Learning Linear Threshold Influence Models

Abhiroop Kodandapur Sanjeeva
The University of Texas at Austin
Department of Electrical and Computer Engineering
Austin, Texas

***Abstract*—We build and evaluate linear threshold influence learners for partial-observation and full-observation influence cascades. Mainly, we expand on the work discussed in [1] to learn underlying influence network parameters by observing influence cascades. Evaluation is also performed on real world Twitter cascades involving the Sydney Morning Herald hashtag. Our work shows that reliable learning of influence cascades in the full-observation setting is possible with small amounts of training data. Learning using partial-observations is shown to be possible when given large amounts of data, and most importantly can be done using a single layer neural network. We also show that the Sydney Morning Herald twitter cascades can be learned reliably in the full-observation setting. All code and data used can be found at https://github.com/Abhiroopks/Influence_Learning**

I. Introduction

A. Background

Influence in a social network is a concept that has been studied extensively for various applications. The diffusion of ideas, opinions, or the adoption of certain behaviors is of particular interest in areas such as viral marketing [1]. Here, marketers are interested in targeting influential members of social networks in the hopes of propagating the purchase of products by their followers. Studies in this area typically involve identifying the individuals of a social network who if initially influenced, will lead to influencing the most number of other individuals after some period of time [2]. This type of marketing can be observed in social media platforms such as Instagram and Youtube, where individuals with a large number of followers and views often advertise and promote products from brands that identified them as highly influential. The value of digital influencers in the modern world is growing rapidly, estimated to reach 15 billion USD in 2022 [3]. Brands will typically target digital influencers based on simple metrics such as number of followers and number of likes/views on posts. However, underlying behavior of social influence may be difficult to predict from these metrics, and deeper knowledge of the social network would be beneficial.

B. Modelling social influence

An important concept of social influence is an influence cascade. Here, influence is propagated through the social network starting from an initial seed. The initial seed is the set of initially influenced nodes of the graph. Each time step will have nodes becoming influenced as a function of their neighbors in the network. An example of a simple cascade with 3 time steps is shown in figure 1.

The cascade can be represented by a matrix of boolean values, such that $M_{ij} \in \{0, 1\}$ represents the state of node j during time step i. Each row is thus a snapshot of the state of all nodes during a particular time step. The social network itself can be modeled as a graph of nodes and edges, such that an edge from node i to node j with weight w_{ij} represents the level of influence individual i has on individual j. Various influence models can be used for modeling the behavior of influence, most notably the linear threshold (LT) and independent cascade (IC) models.

