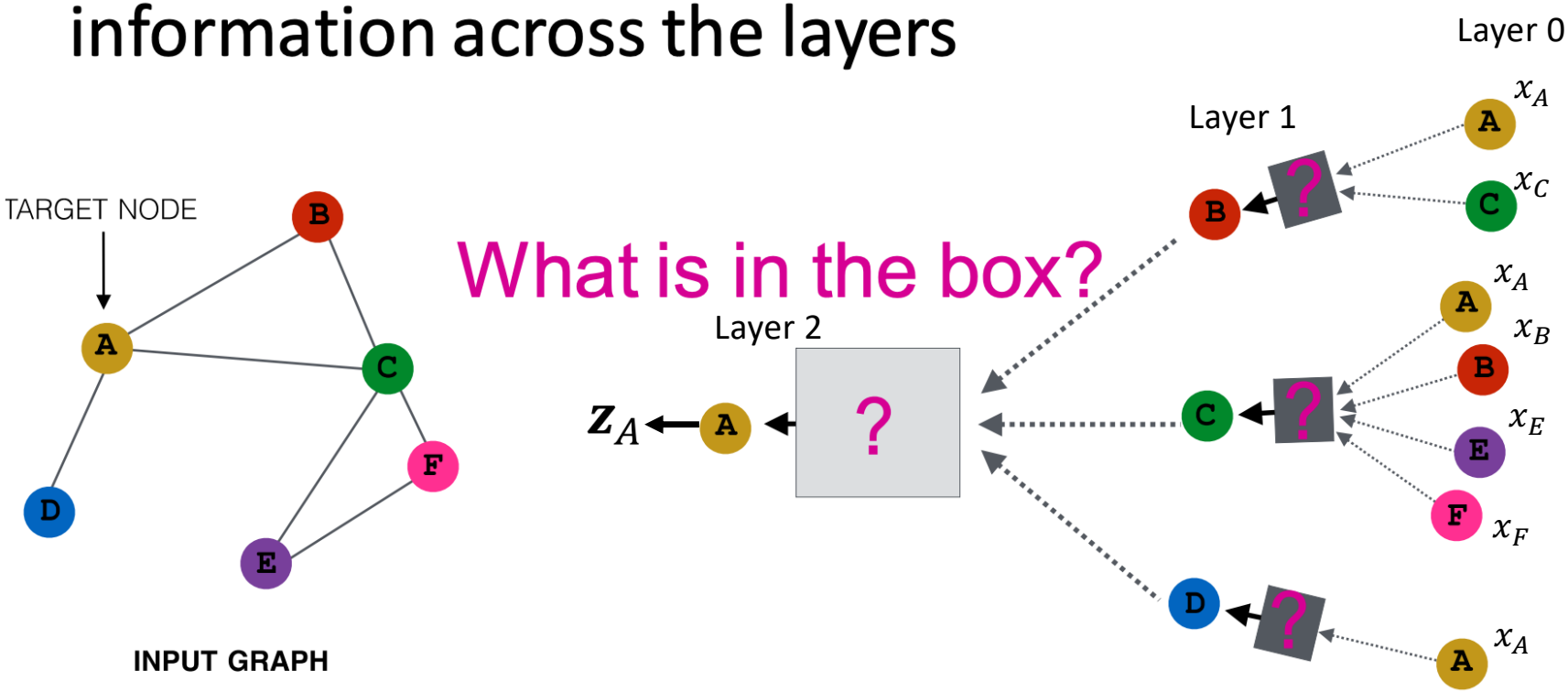
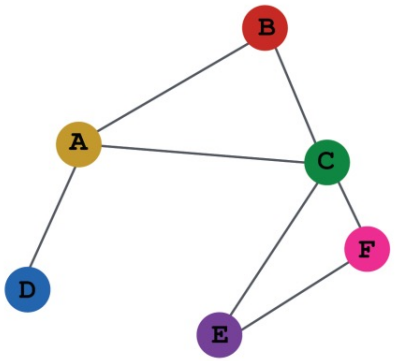
Meta learning and ROLAND

- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers

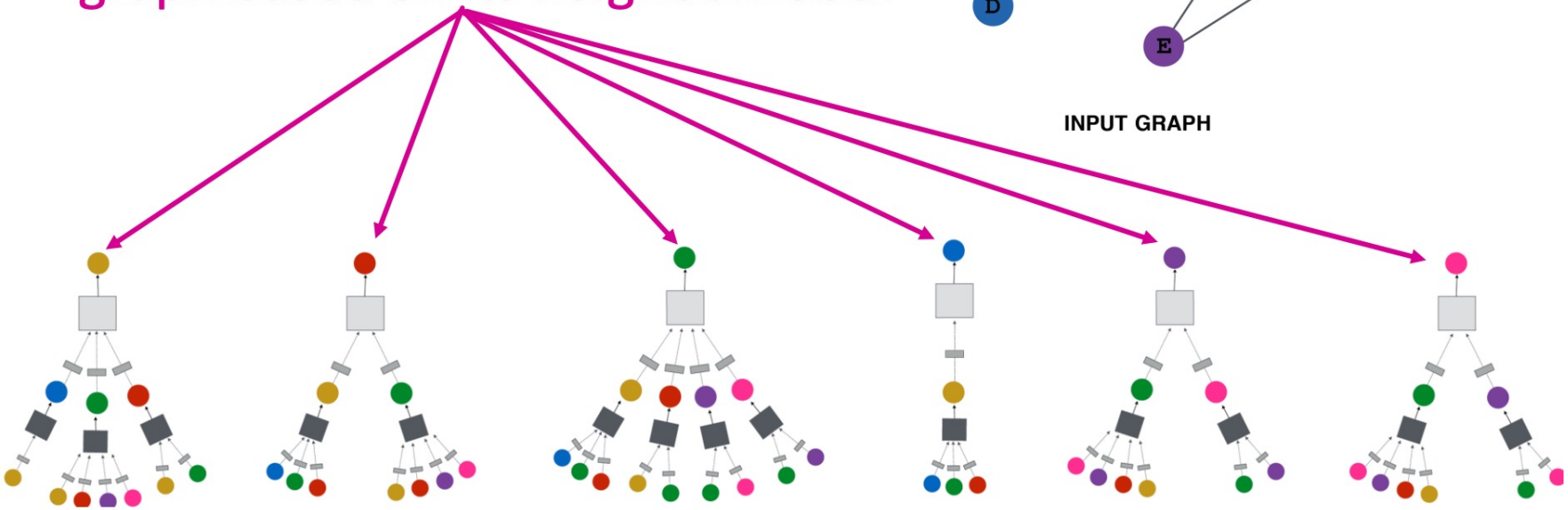


- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



INPUT GRAPH



Trainable weight matrices
(i.e., what we learn)

$$\mathrm{h}_v^{(0)} = \mathrm{x}_v$$

$$\mathrm{h}_v^{(k+1)} = \sigma\left(\mathrm{W}_k \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(k)}}{|\mathrm{N}(v)|} + \mathrm{B}_k \mathrm{h}_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

$$h_v^{(k+1)} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

$$h_v^{(k+1)} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$z_v = h_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Trainable weight matrices
(i.e., what we learn)

$$\mathrm{h}_v^{(0)} = \mathrm{x}_v$$

$$\mathrm{h}_v^{(k+1)} = \sigma\left(\mathrm{W}_k \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(k)}}{|\mathrm{N}(v)|} + \mathrm{B}_k \mathrm{h}_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Trainable weight matrices
(i.e., what we learn)

$$\mathrm{h}_v^{(0)} = \mathrm{x}_v$$

$$\mathrm{h}_v^{(k+1)} = \sigma\left(\mathrm{W}_k \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(k)}}{|\mathrm{N}(v)|} + \mathrm{B}_k \mathrm{h}_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Trainable weight matrices
(i.e., what we learn)

$$\mathrm{h}_v^{(0)} = \mathrm{x}_v$$

$$\mathrm{h}_v^{(k+1)} = \sigma\left(\mathrm{W}_k \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(k)}}{|\mathrm{N}(v)|} + \mathrm{B}_k \mathrm{h}_v^{(k)}\right), \forall k \in \{0..K-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(K)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k: the hidden representation of node v at layer k

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

(1) Graph Convolutional Networks (GCN)

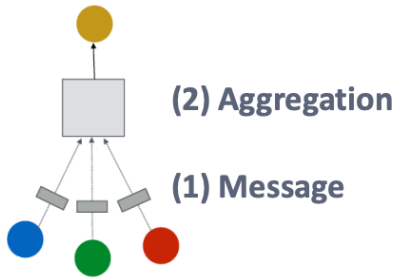
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- **How to write this as Message + Aggregation?**

