My Thesis

Connor Smith

1 Abstract

About 200-300 words long

2 Introduction

A bit of background about the topic. Some information about the current knowledge. The aim of your research (the gap in knowledge that prompted you to write the thesis).

2.1 Terminology?

3 Literature Review

3.1 Traditional Online Social Network Modelling

There has been a lot of research into what governs how edges are formed in an online social network. Some examples of what affects edge formation are a nodes' "age", how much time has passed since it first made an edge; it's edge creation rate is highest shortly after joining and decreases over time. The majority of edge creation in the early stages of a network is driven by new nodes arriving in the network, however this decreases significantly as the network matures. Edges formation follows preferential attachment, but this strength decreases over time as the network expands [4]. Research also indicates that, while preferential attachment (a node is most likely to form an edge with the highest in-degree) can be used to model edge formation in online social networks, it alone does not seem to account for the low number of hops that empirical data suggests edges usually form along. [3] suggest that preferential attachment with node distance as a tie breaker results in a distribution of node distances close to what was seen empirically from their data.

A potential for improvement in this research [[3], [4]] is that they did not model the decay of edges (to the best of my knowledge) and they used a friendship/follow as their edge, which, while may be meaningful in the short term, I don't think is always meaningful after a given amount of time. [find source] A person may add a friend and never speak to them again (hence we use conversations together), this has limitations where a person can't always choose who they talk to, but it might provide a more granular (definition?) snapshot of interactions between users.

3.2 UDEs and NODEs

3.2.1 Theory

UDEs are a combination of a known, structural component of a system, and a universal approximator, often a neural network.[Find reference] Given the flexibility of neural networks, the use of UDEs is natural when modelling a process that is not well understood [On Neural ODEs]
Largely, the SciML ecosystem has been optimised for flexibility and efficiency with respect to models available and training performance. Given there does not seem to be any other ecosystem with this feature, it seemed like an obvious choice. What makes it good (Examples from [9])

3.2.2 Applications [Caution only glanced at these papers]

Whilst there has been much interest in implementing UDEs and neural ODEs, much of this work has been focussed on phisics informed neural network PINNs [12], [11], [13], [10], and on improving modelling of fluids [14] [15]. There has also been a considerable amount of research into converting these discrete step neural networks into continuous depth neural networks, sometimes also referred to as neural ODEs [8] [5] [6]. Along side this, and given UDEs have gained popularity in the last few years, there have been a number of studies exploring their usefulness in modeling the effect of restrictions due to COVID-19 on the virus' spread [7].

Although there has been a large amount of research into the usefulness of these types of models, to the best of this author's knowledge they have never been applied to the problem of predicting temporal social networks. This seems to be a natural fit for UDEs and neural ODEs, given we know relatively little about the processes that govern the evolution of online social networks, and the vast amounts of data that the providers of these services collect, data that can be used to train our neural networks.

Start by illustrating that there is a wide range of applications (Reasonably briefly for not relevant), but little research into UDEs aplicability for modeling how temporal networks behaveb

3.3 Dynamical Systems

3.4 SINDY

4 Methods

4.1 Embedding Our Network, Singular Value Decomposition

We used a singular value decomposition of each matrix to embed our network in a low dimensional space.

$$A = U\Sigma V'$$

Where $\Sigma_{ii} = \sigma_i$ and $\sigma_i > sigma_{i+1}$ and each σ_i is a non-negative square root of the matrices $AA', A'A$. U is a matrix made from the orthonormalised eigenvectors of AA' associated with σ_i, and V is a matrix made from the orthonormalised eigenvectors of $A'A$ associated with σ_i. Each row i of U and V represents the in and out position of node i respectively. This can be truncated at an arbitrary number of eigenpairs to form a low dimentional approximation to A.

$$A \approx \hat{U}\hat{\Sigma}\hat{V}'$$

Where $\hat{U}, \hat{\Sigma}, \hat{V}$ are the first d columns of U, Σ, V [1]. $\hat{U}\sqrt{\hat{\Sigma}}$ is still associated with the in edges of our nodes, and $\sqrt{\hat{\Sigma}}\hat{V}'$ is still associated with the out edges of our nodes, but now we have simplified our system whilst keeping as much information as possible due to our descending order of singular values. These two matrices form our d dimensional embedding.

