Learning Node Representations Using Stationary Flow Prediction on Large Payment and Cash Transaction Networks

Ciwan Ceylan [*1,2] **Salla Franzén** [2] **Florian T. Pokorny** [1]

Abstract

Banks are required to analyse large transaction datasets as a part of the fight against financial crime. Today, this analysis is either performed manually by domain experts or using expensive feature engineering. Gradient flow analysis allows for basic representation learning as node potentials can be inferred directly from network transaction data. However, the gradient model has a fundamental limitation: it cannot represent all types of of network flows. Furthermore, standard methods for learning the gradient flow are not appropriate for flow signals that span multiple orders of magnitude and contain outliers, i.e. transaction data. In this work, the gradient model is extended to a gated version and we prove that it, unlike the gradient model, is a universal approximator for flows on graphs. To tackle the mentioned challenges of transaction data, we propose a multi-scale and outlier robust loss function based on the Student-t log-likelihood. Ethereum transaction data is used for evaluation and the gradient models outperform MLP models using hand-engineered and node2vec features in terms of relative error. These results extend to 60 synthetic datasets, with experiments also showing that the gated gradient model learns qualitative information about the underlying synthetic generative flow distributions.

1. Introduction

A bank will typically have millions of transactions flowing through its systems on a daily basis, and the need to further develop tools to monitor the flows for anomalous behaviours has become a crucial activity. It is estimated that global money laundering activities comprise 2 - 5% of global annual GDP (United Nations office on drugs and crime, 2011), and transaction service providers, such as SEB Group, are required by EU and national regulations to report suspected money-laundering activity to the public authorities (European Parliament and Council, 2018).

A large portion of everyday transactions are payments and cash transactions, e.g. using bank transfers, mobile services or cryptocurrencies. These transactions can be viewed as a dynamic flow on the edges of a network where nodes are accounts of individuals, companies, organisations etc.

Transaction network datasets available for research, however, are scarce due to bank laws and client confidentiality, and machine learning research to aid anti-money laundry (AML) efforts has consequently been impeded. A review of the literature shows a prevalent use of feature engineering, a process which is both time consuming and performance restrictive. Moreover, difficulties in obtaining labelled AML data limits application of supervised deep learning.

In this work, we propose a method to learn representations automatically from network transaction data using a gated extension to the gradient flow model from the literature (Lim, 2020). To automatically learn the representations, we formulate a stationary flow prediction task that provides both a well-defined optimisation objective, which we formalise into an outlier robust, multi-scale loss function fit for the properties of cash transaction data, and a performance measure for objective comparison of our gated gradient model to baselines. To adhere to confidentiality regulation, we here focus on publicly available ethereum transactions which exhibit similar statistical properties to commercial cash and payment transactions (Fig. 1).

The contributions of this work are 1) a gated extension to the gradient flow model capable of learning node representations automatically from network flow data, 2) a proof that this gated model is a universal approximator for flows on graphs, 3) a robust, multi-scale loss function to be used for training flow prediction models on large transactions datasets and 4) empirical evidence using ethereum transactions and synthetic data that our model learns representation with better flow prediction properties than available engineered and learned feature baselines.

[1]RPL, EECS, KTH Royal Institute of Technology, Stockholm, Sweden [2]SEB Group, Stockholm, Sweden. Correspondence to: Ciwan Ceylan <ciwan@kth.se>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

The code, datasets and results are accessible via our project page https://ciwanceylan.github.io/gated-gradient-flow/.

2. Background

The machine learning literature concerning payment and cash transaction data, e.g. for anti-money laundering, is sparse and fragmented due to bank secrecy deterring from publication of benchmark datasets and exchange of AML models (Oeben et al., 2019). In a review on machine learning methods for AML, (Chen et al., 2018) state that all methods rely on feature engineering as an expensive preprocessing step before either unsupervised or supervised machine learning algorithms are applied, e.g. (Le Khac & Kechadi, 2010; Bhattacharyya et al., 2011; Weber et al., 2019). With the advent of deep learning, the computer vision field advanced rapidly by switching from feature engineering to feature learning (LeCun et al., 2015). Yet, even when applying graph convolutional networks to bitcoin transaction data in a supervised AML setting, (Weber et al., 2019) engineer features to use as input to the GCN.

Applying deep learning to naively engineered features can work well if provided with enough labelled data. Obtaining labels in a financial crime prevention setting, however, is difficult and usually only available in the form of customer reported fraud. In an unsupervised AML setting, it is imperative that the features capture relevant aspects of transaction behaviours. (Oeben et al., 2019) note that only one third of the works surveyed include network information in their features, despite the success of network analysis for other network types, e.g. power grids (Albert et al., 2004), affiliation networks (Barabâsi et al., 2002) and social networks (Lewis et al., 2008). Among the works using network features are (Fronzetti Colladon & Remondi, 2017), where an Italian factoring company is monitored using network analysis, and (Savage et al., 2016), who incorporate network features in their supervised learning pipeline for money laundering detection.

During the last decade, cryptocurrencies, e.g. bitcoin and ethereum (Vujičić et al., 2018), have emerged as publicly available transaction datasets since they by necessity maintain a public ledger of all transactions (Zheng et al., 2017). There are several works analysing the properties of these network datasets. (Ron & Shamir, 2013) and (Di Francesco Maesa et al., 2018) analyse the bitcoin transaction graph, both spending significant parts of their papers on how to turn the bitcoin data into a graph where nodes represent users. (Ron & Shamir, 2013) focus on user behaviour and tracing transaction histories while (Di Francesco Maesa et al., 2018) provide many network statistics, e.g. clustering coefficent, average distance and degree distributions, and analyse how these evolve over time. In their systematic study of the ethereum transaction graph, (Chen et al., 2020) provide a wealth of statistics and analyses. They also note that ethereum, unlike bitcoin, includes the concept of accounts and balances, and thus does not require the same difficult preprocessing. Contrary to these works, we use the ethereum transaction data to learn node representations to be used for stationary flow prediction and downstream AML tasks.

To verify that the ethereum data shares some statistical properties with our internal transaction data, and to verify that our synthetic data does not adhere to a significantly different distribution, we visualise the transaction amount densities for these three cases using log-log axes, and with the currency of the internal data converted to eth, in Fig. 1. All three distributions are qualitatively similar with a peak around 10 – 100 eth and span several orders of magnitude, with the ethereum data displaying the heaviest tails.