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

Message

Aggregation



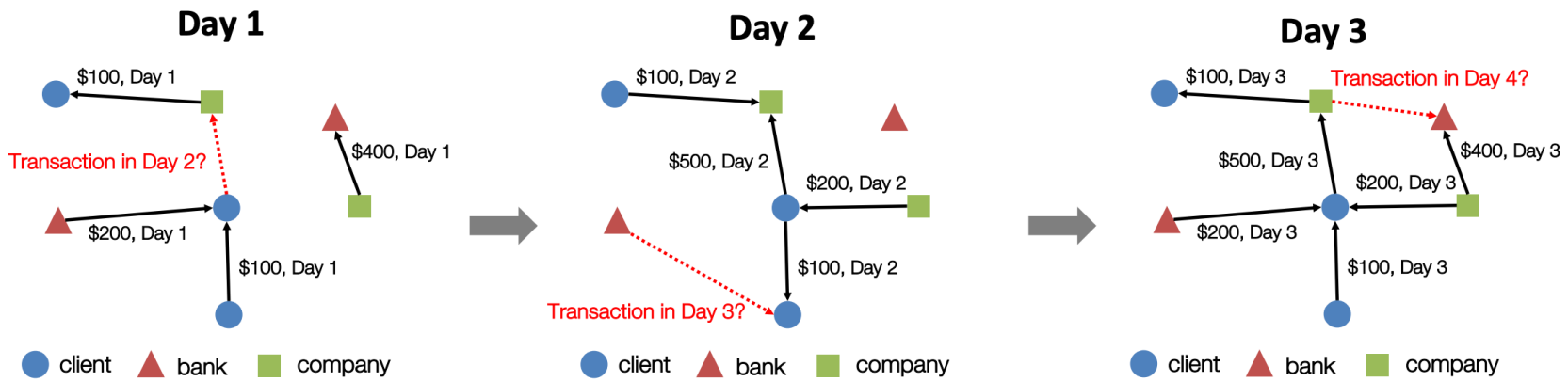
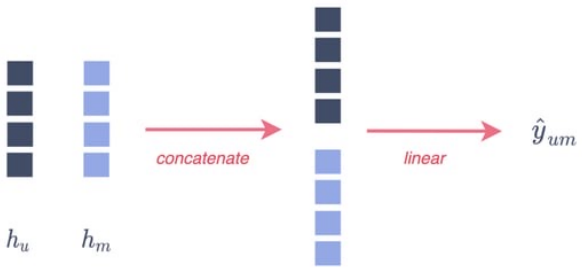


Figure 1: Example ROLAND use case for future link prediction on a dynamic transaction graph. We use information up to time t to predict potential edges at time $t + 1$.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$



Запрос	Ответы	Правильный ответ	Ранг	Обратный ранг
кочерга	кочерг, кочергей, **кочерёг**	кочерёг	3	1/3
попадья	попадь, **попадей**, попадьёв	попадей	2	1/2
турок	**турок**, турков, турчан	турок	1	1

MRR=(1/3 + 1/2 + 1)/3 = 11/18

ROLAND: Мотивация 1

T-GCN

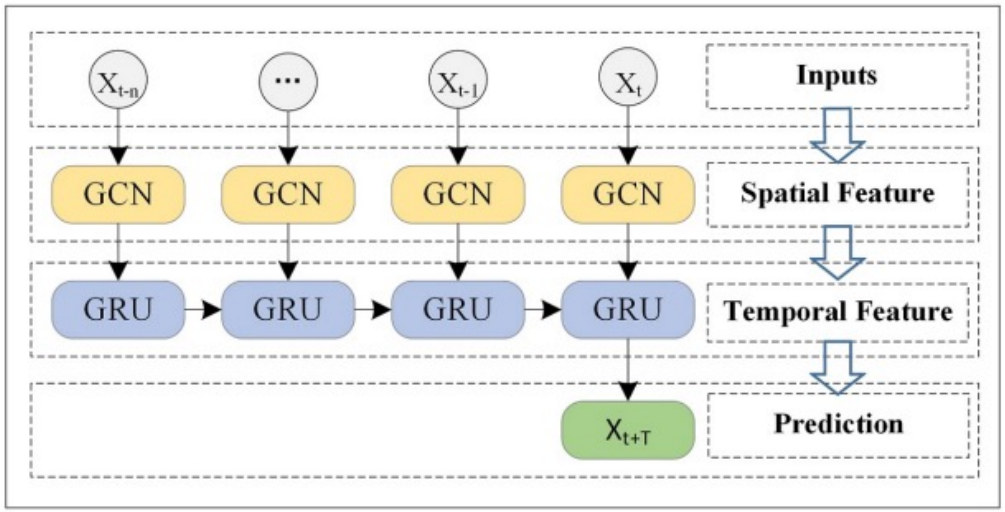
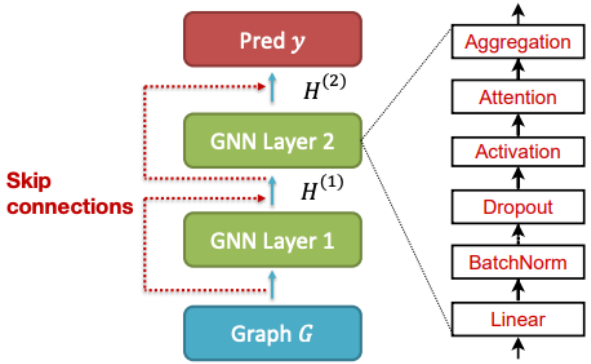


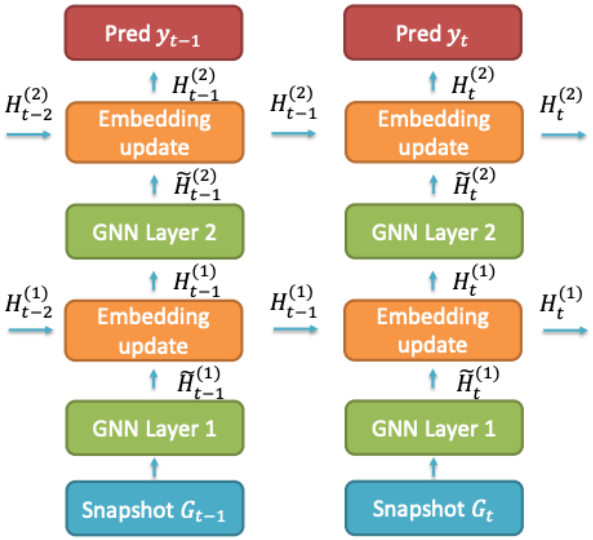
Fig. 3. Overview. We take the historical traffic information as input and obtain the finally prediction result through the Graph Convolution Network and the Gated Recurrent Units model.

https://arxiv.org/pdf/1811.05320.pdf

a) A static GNN with modern architectural design options



b) Extend static GNNs to dynamic GNNs



Hierarchical node state:
$H_t = \{H_t^{(1)}, \ldots, H_t^{(L)}\}$

Update

GRU

$$, H_t^{(l)} = \textsc{Gru}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)}),$$

Moving Avg

$$H_{t,v}^{(l)} = \kappa_{t,v} H_{t-1,v}^{(l)} + (1-\kappa_{t,v})\tilde{H}_{t,v}^{(l)}.$$

$$\kappa_{t,v} = \frac{\sum_{\tau=1}^{t-1} |E_\tau|}{\sum_{\tau=1}^{t-1} |E_\tau| + |E_t|} \in [0,1]$$

MLP:

$$H_t^{(l)} = \textsc{Mlp}(\textsc{Concat}(H_{t-1}^{(l)}, \tilde{H}_t^{(l-1)})).$$

Algorithm 1 ROLAND GNN forward computation

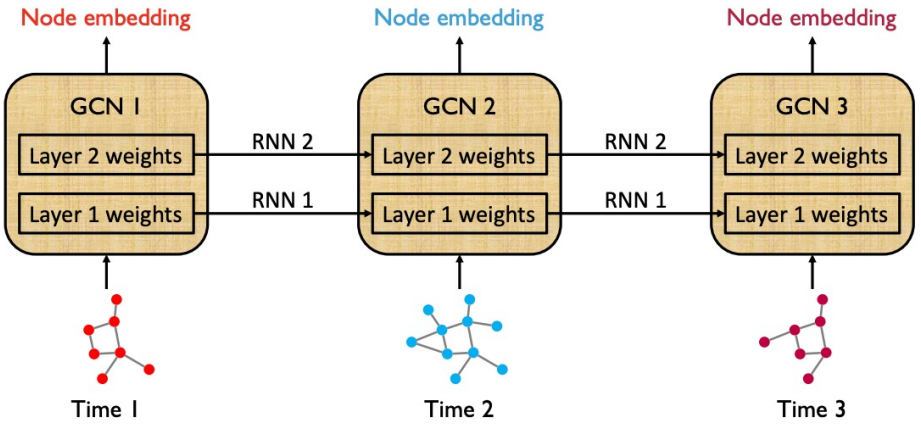
Input: Dynamic graph snapshot G_t, hierarchical node state H_{t-1}
Output: Prediction y_t, updated node state H_t

$H_t^{(0)} \leftarrow X_t$ {Initialize embedding from G_t}
for $l = 1, \ldots, L$ **do**
  $\tilde{H}_t^{(l)} = \textsc{Gnn}^{(l)}(H_t^{(l-1)})$ {Implemented as Equation (4)}
  $H_t^{(l)} = \textsc{Update}^{(l)}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)})$ {Equation (2)}
$y_t = \textsc{Mlp}(\textsc{Concat}(\mathbf{h}_{u,t}^{(L)}, \mathbf{h}_{v,t}^{(L)})), \forall (u,v) \in E$ {Equation (5)}

ROLAND: мотивация 2

$$\underbrace{W_t^{(l)}}_{\text{hidden state}} = \text{GRU}(\underbrace{H_t^{(l)}}_{\text{input}}, \underbrace{W_{t-1}^{(l)}}_{\text{hidden state}}),$$

GCN weights: $W_t^{(l)}$; node embeddings: $H_t^{(l)}$; GCN weights: $W_{t-1}^{(l)}$



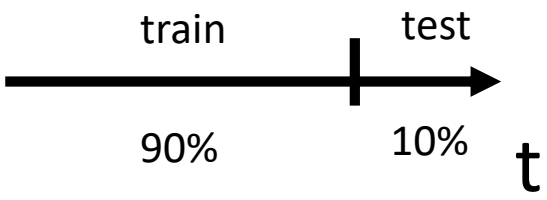
GNN architecture. EvolveGCN [30] proposes to recurrently updates the GNN weights; in contrast, ROLAND recurrently updates hierarchical node embeddings. While the EvolveGCN approach is memory efficient, the explicit historical information is lost (e.g., past transaction information). Our experiments show that EvolveGCN delivers undesirable performance when the number of graph snapshots is large. Moreover, few existing dynamic GNN works have

Figure 1: Schematic illustration of EvolveGCN. The RNN means a recurrent architecture in general (e.g., GRU, LSTM). We suggest two options to evolve the GCN weights, treating them with different roles in the RNN. See the EvolveGCN-H version and EvolveGCN-O version in Figure 2.

https://arxiv.org/abs/1902.10191

https://github.com/maqy1995/EvolveGCN-DGL/blob/master/model.py

Fixed Split



However, the data distribution of dynamic graphs is constantly evolving in the real world. For example, the number of purchase transactions made in the holiday season is higher than usual. Therefore, evaluating models solely based on edges from the last 10% of snapshots can provide misleading results.

