

Graph Representation Learning applied to Transaction Networks

Midterm Assessment

Claudio Moroni 

University of Turin



Outline

- 1** Transaction Networks
- 2** Overview So Far
- 3** Next Goals



Transaction Networks

1. Description
2. Associated Tasks



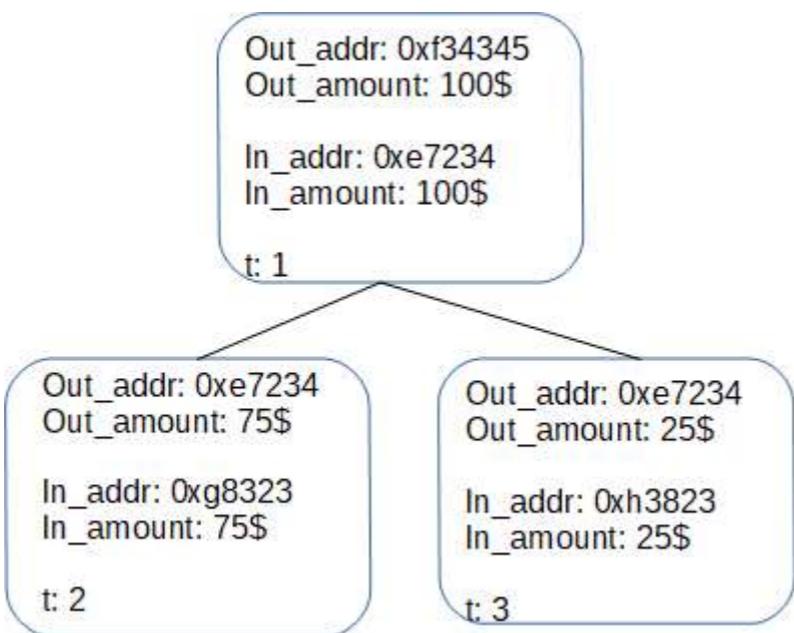
Transaction Networks

1. Description
2. Associated Tasks



Description

- They are **temporal, directed, attributed** network;
- Can be represented as either transactions (edges) between users (nodes) or as **blockchains**:



- There are **hints** that the latter representation could be “**better**”;



Transaction Networks

1. Description
2. Associated Tasks:
 - 2.1 Anti Money Laundering
 - 2.2 Preprocessing with Link Prediction
 - 2.3 Graph Generation



Anti Money Laundering

- **Money Laundering (MoL)**: concealing **illicit** sources of money via complex transaction patterns;
- Estimated yearly **200bn** in US only (2017), and **0.8-2.0tn** yearly worldwide¹, causing **immense** human suffering;
- Responsibility of **detecting** MoL is up to banks and crypto exchanges, who regularly fail paying high prices²;

1. [wikipedia](#), Altman et al. (2023)

2. [wikipedia](#)



Anti Money Laundering

- So far, rule-based, ML, DL and GDL techniques have been deployed¹;
- Many of the above attempts act on static versions of temporal transaction network
 ⇒ information loss;
- Time-aware attempt:
 - Use very complex, **not principled** architectures, sometimes outperformed by (much) simpler ones (TGNNS vs Random Forest);
 - They attempt to **classify** as licit/illicit either nodes or edges (no matching patterns);
 ⇒ We'll propose a framework that **maps** the temporal task to a static task, and then maps back the **predictions**. Advantages:
- Allows for **principled** architectures (GNNs);
- Provides a **framework** for all tasks on temporal graphs.

1. Jullum et al. (2020), Lorenz et al. (2021), Jensen and Iosifidis (2023), Weber et al. (2019), Liu et al. (2023), Johannessen and



Preprocessing with Link Prediction

- Transaction networks are often **incomplete** (interbank transactions, Crypto APIs);
- LP performed with GNNs can make up for it and **improve** downstream tasks¹;
- Transaction Networks are **directed**, and directed LP is problematic²;
- We propose to find **optimal training strategy** for directed LP (loss weighting, optimal fraction of random negative edges);
- We started this research line by **collaborating** with the author.

1. Lin et al. (2022), Li and He (2023)

2. Salha et al. (2019)



Overview So Far

1. Graph Neural Networks
2. Temporal Graph Neural Networks
3. Node Classification
4. Link Prediction



Overview So Far

1. Graph Representation Learning:
 - 1.1 Early Models
 - 1.2 Weisfeiler-Leman Kernel
 - 1.3 Graph Neural Networks (GNNs)
 - 1.4 Expressivity of GNNs
2. Temporal Graph Neural Networks
3. Node Classification
4. Link Prediction



Early Models

- Before Deep Learning, features were **human-engineered** (Machine Learning);
- Early works can be organized in node/edge/graph-level **features**;

Node-level:

- **Features:** Degree, Eigenvector, Betweenness, Closeness **centralities** & Clustering Coefficient (**CC**)/Graphlet Degree Vector (**GDV**)
- **RW-based:** PageRank, **DeepWalk**, **Node2Vec** (not inductive!);

Edge-level: Local/Gobal Neighborhood overlap (**Katz Index**)

Graph-level: (Causal) Anonymous Walks, Graphlet Kernel, **Weisfeiler-Leman Kernel**.



Weisfeiler-Leman Kernel #1

Def.(Graph Isomorphism): Two graphs (V_1, E_1) and (V_2, E_2) are **isomorphic** \iff

$$\exists \text{ bijection } f : V_1 \rightarrow V_2 | (u, v) \in E_1 \implies (f(u), f(v)) \in E_2$$

Obs. : Determining f is extremely **expensive** \implies need for approximate algorithms \implies **WL-kernel**



Weisfeiler-Leman Kernel #2

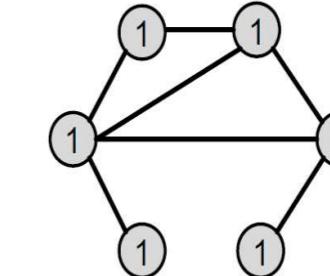
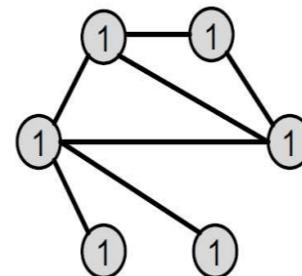
WL-Kernel:



1. Assign initial color $c_v^{(0)}$ to each node v ;

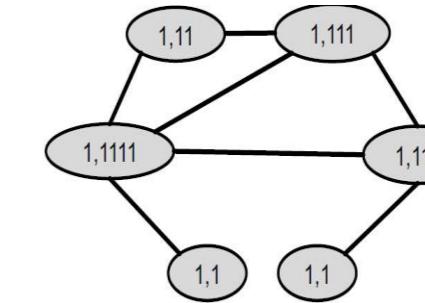
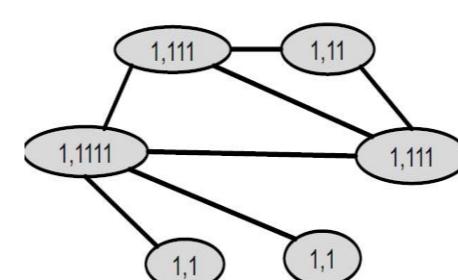
2. Iteratively refine colors via:

$$c_v^{(k+1)} = \text{HASH} \left(c_v^{(k)}, \{c_u^{(k)} | u \in N(v)\} \right)$$