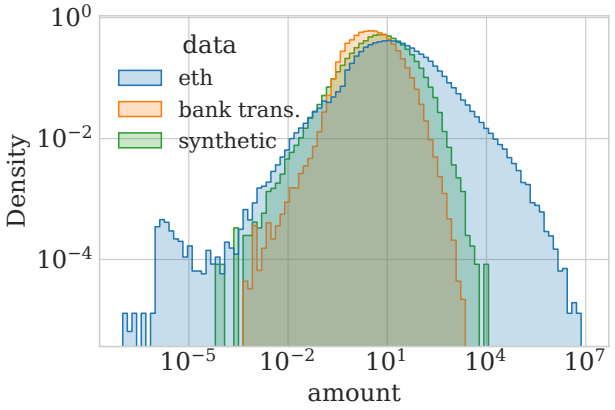

Figure 1. Log-log densities of the preprocessed ethereum transaction amounts, internal bank transaction amounts and synthetic data from the gated gradient model.

3. Related Works

3.1. Node Representation Learning

LINE (Tang et al., 2015), DeepWalk (Perozzi et al., 2014) and node2vec (Grover & Leskovec, 2016) are three well-known node representation learning methods for which embedding similarity corresponds to structural similarity between node neighbourhoods. LINE uses first and second order proximities to define similarity while DeepWalk uses random walks. Node2vec generalises DeepWalk by introducing hyperparameters p and q to control the behaviour of the random walk, thereby allowing different structural similarities to be learned. Unlike our work, these methods learn embeddings unrelated to any flow in the network. The structural embeddings may still be used for stationary flow prediction but the performance will be poor if flow values are independent of structural similarities.

3.2. Link Prediction

Link prediction, filling in missing edges of a graph, is a classic task in the graph mining literature (Newman, 2018, p. 298), and it has been demonstrated that it can be used to learn node representations. (Sarkar et al., 2011) formalise link prediction as a problem of estimating distances between nodes embedded in a latent space and (Rendsburg et al., 2020) propose a graph generator and link predictor via asymmetric factorisation of the adjacency matrix, essentially also learning latent node embeddings. This is similar to our work in the sense that we use stationary flow prediction to learn node representations. However, the approaches differ in both the task formulation, as we assume knowledge of all edges in the graph, and the models used, (Sarkar et al., 2011) use distance models and (Rendsburg et al., 2020) matrix factorisation, while we propose a gradient model.

SEAL (Zhang & Chen, 2018) uses a graph neural network to learn heuristics for link prediction and defines the current state-of-the-art performance for the task. While SEAL can use node representations as input, it does not learn them from data.

3.3. Weight Prediction

Weight prediction is the task of predicting real valued weights on the edges of a network and thus similar to stationary flow prediction as defined in Sec. 4. However, we would like to highlight one conceptual and one practical difference between prediction of network weights and network flow. Conceptually, edge weights indicate concepts like connection strengths, trust and opinion. Flow and transactions, on the other hand, indicate that something finite, e.g. currency, has been transferred from one node to another. Multiple transactions over an edge may result in zero net flow, which should not imply connection strength zero. Practically, weights are assumed to be normalisable to a bounded range, e.g. $[-1, 1]$, (Kumar et al., 2016; Zhu et al., 2016; Hou & Holder, 2018; Agrawal & de Alfaro, 2019). Normalising transactions which span multiple orders of magnitude and contain outliers to a compact interval is non-trivial, e.g. the test data may contain transactions much larger than the largest transaction in the training data. For these reasons, we consider the stationary flow prediction task to be different from the weight prediction task.

(Kumar et al., 2016) propose to use a fairness and goodness score calculated for each node in the network. These scores are multiplied to compute edge weights which are interpreted as trust. (Zhu et al., 2016) instead use neighbourhood set intersections for computing weights and (Agrawal & de Alfaro, 2019) use path aggregations combined with a neural network to perform joint weight and link prediction. The model proposed by (Hou & Holder, 2018) is most similar to our model in the sense that it also learns node embeddings to predict edge weights. However, their embeddings are passed through a two layer MLP, unlike our model which is derived from gradient flow models, see sections 4.1 and 5.

4. Stationary Flow Prediction

Automatic representation learning requires a loss function to be minimised, commonly based on a specific task, e.g. classification loss for supervised learning (LeCun et al., 2015) or reconstruction loss for auto-encoders (Hinton & Salakhutdinov, 2006) and k-means (Murphy, 2012, p.356). Similarly, we define the stationary flow prediction task for network flow data.

Consider an undirected graph $G = (V, E)$ without self-loops, vertex set V, ($n = |V|$) and edge set E, ($m = |E|$). Without loss of generality, we impose a canonical direction on the edges, i.e. each edge is directed from lower to higher indexed nodes, as is standard in flow analysis on networks (Lim, 2020). A flow on the network can then be defined as a vector of values $\mathbf{y} \in \mathbb{R}^m$ where each component $y^{(ij)}$ expresses the magnitude and direction of a flow on edge ij relative to the canonical direction. Furthermore, we also assume that G is connected since each connected component can be treated separately.

The stationary flow prediction task, which we refer to as *flow prediction* here for brevity, is defined as a missing value inference task. It is assumed that an incomplete flow $\mathbf{y}_{\text{train}}$ is observed, only containing values for edges $ij \in E_{\text{train}} \subset E$ and the task is to predict the missing values on edges $E_{\text{test}} = E \setminus E_{\text{train}}$. Since our aim is to learn node representations, we further restrict this task by only considering models on the form $f^{(ij)} = f(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, where $\mathbf{x}^{(i)}$ is a vector representation of node i. This task is stationary since the time component of transaction data is ignored to simplify the problem.

We note that stationary flow prediction is not only a tool for learning node representations, but has potential application in its own right, e.g. filling in missing data or using deviations from predictions as an anomaly score.