Live-Update Evaluation

Algorithm 2 ROLAND live-update evaluation

Input: Dynamic graph $\mathcal{G} = \{G_1, \ldots, G_T\}$, link prediction labels $y_1, \ldots, y_T$, number of snapshots T, GNN$(\cdot)$ defined in Algorithm 1
Output: Performance MRR, model GNN

Initialize hierarchical node state H_0
for $t = 2, \ldots, T$ **do**
  Collect link prediction labels $y_{t-1} = y_{t-1}^{(train)} \cup y_{t-1}^{(val)}, y_t$
  while $\text{MRR}_{t-1}^{(val)}$ is increasing **do**
    $H_{t-1}, \hat{y}_{t-1} \leftarrow \text{GNN}(G_{t-1}, H_{t-2})$, $\hat{y}_{t-1} = \hat{y}_{t-1}^{(train)} \cup \hat{y}_{t-1}^{(val)}$
    Update GNN via backprop based on $\hat{y}_{t-1}^{(train)}, y_{t-1}^{(train)}$
    $\text{MRR}_{t-1}^{(val)} \leftarrow \text{EVALUATE}(\hat{y}_{t-1}^{(val)}, y_{t-1}^{(val)})$
  $H_t, \hat{y}_t \leftarrow \text{GNN}(G_t, H_{t-1})$
  $\text{MRR}_t \leftarrow \text{EVALUATE}(\hat{y}_t, y_t)$
$\text{MRR} = \sum_{t=2}^{T} \text{MRR}_t / (T-1)$

Live-Update Evaluation

Algorithm 2 ROLAND live-update evaluation

Input: Dynamic graph $\mathcal{G} = \{G_1, \ldots, G_T\}$, link prediction labels $y_1, \ldots, y_T$, number of snapshots T, GNN$(\cdot)$ defined in Algorithm 1
Output: Performance MRR, model GNN

Initialize hierarchical node state H_0
for $t = 2, \ldots, T$ **do**
  Collect link prediction labels $y_{t-1} = y_{t-1}^{(train)} \cup y_{t-1}^{(val)}, y_t$
  while $\text{MRR}_{t-1}^{(val)}$ is increasing **do**
    $H_{t-1}, \hat{y}_{t-1} \leftarrow \text{GNN}(G_{t-1}, H_{t-2})$, $\hat{y}_{t-1} = \hat{y}_{t-1}^{(train)} \cup \hat{y}_{t-1}^{(val)}$
    Update GNN via backprop based on $\hat{y}_{t-1}^{(train)}, y_{t-1}^{(train)}$
    $\text{MRR}_{t-1}^{(val)} \leftarrow \text{EVALUATE}(\hat{y}_{t-1}^{(val)}, y_{t-1}^{(val)})$
  $H_t, \hat{y}_t \leftarrow \text{GNN}(G_t, H_{t-1})$
  $\text{MRR}_t \leftarrow \text{EVALUATE}(\hat{y}_t, y_t)$
$\text{MRR} = \sum_{t=2}^{T} \text{MRR}_t / (T-1)$

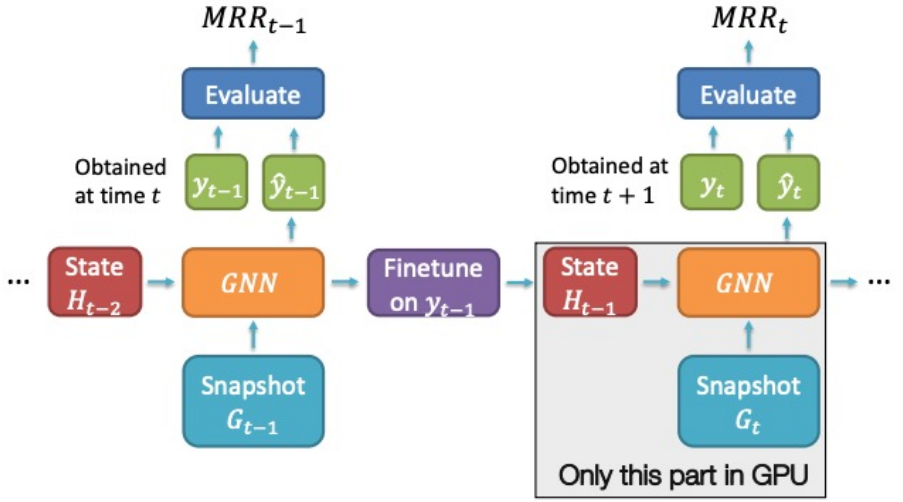
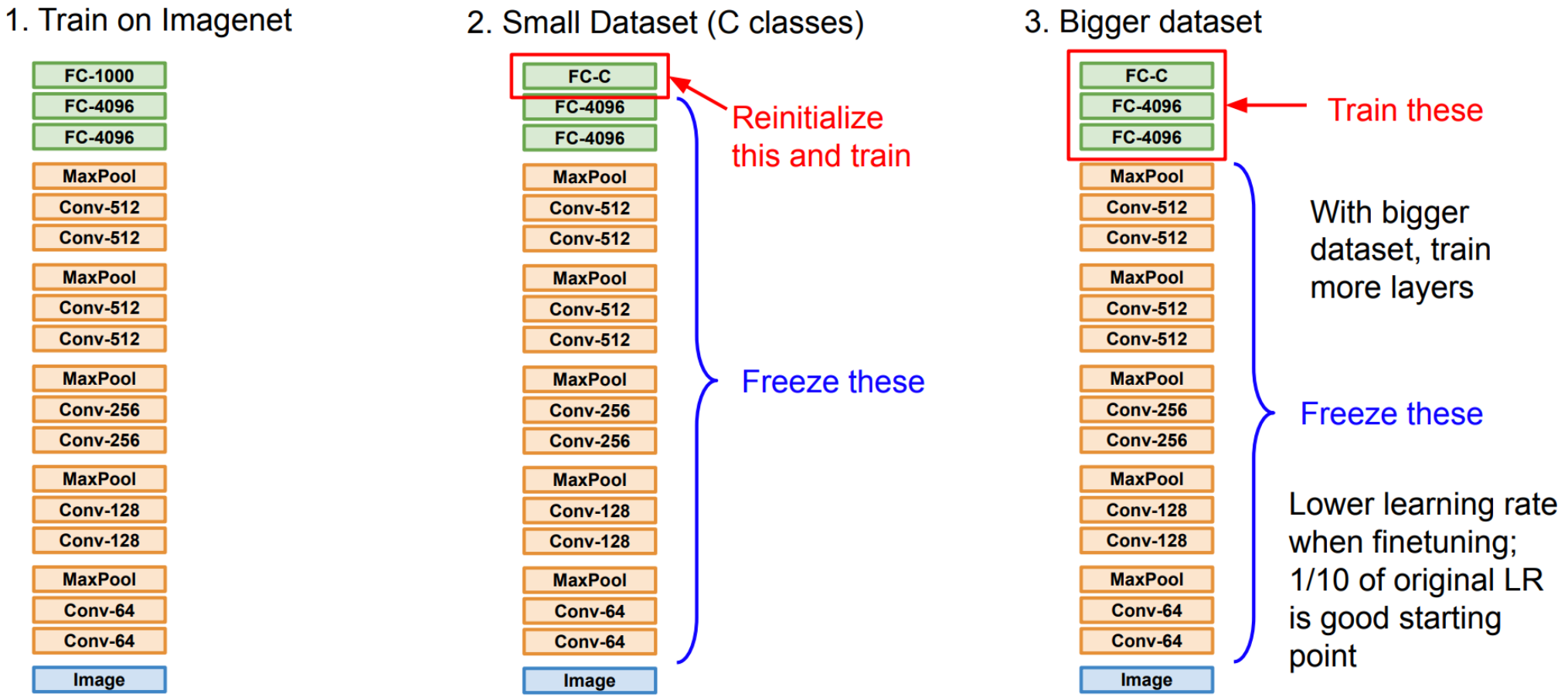


Figure 3: ROLAND live-update evaluation. ROLAND fine-tunes GNN with y_{t-1} and updates node embeddings H_{t-1} for the next prediction task. After obtaining labels y_t, ROLAND evaluates GNN's predictive performance based on historical state H_{t-1} and current snapshot G_t.