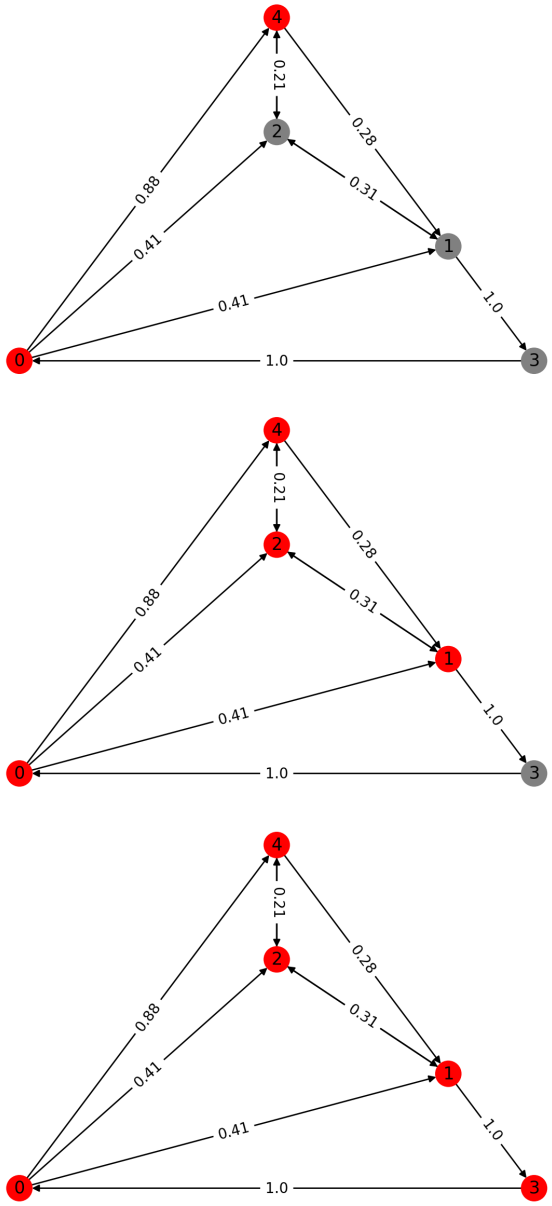
C. Linear Threshold Model

In the LT model, a node j of the social graph will be influenced at time t if:

$$\sum_{i=1}^{N} S_{i,t-1} * w_{ij} \geq \theta_j$$

Intuitively, a node will become influenced when the amount of incoming influence exceeds some personal influence threshold θ_j. $S_{i,t-1}$ is the influence state of node i during the previous time step, denoted by either 0 (not influenced) or 1 (influenced). Note that in our study, nodes remain influenced throughout the cascade once they become influenced. The cascade will terminate when no other nodes can be influenced after a time step, and the state of the system is final. In this model, the longest possible cascade is N-1 time steps, in which N = number of nodes in the graph. This is possible when the initial seed consists of one node, and only a single new node becomes influenced at each time step. The LT model may be useful in contexts such as marketing, in which individuals will purchase a product if they are influenced enough by others in their social network. Going beyond metrics such as social media followers and views/likes on posts, and using metrics such as the sum of outgoing influence (using the graph model, the sum of the weights of edges leaving a node) may help marketers select digital influencers more effectively. Estimating the personal influence thresholds of individuals may also be useful to quantify the relative ease by which individuals adopt a type of product. For example, a brand could target a potential customer with digital advertising for a category of products if that person is known to be easily influenced to make similar purchases. Such information could only be inferred by observing multiple cascades of influence for a category of products, within the same social network. For example, an individual may be observed to have purchased sports-related products often when a few of their

Fig. 1. Example of an influence cascade for an LT model. In this case, the initial seed set = {0,4} and all nodes are influenced at the end. Influenced nodes are shown in red, while uninfluenced nodes are grey. Edge weights show directional influence between nodes. The influence thresholds for each node are given by: [0.13, 0.59, 0.14, 0.45, 0.42].



social network peers have done so previously. If a marketer can learn the extent to which each node influences another (i.e. the weight of every edge in the graph) , as well as the threshold to influence every node, then the marketer can predict influence cascades from initial seeds.

D. Independent Cascade Model

The IC model is useful for contexts such as disease spread [4]. Given a social graph, the weights of edges can represent the probability of transmission from one individual to another. Similarly to the LT model, a cascade of infections can be observed over discrete time steps. Public health officials can make use of these models for contact tracing and infection minimization. This can be thought of as the inverse of the viral marketing problem, where now it is desired to limit the spread of the contagion by identifying individuals who are most likely to propagate the disease, and isolating or removing them from the social network.

II. Objectives

This paper will expand upon the work presented in [1] by making no assumptions about the underlying network structure. Algorithms to learn the social network parameters will thus be implemented for LT models and evaluated without knowledge of the social network structure. In addition, learning of parameters will be done for both the full and partial observation cases. Full observations of an influence cascade give the state of the network at all time steps. Partial observations only give the seed set and the final state. [1] discusses that for LT models, hypotheses with 0 training error can always be learned when given full observations due to the deterministic behavior of LT models. The same is not true for partial observations, but we implement and evaluate a neural network described in [1] to approximately learn the network parameters. In addition, we evaluate the effect of varying training data sizes on the testing error for both partial and full observations. We also evaluate the effect of varying the number of layers in the neural network for partial observations on training and testing error. Finally, our learners will be evaluated on a real-world Twitter cascade data set.