4.1. Gradient Model

Note that any flow can be decomposed into a sum of a gradient flow and a divergence-free (i.e. in $\ker(\text{div})$) flow, see (Lim, 2020). By defining the Edge-Laplacian Δ_1 as in (Lim, 2020), note that this divergence-free part can furthermore be decomposed (by Equation 3.5, (Lim, 2020)) into a harmonic part and a part in the image of curl^* by the Helmholtz decomposition. Of importance for our application is that any flow on a graph can be expressed as $\mathbf{y} = \mathbf{y}_{\text{grad}} + \mathbf{y}_{\text{div}}$, the two terms being orthogonal and $\mathbf{y}_{\text{grad}}$ given by an element in the image of the gradient function,

defined as the natural extension of the gradient on a discrete domain to the edges of a graph,

$$(\text{grad } z)(ij) = y^{(ij)}_{\text{grad}} = z^{(j)} - z^{(i)} \quad \text{for } ij \in E. \tag{1}$$

Here, the $z^{(i)}$ are real scalars called node potentials. In the context of flow prediction, we view them as a 1D node representation for a gradient model formalised as

$$f^{(ij)}_{\text{grad}} = z^{(j)} - z^{(i)}. \tag{2}$$

A consequence of $\mathbf{y}_{\text{grad}}$ and $\mathbf{y}_{\text{div}}$ being orthogonal is that the node potentials can always be inferred from an observed flow, e.g. by minimising a squared error loss $\sum_{i,j}(y^{(ij)}_{\text{train}} - f^{(ij)}_{\text{grad}})^2$ using LSQR (Paige & Saunders, 1982; Virtanen et al., 2020). Note that the gradient flow model cannot capture the divergence-free part of the flow and this limits its application to flow data with significant divergence-free component, e.g. the used ethereum data with a divergence-free component norm of $\sim$ 42M and a gradient component norm of $\sim$ 6M. Moreover, this model can still overfit to training data if not regularised, like most machine learning models. In Sec. 7 & 8, this model is denoted as 'grad'.

4.2. Baseline Models

In addition to comparing our gated gradient model (Sec. 5) to the simpler gradient model described in Sec. 4.1, we include models using engineered features and node2vec features as baselines. As feature engineering, we compute the average sign of incoming/outgoing flows, the average value, the standard deviation, the sum of the absolute values of the flows, the mean of the absolute values and the node degree, making six features in total, $d = 6$. For this, we do not use the canonical orientation of the edges, but rather the convention that incoming flow is positive and outgoing flow is negative for each node. For training and prediction, the sign of the $y^{(ij)}$ of course adhere to the canonical orientation of the edges.

The node2vec features are learned using the implementation in PYTORCH GEOMETRIC (Fey & Lenssen, 2019) and we use the hyperparameters $d = 128$, $r = 10$, $l = 80$ and $k = 10$ as proposed by (Grover & Leskovec, 2016). Moreover, we use $p = 1$, and for the ethereum data we tune q using values $\{0.5, 1, 2\}$ while $q = 1$ for the synthetic flows. For flow prediction using either of these feature sets, we use a two layer deep MLP, similar to (Hou & Holder, 2018). The MLP received the feature vectors $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ to predict the flow on edge ij, and each layer has $2d$ units. This model will be referred to as 'dnn2' combined with either 'feat. engi.' or 'f.e.' for the engineered features or 'n2v' for the node2vec embeddings.

The fairness-goodness weight prediction model by (Kumar et al., 2016) is also included as a baseline since an easy-to-adapt implementation was readily available. This method requires each flow value to be in the range $[-1, 1]$. Since the largest flow value is many orders of magnitude larger than the median flow value, we apply a modified version of the Yeo-Johnson power transform (Yeo & Johnson, 2000) before normalising the flow values, $T(y) = \text{sgn}(y)\log_{10}(1+a|y|)$, where $a \in \{1, 10, 100, 1000\}$ is a hyperparameter. This model will be referred to as 'Kumar et. al.' in the results.

5. Gated Gradient Model

The representation capacity of the gradient flow model in Eq. (2) is restricted by the scalar valued node potentials since the gradient flow on one edge ij cannot be altered without affecting the flow on other neighbours of i or j. To address this, we propose K potentials on each vertex, denoted by the vectors $\mathbf{z}^{(i)} \in \mathbb{R}^K$. For this to work, we also introduce a gate function, $\bar{\sigma} : \mathbb{R}^K \times \mathbb{R}^K \mapsto [0, 1]^K$, which can modulate the flow for each edge and component of $\mathbf{z}$. This gated gradient flow model is expressed as

$$f^{(ij)} = \bar{\sigma}(\mathbf{u}^{(i)}, \mathbf{u}^{(j)})^T \left(\mathbf{z}^{(j)} - \mathbf{z}^{(i)}\right). \tag{3}$$

The following theorem helps us establish that the gated gradient model can approximate any network flow, unlike the gradient model.

Theorem 1. *Let $G = (V, E)$ be any graph following the specifications in Sec. 4, $\mathbf{y}$ an edge flow vector with $y_{\max} = \| \mathbf{y} \|_\infty$ and let $\sigma(\mathbf{u}^{(i)}, \mathbf{u}^{(j)}) : \mathbb{R}^K \times \mathbb{R}^K \mapsto [0, 1]^K$ denote a function in a function class containing elements able to map some set of parameter pairs $(\mathbf{u}^{(i)}, \mathbf{u}^{(j)})$ for $i, j \in \{1, \dots, n\}$ to one-hot vectors $\mathbb{1}_{k_{ij}}$ with bounded error for any fixed choice of $k_{ij} \in \{1, \dots, K\}$ for $i, j \in \{1, \dots, n\}$, i.e. $\sigma(\mathbf{u}^{(i)}, \mathbf{u}^{(j)}) = \mathbb{1}_{k_{ij}} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is an error vector with $\|\boldsymbol{\epsilon}\|_\infty < \epsilon$. Then, for $K = 2\Delta(G)$, σ can be chosen in this function class together with parameters $\mathbf{z}^{(i)} \in \mathbb{R}^K$, such that the resulting gated flow model, Eq. (3), is able to approximate the flow $\mathbf{y}$ such that*

$$\| \mathbf{y} - \mathbf{f} \|_\infty \leq 4\epsilon\Delta(G) y_{\max},$$

where $\Delta(G)$ is the maximum degree of the graph.

Proof. We provide a proof by construction. To assign flow values to each edge, iterate over the nodes in order by node index. Let C be the set of completed nodes, meaning that flow values have been assigned to each edge incident to these nodes. Furthermore, let $Q = V \setminus C$.

Now consider assigning flow values to the node v_i. If v_i already has flow values assigned to each edge, move v_i to C and continue with v_{i+1}. Otherwise, v_i has c neighbours in C and q neighbours in Q, and thus c completed edges and q edges without assigned flow values. Furthermore, this means that c components of $\mathbf{z}^{(i)}$ will have assigned values

and thus $K - c = 2\Delta(G) - c$ components are unassigned. Let J_i be the set of component indices with unassigned values of node v_i. Assign 0 to these $K - c$ components of $\mathbf{z}^{(i)}$.

