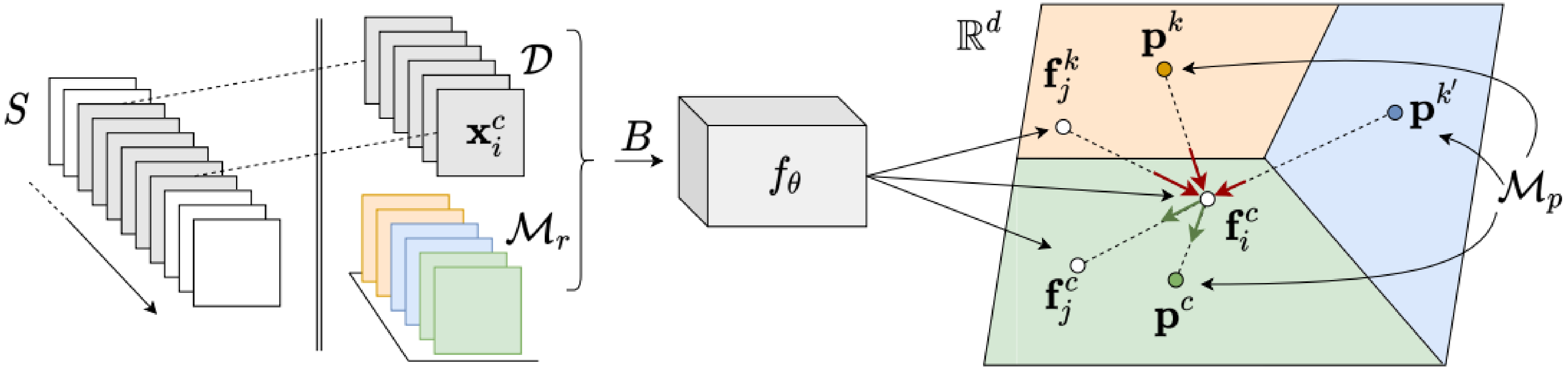
CoPE: Continual Prototype Evolution

- Operates
  - Online
  - Data incremental
  - Imbalanced data



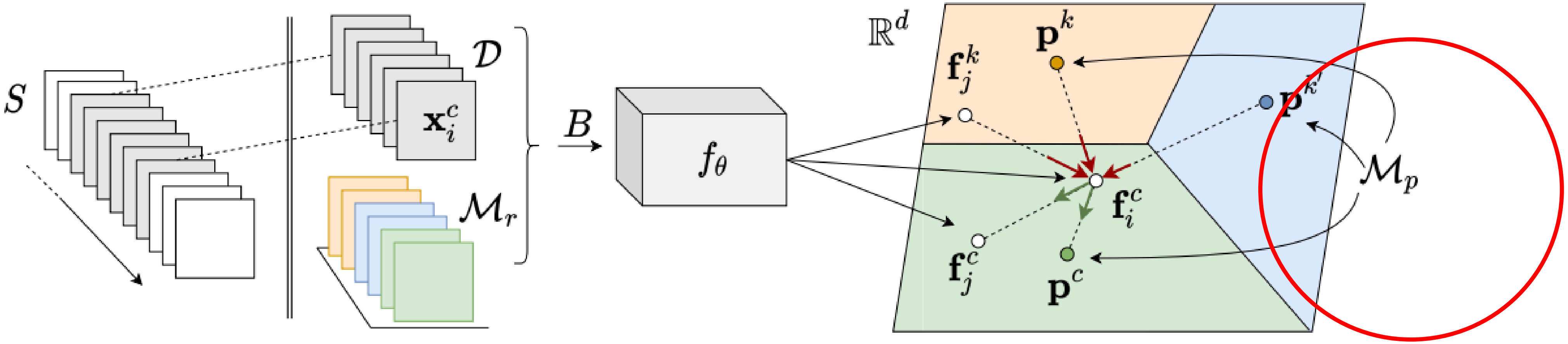
CoPE: Continual Prototype Evolution

- 3 components
  - continually evolving prototypes
  - Pseudo-prototypical proxy loss (PPP-loss)
  - Balanced replay