III. Project Description

For initial testing of the learners, simulated influence cascades are generated and provided as data for learning. To accomplish this, social graphs are generated by assigning random values for edge weights between nodes. Precision of weights is limited to 2 decimal places. Sparsity of graphs was set to 0.5 (half of all edges have weight 0) and influence thresholds for each node are randomly assigned as a value $\in$ [0,1]. Finally, the incoming edge weights for every node are normalized to ensure their sum is 1:

$$\forall j : \sum_{i=1}^{N} w_{ij} = 1$$

To generate cascades, the networks were given a seed set of initially influenced nodes and allowed to simulate the influence cascade until no new nodes became influenced. The resulting cascade for a single simulation is an NxN matrix in which row i is the state of the network at time step i. Note that most cascades do not last N time steps, but the final state of the cascade is redundantly appended until the matrix has N time steps for consistency across all simulations.

Learning requires observations of multiple influence cascades. It can be seen that for a network with N nodes, there are a total of 2^N unique initial seeds. Learners split the data into train/test sets for training and validation.

All work was done in Python 3.9, and major packages utilized include Keras [7], scikit-learn [8], TensorFlow [9], and NumPy [10].

A. Full Observation

The full observation learner makes use of knowing when each node in a cascade gets influenced. That is, it is given multiple cascade matrices and knows the state of all nodes during each time step of every cascade. We can observe that the state of a node in the LT model is a deterministic function of the state of nodes in the previous time step and its influence threshold. As discussed in [1], this reduces to a linear program at every node. Intuitively this means we can learn the incoming edge weights, as well as the influence threshold for each node given enough unique snapshots of the network. A data sample (X_i , Y_i) when learning local parameters for node j is thus:

- X_i: the state of all nodes other than node j at time t-1, for some cascade k.
- Y_i: the state of node j at time t, for the same cascade k.

It does not matter which cascade a data sample comes from under full observation, because the influence network behaves the same regardless. Thus each row of every influence cascade (except for the last row) can be used as an independent data point. This fact greatly increases the volume of available data to use for training. Note that for a node j, cascades in which j is in the seed set are discarded for training. In these cases it is clear that the state of node j throughout the cascade is not a function of other nodes, but is instead set externally before the cascade begins.

To accomplish learning, the full observation learner uses a Perceptron algorithm to linearly separate the data for a single node. The weights being learned are analogous to the incoming influence edge weights from other nodes of the network, as well as the threshold of influence. All weights are initialized to 0, and are allowed to be negative (the state of being uninfluenced is -1 rather than 0 for learning and testing). Because underlying structure of the network is unknown, we must learn all N-1 weights involving other nodes (we ensure there is no self-learning, i.e. the input data Xi for node j does not include the state of node j). Though it can be shown that a set of weights with 0 training error can be learned, a maximum number of iterations through the training data was set to limit time of execution. The algorithm is given in Algorithm 1. Training and testing error is simply calculated as:

$$\sum_{i=1}^{m} \mathbb{1}(sign(\langle w, X_i \rangle) \neq Y_i)$$

Algorithm 1 Perceptron Algorithm

procedure FULLOBSERVATIONLEARNER(X, Y)
  $maxRounds \leftarrow 50$
  $r \leftarrow 0$
  $w \leftarrow \{0\}^N$
  while $r \leq$ maxRounds & $\exists X_i : misclassified$ **do**
    $r \leftarrow r + 1$
    $w \leftarrow X_i Y_i$
  end while
  return w
end procedure

B. Partial Observation

The partial observation learner is given only the initial seed and final state for every cascade. Unlike the full observation setting, we cannot simplify learning to a linear program at each node. Also, a single cascade only provides a single data point (seed and final state), unlike in full observations where we can use every row of the cascade as a data point. We utilize a neural network to estimate model parameters rather than a perceptron algorithm. Each layer of the neural network is fully connected to the next, allowing the model to learn the influence each node has on every other node, as well as thresholds. The activation function for every node in the neural network is the tanh function, allowing us to measure how close to either 1 (influenced) or -1 (not influenced) each node is on a continuous scale. The loss function we define is the mean squared error (mse) loss, giving a surrogate loss for which we can utilize stochastic gradient descent to update model parameters.