Each of the neighbouring nodes $v_j \in Q$ will have at least $K - (\Delta(G) - 1) = \Delta(G) + 1$ unassigned components of $\mathbf{z}^{(j)}$. For each v_j, assign $z_l^{(j)} = y^{(ij)}$ for a component index $l \in J_i \cap J_j$, remove l from J_i and J_j, and let $\bar{\sigma}(i,j) = \bar{\sigma}(j,i)$ approximate $\mathbb{1}_l$. Note that $c + q \leq \Delta(G)$ and thus $|J_i| \geq \Delta(G)$. And since $|J_j| \geq \Delta(G)+1$ and $K = 2\Delta(G)$, such an index l will be available for every node v_j by the pigeonhole principle. Finally, move v_i to C.

To obtain the error bound, note that the flow over each edge can be expressed as

$$f^{(ij)} = (\mathbb{1}_{l_{ij}} + \boldsymbol{\epsilon})^T(\mathbf{z}^{(j)} - \mathbf{z}^{(i)}) = y^{(ij)} + \boldsymbol{\epsilon}^T(\mathbf{z}^{(j)} - \mathbf{z}^{(i)}).$$

Since $z_k^{(i)} \in [-y_{\max}, y_{\max}]$ and $\varepsilon_k \in [-\epsilon, \epsilon]$ we have

$$|y^{(ij)} - f^{(ij)}| = |\boldsymbol{\epsilon}^T(\mathbf{z}^{(j)} - \mathbf{z}^{(i)})| \leq 2\epsilon K y_{\max}.$$

□

Note that if $\bar{\sigma}$ is a universal approximator with range $(0, 1)$, e.g. a sufficiently large deep neural network with a sigmoid function output layer, Theorem 1 implies that the resulting gated gradient model is a universal approximator for flows on graphs. The vectors $\{\mathbf{u}^{(i)}\}_i$ could either consist of vertex data or, like $\{\mathbf{z}^{(i)}\}_i$, be parameters which are learned via optimisation. There is no additional vertex data available in our case, so we let the $\{\mathbf{u}^{(i)}\}_i$ be parameters. In practice, overfitting is a larger issue than model bias for transaction data, so we choose a simple form for $\bar{\sigma}$:

$$[\bar{\sigma}(\mathbf{u}^{(i)}, \mathbf{u}^{(j)})]_k = \sigma(u_k^{(i)}, u_k^{(j)}) = \frac{1}{1 + e^{-(u_k^{(i)} + u_k^{(j)})}}. \quad (4)$$

This choice has two motivations: interpretability (see Sec. 5.1) and numerical practicality, namely that its gradients are non-zero at the origin. Also note that the gated model, Eq. (3), reduces to the gradient flow model, Eq. (2), if $\mathbf{u}^{(i)}$ is a constant. In the future, other forms for $\bar{\sigma}$ should be explored.

The space complexity of the gated flow model is $\mathcal{O}(Kn)$, where K is in practice typically a small fixed constant, $K \leq 3$ in our case. The model is hence linear and very space efficient for large graphs.

5.1. Model Interpretation

Transactions are governed by human incentive which does not closely follow any known physical law. Nevertheless, the price dynamics on trade networks can be modelled by combining supply-demand dynamics with game theory (Kakade et al., 2004; 2005), and likewise, stability of commodity transport networks can be analysed using current laws under an optimal supply-demand network assumption (Rubido et al., 2014). This motivates us to also interpret the gated gradient model from a supply-demand perspective, in which the $\mathbf{z}^{(i)}$ represent supply/demand indicators for K different goods and services, with higher values indicating larger supply and low values lack of supply or demand. Note that the model defines positive flow as directed from low supply to high supply since the observed data tends to be the payment for a product or service.

The $[\bar{\sigma}(\mathbf{u}^{(i)}, \mathbf{u}^{(j)})]_k$ are then a form of impedance for transactions of the kth product/service along edge ij. With our choice of $\bar{\sigma}$ in Eq. (4), the parameters $u_k^{(i)}$ indicate vertex i's willingness to trade in product k, with negative values indicating reluctance and positive values eagerness.

6. Learning with Multiscale Transaction Data

6.1. Loss Function

Observe in Fig. 1 that the densities of transaction amounts span multiple orders of magnitude and appear to be relatively heavy-tailed. To better account for this situation, we propose to utilize the following custom relative error $\varepsilon : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$:

$$\varepsilon(y, f) = (y - f)^2/|y|, \quad \frac{\partial \varepsilon}{\partial f}(y, f) = 2(f - y)/|y|. \quad (5)$$

This error has the, we believe, desirable property that its partial derivative above is invariant under a common scaling of the magnitude of y and f by a constant $\alpha > 0$ while $\varepsilon(y, f)$ itself still decreases with decreasing overall scaling factor $\alpha > 0$, as is the case with the standard squared error. We believe this to be beneficial to mitigate the otherwise dramatic variations in gradient norms for the classical squared error over the multiple magnitudes of scale observed in the input data. To avoid division by zero and numerical instabilities, we furthermore truncate the denominator of ε at a threshold specified per dataset which has to be chosen in relation to the desired precision.

To address the heavy tails, we propose using a Student-t log-likelihood (Lange et al., 1989) of the specific form

$$\ell(\mathbf{y}, \mathbf{f}) = \frac{1 + \nu}{2m} \sum_{\substack{i,j \in E, \\ i<j}} \log\left(1 + \varepsilon(y^{(ij)}, f^{(ij)})/\nu\right), \quad (6)$$

where ν is a parameter (also referred to as dof/degree of freedom in Sec. 8) for the Student-t distribution and the denominator of ε is truncated at a constant $\tau > 0$ as mentioned above. The probabilistic motivation for this loss is the assumption that our errors are well modelled as a Student-t distribution scaled by the square-root of the absolute value of the target variables.

The choice of ν allows for controlling the growth of log terms in the sum above. Based on our preliminary experiments, setting ν to the median of $|y^{(ij)}_{\text{train}}|$ provided a balance that did not over-emphasize the largest ε values occurring in the loss function.