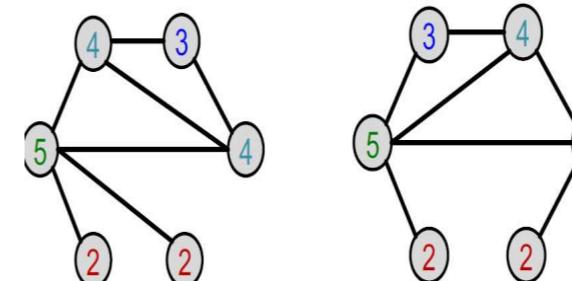


3. Terminate when colors **stabilize** (#nodes iterations maximum);

4. Assign to each graph the empirical color distribution (histogram)



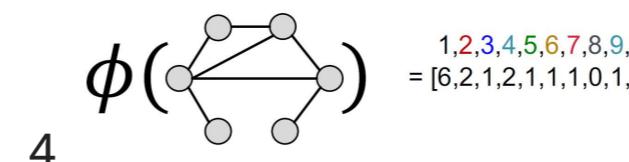
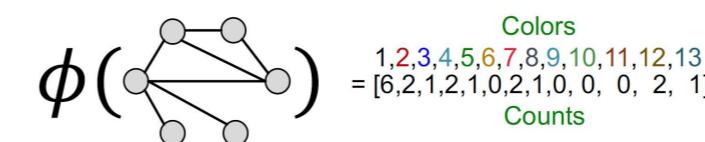
2.



Hash table

1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

3.

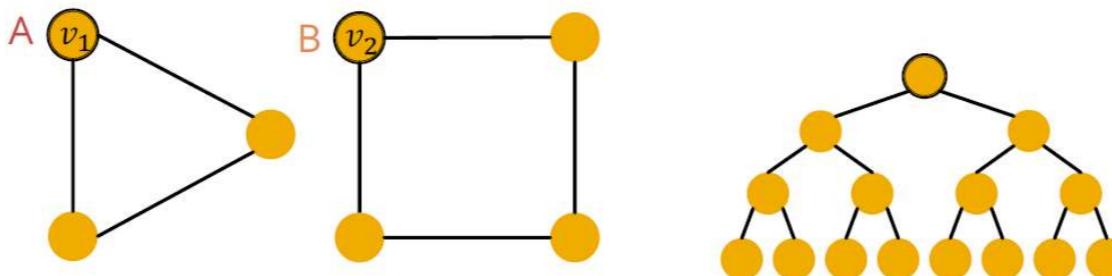


4

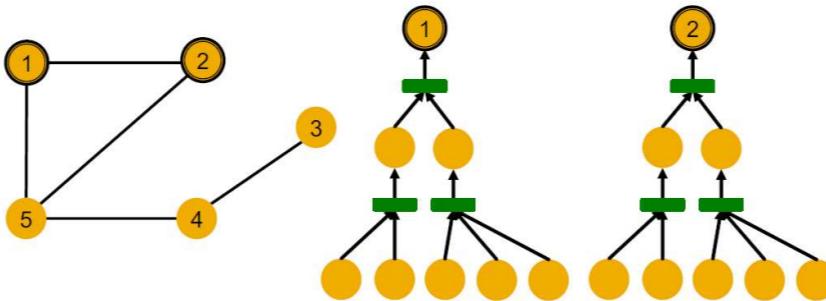


Weisfeiler-Leman Kernel #3

Obs. WL-Kernel cannot distinguish all non-isomorphic graphs e.g.:



And computes same color for different nodes:



Obs. : GNNs will be **differentiable** versions of *HASH* \implies less expressive.



Graph Neural Networks (GNNs) #1

Disadvantages of previous methods:

- Inherently **transductive**;
- Do not **incorporate** node/edge/graph features
- Computed node/edge/graph features are **task-independent**;

⇒ Need for Graph Deep Learning i.e. **Graph Neural Networks!**



Graph Neural Networks (GNNs) #1

Historical derivation of GNNs:

1. Simultaneous discretization of a Riemannian Manifold \mathcal{M} & Laplace-Beltrami operator yields a graph G and its laplacian $L = D - A^1$
2. $\hat{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - \hat{A}$'s eigenvectors U used to define the Graph Fourier Transform:

$$\mathcal{F}(\vec{x}) = U^T \vec{x}$$

3. \mathcal{F} used to define Graph Convolution:

$$\vec{g}_{\vec{\theta}} * \vec{x} = U \vec{g}_{\vec{\theta}} U^T \vec{x}$$



4. Interpret $\vec{g}_{\vec{\theta}}$ as function of \hat{L} 's eigenvalues $\hat{\Lambda}$, and approximate using Chebishev's polynomials:

$$\vec{g}_{\vec{\theta}} \approx \sum_{k=0}^K \theta_k T_k(\hat{\Lambda})$$

5. Truncate to **first-order** term, **group** parameters, **renormalize** and apply non-linearity:

$$\vec{g}_{\vec{\theta}} * \vec{x} \approx \text{ReLU}(\tilde{A}X\Theta)$$

where $\tilde{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$

Obs.. Historical approach, albeit interesting, is **unnecessarily** mathematically heavy and rigid: need for more flexible framework \implies **MPNNs**³

1. Nahmias (2017), Hein, Audibert, and Luxburg (2005), Burago, Ivanov, and Kurylev (2014), Lu (2020), Aubry (2013), García Trillos et al. (2019)
2. Kipf and Welling (2017)
3. Gilmer et al. (2017)



Graph Neural Networks (GNNs) #2

MPNN framework prescribes that a GNN is a **stacking of layers** of the form:

$$\vec{h}_v^{(t+1)} = \text{COMB}_t \left(\vec{h}_v^{(t)}, \text{AGGR}_t \left(\left\{ \text{MSG}_t \left(\vec{h}_v^{(t)}, \vec{h}_u^{(t)}, e_{uv} \right) \mid u \in N(v) \right\} \right) \right)$$

Kipf's convolution recovered by:

$$\begin{cases} \text{COMB}_t & = \text{ReLU}(\cdot) \text{ (no self-information)} \\ \text{AGGR}_t & = \sum \\ \text{MSG}_t \left(\vec{h}_v^{(t)}, \vec{h}_u^{(t)}, e_{uv} \right) & = \frac{W_t}{\sqrt{\deg(u)} \sqrt{\deg(v)}} \vec{h}_u^{(t)} \end{cases}$$

Obs. All architectures within this framework are **permutation-invariant**;

Obs. The above graph convolution generalizes **CNNs' convolution** (minus pooling)¹

