GRNUlar: Gene Regulatory Network reconstruction using Unrolled algorithm from Single Cell RNA-Sequencing data

Harsh Shrivastava 1,∗, Xiuwei Zhang 1, Srinivas Aluru 1 and Le Song 1

1Department of Computational Science & Engineering, Georgia Institute of Technology, Atlanta, 30332, USA.

∗To whom correspondence should be addressed.

Abstract
Motivation: Gene regulatory networks (GRNs) are graphs that specify the interactions between transcription factors (TFs) and their target genes. Understanding these interactions is crucial for studying the mechanisms in cell differentiation, growth and development. Computational methods are needed to infer these networks from measured data. Although the availability of single cell RNA-Sequencing (scRNA-Seq) data provides unprecedented scale and resolution of gene-expression data, the inference of GRNs remains a challenge, mainly due to the complexity of the regulatory relationships and the noise in the data.

Results: We propose GRNUlar, a novel deep learning architecture based on the unrolled algorithms idea for GRN inference from scRNA-Seq data. Like some existing methods which use prior information of which genes are TFs, GRNUlar also incorporates this TF information using a sparse multi-task deep learning architecture. We also demonstrate the application of a recently developed unrolled architecture GLAD to recover undirected GRNs in the absence of TF information. These unrolled architectures require supervision to train, for which we leverage the existing synthetic data simulators which generate scRNA-Seq data guided by a GRN. We show that unrolled algorithms outperform the state-of-the-art methods on synthetic data as well as real datasets in both the settings of TF information being absent or available.

Availability: Github link to GRNUlar - https://github.com/Harshs27/GRNUlar

Contact: hshrivastava3@gatech.edu

1 Introduction

In molecular biology, it is known that the expression level of a gene is controlled by its transcription factors (TFs). Transcription factors are proteins which regulate the expression levels of their target genes in a given cell at a given time. These regulatory relationships can be represented by a graph, called a gene regulatory network (GRN), where nodes represent genes, and an edge from gene A to gene B means that the protein product of gene A is a TF for regulating gene B. This network governs transcription and further decides the way cell behave, and it is of great interest to decipher the interactions in this network.

It has been a long-standing challenge to reconstruct these networks computationally from gene-expression data (Chen et al., 1998; Kim et al., 2003). Recently, single cell RNA-Sequencing (scRNA-Seq) technologies provide unprecedented scale of genome-wide gene-expression data from thousands of single cells, which can lead to the inference of more reliable and detailed regulatory networks (Chen and Mar, 2018; Pratapa et al., 2020). GRNBoost2 (Moerman et al., 2019) and GENIE3 (Ván Anh Huynh-Thu et al., 2010) are among the top performing methods for GRN inference (Chen and Mar, 2018; Pratapa et al., 2020). They are based on fitting a regression function between the expression values of the TFs and other genes. These methods use the information of which genes (corresponding to nodes in the network) are TF genes (thus can have outgoing edges) to achieve better performance. An alternate approach to the inference of regulatory networks can be to pose the sparse graph recovery problem as a graphical lasso problem with $l_1$ regularization Friedman et al. (2008). These approaches are primarily unsupervised in nature and it is not straightforward to include supervised information or prior knowledge of underlying GRNs in their algorithms.

While methods for GRN inference from scRNA-Seq data are improving, simulators which generate synthetic scRNA-Seq data guided by GRNs are also progressing (Dibaeinia and Sinha, 2019; Pratapa et al., 2020). These scRNA-Seq simulators can generate realistic looking data, and have modeled sources of variation in single cell RNA-Seq data, such as noise intrinsic to the process of transcription, extrinsic variation indicative of different cell states, technical variation and measurement noise and
Fig. 1: Using the neural network in a multi-task learning framework which acts as non-linear regression functions between TFs and other genes. We start with a fully connected NN indicating all genes are dependent on all the input TFs (dotted black lines). Assume that in the process of discovering the underlying sparse GRN our algorithm zeroes out all the edge weights except the blue ones. Now, if there is a path from an input TF to an output gene, then we conclude that the output gene is dependent on the corresponding input TF.

bias. The primary application of these realistic simulators is to benchmark the performance of GRN inference methods (Dibaeinia and Sinha, 2019; Chen and Mar, 2018; Pratapa et al., 2020). These evaluations show that the performance of current method for GRN inference is not satisfying even with synthetic data. Indeed, GRN inference is hindered by multiple factors, including the potentially nonlinear relationships between TFs and their target genes, and the intrinsic and technical noise in scRNA-Seq data (Vallejos et al., 2017).