As mentioned in Sec. 5, overfitting is a challenge to generalisation in the flow prediction task. To address this, we introduced additional regularisation using the L1-loss on the model parameter. Compared to the L2-loss, the L1 loss assigns lower penalty to large parameter values. This is especially important for the $\mathbf{z}^{(i)}$ parameters since they need to be large to successfully reproduce large flows. The complete loss function, where λ_z and λ_u are regularisation strength hyperparameters, is

$$L(\mathbf{z}, \mathbf{u}) = \ell(\mathbf{y}, \mathbf{f}(\mathbf{z}, \mathbf{u})) + \frac{\lambda_z}{nK} \| \mathbf{z} \|_1 + \frac{\lambda_u}{nK} \| \mathbf{u} \|_1. \quad (7)$$

6.2. Optimisation

The gated flow model is implemented in PYTORCH (Paszke et al., 2019) and the loss function is minimised using the library's implementation of AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017), using learning rate 0.01, weight decay 0.01 and amsgrad (Reddi et al., 2019). Full batches are used and the models are trained until convergence, defined by a relative tolerance threshold on the training loss set to 10^{-5}. However, preliminary experiments indicated that the loss function has stationary points not necessarily close to the global minimum and the optimiser tended to get stuck with both high training and validation losses. To mitigate this, we investigate different initialisation strategies since these have proved important for non-convex optimisation of neural networks (Sutskever et al., 2013).

The poor performance observed when using noise from a normal distribution to initialise $\mathbf{z}$ led to the conclusion that the shape of the loss function makes it difficult for the $\mathbf{z}$ to move away from the origin. To alleviate this, we took advantage of the efficiency in inferring $z^{(i)}$ for the gradient model, Eq. (2), when minimising a MSE loss using LSQR (Paige & Saunders, 1982; Virtanen et al., 2020), and subsequently set $z^{(i)}_k = 2z^{(i)}$ for each k for the gated model. The factor 2 compensates for the factor 0.5 coming from $\sigma(u^{(i)}_k, u^{(j)}_k)$ as $\mathbf{u}$ is initialised at the origin. This strategy, however, has the issue that the model is initialised in a subspace of the parameter space, and by looking at the gradients of gated model (Eq. (3)) one sees that it is impossible to escape this subspace. We address this by adding noise to the $\mathbf{z}$ after first initialising using LSQR, where the noise is scaled by the magnitude of each $z^{(i)}_k$. We call this third initialisation strategy LSQR+ and the three strategies are compared using synthetic data in Sec. 8.

6.3. Preprocessing and Flow Prediction Evaluation

The transaction graphs are preprocessed by first extracting the largest connected component and then trimming the graph, meaning that all nodes of degree one are iteratively removed. This is done since it is not possible to learn to predict the flow to/from a node of degree one without additional vertex data. The edges directed from lower to higher indexed nodes, and the sign of flow values are adjusted accordingly. The net flow is calculated in the case of transactions going in opposite directions.

To create training, validation and test splits of the flow, a random spanning tree of the graph is created and all edges on the tree are put into the training set. The spanning tree is necessary to ensure that the different splits belong to the same connected graph. Additional edges are then added to the training set until it reaches the desired size, e.g. 80% of all edges, and the remaining edges are split into validation and test sets. The graph preprocessing is done using GRAPH-TOOL (Peixoto, 2014) and the splits using SCIPY (Virtanen et al., 2020).

To evaluate the flow prediction performance across multiple scales, the relative error

$$\delta^{(ij)} = |y^{(ij)} - f^{(ij)}|_{10^{-6}} / |y^{(ij)}|_{10^{-6}} \quad (8)$$

For detailed analysis of the prediction performance, the cumulative distributions of $\delta^{(ij)}$ across all test edges are studied, see Figs. 2 and 5. Note that $\delta^{(ij)} = 0$ can be achieved by trivially predicting $f^{(ij)} = 0$. The numerator and denominator in Eq. (8) are truncated at $1\text{e}{-6}$ to avoid complications with division by zero. To summerise the error into a single value, we use the median absolute error.

7. Experiments on Ethereum Transactions

Google's BigQuery was used to extract all ethereum transactions between 2018-06-01 and 2020-12-01 with a value greater than zero (in particular contract transactions without additional ether were excluded). The transactions were grouped by directed edges and filtered by removing edges with fewer than 10 transactions over the time period. The time period was chosen based on the relative price stability of around 100$/eth to 500$/eth. All transactions were converted from wei to ether. The resulting transaction graph contains 452862 vertices and 1107858 edges, and the distribution of amounts can be seen in Fig. 1.

The edges are split into a train set (70%) for optimisation, a validation set (15%) for hyperparameter selection and a test set (15%) for evaluation. LSQR+ initialisation is used for the gated and grad models and we perform a grid search over the regularisation hyperparameters, λ_u and λ_z. For the gated model we additionally include the dimension K in the search. We use dimension $\{1, 2, 3\}$ for K and

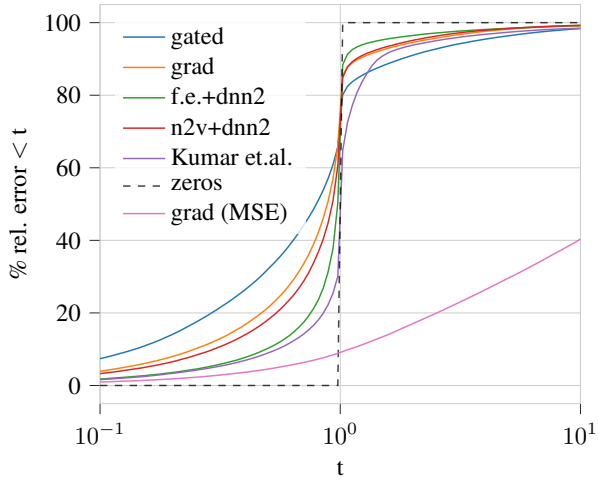

Figure 2. Cumulative relative error distributions for the ethereum data. The trivial prediction of zero for all edges is shown as a dashed black curve. The larger the area under the curve to the left of $t = 1$ the better.

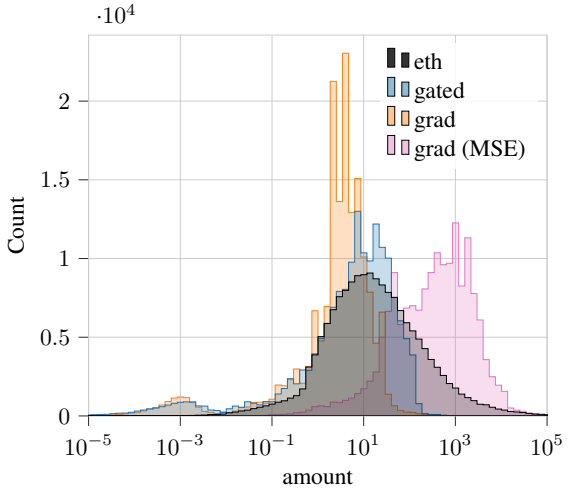

Figure 3. The histograms of ethereum transaction amounts and predictions by the gradient models on the test set.

values $\{3., 1, 0.3, 0.1\}$ for λ_u and λ_z, resulting in a grid search over 48 models. Each model takes around 30 min to train on a consumer grade GPU (e.g. Nvidia RTX 2070). Hyperparameter search is also performed for the baseline models, using same regularisation strengths for the 'dnn2' models, including q in the search for the node2vec feature and tuning a for the 'Kumar et.al.' model.