Other architectures are **GAT**, **GraphSAGE**, **SGCN**, etc²

1. Bianchi and Lachi (2023)



2. Hamilton, Ying, and Leskovec (2017), Veličković et al. (2018), Chanpuriya and Musco (2022)



GNN Expressivity #1

- We may **choose** to quantify GNN “expressivity” by the number of (sub)graphs it may distinguish when **no features** are present;
- In this case, GNNs are at most expressive as the WL-kernel **by construction**;
- GNNs may match WL-kernel
 $\iff \xrightarrow{\text{COMB}_t}, \xrightarrow{\text{AGGR}_t}, \xrightarrow{\text{MSG}_t}$ are all **injective** on their domains;



GNN Expressivity #2

- By **Kolmogorov-Arnold's theorem**¹, every symmetric function can be represented as:

$$f(x_1, \dots, x_n) = \rho \left(\sum_{i=1}^n \phi(x_i) \right) \text{ where } \rho \text{ and } \phi \text{ are injective nonlinearities}$$

- Thus, by the **Universal Approximation Theorem**, the following architecture² has a chance to match WL expressivity:

$$\vec{h}_v^{(t+1)} = \text{MLP} \left((1 + \epsilon) \vec{h}_v^{(k)} + \sum_{u \in N(v)} \vec{h}_u^{(k)} \right)$$

- More expressive GNNs may be obtained by mimicking k-WL kernel, or by extending the MPNN framework³.

1. Wagstaff et al. (2022), Zaheer et al. (2017), [wikipedia](#)



2. Xu et al. (2019)

3. Morris et al. (2021), Du et al. (2018), Michel et al. (2023)



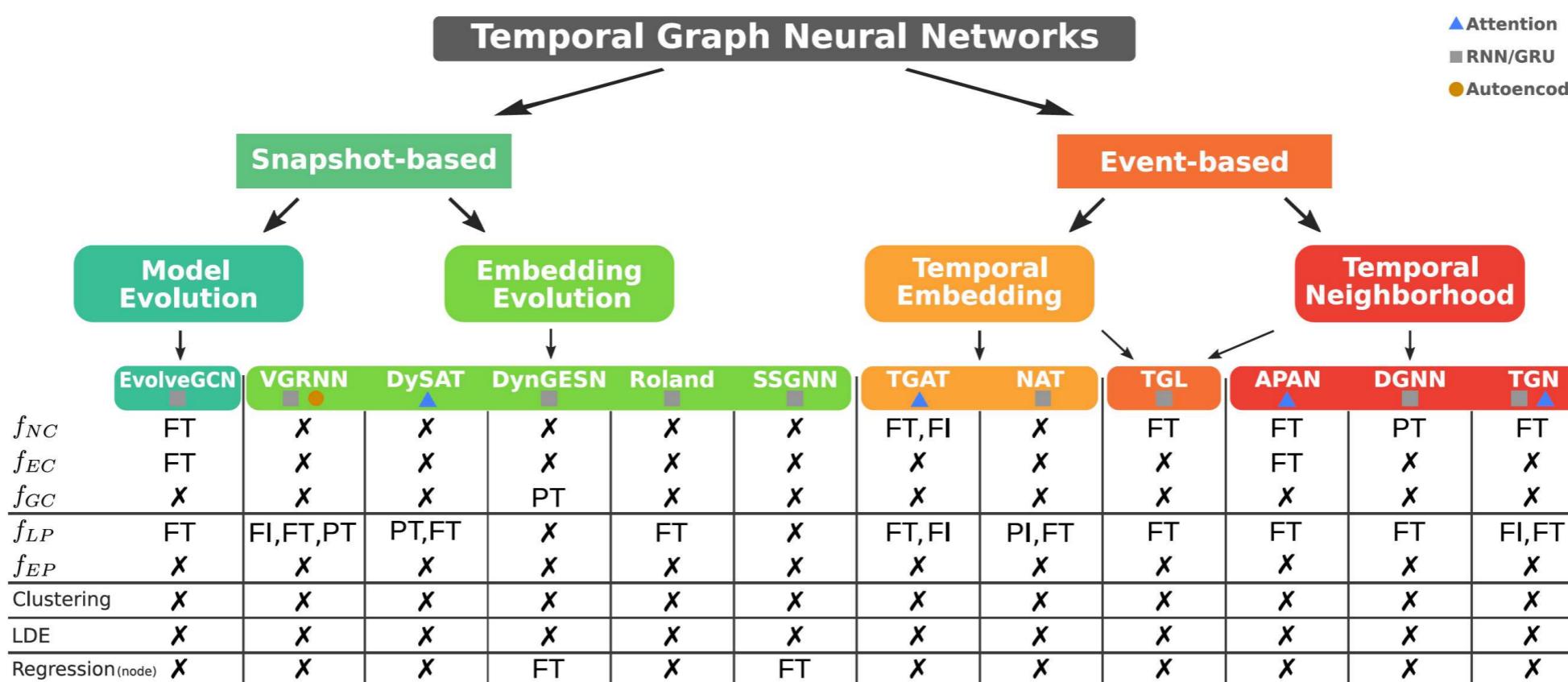
Overview So Far

1. Graph Representation Learning
2. Temporal Graph Neural Networks:
 - 2.1 Framework
 - 2.2 ROLAND
 - 2.3 TGAT
3. Node Classification
4. Link Prediction



Framework

Reviews: Longa et al. (2023), Gravina and Bacciu (2023), Gupta and Bedathur (2022)



ROLAND

- A meta-model $\text{GNN}^{(meta)}$, responsible for parameter initialization before **fine tuning** on each snapshot, is trained via **Meta-Learning** up to snapshot t ;
- The GNN_{t+1} at snapshot $t + 1$ is initialized $\text{GNN} \leftarrow \text{GNN}^{(meta)}$ and **fine-tuned** on snapshot $t + 1$ via:

$$H_{(k+1)}^{(t+1)} = \text{GRU}(H_{(k+1)}^{(t)}, \tilde{H}_{(k)}^{(t+1)})$$

So only **current snapshot** and **previous final embeddings** are needed on GPU (**live-update**)

- The meta-model is updated:

$$\text{GNN}^{(meta)} \leftarrow (1 - \alpha)\text{GNN}^{(meta)} + \alpha\text{GNN}_{t+1}$$

And the cycle repeats.