Intuitively, each layer of the neural network simulates a single time step during the influence cascade, as discussed in [1]. Since LT models do not have varying edge weights across time steps, we can simplify the neural network by repeating the same layer multiple times. This effectively allows each layer to share the same weights, and simplifies SGD substantially. However, our study will vary the number of intermediate layers of the neural network to assess how this affects training and testing accuracy.

C. Evaluation

The real-world dataset for which we evaluate both the full and partial observation learners is the SMH dataset from [5]. This dataset has 20 cascades involving 1695 unique twitter users. Each event in the cascade is a tweet by a user involving the Sydney Morning Herald newspaper, collected in 2017. Unlike the simulated cascades, it is unknown whether the underlying social mechanism behind these tweet cascades follows an LT model. The cascades are processed in such a way that every row contains just a single newly influenced node. This includes the initial seed, in which we choose only the very first tweet of the cascade. In addition, the largest cascade contains only 317 time steps, and most contain much less. For the partial observation learner, we vary the number of layers in the neural network between 1 and 5 to measure the effect this has on training/testing error. We did not test for more layers due to limitations on computing power and available time.

In addition, we provide evaluation of the models using simulated influence cascades. We measure training and testing errors over variables such as number of layers (partial observation learner) and sample size (both learners). Sample size in this case is given as a fraction in (0,1], where we use

only this specified fraction of the training data. The size of the social networks used for these simulated results is kept constant at N=10 nodes.

In addition to evaluating the learners, we evaluate the performance of ”naive” predictors which blindly classify all nodes as uninfluenced or influenced. For all evaluations, 25% of the given data is used for validation of the learned model. The partial observation learner was also limited to 50 epochs over the training data.

IV. Results

A. Simulated Cascades

We first evaluate performance of the learners with simulated cascades. For a network of N=10 nodes, the full-observation learner achieves relatively low error rates across all sample sizes. Figure 2 shows that the training and testing errors for all sample size fractions. Note that the error rates here are the **mean error rates** over all nodes. That is, the learner only infers the local parameters of a single node and calculates the classification error rate for it. The final error rate is the average over all nodes. Although error rates for naive predictors were also calculated, we found them to have error rates much higher than the full observation learner, and omitted them from visualization. For the simulation in Figure 2, the naive 0 and naive 1 classifiers had an error rate of **0.87** and **0.13**, respectively. The low error rate of the full-observation learner in the simulated data setting is expected, as the perceptron algorithm reliably learns network parameters with enough samples. It is surprising however that the testing error is below 1% even when only given 10% of possible training data. This can be explained by the fact that although there are 2^N unique cascade seeds, many of the intermediate snapshots of the cascades are not unique. Upon further examination, we found that the average number of unique intermediate snapshots for 500 different (randomly generated) graphs of size N=10 nodes was just 194. Thus, nearly 80% of the the given data for this size graph is redundant data. Some redundancy is expected within cascades due to the data processing step, which appends the last unique row to each cascade until there are N rows. However, we believe a major portion of overall redundancy is also between different cascades. Intuitively, this means most of the simulated social networks have a small number of unique states, drastically reducing the required number of samples necessary for learning.

Figure 3 shows the error rates for the partial-observation learner over a varying number of layers in the neural network. Naive 1 classifier error is given as reference, while naive 0 classifier error was omitted due to it being much larger than others (close 0.9). Note that the error rates reported here are the binary classification errors, while the learner uses mean squared error for training. We can see that for a single layer (0 intermediate layers), train and test errors are low and close to 0. However, both errors increase drastically as the number of layers increase. This is a surprising result, as we expect more layers to closer simulate the behavior of the actual influence cascade over multiple time steps. However, the low error using a single layer shows that the final state of a cascade can be reliably learned using only the initial state, given a sufficient number of samples. This can mean that influence cascades can be reduced to a simple 1-step influence from the initial state, assuming the underlying behavior is in fact an LT system.

Fig. 2. Error rates for the full observation learner. Evaluated on simulated data where N=10 nodes.