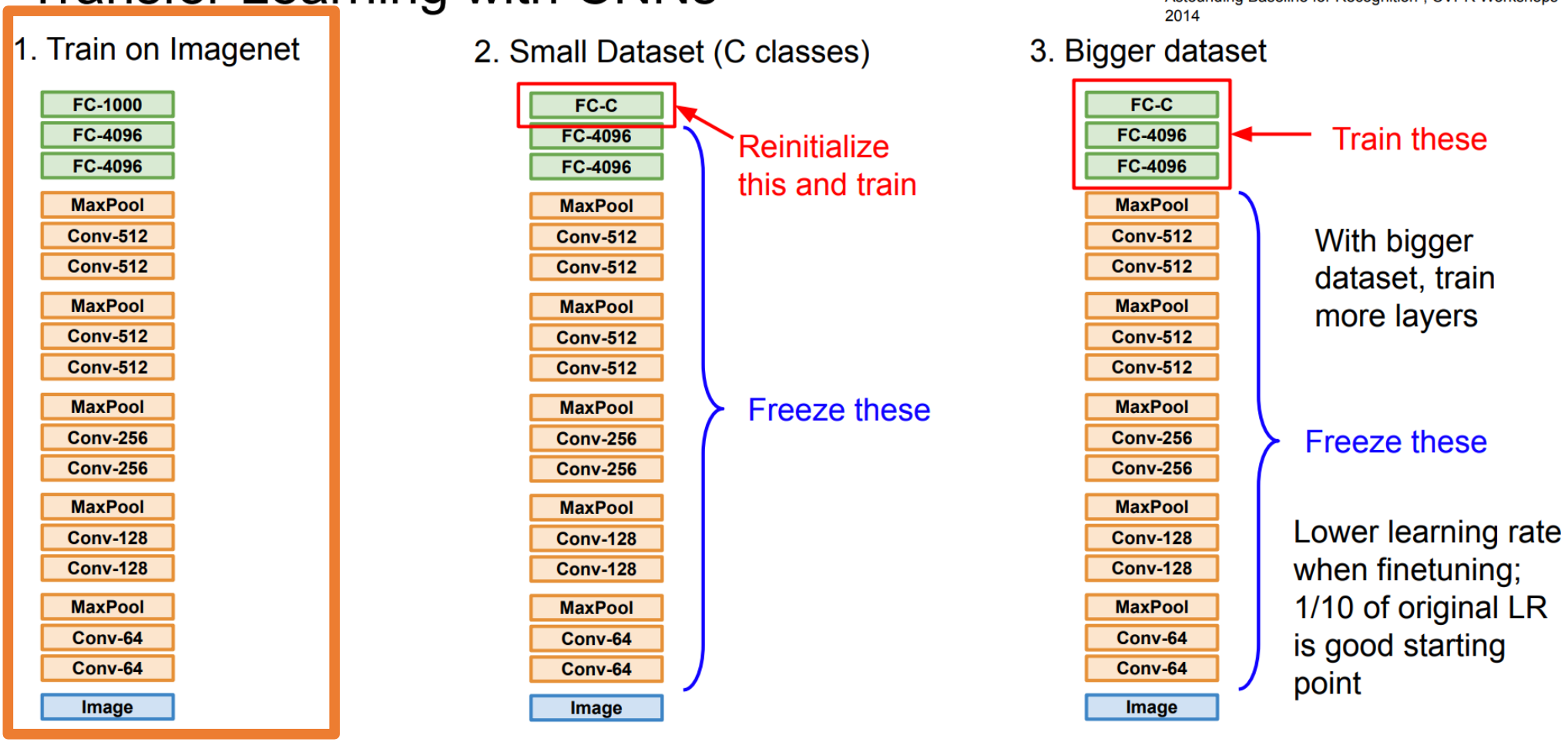
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



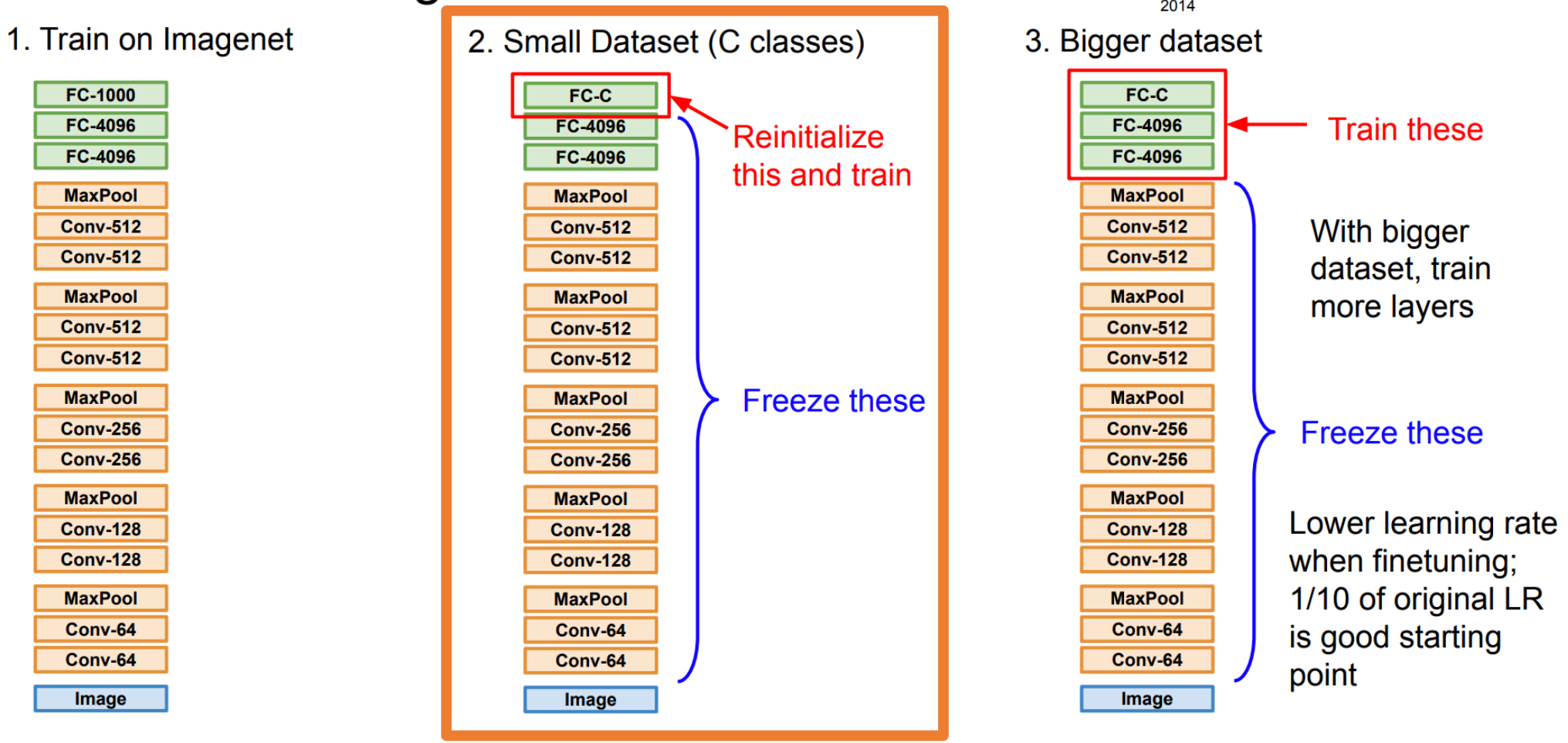
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



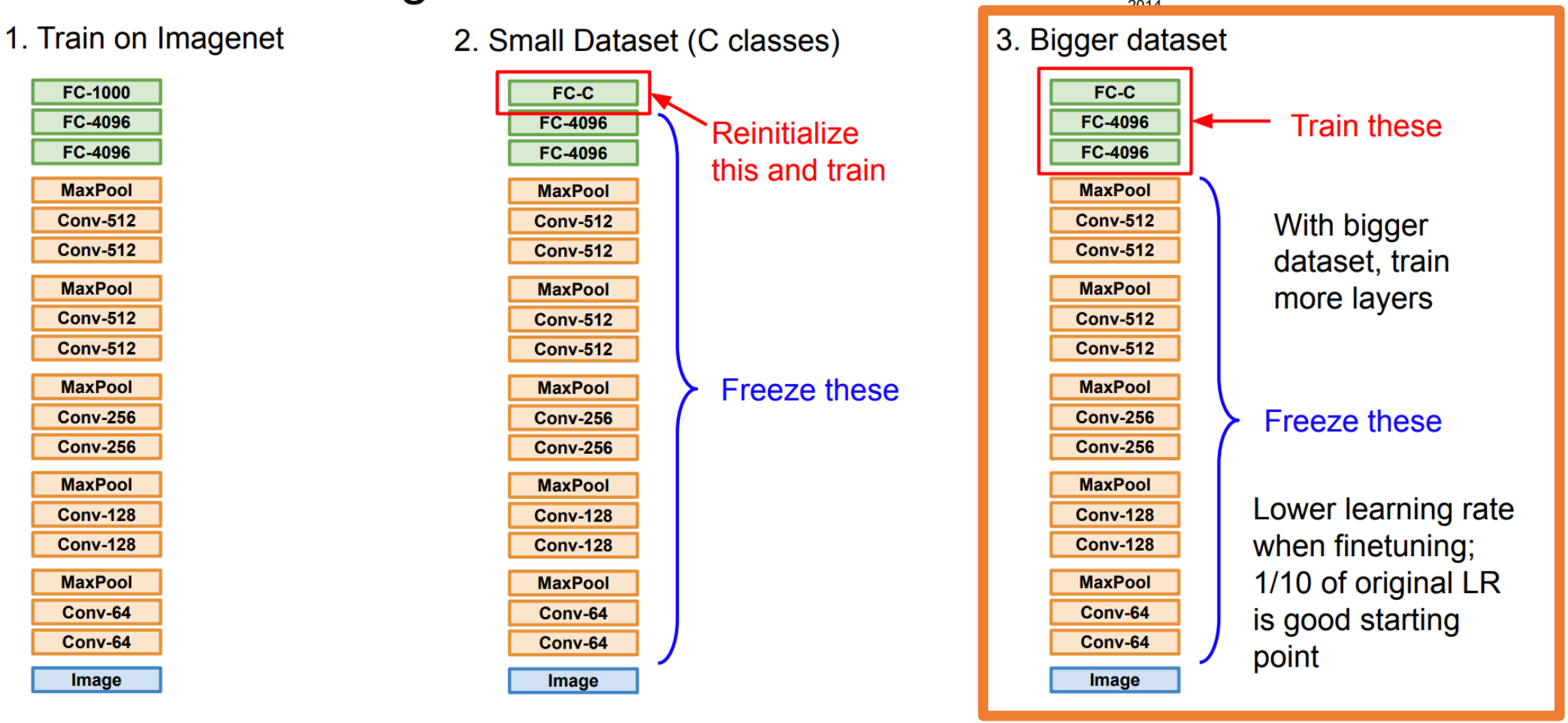
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



From Transfer Learning to Meta-Learning

Transfer learning: Initialize model. Hope that it helps the target task.