TGAT

Time differences are represented via Bochner's theorem (translation invariance can be proved):

$$\Phi(\Delta t) = \sqrt{\frac{1}{d}} [\cos(\omega_1 \Delta t), \sin(\omega_1 \Delta t), \dots, \cos(\omega_d \Delta t), \sin(\omega_d \Delta t)]$$

Given node v , we define:

$$\begin{aligned} Z^{(l-1)}(t) &:= [\vec{h}_v^{(l-1)}(t) || \vec{x}_{(v,v)} || \Phi(0), \vec{h}_{u_1}^{(l-1)}(t_1) || \vec{x}_{(u_1,v)} || \Phi(t - t_1), \dots \\ &\quad , \vec{h}_{u_1}^{(l-1)}(t_N) || \vec{x}_{(u_N,v)} || \Phi(t - t_N)]^t \\ \vec{q}_v^{(l-1)}(t)^t &:= \vec{Z}_v^{(l-1)}(t) W_Q := Z^{(l-1)}(t)[0,:] W_Q \\ K^{(l-1)}(t) &:= Z^{(l-1)}(t)[1:N,:] W_K \\ V^{(l-1)}(t) &:= Z^{(l-1)}(t)[1:N,:] W_V \end{aligned}$$



- A **convolutional self-attention¹** layer is defined:

$$\tilde{\vec{h}}_v^{(l)}(t) = \underbrace{\sum_{u \in N(v; t)} \text{softmax} \left(\frac{\vec{q}_v^{(l-1)}(t)^T \vec{K}_u^{(l-1)}(t)}{\sqrt{d}}; N(v; t) \right) \vec{V}_u^{(l-1)}(t)}_{\tilde{h}_{uv}^{(l)}(t)}$$

- Final embedding for v is:

$$\vec{h}_v^{(l)}(t) = \text{ReLU}([\tilde{\vec{h}}_v^{(l)}(t) | \vec{h}_v^{(0)}(t)] W_0^{(l)} + \vec{b}_0^{(l)}) W_1^{(l)} + \vec{b}_1^{(l)}$$

1. Vaswani et al. (2017)



Overview So Far

1. Graph Representation Learning
2. Temporal Graph Neural Networks
3. Node Classification:
 - 3.1 Models
 - 3.2 Reddit NC Task
 - 3.3 OGBN-Arxiv NC Task
4. Link Prediction



Models #1

GraphSAGE GCN:

$$\vec{h}_v^{(k)} = \text{L2-NORMALIZE} \left(\sigma \left(W^{(k)} \text{MEAN}(\{\vec{h}_v^{(k-1)}\} \cup \{\vec{h}_u^{(k-1)} | u \in N(v)\}) \right) \right)$$

```
1  class GraphSAGEConv_GCN(pyg.nn.MessagePassing):
2      def __init__(self, in_channels, out_channels):
3          super().__init__(aggr = "mean")
4          self.W = torch.nn.Linear(in_channels, out_channels, bias = False)
5          self.activation = torch.nn.ReLU()
6
7      def message_and_aggregate(self, adj_t, x):
8
9          return spmm(adj_t, x, reduce=self.aggr)
10
11     def message(self, x_j):
12         return x_j
13
14     def update(self, inputs):
15
16         unnormalized_embeddings = self.activation( self.W(inputs) ) # assumes add_self_loops was run before
17
18         normalized_embeddings = normalize(unnormalized_embeddings, p =2 , dim = 1)
19
20         return normalized_embeddings
21
22     def forward(self, x, edge_index): #x, edge_index
```



Models #2

GraphSAGE MEAN:

$$\vec{h}_v^{(k)} = \text{L2-NORMALIZE} \left(\sigma \left(W_N^{(k)} \text{MEAN}(\{\vec{h}_u^{(k-1)} | u \in N(v)\}) + W_S^{(k)} \vec{h}_v^{(k-1)} \right) \right)$$

```
1  class GraphSAGEConv_MEAN(pyg.nn.MessagePassing):
2      def __init__(self, in_channels, out_channels):
3          super().__init__(aggr = "mean")
4          self.W_neigh = torch.nn.Linear(in_channels, out_channels, bias = False)
5          self.W_self = torch.nn.Linear(in_channels, out_channels, bias = False)
6          self.activation = torch.nn.ReLU()
7
8      def message_and_aggregate(self, adj_t, x):
9
10         return spmm(adj_t, x, reduce=self.aggr)
11
12     def message(self, x_i, x_j):
13         return x_j
14
15
16     def forward(self, x, edge_index): #x, edge_index
17
18
19         neigh_means = self.propagate( edge_index, x=x)
20
21         neigh_means_transformed = self.W_neigh(neigh_means)
22
23         self_vecs_transformed = self.W_self(x)
24
25         unnormalized_embeddings = self.activation(self_vecs_transformed + neigh_means_transformed)
```



Models #3

ModGIN:

$$\vec{h}_v^{(k)} = \text{MLP}_C \left(\text{CONCAT} \left(\sum_{u \in N(v)} \text{MLP}_N(\vec{h}_u^{(k-1)}, \vec{h}_v^{(k-1)}) \right) \right)$$

```
1  class ModGINConv(MessagePassing):
2      def __init__(self, in_channels, out_channels):
3          super().__init__(aggr = "add")
4
5          self.in_channels = in_channels
6          self.out_channels = out_channels
7
8          self.neigh_net = Sequential(
9              Linear(self.in_channels, self.out_channels),
10             LeakyReLU(),
11             Linear(self.out_channels, self.out_channels),
12             LeakyReLU(),
13         )
14
15          self.concat_net = Sequential(
16              Linear(self.out_channels + self.in_channels, self.out_channels),
17              LeakyReLU(),
18              Linear(self.out_channels, self.out_channels),
19              LeakyReLU(),
20         )
21
22      def message_and_aggregate(self, adj_t, x):
```



```
23     return pyg.utils.spmm(adj_t, self.neigh_net(x), self.aggr) # torch_sparse.spmm
24
25 def forward(self, x, edge_index):
26
```



Models #4

ModGIN_Att_Norm:

$$\tilde{\vec{h}}_v^{(k)} = \text{MLP}_C \left(\left(\sum_{u \in N(v)} \text{MLP}_N(\vec{h}_u^{(k-1)} || \vec{h}_v^{(k-1)}) \right) || \vec{h}_v^{(k-1)} \right)$$

```
1  class ModGINConv(MessagePassing):
2      def __init__(self, in_channels, out_channels):
3          super().__init__(aggr = "add")
4
5          self.in_channels = in_channels
6          self.out_channels = out_channels
7
8          self.neigh_net = Sequential(
9              Linear(2*self.in_channels, self.out_channels),
10             LeakyReLU(),
11             Linear(self.out_channels, self.out_channels),
12             LeakyReLU(),
13         )
14
15          self.concat_net = Sequential(
16              Linear(self.out_channels + self.in_channels, self.out_channels),
17              LeakyReLU(),
18              Linear(self.out_channels, self.out_channels),
19              LeakyReLU(),
20         )
21
22      def message(self, x_i, x_j):
```



```
23     return self.neigh_net(torch.cat((x_j,x_i), dim = 1))  
24  
25 def forward(self, x, edge_index):  
26
```



Reddit NC Task #1

41-way node classification on Reddit Dataset.

Reddit Dataset:

- Nodes are **posts** (232,965 total, 492 average degree)
- Two posts are linked if the **same user** comments below both (114615892 total edges)
- Label to predict is the **subreddit** a post belongs to
- Other node features: 300-dimensional **word embedding** of title || comments || score and number of comments
- **Metric:** micro-averaged F1 score
- **Training Set** size: 153431, **Validation Set** size: 23831, **Test Set** size: 55703



Reddit NC Task #2

- Minibatch + Neighbor Sampling;
- 10 epochs, 512 batch size
- $K = 2$
- $S_1 = 25, S_2 = 10$
- Embedding dimension: 256 (big), 128 (small)
- $\sigma = \text{ReLU}$
- Adam Optimizer, LR Hyperparameter Search: $\{0.01, 0.001, 0.0001\}$
- Loss ℓ : LogSoftmax + Negative Log Likelihood;



Reddit NC Task #3

Model	NSI (F1)	FNI (F1)	epochs	secs/epoch
GraphSAGE_GCN	0.941	0.947	5	16.44
GraphSAGE_GCN_Sparse	0.941	0.948	10	14.68
GraphSAGE_MEAN	0.957	0.962	3	19.15
GraphSAGE_MEAN_Sparse	0.960	0.963	2	15
ModGIN	0.961	0.7	25	25.22
ModGIN_Att_Norm	0.959	0.964	10	46.27



OGBN-Arxiv NC Task #1

TASK: 40-way node classification on OGBN-Arxiv.