For the gated model, $\lambda_u = \lambda_z = 0.3$ and $K = 1$ perform best with the model overfitting for higher dimensions and lower regularisation strengths. For the gradient model $\lambda_z = 0.3$ is best for the 'dnn2' models with $q = 0.5$ for 'n2v'. For 'Kumar et. al.', $a = 1000$ is best.

In Fig. 2, the cumulative distributions of the relative errors on the test split are shown for the different models. Our gated model performs best, in the sense that it has the largest mass of errors smaller than one, followed by the gradient flow model and the 'dnn2' baseline using node2vec features. The 'dnn2' model using the engineered features and the weight prediction model by (Kumar et al., 2016) lie close to the trivial performance indicated by the zeros curve. This is also apparent in Table 1 where the median absolute errors are shown. Also included is the gradient model when trained with mean square error loss instead of our robust multi-scale loss. Its performance in terms of relative error and median absolute error is very poor since it overfits to the few very large transactions.

The transaction amount densities for the test data and the models' predictions provide further insight into their behaviour, see Fig. 3. Again, we see how the gradient model trained using a squared loss is biased towards large amounts. Conversely, both models trained using the multi-scale loss with L1 regularisation are biased towards smaller amounts and fail to capture the right hand tail of the distribution.

8. Experiments on Synthetic Flow Data

As discussed in Sec. 2, publicly available transaction datasets outside of cryptocurrencies are rare and machine learning research in AML will likely need to rely on synthesised transaction data to a large extent (Oeben et al., 2019). Synthetic datasets also come with the advantage that the ground truth about the data generation process is known. We choose to use 60 different synthetic transaction network datasets to further investigate the properties of the gradient models and baselines. The datasets are generated using three different graphs, and two different parameter distributions for generating flows. For each of these six combinations, we sample 10 sets of flows.

The aim of these experiments is to verify that the flow prediction results for the ethereum data extend to other graphs, flow distributions and flow samples, and, since we have access to the ground truth flow generative distribution, to investigate if the gated flow model is able to learn qualitative information about this distribution. Additionally, we perform an ablation study to see how L1 regularisation and the initialisation strategies discussed in Sec. 6.2 affect training and validation errors.

Table 1. Median absolute errors on reserved test transactions for the ethereum dataset. 'Zeros' is the trivial model of prediction zero for each edge.

GATED	GRAD	F.E.+ DNN2	N2V+ DNN2	KUMAR ET. AL.	ZEROS	GRAD (MSE)
9.34	10.39	13.10	11.25	14.19	14.61	397

Table 2. Mean and standard deviations of the $\log_{10}$ median relative error for the validation splits on the synthetic flow data. Lower values are better. The means and standard deviations are calculated over 10 generated flows for each graph.

	UNIMODAL			MULTIMODAL		
MODELS	CORA	BITCOIN	COMPLETE	CORA	BITCOIN	COMPLETE
GATED	$\mathbf{-0.07 \pm 0.00}$	-0.06 ± 0.01	$\mathbf{-0.28 \pm 0.07}$	$\mathbf{-1.32 \pm 0.02}$	$\mathbf{-0.81 \pm 0.06}$	$\mathbf{-2.22 \pm 0.18}$
GRAD	$\mathbf{-0.07 \pm 0.00}$	$\mathbf{-0.07 \pm 0.01}$	-0.14 ± 0.03	-0.57 ± 0.00	-0.56 ± 0.02	-0.62 ± 0.04
F.E.+DNN2	$\mathbf{-0.07 \pm 0.01}$	-0.04 ± 0.01	-0.12 ± 0.03	-0.63 ± 0.06	-0.01 ± 0.02	-1.20 ± 0.15
N2V+DNN2	-0.00 ± 0.00	-0.02 ± 0.01	-0.26 ± 0.05	-0.00 ± 0.00	-0.00 ± 0.00	-0.92 ± 0.48
KUMAR ET.AL.	-0.01 ± 0.00	-0.00 ± 0.00	-0.02 ± 0.01	-0.00 ± 0.00	-0.00 ± 0.00	-0.00 ± 0.00

The three graphs used are the cora citation graph (McCallum et al., 2000) which is commonly used in graph-machine learning research (Kipf & Welling, 2016; Rendsburg et al., 2020), a version of the bitcoin transaction graph (Fire & Guestrin, 2017) (which only contains the links and no actual transaction amounts), and the complete graph on 40 vertices (780 edges). After the preprocessing steps, the cora graph contains 19727 nodes and 85718 edges and the bitcoin graph 12805 nodes and 36327 edges.

To generate flows, we sample values for $\mathbf{z}^{(i)}$ and $\mathbf{u}^{(i)}$ and then run a forward pass through the gated flow model, Eq. (3). We consider a unimodal and a multimodal setting for the parameter distributions. The unimodal setting is designed to mimic the transaction amount densities observed for real data, see Fig. 1 for a comparison. This is achieved by using student-t distributions with mean 0, scale 100 and dof 2 for the $\mathbf{z}^{(i)}$, and mean -2, scale 1 and dof 4 for $\mathbf{u}^{(i)}$, and $K = 3$. In the multimodal setting, mixture of student-t distributions are used for both $\mathbf{z}$ and $\mathbf{u}$, with three components and dof 2 for each. For $\mathbf{z}^{(i)}$, the components are centred at $(100, 100), (0, 0), (0, -50)$, and for $\mathbf{u}^{(i)}$ they are centred at $(4, 4), (0, -5), (-5, 1)$, and scaled by 0.01. We use $K = 2$ and an example of sampled $\mathbf{z}^{(i)}$ are shown in Fig. 4. This setup results in a flow distribution with three sharp peaks: one large peak close to zero from small transactions within each component, one for transactions between component 0 and 1 and one for transactions between component 0 and 2. A visualisation is available in the supplementary material.