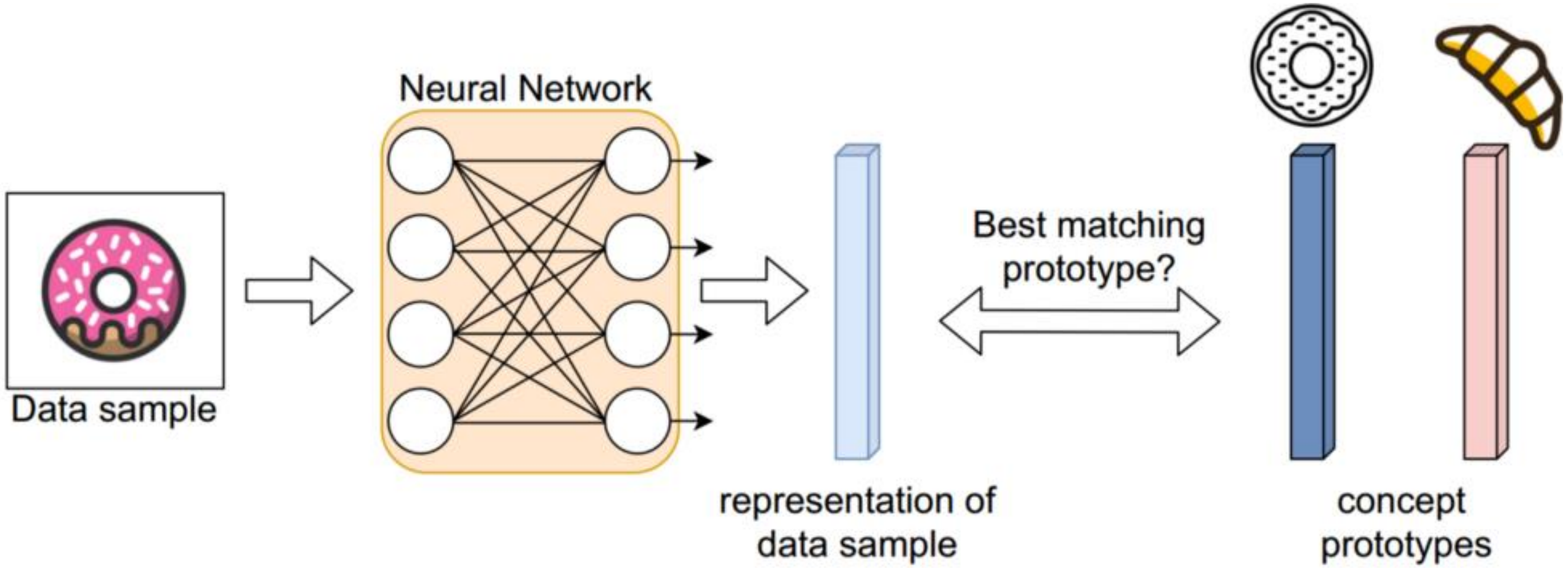
CoPE: Component 1, Prototypes

Prototypical Memory



CoPE: Component 1, Prototypes

- Prototypes → Nearest Neighbour classifier



CoPE: Component 1, Prototypes

- CL literature: recalculated <u>on task transitions</u> with the FULL memory
  ✗ Exhaustive recalculation
  ✗ Dependent on task transitions
  ✗ Static and outdated between task transitions!

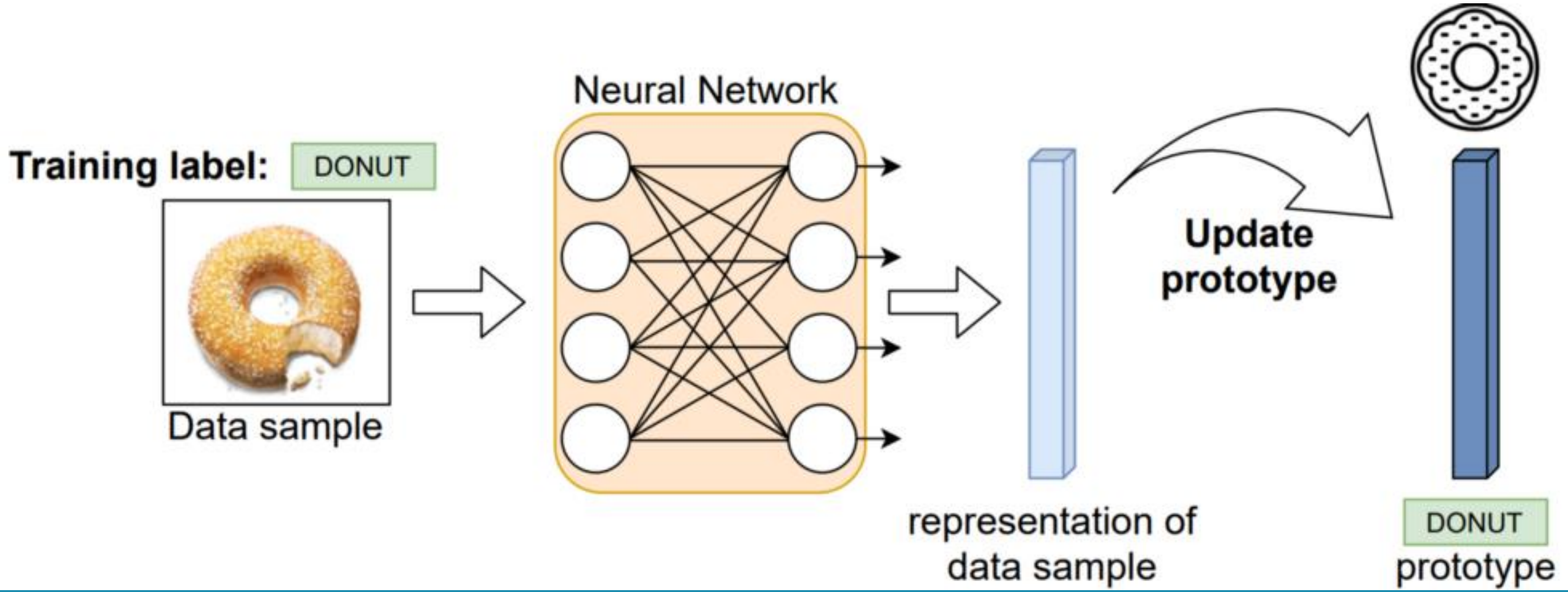
- CoPE updates <u>online batch-wise</u> with high momentum
  ✓ Low resource usage
  ✓ Only dependent batch transition
  ✓ Always representative!