Meta-learning: Can we explicitly ***optimize*** for transferability?

Given a set of training tasks, can we optimize for the ability to learn these tasks quickly?
so that we can learn ***new*** tasks quickly too

Learning a task: $\mathcal{D}_i^{tr} \rightarrow \theta$

Can we optimize this function?

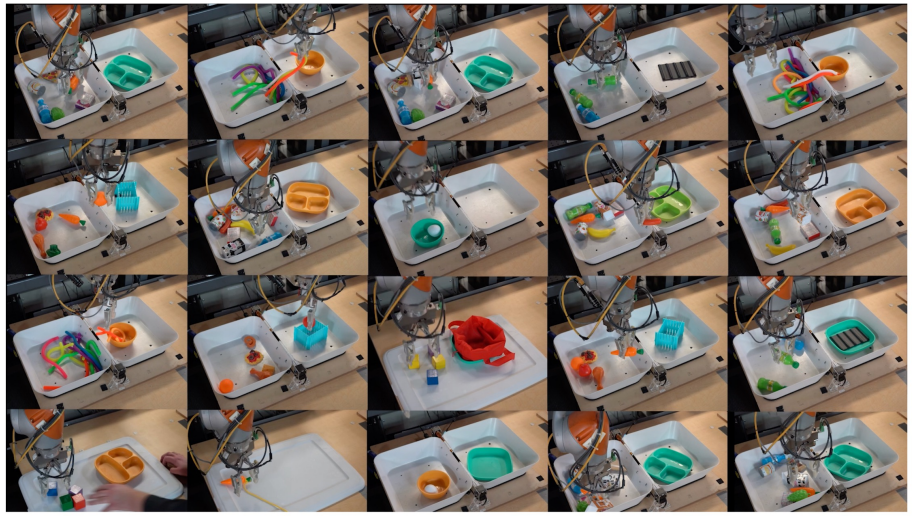
(for small $\mathcal{D}_i^{tr}$)

Meta learning[1][2] is a subfield of machine learning where automatic learning algorithms are applied to metadata about machine learning experiments. (Wikipedia)

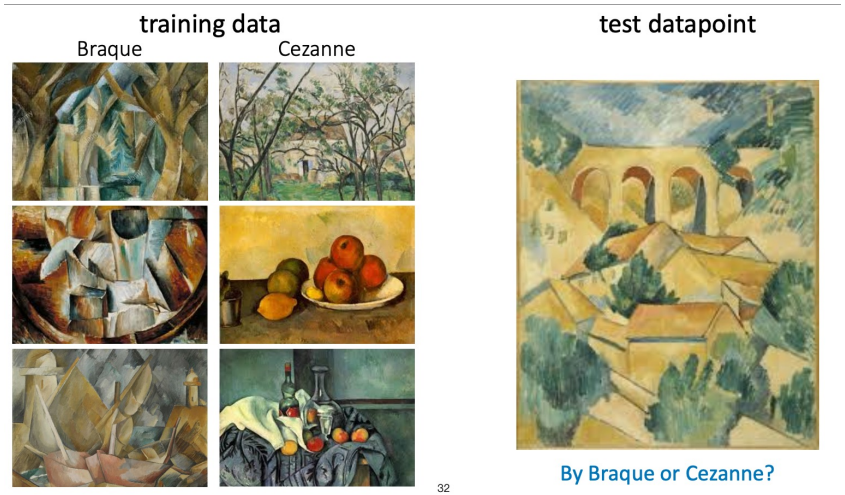
https://cs330.stanford.edu/lecture_slides/cs330_transfer_meta_learning.pdf

Meta Learning

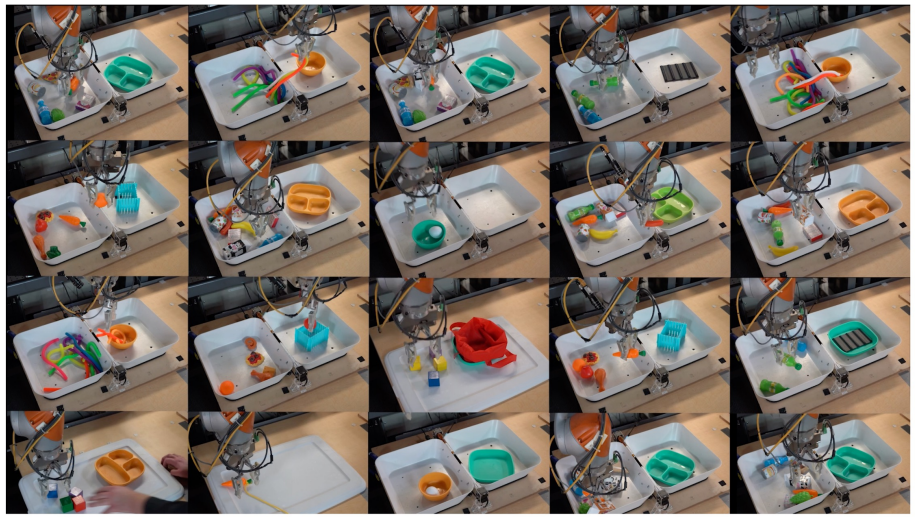
Meta Reinforcement Learning

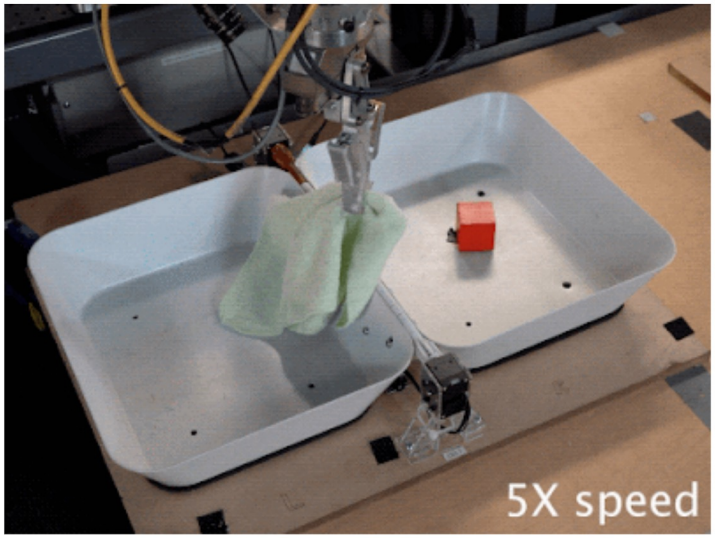


Few Shot Learning



Meta Reinforcement Learning



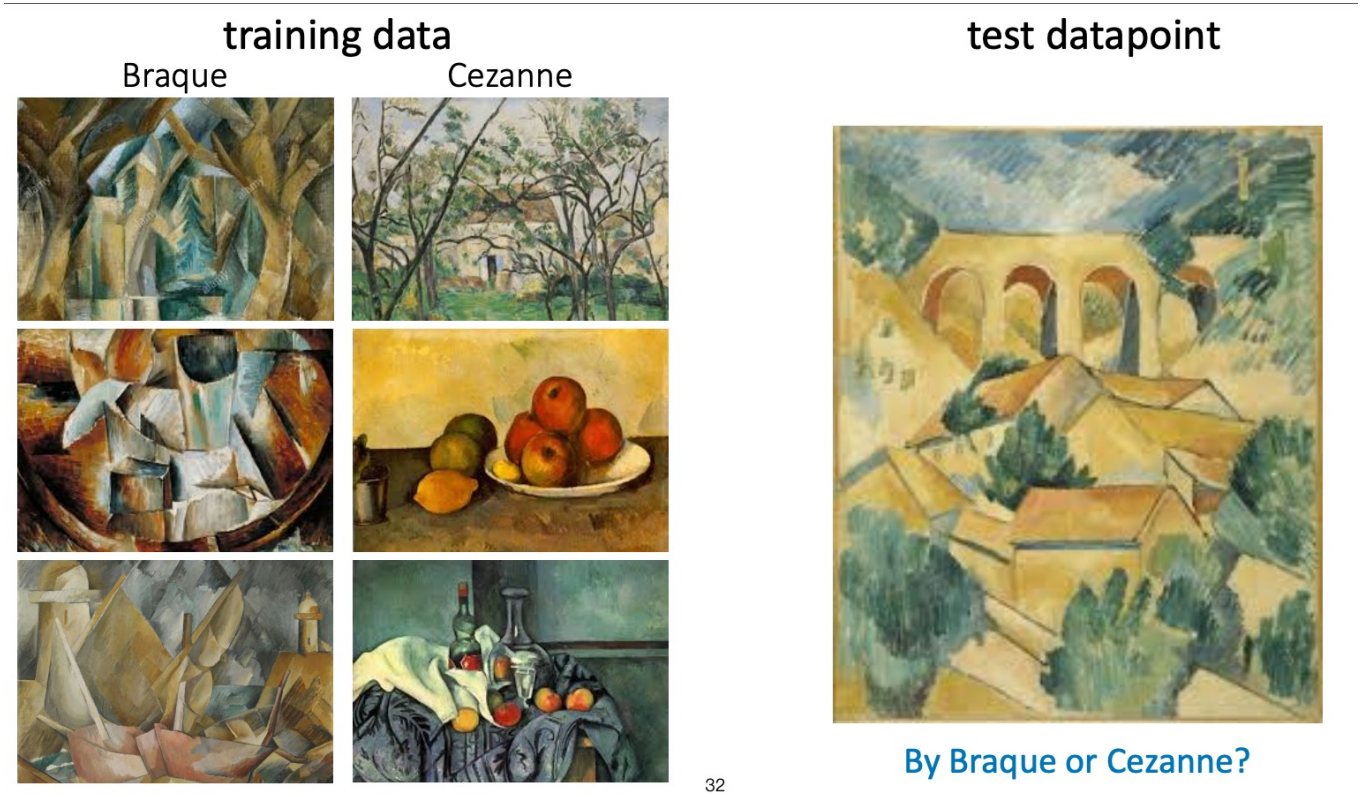


Обучали роборуки на разных задачах,
они усвоили общие концепции: захватывать,
следить за чем-либо, размещать, выравнивать,
переставлять.