For each of the six setups, 10 different ground truth parameter sets are sampled and the 10 resulting flows are split into 80-20 training/validation sets. The experiments are performed jointly with the ablation study which consists of six steps. First, all models are trained without regularisation and noise from a normal distribution is used as initialisation for the gradient models. Then the gradient models are retained using LSQR and LSQR+. Using LSQR+ for the following steps, L1 regularisation is added to $\mathbf{u}^{(i)}$ only, then to $\mathbf{z}^{(i)}$ only, and finally to both parameter sets and the weights of the baseline models. Regularisation strengths $\lambda_z = \lambda_u = 0.5$ were used in the unimodal case and 0.05 in the multimodal one. The training time ranges from 1-100s per ablation step, depending on the data and model, on a customer grade CPU and GPU.

Table 3. Clustering agreement scores averaged over 10 flow samples. K-means clustering is applied to the ground truth and model representations separately and the score is the cluster assignment overlap. Scores lie in $[0, 1]$, higher is better.

MODEL	CORA	BITCOIN	COMPLETE
GATED	0.92 ± 0.00	0.90 ± 0.03	0.95 ± 0.10
GRAD	0.69 ± 0.00	0.71 ± 0.04	0.57 ± 0.05
FEAT. ENGI.	0.68 ± 0.00	0.62 ± 0.01	1.00 ± 0.00
NODE2VEC	0.34 ± 0.00	0.34 ± 0.00	0.41 ± 0.04

Table 2 shows the flow prediction performance results on the validation sets in terms of means with standard deviations over the 10 different flows. The performance is measured using the $\log_{10}$ median relative error, i.e. $\log_{10} \text{median}_{ij}\ \delta^{(ij)}$. These errors are visualised using vertical, dashed, coloured lines in Fig. 5 for one flow sample. In the multimodal setting, the gated flow model outperforms the other models by a wide margin and achieves low relative errors on average for all three graphs. The unimodal setting appears to be more difficult with models generally having higher errors. Looking at the training set errors, we observe larger discrepancies between the validation set errors in the unimodal setting, indicating that overfitting is a partial reason to the larger errors. We hypothesise that the discrete nature of the multimodal setting reduces the number of symmetries and local minima of the training loss surface, resulting in better generalisation. We further observe that overfitting is less of an issue for the complete graph and that all models overfit more on the bitcoin graph compared to cora. The reason is believed to relate to the graph sparsity or possibly the clustering coefficient. Further analysis is left as future work.

To determine if the gated gradient model can learn qualitative information about the multimodal distribution, we first look at one example visualised in Fig. 4. The figure

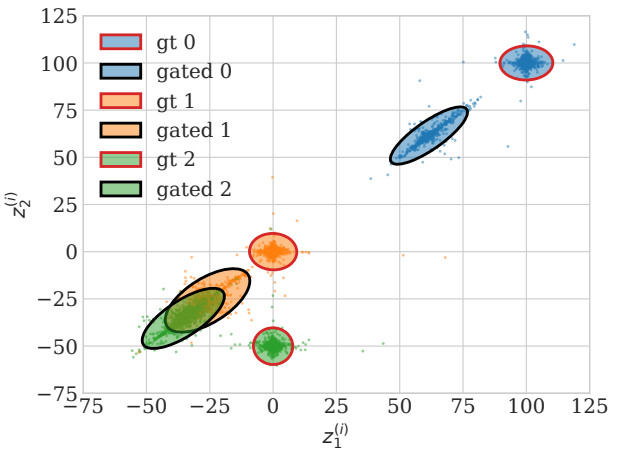

Figure 4. One sample set of $\{\mathbf{z}^{(i)}\}_i$ from the multimodal ground truth distribution (red ellipses) for generating synthetic flow, and the learned $\mathbf{z}^{(i)}$ parameters (black ellipses) on the cora graph.

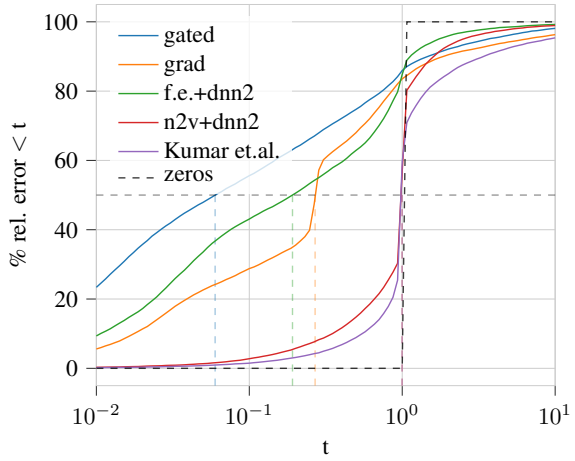

Figure 5. Cumulative relative error distributions over the validation set of one synthetic flow sample generated from the mulitmodal distribution on the cora graph. Median relative errors are marked by dashed, coloured, vertical lines.

shows the ground truth and learned $\mathbf{z}^{(i)}$ parameters for the same multimodal flow sample underlying the error curves in Fig. 5. The different components are highlighted using ellipses with red border for the ground truth components and black border for the learned representations. We see that the gated gradient model is able to separate the three components albeit along the diagonal on which the parameters are initialised. To quantify the performance across all the flow samples, we calculate a clustering agreement score, see Table 3. This score is calculated by running k-means clustering separately on the ground truth and learned $\mathbf{z}^{(i)}$ and computing the fraction of nodes which get assigned to the same cluster. The gated model receives the highest score on the cora and bitcoin graphs and is only marginally worse than the hand-engineered features which receive a perfect score on the complete graph.

The ablation study showed that LSQR+ performs the best out of the three initialisation strategies. LSQR performs similarly to LSQR+ on the multimodal data, but worse on the unimodal data, and vice versa for the normal noise initialisation. Since overfitting is the bottleneck for the unimodal data, L1 regularisation improves the validation errors slightly for the cora and bitcoin graph, while having a detrimental effect in the multimodal case. The complete results are displayed in the supplementary materials.

9. Conclusions

We have extended the gradient flow model to a gated version capable of learning node representations automatically from network flow data, not requiring hand-engineered features. We have shown that our gated extension can be interpreted as a universal approximator for flows on graphs, unlike the gradient flow model. Furthermore, we propose a robust, multi-scale loss function to be used for training flow prediction models on large cash transactions datasets, which both span multiple order of magnitude and contain outliers. When applied to ethereum transaction data, our gated model outperforms both the gradient model and neural networks using engineered features. These results hold when the models are trained on 60 synthetic network flow datasets.

For the synthetic data, we observe that the learned node representations capture some qualitative properties of the flow generating distributions, e.g. the different modes of the multimodal case. This is encouraging for the prospects of using the learned representations for downstream tasks, e.g. anomaly detection for financial crime prevention.