Questions? matthias.delange@kuleuven.be

CoPE: Component 1, Prototypes

- CoPE updates online batch-wise with high momentum

$$\mathbf{p}^c \leftarrow \alpha \mathbf{p}^c + (1-\alpha)\bar{\mathbf{p}}^c, \text{ s.t.} \quad \bar{\mathbf{p}}^c = \frac{1}{|B^c|} \sum_{\mathbf{x}^c \in B^c} f_\theta(\mathbf{x}^c)$$



CoPE: Component 1, Prototypes

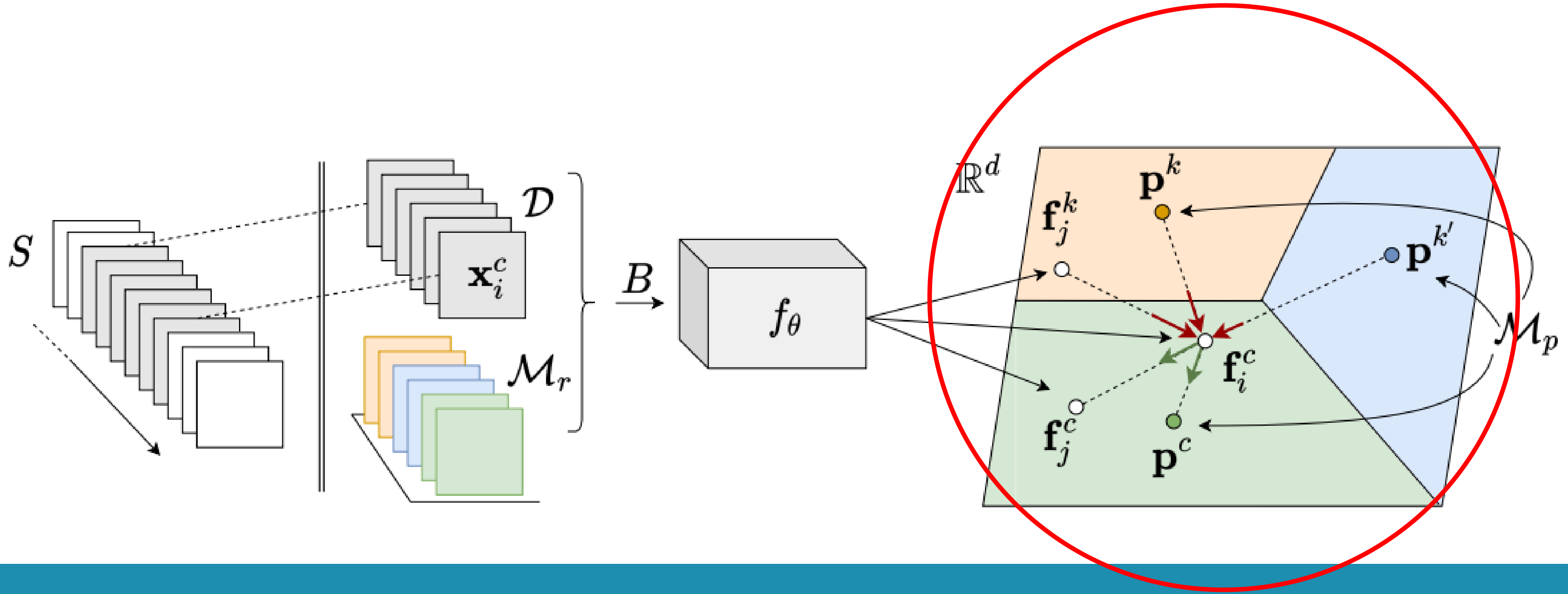
- But, **how** do the prototypes **remain representative**?