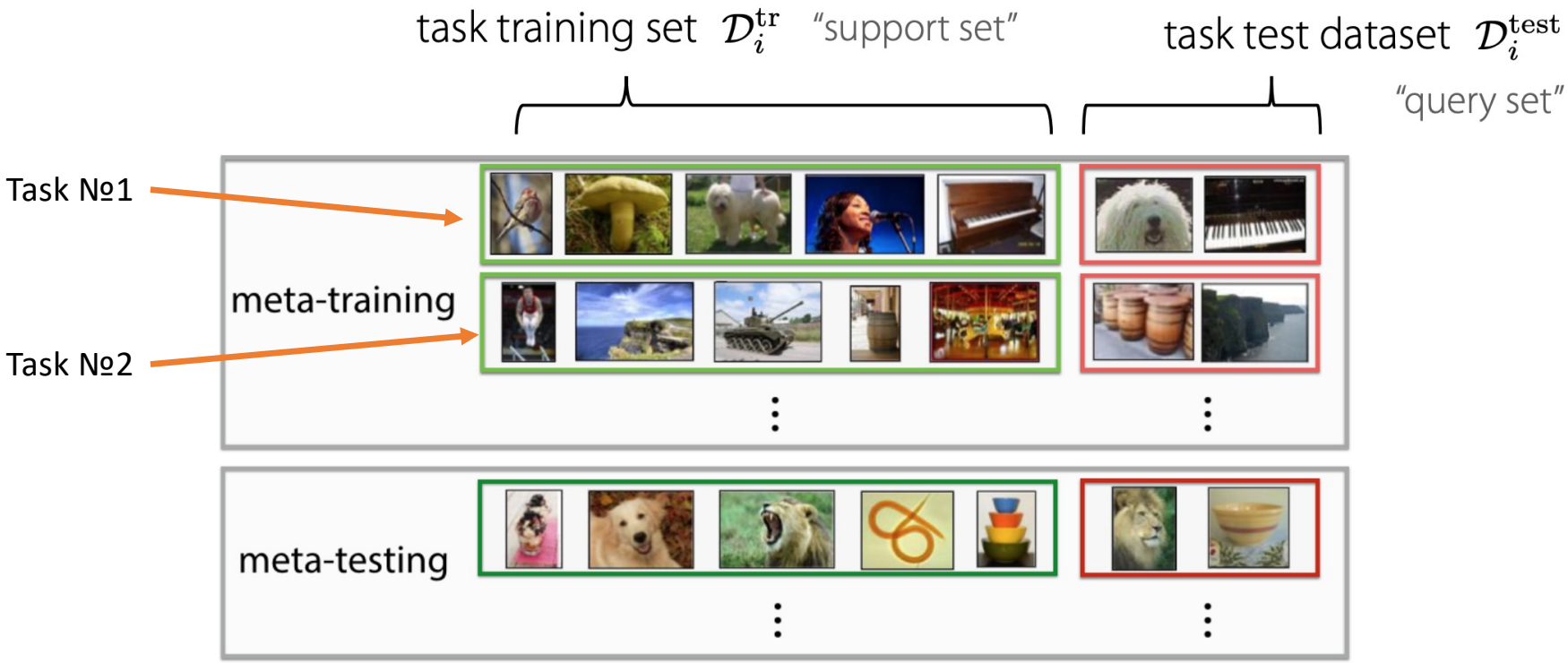
Перенесли опыт всех роборук в одну
роборуку и это ускорило обучение на
совершенно новой задаче:
накрывать тряпочкой.

https://ai.googleblog.com/2021/04/multi-task-robotic-reinforcement.html

Few Shot Learning



2-way 3-shot Classification



k-shot learning: learning with **k** examples per class
(or **k** examples total for regression)

N-way classification: choosing between N classes

Question: What are k and N for the above example?

Fine-tuning
[test-time]

$$\phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

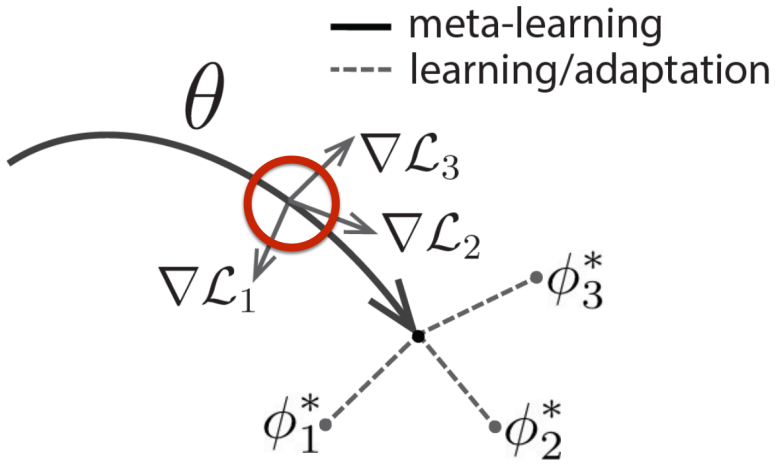
pre-trained parameters

training data for new task

Meta-learning $\min_\theta \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{ts}})$

θ parameter vector being meta-learned

ϕ_i^* optimal parameter vector for task i



Fine-tuning [test-time]

$$\phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

pre-trained parameters

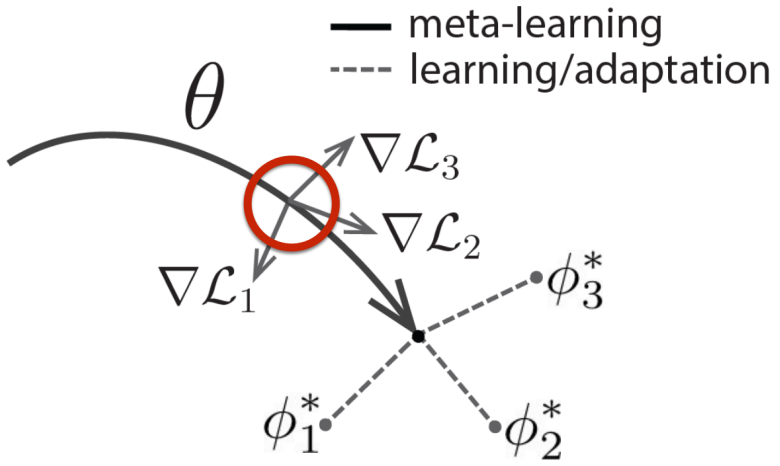
training data for new task

Meta-learning

$$\min_\theta \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{ts}})$$

θ parameter vector being meta-learned

ϕ_i^* optimal parameter vector for task i



Fine-tuning [test-time]

$$\phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

pre-trained parameters

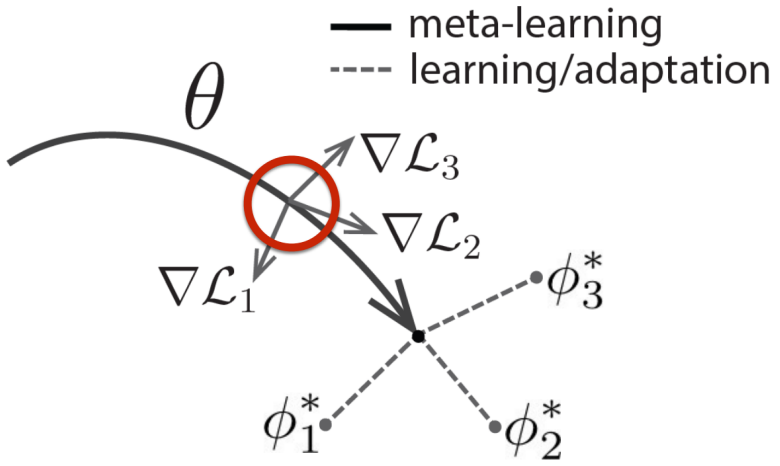
training data for new task

Meta-learning

$$\min_\theta \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{ts}})$$

θ parameter vector being meta-learned

ϕ_i^* optimal parameter vector for task i



MAML

1. Sample task τ_i
2. Sample disjoint datasets D_i^s, D_i^q from D_i
3. Optimize $\varphi_i \leftarrow \theta - \alpha \nabla_\theta L(\theta, D_i^s)$
4. Update θ using $\nabla_\theta L(\varphi_i, D_i^q)$

A significant computational expense in MAML comes from the use of second derivatives when backpropagating the meta-gradient through the gradient operator in the meta-objective (see Equation (1)). On MiniImagenet,

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \qquad (1)$$

Reptile

1. Sample task τ_i
2. Optimize $\varphi_i \leftarrow U(\theta, k, D_i)$
3. Update θ: $\theta \leftarrow \theta + \alpha\,(\varphi_i - \theta)$

$U(\theta, k, D_i)$ – Perform k steps of SGD or Adam on task τ_i (so data D_i), starting with parameters θ, resulting in parameters φ_i

Algorithm 1 Reptile (serial version)

Initialize ϕ, the vector of initial parameters
for iteration $= 1, 2, \ldots$ **do**
  Sample task τ, corresponding to loss L_τ on weight vectors $\widetilde{\phi}$
  Compute $\widetilde{\phi} = U_\tau^k(\phi)$, denoting k steps of SGD or Adam
  Update $\phi \leftarrow \phi + \epsilon(\widetilde{\phi} - \phi)$
end for

https://arxiv.org/abs/1703.03400
Reptile: a Scalable Metalearning Algorithm. Alex Nichol and John Schulman.
https://arxiv.org/abs/1803.02999
https://habr.com/ru/companies/sberbank/articles/649609/