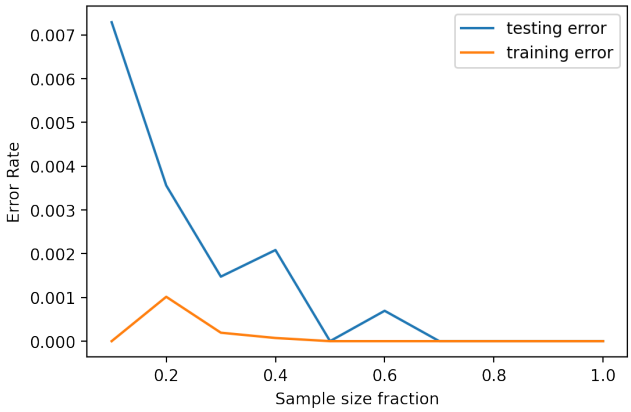


Fig. 3. Error rates for the partial observation learner. Evaluated on simulated data where N=10 nodes with varying number of layers in the neural network.

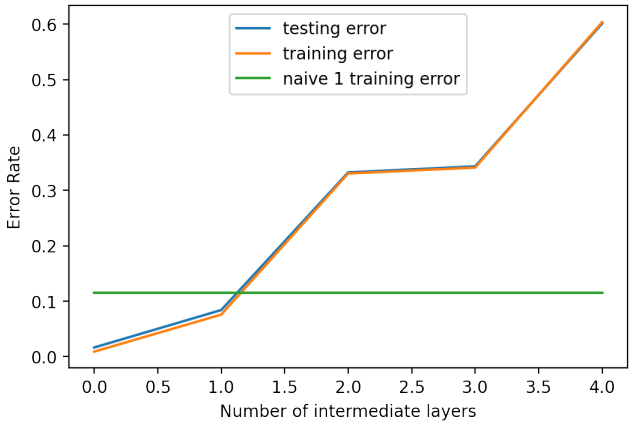
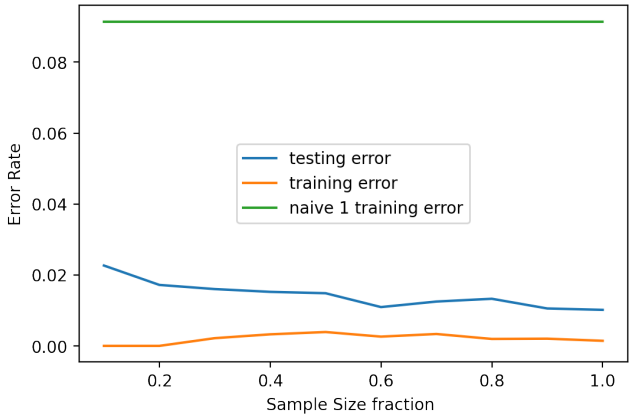


Figure 4 shows the error rates of the partial-observation learner over a varying sample size fraction (the simulated social network used here is different from Figure 3). The number of layers in the neural network is fixed to 1, as we have shown that increasing this number does not lead to better learning. Naive 1 classifier error is also shown for reference, while the naive 0 classifier is omitted due to its error being extremely high (close to 0.9). We can see that test and train errors are relatively low across all sample sizes, even when using only 10% of the training set. This is a similar outcome to the full-observation learner, with only slightly higher overall error rates.

B. SMH Dataset

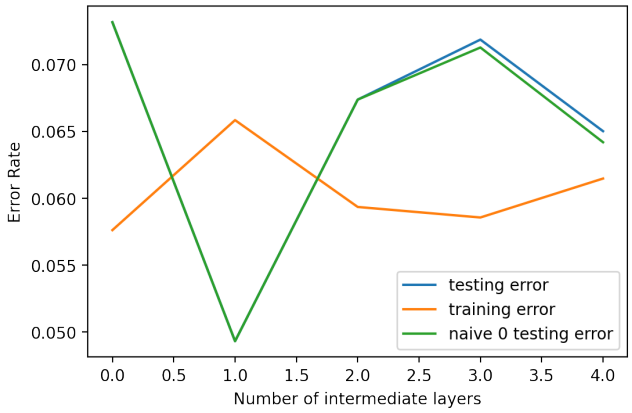
Figure 5 shows the performance of the partial-observation learner over a varying number of neural network layers. We