  → Ever evolving latent space with each update

  → Non-stationary data → Catastrophic forgetting

- Other 2 components:
  - PPP-loss
  - Balanced replay

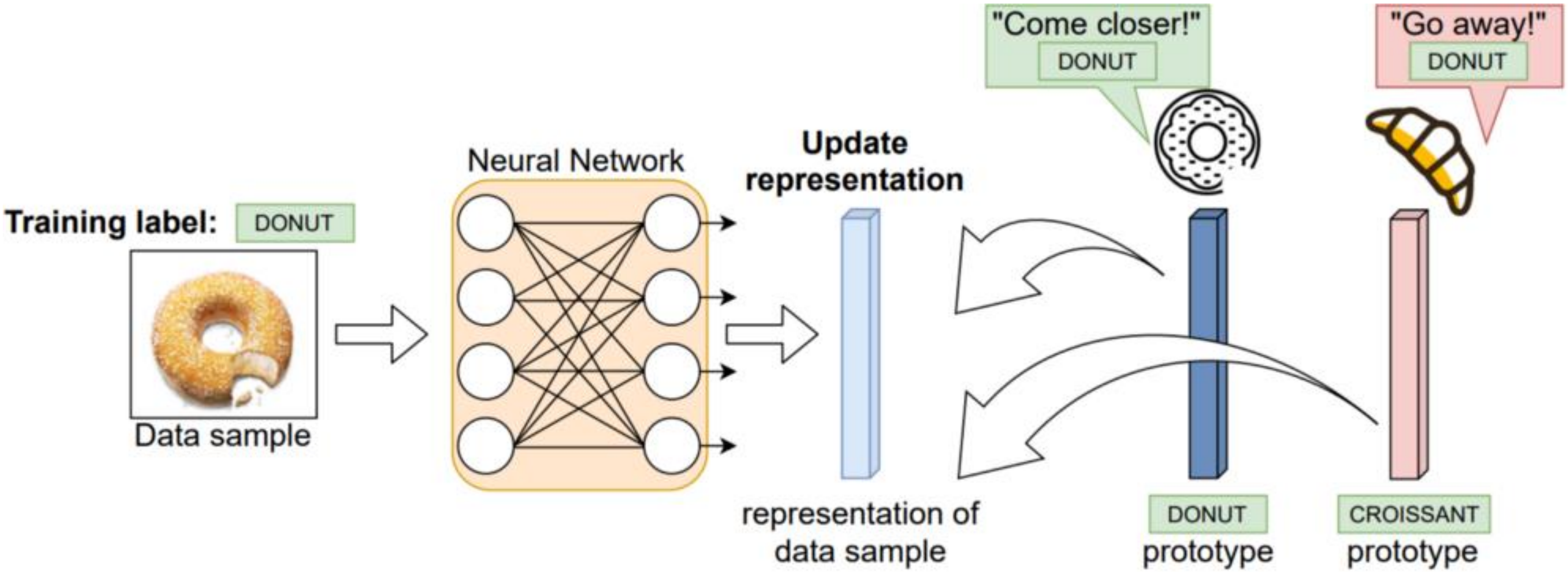
CoPE: Component 2, PPP-loss



KU LEUVEN

CoPE: Component 2, PPP-loss

How to update representations?



CoPE: Component 2, PPP-loss

- Pseudo-Prototypical Proxy loss
- Batch *B* not only gives supervision about instance category
  → Also relational information in the latent space!
- Construct per instance, one-against-all subsets:

$B^c = \{(\mathbf{x}_i, y_i = c) \in B\}$ and B^k

Questions? matthias.delange@kuleuven.be

CoPE: Component 2, PPP-loss

- **Main idea**
  For each instance $\mathbf{x}_i^c$ in the batch, we want it to

  1. Be close to its prototype and remaining class-instances in the batch
     → Attractor set: $\mathbb{P}_i^c = \{\mathbf{p}^c\} \cup \{\hat{\mathbf{p}}_j^c = f_\theta(\mathbf{x}_j^c) \mid \forall \mathbf{x}_j^c \in B^c,\ i \neq j\}$

  2. Push other-class instances away
     → Repellor set: $\mathbb{U}_i^c = \{\mathbf{p}^c,\ \hat{\mathbf{p}}_i^c = f_\theta(\mathbf{x}_i^c)\}$