The approximation given by this formula can be interpreted as a graph where the entry (i, j) of the matrix are the probability that an edge exists from $i \to j$.This approach creates a random dot product graph, a class of Latent Position Models [2]. Finding the embedding is not trivial; we need to find the eigenpairs of each of AA' and $A'A$. Because of the computational complexity we use the SVD function in Arpack.jl to approximate each matrix. This then changes the problem of discovering edge formation and destruction into learning a dynamical system.
It is important to note that the SVD is not unique. The distances between points will be the same for every embedding, but the orientation may be different, and so, if the SVD of each time step is not alligned, learning will be effectively impossible [¡- show this]. Since our approximation method is an iterative approach, to align each SVD, we can simply pick the same initial vector guess for each [probably worth getting a reference].

Once we have this embedding of points, we look to model the trajectories of each point. That is, with some input data, point location, distance to neighbours, etc, we want to model the ODE of each point. To do this we make use of the SciML ecosystem of packages in the Julia programming language [9]. To explore which architectures perform best for this system we then test various types.

With our trained NODE, we have a black box that approximates the system that governs the movement of Twitter users. To make this more interpretable we then look to decompose this into linear combinations of known functions using SINDY. For this process we need to define a search space of functions which the algorithm will use to find the best approximation to our NODE with.

Initially, we decided the best course of action would be to implement a toy example graph as a proof of concept. This would allow us to check that it is indeed possible to use neural networks to model the progession of temporal networks. It would also allow us to compare and contrast the use of a "known"

	training loss	computation time
super small		
small		
medium		

Table 1: Caption

function in our model, which will be combined with the neural network to model the evolution of the system over time.

[On Neural Ordinary Differential Equations ch. 2] Potential for starting with a portion of data to minimise getting stuck in local minima.

4.2 Testing Different Things

Before we begin training our model on the whole data set, we first perform a series of comparisons of different model setups on a subset of our data (1 node 0 knn, 15 nodes 5 knn, 50 nodes 20 knn). Each of these comparisons will be run for ?20? epochs and a ttraining data size of 25 weeks. We will compare both the accuracy and the training time, to develope a model that is likely to perform well on the whole data set. The base model we will be comparing against will have no proposed known component, will have 1 NN with ?2? hidden layers of width 16 each, with an activation function of ?tanh?. The results of this model are

4.3 Proposed Known Components

4.3.1 Everything Towards Zero

4.3.2 Including Time Component

4.4 Comparison of Methods on Deterministic Toy Graphs

4.4.1 Toy Graph

Toy Graph 1

Toy Graph 2
Our second toy network was created from a known vector field defined by

$$\frac{dx}{dt} = (x^2 + y^2 - \frac{1}{4})y \tag{1}$$

$$\frac{dy}{dt} = -(x^2 + y^2 - \frac{1}{4})x \tag{2}$$

With 4 initial points $(x, y) = (0.5, 0.5), (0.125, -0.125), (-0.5, 0.0), (-0.5, -0.5)$. One minus the euclidean distance between point i and j is the entry (i, j) in our associated adjacency matrix and corresponds to the probability of an edge existing between these two nodes. Having a network defined by such a system allows us to easily see whether our network (when run through SINDY) is recovering the correct dynamics of the movement of edges between nodes.

Explain how the graph is defined and moves over time

4.4.2 Tweaks to Method

4.4.3 Aligned vs Unaligned Embeddings

How does model accuracy, training time , etc change when the embedding is aligned compared to unaligned? Does alignment method matter?

4.4.4 Training on Embedded Data vsThe True Graph

Does training the model using the embedded data to calculate loss change the test loss of the model compared to us using the true network matrix?

4.4.5 One Neural Network Compared to One For In Edges and One For Out Edges

Does test error change if we have one NN for the whole graph compared to one for the L and one for the R embeddings? Does it have much impact on computing time? Keep total number of parameters the same (maybe ask Giulio)?

4.4.6 Random Noise

If we introduce a stochastic component, how does the model cope?

4.4.7 Different Dimension Embeddings

As we vary the number of dimensions we embed, how does accuracy and computation time change?

4.4.8 Training Loss Against Embedding Loss

Does the information captured in the SVD embedding correlate with the accuracy of the NN?

4.4.9 SINDY

If we include a SINDY step, does this improve the accuracy of our model at times far from the training data?