Fig. 4. Error rates for the partial observation learner. Evaluated on simulated data where N=10 nodes with varying sample size fraction. Number of neural network layers fixed to 1.



find that the SMH dataset is difficult to learn using partial-observations due to the small number of cascades - just 20. In addition, the largest number of time steps in any cascade is just 317 - meaning a vast majority of the nodes are never observed to become influenced. This translates to minimal learning, as the learner simply predicts most nodes to be uninfluenced no matter the initial seed. Figure 5 shows that the naive 0 classifier error closely follows the testing error of the learner to illustrate this behavior.

Fig. 5. Error rates for the partial observation learner - evaluated on the SMH dataset with a varying number of layers in the neural network. Error rate of naive 0 classifier given for reference.



The full-observation learner was also evaluated using the SMH dataset. The observed error rates are:

- test error: 6.938e-05
- train error: 3.896e-05
- naive 0 error: 0.059
- naive 1 error: 0.941

We can see that using full observations, we can reliably learn the state of each node at any time step given the state of other nodes in the previous time step. This is an interesting result, as it shows that the twitter users of the SMH dataset may have tweeted based on an underlying LT influence network. In fact, the observed error rates here are even lower than what is observed in the simulated cascades.

V. Future Work

Further research should be done to investigate the efficacy of using a multi-layer neural network to learn LT influence network parameters using partial observations. Based on this study, we found that the accuracy of the learner only decreases as the number of layers increases. However, the error rate for a single layer was found to be reliably accurate for simulated influence cascades, and this should be verified against larger real-world data sets.

Evaluation of the learners on real-world data was hampered by extremely high computation times and memory required to process and learn large data sets. An example of such datasets can be found in [6], which hold data regarding citations of academic papers. Influence can be inferred here by analyzing the cascade of paper citations of a particular paper following its publication. For example, a researcher publishes a paper on a certain date, and is cited on many other papers in the future by other researchers. This type of cascade can be analyzed to estimate the influence that researchers have on others. The smallest dataset here contains over 630K papers and citations, which is well beyond our capabilities to analyze.

Finally, this study took a very in-depth view of predicting influence cascade outcomes. In particular, we try to predict the state of every node in the social network at all time steps for full-observations, or at the end of the cascade for partial-observations. A simpler macro-scale approach can be taken that would predict the total number of influenced nodes given the initial state of the network. This may be more than enough for purposes such as viral marketing, where marketers are mainly concerned with volume of sales rather than specifically who was influenced. Indeed, the problem of influence maximization by choosing the smallest seed sets has been studied extensively [2].

References

[1] Narasimhan, H., Parkes, D., and Singer, Y. (2015). Learnability of Influence in Networks. NIPS.
[2] David Kempe, Jon M. Kleinberg, and Eva Tardos. Maximizing the spread of influence through a social network. In KDD, 2003.
[3] Wielki, J. Analysis of the Role of Digital Influencers and Their Impact on the Functioning of the Contemporary On-Line Promotional System and Its Sustainable Development. Sustainability 2020, 12, 7138. https://doi.org/10.3390/su12177138
[4] Kumar, Raj Gaurav Ballabh, "Evaluating the role of critical nodes in disrupting diffusion in independent cascade diffusion model" (2019). Graduate Theses and Dissertations. 17723. https://lib.dr.iastate.edu/etd/17723
[5] Rizoiu, M.-A., Graham, T., Zhang, R., Zhang, Y., Ackland, R., & Xie, L, Cascade Influence, (2018), GitHub Repository, https://github.com/computationalmedia/cascade-influence
[6] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In Proceedings of the Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'2008). pp.990-998
[7] Chollet, François. Keras, 2015. Github. https://github.com/fchollet/keras
[8] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 0.1038/s41586-020-2649-2.