KU LEUVEN

CoPE: Component 2, PPP-loss

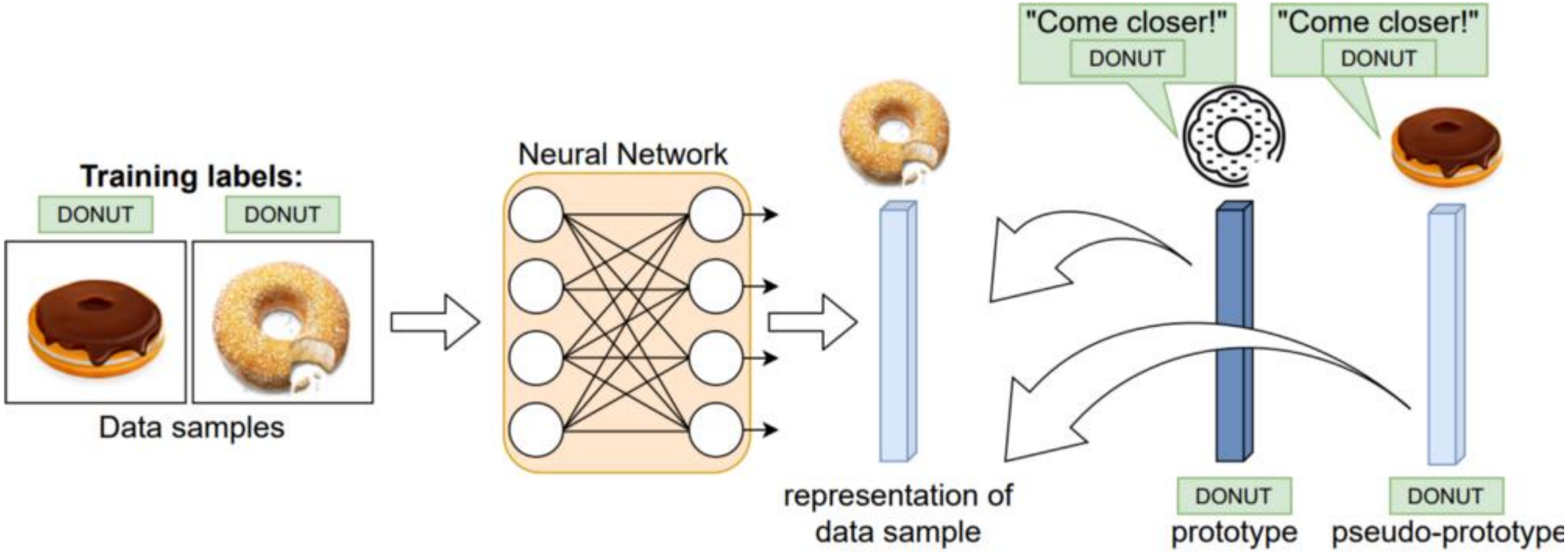
The Pseudo-Prototypical Proxy Loss:

$$\mathcal{L} = -\frac{1}{|B|}\left[\sum_i \log P(c|\mathbf{x}_i^c) + \sum_i \sum_{\mathbf{x}_j^k} \log(1 - P_i(c|\mathbf{x}_j^k))\right]$$

See paper for details.

CoPE: Component 2, PPP-loss

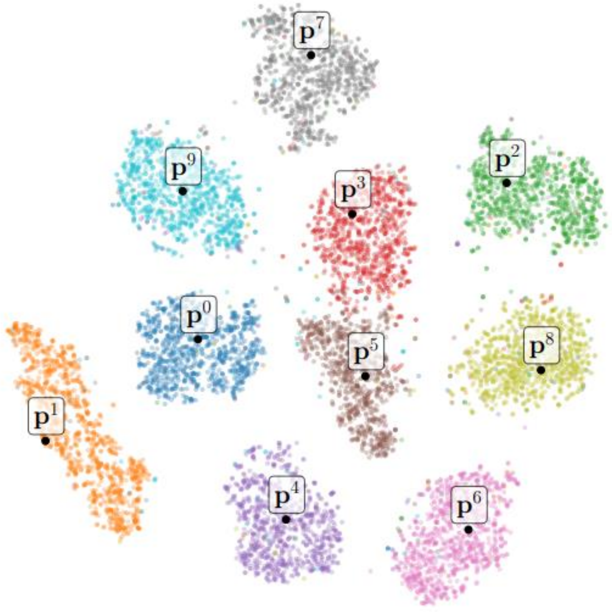
The Pseudo-Prototypical Proxy Loss:



PPP-loss ablation

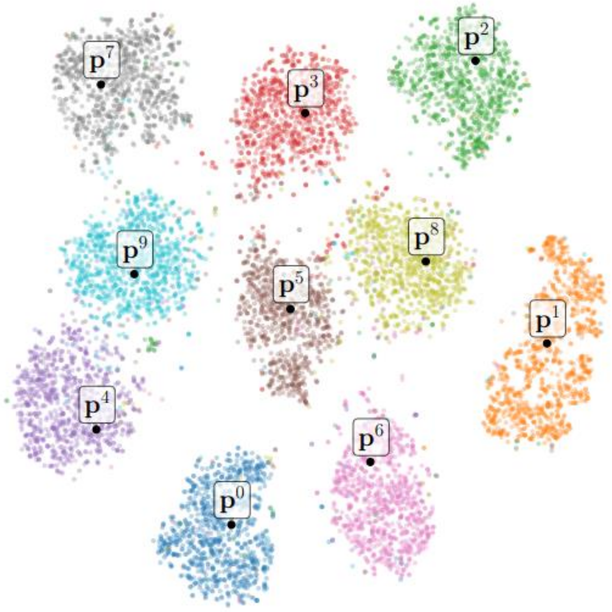
- Including/excluding pseudo-prototypes in PPP-loss

94.52%



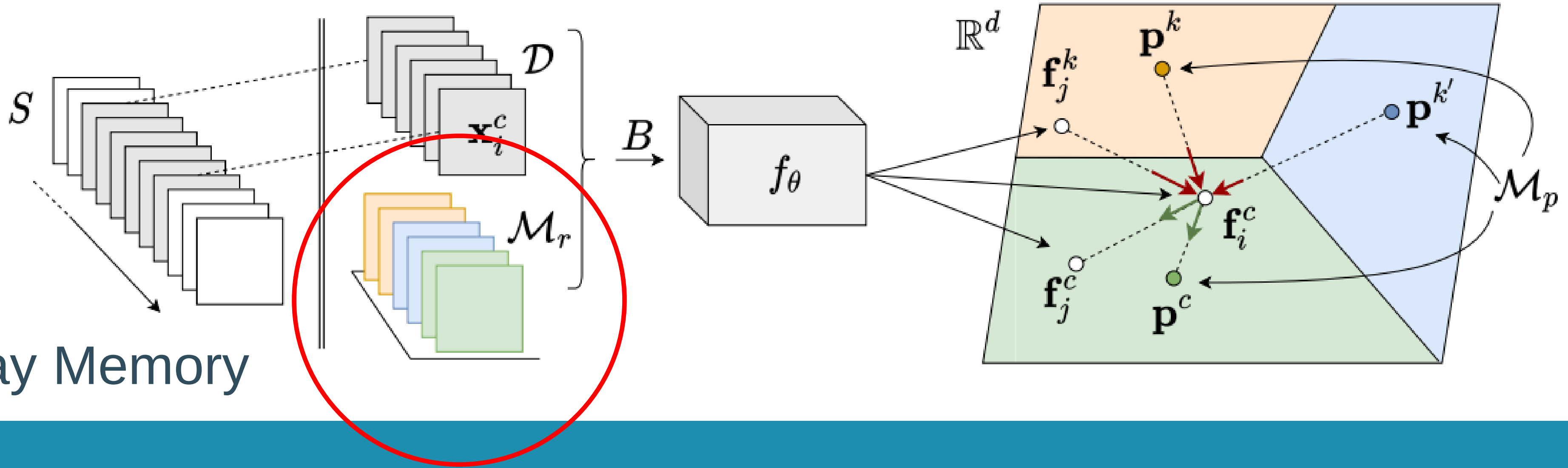
(a) PPP-loss – *incl.* $\hat{\mathbf{p}}$

90.86%



(b) PPP-loss – *excl.* $\hat{\mathbf{p}}$

CoPE: Component 3, Balanced replay



Replay Memory

CoPE: Component 3, Balanced replay

- Prior: deem each class equally important
  - **Storage**: Dynamic class memory $\mathcal{M}_r^c$ based on reservoir sampling
  - **Easy Retrieval**: Uniform = class-balanced batch
    → Keeps all class-prototypes up-to-date

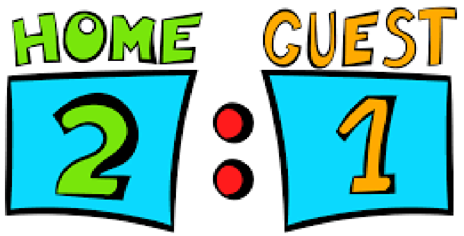
- **Replay benefits:**
  1. Standard replay: Make input batches more iid

CoPE: Component 3, Balanced replay

- Prior: deem each class equally important
  - **Storage**: Dynamic class memory $\mathcal{M}_r^c$ based on reservoir sampling
  - **Easy Retrieval**: Uniform = class-balanced batch
    → Keeps all class-prototypes up-to-date

- **Replay benefits:**
  1. Standard replay: Make input batches more iid

  2. In representation learning:
  Latent batch information for all classes



Experiments

- Learner:
  - Online processing with |B|=10
  - S subdivided in task-like sequence (to compare with iCaRL/GEM)
    → CoPE learner is unaware of this! (not provided)

- Evaluator:
  - held-out dataset of static concepts in S_eval, evaluating with the subset of seen concepts Y in D_eval using the accuracy metric.