ROLAND: Training

Algorithm 3 ROLAND training algorithm

Input: Graph snapshot G_t, link prediction label y_t, hierarchical node state H_{t-1}, smoothing factor α, meta-model $\text{GNN}^{(meta)}$

Output: Model GNN, updated meta-model $\text{GNN}^{(meta)}$

1: $\text{GNN} \leftarrow \text{GNN}^{(meta)}$
2: Move GNN, G_t, H_{t-1} to GPU
3: **while** $\text{MRR}_t^{(val)}$ is increasing **do**
4: $\quad H_t, \hat{y}_t \leftarrow \text{GNN}(G_t, H_{t-1}), \hat{y}_t = \hat{y}_t^{(train)} \cup \hat{y}_t^{(val)}$
5: $\quad$ Update GNN via backprop based on $\hat{y}_t^{(train)}, y_t^{(train)}$
6: $\quad \text{MRR}_t^{(val)} \leftarrow \text{EVALUATE}(\hat{y}_t^{(val)}, y_t^{(val)})$
7: Remove GNN, G_t, H_{t-1} from GPU
8: $\text{GNN}^{(meta)} \leftarrow (1-\alpha)\text{GNN}^{(meta)} + \alpha\text{GNN}$

Reptile

1. Sample task τ_i
2. Optimize $\varphi_i \leftarrow U(\theta, k, D_i)$
3. Update θ: $\theta \leftarrow \theta + \alpha\,(\varphi_i - \theta)$

$U(\theta, k, D_i)$ – Perform k steps of SGD or Adam on task τ_i (so data D_i), starting with parameters θ, resulting in parameters φ_i

Table 3: Results in the live-update settings. We run experiments (except for BSI-ZK) with 3 random seeds to report the average and standard deviation of MRRs. We attempted each experiment five times before concluding the out-of-memory (OOM) error. The top part of the results consists of baseline models trained using BPTT, the middle and bottom portions summarize the performance of baselines, and our models using ROLAND incremental training.

	BSI-ZK	AS-733	Reddit-Title	Reddit-Body	BSI-SVT	UCI-Message	Bitcoin-OTC	Bitcoin-Alpha
Baseline Models with standard training								
EvolveGCN-H	N/A, OOM	N/A, OOM	N/A, OOM	**0.148 ± 0.013**	**0.031 ± 0.016**	0.061 ± 0.040	0.067 ± 0.035	0.079 ± 0.032
EvolveGCN-O	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.015 ± 0.006	0.071 ± 0.009	0.085 ± 0.022	0.071 ± 0.025
GCRN-GRU	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.080 ± 0.012	N/A, OOM	N/A, OOM
GCRN-LSTM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	**0.083 ± 0.001**	N/A, OOM	N/A, OOM
GCRN-Baseline	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.069 ± 0.004	**0.152 ± 0.011**	**0.141 ± 0.005**
TGCN	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.054 ± 0.024	0.128 ± 0.049	0.088 ± 0.038
Baseline Models with ROLAND Training								
EvolveGCN-H	N/A, OOM	0.251 ± 0.079	0.165 ± 0.026	0.102 ± 0.010	0.032 ± 0.008	0.057 ± 0.012	0.076 ± 0.022	0.054 ± 0.015
EvolveGCN-O	0.396	0.163 ± 0.002	0.047 ± 0.004	0.033 ± 0.001	0.018 ± 0.003	0.066 ± 0.012	0.032 ± 0.004	0.034 ± 0.002
GCRN-GRU	N/A, OOM	**0.344 ± 0.001**	0.338 ± 0.006	0.217 ± 0.004	0.050 ± 0.004	0.089 ± 0.004	0.173 ± 0.003	0.140 ± 0.004
GCRN-LSTM	N/A, OOM	0.341 ± 0.001	0.344 ± 0.005	0.216 ± 0.000	0.051 ± 0.002	0.091 ± 0.010	0.174 ± 0.004	**0.146 ± 0.005**
GCRN-Baseline	0.754	0.336 ± 0.002	0.351 ± 0.001	0.218 ± 0.002	0.054 ± 0.002	**0.095 ± 0.008**	**0.183 ± 0.002**	0.145 ± 0.003
TGCN	**0.831**	0.343 ± 0.002	**0.391 ± 0.004**	**0.251 ± 0.001**	**0.157 ± 0.004**	0.080 ± 0.015	0.083 ± 0.011	0.069 ± 0.013
ROLAND results								
Moving Average	0.819	0.309 ± 0.011	0.362 ± 0.007	0.289 ± 0.038	0.177 ± 0.006	0.075 ± 0.006	0.120 ± 0.002	0.0962 ± 0.010
MLP-Update	0.834	0.329 ± 0.021	0.395 ± 0.006	0.291 ± 0.008	**0.217 ± 0.003**	0.103 ± 0.010	0.154 ± 0.010	0.148 ± 0.012
GRU-Update	**0.851**	**0.340** ± 0.001	**0.425** ± 0.015	**0.362** ± 0.002	0.205 ± 0.014	**0.112** ± 0.008	**0.194** ± 0.004	**0.157** ± 0.007
Improvement over the best baseline	2.40%	-1.16%	8.70%	44.22%	38.21%	17.89%	6.01%	7.53%

Table 4: Ablation study on the effectiveness of meta-learning. Except for the BSI-ZK dataset, the results are averaged over 3 random seeds. "Gain" refers to the MRR improvement from the best meta-learning setting over the non-meta-learning setting.

Model	BSI-ZK		AS-733		Reddit-Title		Reddit-Body		BSI-SVT		UCI-Message		Bitcoin-OTC		Bitcoin-Alpha		Average
	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	α	Gain	
Moving Average	0.5	0.99%	0.4	4.94%	0.7	2.68%	0.5	8.52%	0.5	3.27%	0.7	23.73%	0.9	7.45%	0.6	6.94%	7.33%
MLP-Update	0.5	4.04%	0.9	18.05%	0.7	1.09%	0.4	2.51%	0.4	11.40%	0.2	27.73%	0.3	33.76%	0.6	6.98%	13.19%
GRU-Update	0.7	0.56%	0.5	5.15%	0.1	2.97%	0.5	8.18%	0.8	2.10%	0.5	0.17%	0.9	2.82%	0.8	0.77%	2.84%

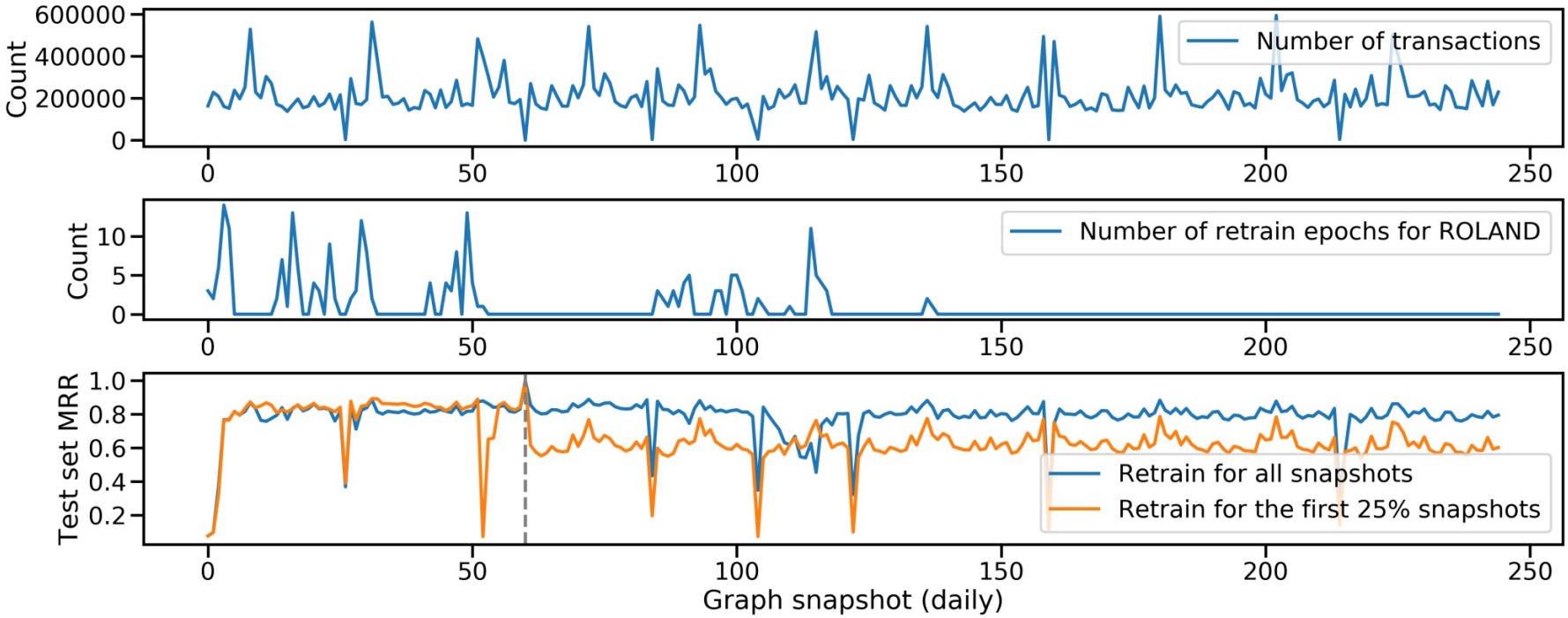


Figure 5: Effectiveness of ROLAND model retraining on BSI-ZK dataset. Top: number of transactions in each daily graph snapshot. Middle: number of training epochs before early stopping. Bottom: ROLAND's performance on the test set. We compare the option of retraining for all snapshots (blue curve), with a baseline that stopping retraining after training the first 25% snapshots (organ curve).