4.4.10 trying to predict self loops vs not

If we ignore the models prediction of self loops, does this improve the accuracy for the rest of the model?

4.5 Including "Known" Differential Equation Component

How does the inclusion of the known component effect the accuracy and training time of the model. From [3] Factors that have an impact on edge generation:

4.6 Including Other Information

Within the literature, there is evidence of the rate at which a new node creates edges depending on when it was first created, hence it may be worth including this, and other information, (for example the communities a user is in) as an input for the NN.

4.7 Variation in the known component

[9] Demonstrates that if we have a differential equation of the form

$$\dot{x} = f_1(x, y) + g_1(x, y) \tag{3}$$
$$\dot{y} = f_2(x, y) + g_2(x, y) \tag{4}$$

g_1, g_2 can be replaced with neural networks $U_1(\theta, x, y), U_2(\theta, x, y)$. These neural networks can be trained, and using some type of sparse identification (SINDY), the original g_1, g_2 can be recovered. This example was given using the LotkaVolterra equations in which we knew that f_1, f_2 were exactly correct. Given we know very little about the systems of differential equations that govern social networks, we wish to briefly explore what is recovered when the known functions f_1, f_2 are perturbed to $\hat{f}_1, \hat{f}_2$.

5 Thesis Results

5.1 Exploratory Analysis

- A brief overview of the data following the methodology of [4] and [3]. (Might need to find source code?)

6 Discussion

7 Conclusion

References

1. Golub, G. H. & Reinsch, C. in *Linear algebra* 134–151 (Springer, 1971).
2. Hoff, P. D., Raftery, A. E. & Handcock, M. S. Latent space approaches to social network analysis. *Journal of the american Statistical association* **97,** 1090–1098 (2002).
3. Garg, S., Gupta, T., Carlsson, N. & Mahanti, A. *Evolution of an online social aggregation network: an empirical study* in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (2009), 315–321.
4. Zhao, X. *et al. Multi-scale dynamics in a massive online social network* in *Proceedings of the 2012 Internet Measurement Conference* (2012), 171–184.
5. Poli, M. *et al.* Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532* (2019).
6. Cranmer, M. *et al. Discovering Symbolic Models from Deep Learning with Inductive Biases* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) **33** (Curran Associates, Inc., 2020), 17429–17442. `https://proceedings.neurips.cc/paper/2020/file/c9f2f917078bd2db12f23c3b413d9cba-Paper.pdf`.
7. Dandekar, R. & Barbastathis, G. Quantifying the effect of quarantine control in Covid-19 infectious spread using machine learning. *medRxiv*. eprint: `https://www.medrxiv.org/content/early/2020/04/06/2020.04.03.20052084.full.pdf`. `https://www.medrxiv.org/content/early/2020/04/06/2020.04.03.20052084` (2020).
8. Massaroli, S., Poli, M., Park, J., Yamashita, A. & Asama, H. Dissecting neural odes. *Advances in Neural Information Processing Systems* **33,** 3952–3963 (2020).
9. Rackauckas, C. *et al.* Universal Differential Equations for Scientific Machine Learning. *CoRR* **abs/2001.04385.** arXiv: `2001.04385`. `https://arxiv.org/abs/2001.04385` (2020).
10. Roehrl, M. A., Runkler, T. A., Brandtstetter, V., Tokic, M. & Obermayer, S. Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics. *IFAC-PapersOnLine* **53,** 9195–9200 (2020).
11. Gao, H., Sun, L. & Wang, J.-X. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *Journal of Computational Physics* **428,** 110079. ISSN: 0021-9991. `https://www.sciencedirect.com/science/article/pii/S0021999120308536` (2021).
12. Karniadakis, G. E. *et al.* Physics-informed machine learning. *Nature Reviews Physics* **3,** 422–440 (2021).
13. Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R. & Mahoney, M. W. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems* **34,** 26548–26560 (2021).
14. Mahmoudabadbozchelou, M. *et al.* Data-driven physics-informed constitutive metamodeling of complex fluids: A multifidelity neural network (MFNN) framework. *Journal of Rheology* **65,** 179–198 (2021).
15. Nguyen, T. N. K., Dairay, T., Meunier, R. & Mougeot, M. Physics-informed neural networks for non-Newtonian fluid thermo-mechanical problems: an application to rubber calendering process. *arXiv preprint arXiv:2201.13389* (2022).