Prior Work

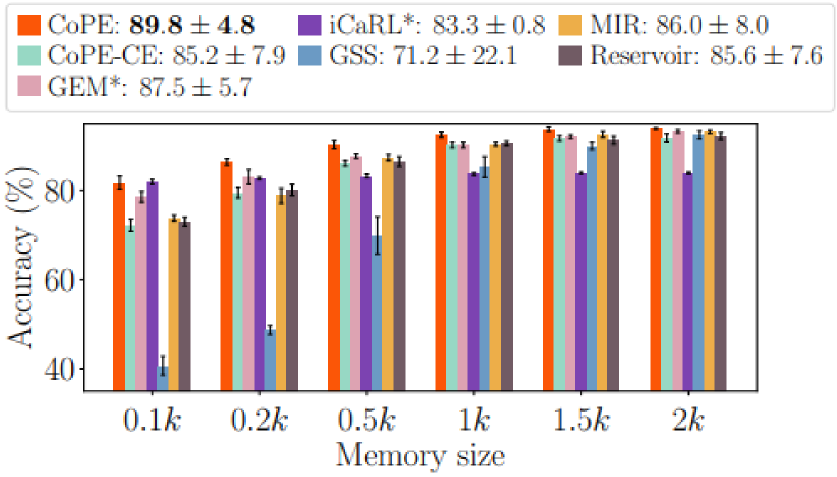
- **Online data incremental learning** ($\mathcal{D} = B$)
  - Replay: Reservoir, GSS , MIR
  - Parameter isolation methods: CURL, CN-DPM
- **Class incremental**: iCaRL, GEM

Balanced data streams

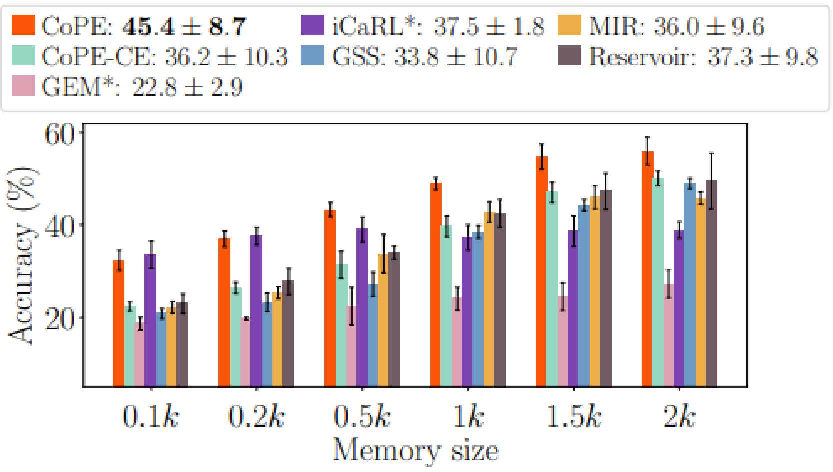
	Split-MNIST	Split-CIFAR10	Split-CIFAR100
iid-offline	98.44 ± 0.02	83.02 ± 0.60	50.28 ± 0.66
iid-online	96.57 ± 0.14	62.31 ± 1.67	20.10 ± 0.90
finetune	19.75 ± 0.05	18.55 ± 0.34	3.53 ± 0.04
GEM	93.25 ± 0.36	24.13 ± 2.46	11.12 ± 2.48
iCARL	83.95 ± 0.21	37.32 ± 2.66	10.80 ± 0.37
CURL (Rao et al., 2019)	92.59 ± 0.66	–	–
DN-CPM (Lee et al., 2020)	93.23 ± 0.09	45.21 ± 0.18	20.10 ± 0.12
reservoir	92.16 ± 0.75	42.48 ± 3.04	19.57 ± 1.79
MIR	93.20 ± 0.36	42.80 ± 2.22	20.00 ± 0.57
GSS	92.47 ± 0.92	38.45 ± 1.41	13.10 ± 0.94
CoPE-CE	91.77 ± 0.87	39.73 ± 2.26	18.33 ± 1.52
CoPE (ours)	$\mathbf{93.94 \pm 0.20}$	$\mathbf{48.92 \pm 1.32}$	$\mathbf{21.62 \pm 0.69}$

“Sure dude, but you just tweaked the buffer size?”