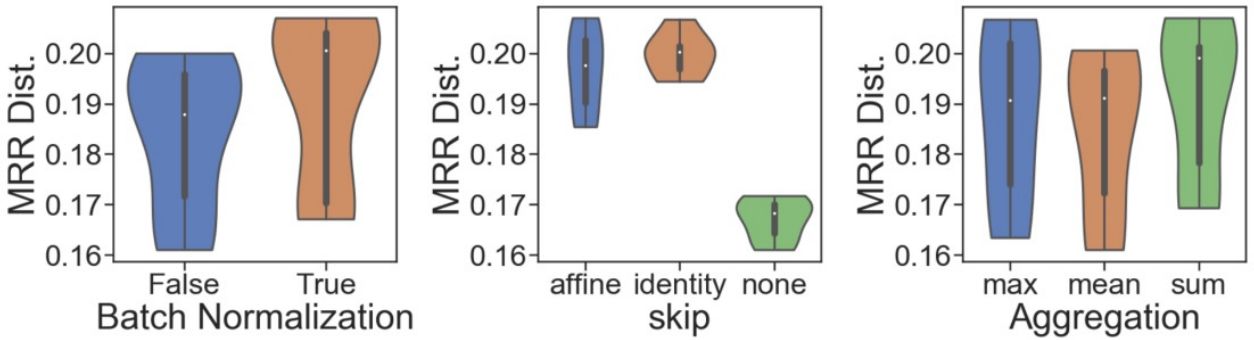


Figure 4: Effectiveness of ROLAND architectural design. We run experiments with different GNN models and analyze the effect of differ design dimensions. We show the MRR distribution of all the models under different design options. Results show that batch normalization, skip-connection and max aggregation are desirable for GNN architectural design.

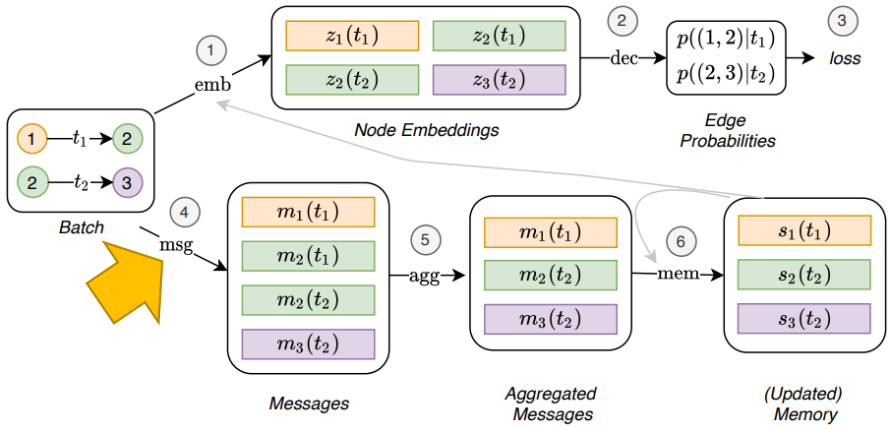
Материалы

- https://cs330.stanford.edu/
- LEARNING-TO-LEARN PERSONALISED HUMAN AC- TIVITY RECOGNITION MODELS , Anjana Wijekoon, Nirmalie Wiratunga
- https://research.samsung.com/blog/Meta-Learning-in-Neural-Networks
- https://github.com/sudharsan13296/Hands-On-Meta-Learning-With-Python
- https://science.sciencemag.org/content/350/6266/1332
- https://arxiv.org/abs/1605.06065
- Jurgen Schmidthuber, Simple methods of meta learning
- Reptile: a Scalable Metalearning Algorithm, Alex Nichol and John Schulman, OpenAI
- https://arxiv.org/abs/2203.14883
- ROLAND: Graph Learning Framework for Dynamic Graphs
- https://github.com/twitter-research/tgn

Спасибо за внимание!

Appendix A

Dynamic GNN : TGN



$$\mathbf{m}_i(t) = \mathrm{msg}_\mathrm{s}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)\right), \qquad \mathbf{m}_j(t) = \mathrm{msg}_\mathrm{d}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)\right) \quad (1)$$

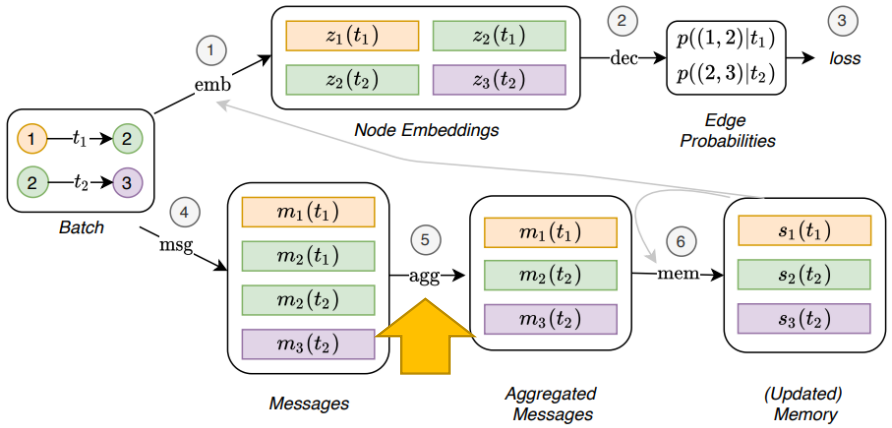
$$\mathbf{m}_i(t) = \mathrm{msg}_\mathrm{n}\left(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\right). \quad (2)$$

$$\bar{\mathbf{m}}_i(t) = \mathrm{agg}\left(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\right). \quad (3)$$

$$\mathbf{s}_i(t) = \mathrm{mem}\left(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\right). \quad (4)$$

$$\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right), \quad (5)$$

Dynamic GNN: TGN



$$\mathbf{m}_i(t) = \mathrm{msg}_\mathrm{s}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)\right), \qquad \mathbf{m}_j(t) = \mathrm{msg}_\mathrm{d}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)\right)$$

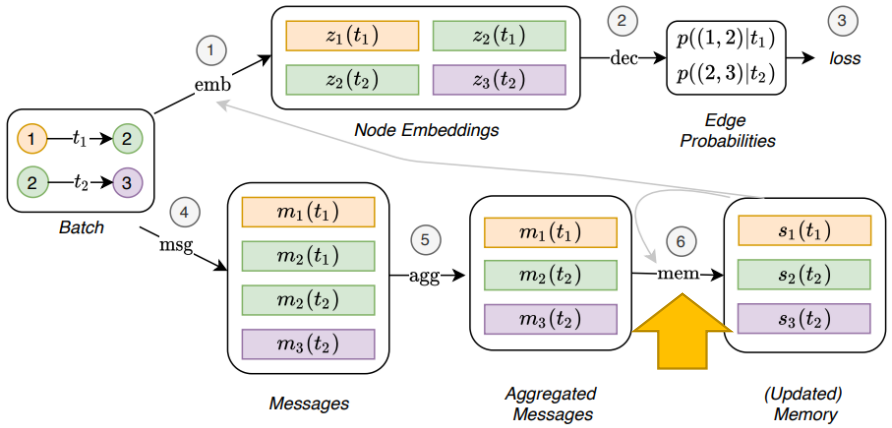
$$\mathbf{m}_i(t) = \mathrm{msg}_\mathrm{n}\left(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\right).$$

$$\bar{\mathbf{m}}_i(t) = \mathrm{agg}\left(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\right).$$

$$\mathbf{s}_i(t) = \mathrm{mem}\left(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\right).$$

$$\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right),$$

Dynamic GNN: TGN



$$\mathbf{m}_i(t) = \text{msg}_\text{s}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)\right), \qquad \mathbf{m}_j(t) = \text{msg}_\text{d}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)\right)$$

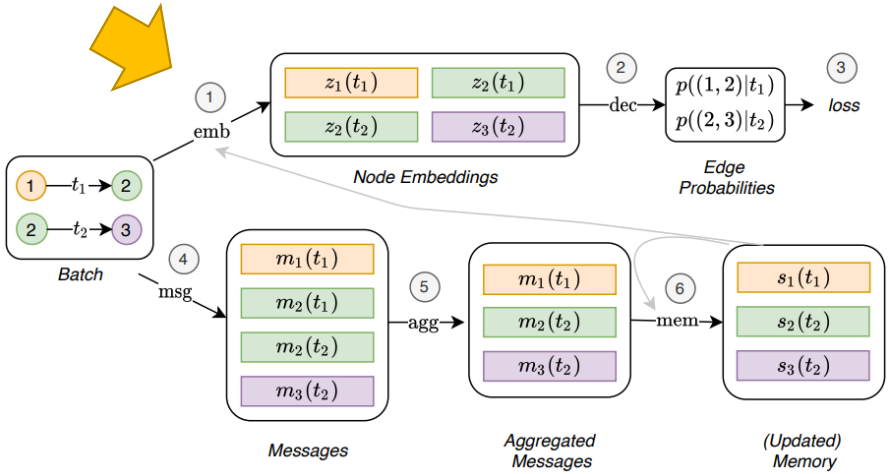
$$\mathbf{m}_i(t) = \text{msg}_\text{n}\left(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\right).$$

$$\bar{\mathbf{m}}_i(t) = \text{agg}\left(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\right).$$

$$\mathbf{s}_i(t) = \text{mem}\left(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\right).$$

$$\mathbf{z}_i(t) = \text{emb}(i,t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right),$$

Dynamic GNN: TGN



$$\mathbf{m}_i(t) = \text{msg}_\text{s}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)\right), \qquad \mathbf{m}_j(t) = \text{msg}_\text{d}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)\right)$$

$$\mathbf{m}_i(t) = \text{msg}_\text{n}\left(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\right).$$

$$\bar{\mathbf{m}}_i(t) = \text{agg}\left(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\right).$$

$$\mathbf{s}_i(t) = \text{mem}\left(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\right).$$

$$\mathbf{z}_i(t) = \text{emb}(i,t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right),$$