OGBN-Arxiv:

- Nodes are **articles** (169343 total)
- Two articles are linked if one cites the other (directed graph, 1166243 total edges)
- Label to predict is the **subject area** an article belongs to (e.g. cs.AI, cs.OS, etc)
- Other node features: 128-dimensional feature vector obtained by **averaging** the embeddings of words in its title and abstract.
- **Metric:** micro-averaged F1 score (accuracy)
- **Training Set:** until 2017 (included), **Validation Set:** 2018, **Test Set:** 2019
- Evaluation same as Reddit



OGBN-Arxiv NC Task #2

Model	Accuracy	epochs	secs/epoch
GraphSAGE_MEAN_OTS	0.7164	154	0.12
GraphSAGE_MEAN_MB_NS	0.704	10	6.64



Overview So Far

1. Graph Representation Learning
2. Temporal Graph Neural Networks
3. Node Classification
4. Link Prediction:
 - 4.1 Overview;
 - 4.2 OGBL-DDI;
 - 4.3 (V)GAE;
 - 4.4 (V)GNAE;
 - 4.5 Gravity-(V)GAE;
 - 4.6 ST-(V)GAE;
 - 4.7 IT-AML;
 - 4.8 Conclusions;



Overview

- **Link Prediction** consists in predicting the probability of an unseen edge between two nodes;
- The models we'll be using are **(Variational) Graph Autoencoders**, they're composed of:
 - An **Encoder** that computes the embeddings of all nodes $H = Enc(A, X)$;
 - A **Decoder** that emits the probability of an edge $p_{ij} = Dec(\vec{h}_i, \vec{h}_j)$.
- The Variational models are also capable of **graph generation** through the added **stochasticity**;
- To perform LP, edges are split between **message**, **supervision**, **validation** and **test** sets.



OGBL-DDI

```
1
2 pyg.seed_everything(12345)
3
4 # Set device
5 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
6 cpu_device = torch.device("cpu")
7
8
9 # Load dataset https://ogb.stanford.edu/docs/linkprop/#ogbl-ddi
10 dataset = PygLinkPropPredDataset(name = "ogbl-ddi", root = data_path, transform = T.ToSparseTensor())
11
12
13 split_edge = dataset.get_edge_split()
14 train_edge, valid_edge, test_edge = split_edge["train"], split_edge["valid"], split_edge["test"]
15 data = dataset[0]
16
17 data.to(device, "x", "adj_t")
18
19
20 negative_sampling_amount = 1
21 batch_size = 64 * 1024
22 num_neighbors = [-1,-1]
23
24
25
26 train_loader = LinkNeighborLoader(
    . . .
```

We fully reproduce performance of OGB Team.



(V)GAE

- **Dataset:** Cora (Undirected)
- **Decoder:** $p_{ij} = \sigma(\vec{h}_i \cdot \vec{h}_j)$
- **Training Strategy:** Random Split + “General Setting”

Ours:

	GAE	VGAE
AUC	0.92 +- 0.01	0.91 +- 0.006
F1	0.81 +- 0.01	0.799 +- 0.007
hitsk	0.66 +- 0.07	0.6 +- 0.05
AP	0.92 +- 0.02	0.905 +- 0.008
MRR	0.48 +- 0.06	0.42 +- 0.02

Theirs:

Method	Cora	
	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00
DW [6]	83.1 ± 0.01	85.0 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01
VGAE*	84.0 ± 0.02	87.7 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03
VGAE	91.4 ± 0.01	92.6 ± 0.01



(V)GNAE #1

- GAE and VGAE embed **isolated** nodes near $\vec{0}$ \implies all isolated nodes are **similar**;
- The authors propose to **normalize** the embeddings before **propagating (GNCN)**:

$$\vec{h}_v^l = \frac{1}{d_i + 1} s \frac{\vec{h}_v^{l-1}}{\|\vec{h}_v^{l-1}\|} + \sum_{u \in N(v)} \frac{1}{\sqrt{d_v + 1} \sqrt{d_u + i}} s \frac{\vec{h}_u^{l-1}}{\|\vec{h}_u^{l-1}\|}$$

Encoder:

1. One-layer GNCN for the GNAE;
2. A GNCN layer (μ) and a GCN layer ($\log\sigma$) for the VGNAE.

Decoder: $p_{ij} = \sigma(\vec{h}_i \cdot \vec{h}_j)$



(V)GNAE #2

Task: Link prediction on undirected Cora

Results

Ours:

	GNAE	VNGAE
AUC	0.930 +- 0.004	~0.9
F1	0.793 +- 0.003	~0.76
AP	0.940 +- 0.003	~0.92

Theirs:

Data	Metric	Train	GAE	LGAE	ARGA	GIC	sGraph	GNAE	VGNAE
Cora	AUC	20%	0.782	0.866	0.795	0.880	0.845	0.887	0.890
		40%	0.856	0.908	0.844	0.914	0.840	0.926	0.929
		80%	0.922	0.938	0.919	0.933	0.885	0.956	0.954
	AP	20%	0.793	0.878	0.806	0.881	0.829	0.901	0.901
		40%	0.861	0.915	0.856	0.911	0.828	0.936	0.933
		80%	0.930	0.945	0.927	0.929	0.867	0.957	0.958



Gravity-(V)GAE #1

- **Dataset:** Cora (Directed)
- **Decoder:** $\hat{A}_{ij} = \sigma(m_j - \lambda \log(\|\vec{z}_i - \vec{z}_j\|_2^2))$,
 $m_i = Enc(\vec{x}_i)[-1]$, $\vec{z}_i = Enc(\vec{x}_i)[:-1]$
- Directed LP is divided in three tasks: **General**, **Biased** and **Bidirectional**
- **Improvements** over original: **Early Stopping**, Trainable λ , **better Train Set** for Biased and Bidirectional



Gravity-(V)GAE #2

Ours:

General Link Prediction:

	Gravity-GAE	Gravity-VGAE
AUC	0.857 +- 0.002	0.9041 +- 0.0005
F1	0.74 +- 0.01	0.741 +- 0.007
hitsk	0.72 +- 0.01	0.754 +- 0.004
AP	0.902 +- 0.002	0.9279 +- 0.0007

Bidirectionality Prediction:

Biased Negative Samples Link Prediction:

	Gravity-GAE	Gravity-VGAE	Gravity-GAE_altered
AUC	0.83 +- 0.01	0.822 +- 0.003	0.848 +- 0.001
F1	0.722 +- 0.003	0.742 +- 0.002	0.751 +- 0.006
hitsk	0.445 +- 0.005	0.44 +- 0.01	0.37 +- 0.02
AP	0.832 +- 0.005	0.82 +- 0.002	0.833 +- 0.003



	Gravity-GAE	Gravity-VGAE	Gravity-GAE_altered	Gravity-VGAE_altered
AUC	0.775 +- 0.003	0.753 +- 0.01	0.794 +- 0.002	0.774 +- 0.003
F1	0.755 +- 0.006	0.74 +- 0.01	0.32 +- 0.01	0.4 +- 0.01
hitsk	0.55 +- 0.01	0.52 +- 0.04	0.57 +- 0.007	0.57 +- 0.02
AP	0.744 +- 0.003	0.7 +- 0.01	0.778 +- 0.004	0.742 +- 0.005

Theirs:

Dataset	Model	Task 1: General Link Prediction		Task 2: B.N.S. Link Prediction		Task 3: Bidirectionality Prediction	
		AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Cora	<i>Gravity Graph VAE (ours)</i>	91.92 ± 0.75	92.46 ± 0.64	83.33 ± 1.11	84.50 ± 1.24	75.00 ± 2.10	73.87 ± 2.82
	<i>Gravity Graph AE (ours)</i>	87.79 ± 1.07	90.78 ± 0.82	83.18 ± 1.12	84.09 ± 1.16	75.57 ± 1.90	73.40 ± 2.53



ST-(V)GAE

- Everything the same as Gravity-(V)GAE (same paper);
- **Decoder:**

$$\hat{A}_{ij} = \sigma(\vec{s}_i \cdot \vec{t}_j)$$

where

$$\begin{aligned}\vec{s}_i &= Enc(\vec{x}_i)[: emb_size/2] \\ \vec{t}_i &= Enc(\vec{x}_i)[emb_size/2 :]\end{aligned}$$

- Alternative ([Kollias et al. 2022](#)).



ST-(V)GAE

Ours:

General Link Prediction:

	SourceTarget-GAE
AUC	0.86 +- 0.02
F1	0.5 +- 0.08
hitsk	0.66 +- 0.03
AP	0.894 +- 0.008

Biased Negative Samples Link Prediction:

	SourceTarget-GAE	SourceTarget-GAE_altered
AUC	0.61 +- 0.01	0.878 +- 0.006
F1	0.39 +- 0.09	0.8 +- 0.008
hitsk	0.17 +- 0.02	0.56 +- 0.02
AP	0.63 +- 0.01	0.869 +- 0.009

Bidirectionality Prediction:

	SourceTarget-GAE
AUC	0.74 +- 0.02
F1	0.69 +- 0.01
hitsk	0.52 +- 0.07



	Source	Target-GAE
AP	0.72	± 0.03

Theirs:

Source/Target Graph AE	82.67 ± 1.42	83.25 ± 1.51	57.81 ± 2.64	57.66 ± 3.35	65.83 ± 3.87	63.15 ± 4.58
------------------------	------------------	------------------	------------------	------------------	------------------	------------------



IT-AML #1

- Paper: Altman et al. (2023)
- ABM model (*AMLworld*) that simulates transactions in a complex financial environment (individuals, banks, companies, etc.)
- Model's output is a temporal sequence of transactions with labelled laundering patterns:

	Timestamp	From Bank	Account	To Bank	Account.1	Amount Received	Receiving Currency	Amount Paid	Payment Currency	Payment Format	Is Laundering
0	2022/09/01 00:08	11	8000ECA90	11	8000ECA90	3.195403e+06	US Dollar	3.195403e+06	US Dollar	Reinvestment	0
1	2022/09/01 00:21	3402	80021DAD0	3402	80021DAD0	1.858960e+03	US Dollar	1.858960e+03	US Dollar	Reinvestment	0
2	2022/09/01 00:00	11	8000ECA90	1120	8006AA910	5.925710e+05	US Dollar	5.925710e+05	US Dollar	Cheque	0
3	2022/09/01 00:16	3814	8006AD080	3814	8006AD080	1.232000e+01	US Dollar	1.232000e+01	US Dollar	Reinvestment	0
4	2022/09/01 00:00	20	8006AD530	20	8006AD530	2.941560e+03	US Dollar	2.941560e+03	US Dollar	Reinvestment	0
...	
6924044	2022/09/10 23:39	71696	81B2518F1	71528	81C0482E1	3.346900e-02	Bitcoin	3.346900e-02	Bitcoin	Bitcoin	0
6924045	2022/09/10 23:48	271241	81B567481	173457	81C0DA751	1.313000e-03	Bitcoin	1.313000e-03	Bitcoin	Bitcoin	0
6924046	2022/09/10 23:50	271241	81B567481	173457	81C0DA751	1.305800e-02	Bitcoin	1.305800e-02	Bitcoin	Bitcoin	0
6924047	2022/09/10 23:57	170558	81A2206B1	275798	81C1D5CA1	4.145370e-01	Bitcoin	4.145370e-01	Bitcoin	Bitcoin	0
6924048	2022/09/10 23:31	170558	81A2206B1	275798	81C1D5CA1	3.427700e-02	Bitcoin	3.427700e-02	Bitcoin	Bitcoin	0

[6924049 rows x 11 columns]



IT-AML #2

Laundering Data:

```
BEGIN LAUNDERING ATTEMPT - FAN-IN: Max 3-degree Fan-In
2022/09/01 02:38,001812,80279F810,0110,8000A94C0,10154.74,Australian Dollar,10154.74,Australian Dollar,ACH,1
2022/09/02 14:36,0022595,80279F8B0,0110,8000A94C0,5326.79,Australian Dollar,5326.79,Australian Dollar,ACH,1
2022/09/03 14:09,001120,800E36A50,0110,8000A94C0,4634.81,Australian Dollar,4634.81,Australian Dollar,ACH,1
END LAUNDERING ATTEMPT - FAN-IN

BEGIN LAUNDERING ATTEMPT - FAN-IN: Max 8-degree Fan-In
2022/09/01 03:17,003671,801BF8E70,002557,8016B3750,8099.96,Euro,8099.96,Euro,ACH,1
2022/09/01 06:27,015,80074C7E0,002557,8016B3750,10468.56,Euro,10468.56,Euro,ACH,1
2022/09/01 10:04,002557,80107C9A0,002557,8016B3750,10270.07,Euro,10270.07,Euro,ACH,1
2022/09/02 06:35,012,800A9B180,002557,8016B3750,15645.21,Euro,15645.21,Euro,ACH,1
2022/09/03 09:12,021393,801271170,002557,8016B3750,14139.75,Euro,14139.75,Euro,ACH,1
2022/09/03 13:45,002175,801E25F20,002557,8016B3750,6276.26,Euro,6276.26,Euro,ACH,1
2022/09/03 16:17,020,800043BE0,002557,8016B3750,1042.63,Euro,1042.63,Euro,ACH,1
2022/09/03 23:09,022124,8011D0180,002557,8016B3750,12795.57,Euro,12795.57,Euro,ACH,1
END LAUNDERING ATTEMPT - FAN-IN
```