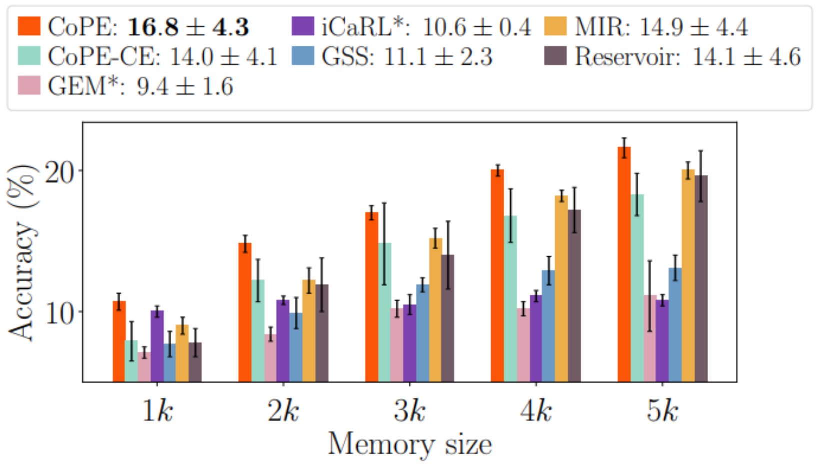
Split-MNIST



Split-CIFAR10



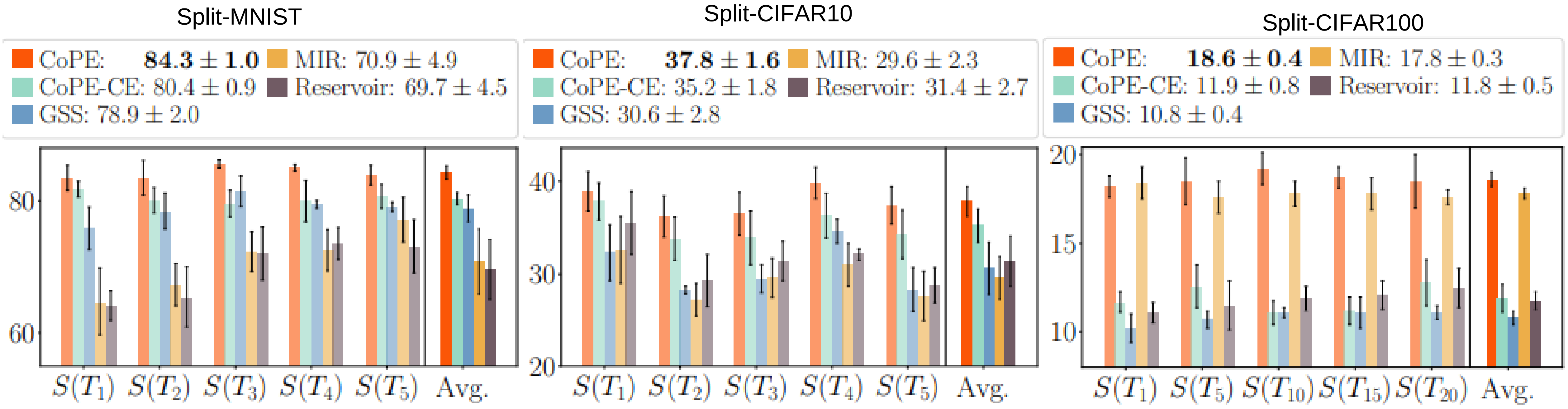
Split-CIFAR100



- Discrepancy CoPE-CE / CoPE → Efficacy prototypical approach

Imbalanced experiments

- Not just the balancing memory scheme (CoPE-CE)
- The PPP-loss encourages prototype-based clusters each update



Summary

- Learner-evaluator framework → 2 agents, horizon (~~task~~), concept drift
- Data incremental learning → Any data stream (~~task info~~)
- CoPE → Online data incremental
  → Continually evolving prototypes
  → Balanced replay
  → PPP-loss
- Future? → Apply for concept drift, beyond classification/supervised learning

Code

https://github.com/mattdl/ContinualPrototypeEvolution

Blog

https://ai.kuleuven.be/stories/post/2021-05-10-continual-learning/

Questions? matthias.delange@kuleuven.be

ContinualAI

https://www.continualai.org/

Free-Access Workshop CVPR

https://sites.google.com/view/clvision2021/overview