The main bottleneck for flow prediction generalisation appears to be overfitting and experiments suggest both the flow distribution and graph topology affect the overfitting severity. We hypothesise that the risk of overfitting generally increases for sparse graphs with lower global clustering coefficient. Applying L1 regularisation does reduce the overfitting, but we assert that there exist less invasive regularisation methods which would allow to train larger gated gradient flow models, cf. dropout for DNNs (Srivastava et al., 2014).

Other potential areas for future research are unification of link and stationary flow prediction into a single model, and to extend the gated flow model to a full generative model.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Agrawal, R. and de Alfaro, L. Learning edge properties in graphs from path aggregations. In *The World Wide Web Conference*, pp. 15–25, 2019.

Albert, R., Albert, I., and Nakarado, G. L. Structural vulnerability of the north american power grid. *Physical review E*, 69(2):025103, 2004.

Barabâsi, A.-L., Jeong, H., Néda, Z., Ravasz, E., Schubert, A., and Vicsek, T. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3-4):590–614, 2002.

Bhattacharyya, S., Jha, S., Tharakunnel, K., and Westland, J. C. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, 2011. ISSN 0167-9236.

Chen, T., Li, Z., Zhu, Y., Chen, J., Luo, X., Lui, J. C.-S., Lin, X., and Zhang, X. Understanding Ethereum via Graph Analysis. *ACM Transactions on Internet Technology*, 20 (2):18:1–18:32, 2020.

Chen, Z., Teoh, E. N., Nazir, A., Karuppiah, E. K., Lam, K. S., et al. Machine learning techniques for anti-money laundering (aml) solutions in suspicious transaction detection: a review. *Knowledge and Information Systems*, 57(2):245–285, 2018.

Di Francesco Maesa, D., Marino, A., and Ricci, L. Data-driven analysis of Bitcoin properties: Exploiting the users graph. *International Journal of Data Science and Analytics*, 6(1):63–80, 2018.

European Parliament and Council. Directive (EU) 2018/843 of the European Parliament and of the Council of 30 May 2018 amending Directive (EU) 2015/849 on the prevention of the use of the financial system for the purposes of money laundering or terrorist financing, and amending Directives 2009/138/EC and 2013/36/EU (Text with EEA relevance), 2018. URL http://data.europa.eu/eli/dir/2018/843/oj/eng. http://data.europa.eu/eli/dir/2018/843/oj/eng.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Fire, M. and Guestrin, C. The rise and fall of network stars. *arXiv preprint arXiv:1706.06690*, 2017.

Fronzetti Colladon, A. and Remondi, E. Using social network analysis to prevent money laundering. *Expert Systems with Applications*, 67:49 – 58, 2017.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Hou, Y. and Holder, L. B. Link weight prediction with node embeddings. Submitted for review, ICLR18, 2018.

Kakade, S. M., Kearns, M., and Ortiz, L. E. Graphical economics. In *International Conference on Computational Learning Theory*, pp. 17–32. Springer, 2004.

Kakade, S. M., Kearns, M., Ortiz, L. E., Pemantle, R., and Suri, S. Economic properties of social networks. In Saul, L., Weiss, Y., and Bottou, L. (eds.), *Advances in Neural Information Processing Systems*, volume 17, pp. 633–640. MIT Press, 2005.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kumar, S., Spezzano, F., Subrahmanian, V. S., and Faloutsos, C. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 221–230, 2016.

Lange, K. L., Little, R. J. A., and Taylor, J. M. G. Robust statistical modeling using the t distribution. *Journal of the American Statistical Association*, 84(408):881–896, 1989. ISSN 01621459.

Le Khac, N. A. and Kechadi, M.-T. Application of data mining for anti-money laundering detection: A case study. In *2010 IEEE International Conference on Data Mining Workshops*, pp. 577–584, 2010.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Lewis, K., Kaufman, J., Gonzalez, M., Wimmer, A., and Christakis, N. Tastes, ties, and time: A new social network dataset using facebook.com. *Social networks*, 30 (4):330–342, 2008.

Lim, L.-H. Hodge Laplacians on graphs. *Siam Review*, 62 (3):685–715, 2020.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Newman, M. *Networks*. Oxford university press, 2018.

Oeben, M., Goudsmit, J., and Marchiori, E. Prerequisites and AI challenges for model-based Anti-Money Laundering. In *AI for Social Good - IJCAI 2019 Workshop*, pp. 1–4, 2019.

Paige, C. C. and Saunders, M. A. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1): 43–71, 1982.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Peixoto, T. P. The graph-tool python library. *figshare*, 2014. URL http://figshare.com/articles/graph_tool/1164194.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Rendsburg, L., Heidrich, H., and Von Luxburg, U. NetGAN without GAN: From random walks to low-rank approximations. In *International Conference on Machine Learning*, pp. 8073–8082. PMLR, 2020.

Ron, D. and Shamir, A. Quantitative Analysis of the Full Bitcoin Transaction Graph. In Sadeghi, A.-R. (ed.), *Financial Cryptography and Data Security*, pp. 6–24. Springer, 2013.

Rubido, N., Grebogi, C., and Baptista, M. S. Resiliently evolving supply-demand networks. *Phys. Rev. E*, 89: 012801, Jan 2014.

Sarkar, P., Chakrabarti, D., and Moore, A. W. Theoretical justification of popular link prediction heuristics. In *IJCAI proceedings-international joint conference on artificial intelligence*, volume 22, pp. 2722. Citeseer, 2011.

Savage, D., Wang, Q., Chou, P., Zhang, X., and Yu, X. Detection of money laundering groups using supervised learning in networks. *arXiv preprint arXiv:1608.00708*, 2016.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147. PMLR, 2013.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, 2015.

United Nations office on drugs and crime. UNODC report: Estimating illicit financial flows resulting from drug trafficking and other transnational organized crimes, 2011. https://www.unodc.org/unodc/en/frontpage/2011/October/illicit-money_-how-much-is-out-there.html.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Vujičić, D., Jagodić, D., and Ranđić, S. Blockchain technology, bitcoin, and ethereum: A brief overview. In *2018 17th international symposium infoteh-jahorina (infoteh)*, pp. 1–6. IEEE, 2018.

Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

Yeo, I.-K. and Johnson, R. A. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 557–564, 2017.

Zhu, B., Xia, Y., and Zhang, X.-J. Weight prediction in complex networks based on neighbor set. *Scientific reports*, 6(1):1–10, 2016.