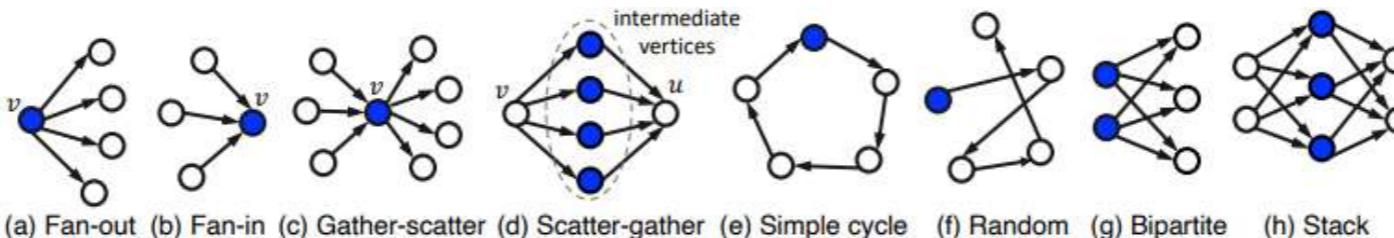


Figure 2: Laundering Patterns Modelled



IT-AML #3

- The ABM is initialized via distributions, not real-world data;
- Modeling is detailed: transactions stratified by reason (e.g. salaries are monthly), companies own other companies, etc...
- 6 synthetic datasets are available [online](#)

Table 4: Public Synthetic Data Statistics. **HI** = Higher Illicit (more laundering). **LI** = Lower Illicit.

Statistic	Small		Medium		Large	
	HI	LI	HI	LI	HI	LI
# of Days Spanned	10	10	16	16	97	97
# of Bank Accounts	515K	705K	2077K	2028K	2116K	2064K
# of Transactions	5M	7M	32M	31M	180M	176M
# of Laundering Transactions	3.6K	4.0K	35K	16K	223K	100K
Laundering Rate (1 per N Trans)	981	1942	905	1948	807	1750



IT-AML #4

Link Prediction Pass

Data Processing:

- Dataset size required vectorized methods
- Duplicate edges removed and temporal information discarded

Model. Gravity-GAE (best so far)

Split. Directed `torch.geometric.transforms.RandomLinkSplit` with
`neg_sampling = 1`



General Setting

Gravity-GAE	
AUC	0.894 +- 0.003
F1	0.8399 +- 0.0009
hitsk	0.00048 +- 3e-05
AP	0.86 +- 0.01
AUC_rev	0.617 +- 0.003
F1_rev	0.707 +- 0.002
hitsk_rev	5e-05 +- 1e-05
AP_rev	0.545 +- 0.003

Biased Setting

Gravity-GAE	
AUC	0.7827 +- 0.0002
F1	0.7416 +- 0.0005
hitsk	2e-05 +- 0.0
AP	0.7252 +- 0.0002



IT-AML #4

“Temporal” Link Prediction Pass



Conclusions

- Gravity-(V)GAE and ST-(V)GAE are from the **same paper** ([Salha et al. 2019](#));
- For the ST models, we also implemented the **ST-VGAE** (not reported);
- We started a **collaboration** with the author to:
 - Implement a **Pytorch Geometric** version of his code;
 - **Confirm** the improvements suggested above;
 - (Possibly) find a **single training strategy** that fits all three tasks.
- Would be nice to try ([Kollias et al. 2022](#)), Edge Convolution/Line Graphs ([Cai et al. 2020](#)), Topology Adaptive GNN ([Du et al. 2018](#)) and experimenting with bidirectional heterogeneity.



Next Goals

1. Directed Link Prediction
2. Graph Representation Learning via Static Representations
3. Anti Money Laundering via Subgraph Matching



Next Goals

1. Directed Link Prediction
2. Graph Representation Learning via Static Representations
3. Anti Money Laundering via Subgraph Matching



Directed Link Prediction

- ([Salha et al. 2019](#)) splits directed LP into three tasks;
- But comes short of finding a setup that produces a good representation for **all** three;
- We began exploring this avenue initiating a **collaboration** with the author;
- (If time allows), I'd try **weighting** reciprocal vs non-reciprocal vs random neagative edges in the loss and/or finding the **optimal amount** of random negatives;
- We'll also apply directed LP to a **graph generation** task in the near future.



Next Goals

1. Directed Link Prediction
2. Graph Representation Learning via Static Representations
3. Anti Money Laundering via Subgraph Matching



Graph Representation Learning via Static Representations

- Current TGNNs suffer from [low performances](#), and are very **complex** and difficult to **train**;
- ([Oettershagen et al., n.d.](#)) proposes three ways to **statically represent** a graph s.t. the WL kernel retains most temporal information;
- Might be worth **extending** the framework to GNNs and other tasks;
- We have initiated a(nother) **collaboration** with researchers from **UniTrento** to implement these ideas.



Next Goals

1. Directed Link Prediction
2. Graph Representation Learning via Static Representations
3. Anti Money Laundering via Subgraph Matching



Anti Money Laundering via Subgraph Matching

- Current Machine/Deep Learning/(T)GNNs approach to AML make use of **node/edge classification**
- But laundering, as IT-AML confirms, is a matter of patterns i.e. **temporal motifs**
- **Statistical/Algorithmic** approaches to Subgraph Matching have been tried¹ but they're not **inductive** and may **scale** suboptimally
- For this reasons, **Neural** (approximate) Subgraph Matching/Mining/Counting algorithms have been developed [Nguye2023](#), [Lan2022](#), [Xu2022](#), ([Roy et al. 2022](#)), ([Fey et al. 2020](#)), ([Rex et al. 2020](#))
- These methods may be applied to **static representations** of temporal networks.



Thanks



References

- Alarab, Ismail, and Simant Prakoonwit. 2022. “Graph-Based LSTM for Anti-Money Laundering: Experimenting Temporal Graph Convolutional Network with Bitcoin Data.” *Neural Processing Letters* 55 (1): 689–707. <https://doi.org/10.1007/s11063-022-10904-8>.
- Altman, Erik, Jovan Blanuša, Luc von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. 2023. “Realistic Synthetic Financial Transactions for Anti-Money Laundering Models.” <https://arxiv.org/abs/2306.16424>.
- Aubry, Erwann. 2013. “Approximation of the Spectrum of a Manifold by Discretization.” arXiv. <https://doi.org/10.48550/ARXIV.1301.3663>.
- Bianchi, Filippo Maria, and Veronica Lachi. 2023. “The Expressive Power of Pooling in Graph Neural Networks.” <https://arxiv.org/abs/2304.01575>.
- Burago, Dmitri, Sergei Ivanov, and Yaroslav Kurylev. 2014. “A Graph Discretization of the Laplace–Beltrami Operator.” *Journal of Spectral Theory* 4 (4): 675–714. <https://doi.org/10.4171/jst/83>.
- Cai, Lei, Jundong Li, Jie Wang, and Shuiwang Ji. 2020. “Line Graph Neural Network for Link Prediction.” <https://arxiv.org/abs/2010.10046>.



Chanpuriya, Sudhanshu, and Cameron N Musco. 2022. “Simplified Graph Convolution with Heterophily.” In *Advances in Neural Information Processing Systems*, edited by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. <https://openreview.net/forum?id=jRrpjqxtrWm>.

Du, Jian, Shanghang Zhang, Guanhong Wu, Jose M. F. Moura, and Soumya Kar. 2018. “Topology Adaptive Graph Convolutional Networks.” <https://arxiv.org/abs/1710.10370>.

Fey, Matthias, Jan E. Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. 2020. “Deep Graph Matching Consensus.” In *International Conference on Learning Representations*. <https://openreview.net/forum?id=HyeJf1HKvS>.

García Trillos, Nicolás, Moritz Gerlach, Matthias Hein, and Dejan Slepčev. 2019. “Error Estimates for Spectral Convergence of the Graph Laplacian on Random Geometric Graphs Toward the Laplace–Beltrami Operator.” *Foundations of Computational Mathematics* 20 (4): 827–87. <https://doi.org/10.1007/s10208-019-09436-w>.

Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. “Neural Message Passing for Quantum Chemistry.” In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 1263–72. ICML’17. Sydney, NSW, Australia: JMLR.org.



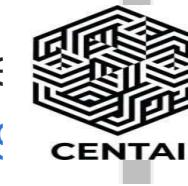
- Gravina, Alessio, and Davide Bacciu. 2023. “Deep Learning for Dynamic Graphs: Models and Benchmarks.” <https://arxiv.org/abs/2307.06104>.
- Gupta, Shubham, and Srikanta Bedathur. 2022. “A Survey on Temporal Graph Representation Learning and Generative Modeling.” <https://arxiv.org/abs/2208.12126>.
- Hamilton, William L., Rex Ying, and Jure Leskovec. 2017. “Inductive Representation Learning on Large Graphs.” In, 1025–35. NIPS’17. Red Hook, NY, USA: Curran Associates Inc.
- Hein, Matthias, Jean-Yves Audibert, and Ulrike von Luxburg. 2005. “From Graphs to Manifolds – Weak and Strong Pointwise Consistency of Graph Laplacians.” In *Lecture Notes in Computer Science*, 470–85. Springer Berlin Heidelberg. https://doi.org/10.1007/11503415_32.
- Jensen, Rasmus Ingemann Tuffveson, and Alexandros Iosifidis. 2023. “Qualifying and Raising Anti-Money Laundering Alarms with Deep Learning.” *Expert Systems with Applications* 214: 119037. <https://doi.org/10.1016/j.eswa.2022.119037>.
- Johannessen, Fredrik, and Martin Jullum. 2023. “Finding Money Launderers Using Heterogeneous Graph Neural Networks.” <https://arxiv.org/abs/2307.13499>.
- Jullum, Martin, Anders Løland, Ragnar Bang Huseby, Geir Ånonsen, and Johannes Lorentzen. 2020. “Detecting Money Laundering Transactions with Machine



- Learning.” *Journal of Money Laundering Control* 23 (1): 173–86.
<https://doi.org/10.1108/jmlc-07-2019-0055>.
- Kipf, Thomas N., and Max Welling. 2017. “Semi-Supervised Classification with Graph Convolutional Networks.” <https://arxiv.org/abs/1609.02907>.
- Kollias, Georgios, Vasileios Kalantzis, Tsuyoshi Idé, Aurélie Lozano, and Naoki Abe. 2022. “Directed Graph Auto-Encoders.” <https://arxiv.org/abs/2202.12449>.
- Li, Zhiyuan, and Enhan He. 2023. “Graph Neural Network-Based Bitcoin Transaction Tracking Model.” *IEEE Access* 11: 62109–20.
<https://doi.org/10.1109/ACCESS.2023.3288026>.
- Lin, Dan, Jiajing Wu, Qi Xuan, and Chi K. Tse. 2022. “Ethereum Transaction Tracking: Inferring Evolution of Transaction Networks via Link Prediction.” *Physica A: Statistical Mechanics and Its Applications* 600: 127504.
<https://doi.org/https://doi.org/10.1016/j.physa.2022.127504>.
- Liu, Jiayi, Changchun Yin, Hao Wang, Xiaofei Wu, Dongwan Lan, Lu Zhou, and Chunpeng Ge. 2023. “Graph Embedding-Based Money Laundering Detection for Ethereum.” *Electronics* 12 (14): 3180.
<https://doi.org/10.3390/electronics12143180>.
- Longa, Antonio, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. 2023. “Graph Neural Networks for Temporal Graphs: State of the Art, Open Challenges, and Opportunities.” <https://arxiv.org/abs/2302.01018>.



- Lorenz, Joana, Maria Inês Silva, David Aparício, João Tiago Ascensão, and Pedro Bizarro. 2021. “Machine Learning Methods to Detect Money Laundering in the Bitcoin Blockchain in the Presence of Label Scarcity.” <https://arxiv.org/abs/2005.14635>.
- Lu, Jinpeng. 2020. “Graph Approximations to the Laplacian Spectra.” *Journal of Topology and Analysis* 14 (01): 111–45. <https://doi.org/10.1142/s1793525320500442>.
- Michel, Gaspard, Giannis Nikolentzos, Johannes Lutzeyer, and Michalis Vazirgiannis. 2023. “Path Neural Networks: Expressive and Accurate Graph Neural Networks.” <https://arxiv.org/abs/2306.05955>.
- Morris, Christopher, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2021. “Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks.” <https://arxiv.org/abs/1810.02244>.
- Nahmias, David. 2017. “An Exposition of Spectral Graph Theory.” <https://dnahmias.com/docs/harmonicfinalpaper.pdf>.
- Oettershagen, Lutz, Nils M. Kriege, Christopher Morris, and Petra Mutzel. n.d. “Temporal Graph Kernels for Classifying Dissemination Processes.” In *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, 496–504. <https://doi.org/10.1137/1.9781611976236.56>.
- Rex, Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. “Neural Subgraph Matching.” <https://arxiv.org/abs/2007.03050>.



- Roy, Indradymna, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. 2022. “Interpretable Neural Subgraph Matching for Graph Retrieval.” *Proceedings of the AAAI Conference on Artificial Intelligence* 36 (7): 8115–23. <https://doi.org/10.1609/aaai.v36i7.20784>.
- Salha, Guillaume, Stratis Limnios, Romain Hennequin, Viet-Anh Tran, and Michalis Vazirgiannis. 2019. “Gravity-Inspired Graph Autoencoders for Directed Link Prediction.” In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 589–98. CIKM ’19. New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/3357384.3358023>.
- Starnini, Michele, Charalampos E. Tsourakakis, Maryam Zamanipour, André Panisson, Walter Allasia, Marco Fornasiero, Laura Li Puma, et al. 2021. “Smurf-Based Anti-Money Laundering in Time-Evolving Transaction Networks.” In *Lecture Notes in Computer Science*, 171–86. Springer International Publishing. https://doi.org/10.1007/978-3-030-86514-6_11.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention Is All You Need.” In *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc.





CENTAI