In this paper, we present deep learning frameworks which accommodate the nonlinearity in regulatory relationships, and which are shown to be relatively more robust to technical noise in the data. Moreover, our approach is based on the idea of incorporating prior knowledge of the problem as an inductive bias to design data-driven models (Shrivastava et al., 2018, 2020). We make use of recently proposed unrolled algorithms technique which employs optimization algorithms for the objective function of the problem under consideration as templates for designing deep architectures. The key advantages of unrolled algorithms are (a) few learnable parameters (b) less supervised data points required for training (c) comparable or better performance than existing state-of-the-art methods and (d) more interpretable. Unrolled algorithms have been successfully used in other recent works, including REEfold for RNA secondary structure prediction (Chen et al., 2020) and GLAD for sparse graph recovery (Shrivastava et al., 2020). Here we present a novel unrolled algorithm for our deep learning framework, GRNUlar (pronounced “granular”) Gene Regulatory Network Unrolled algorithm), for GRN inference. GRNUlar works in the setting where TF information is given. GLAD can be applied to GRN inference without using the TF information, and we also provide a modification to it, called GLAD-TF, for it to benefit from the TF information.

We utilize the GRN-guided simulators in a novel way: we use a simulator to generate a corpus of simulated data pairs consisting of expression data and the corresponding GRN. This allows us to leverage supervised learning to learn the unrolled algorithm (or neural algorithm) for reconstructing the GRNs from the input gene expression data. It has been argued in the recent works by (Belilovsky et al., 2017; Shrivastava et al., 2020) that a data-driven neural algorithm may be able to leverage this distribution of problem instances, and learn an algorithm which performs better than traditional manually designed algorithms. SERGIO (Dibaeinia and Sinha, 2019) is the most realistic GRN-guided simulator for scRNA-Seq data to date, and is the main simulator we use for training and evaluation of our models. SERGIO generates realistic gene expression data by incorporating known principals of TF-gene regulatory interactions that underlie expression dynamics, and it models the stochastic nature of transcription as well as simulates the non-linear influences of multiple TFs.

We show that our proposed algorithms perform better than state-of-the-art methods on both simulated data and real data from species including human and mouse. Our learned neural algorithms is comparably more robust to high levels of technical noises often observed in realistic settings. We demonstrate that our methods benefit from the supervision obtained through synthetic data simulators, and to the best of our knowledge, we are the first to use the simulators to train neural algorithms for GRN inference from scRNA-seq data.

2 Methods

In this section, we first formulate the problem of GRN inference, and briefly introduce some existing approaches. We then present GRNUlar-base, which is the “basic” version of GRNUlar, and the difference between the two is that the latter uses a technique for initialization and thus improves the runtime of the former. We also provide an insight into a novel loss function which we developed specifically for GRN inference task. Then, we briefly introduce how GLAD works for GRN inference and how it is extended to GLAD-TF, which makes use of the TF information.

Problem Setting: We consider the input gene expression data to have $D$ genes and $M$ samples, $X \in \mathbb{R}^{M \times D}$. Let $G = [1, D]$ be the set of genes and $T \subseteq G$ be those which are TFs. We aim to identify the directed interactions of the form $(t, o)$, where $t \in T$ and $o \in G$.

We note that there can be interactions between TFs themselves. For our method, we assign directed edges between the TFs and other genes and the interactions between TFs are represented by undirected edges. We thus output Completed PDAGs, where PDAGs represent equivalence classes of DAGs and stands for “partially directed acyclic graphs” (Chickering, 2002).

Existing approaches: The common approach followed by many state-of-the-art methods for GRN inference is based on fitting regression functions between the expression values of TFs and the other genes. Usually, a sparsity constraint is also associated with the regression function to identify the top influencing TFs for every gene.

Generally, the form of objective function for GRN recovery used in various methods is a variant of the equation given below

$$ X_o = f_k(X_T) + \epsilon \tag{1} $$

We can view Eq. 1 as fitting a regression between each gene’s expression value as a function of the TFs and some random noise. Simplest model will be to assume that the function $f_k$ is linear. One of the state-of-the-art methods TIGRESS by Haury et al. (2012) assumes a linear function of the following form for every gene

$$ f_k(X_T) = \sum_{t \in T} \beta_{t,k} X_t \tag{2} $$

Another top performing method GENIE3 by Van Anh Huynh-Thu et al. (2010), assumes each $f_k$ to be a random forest. GRNBoost2 by Moerman et al. (2019) further uses gradient boosting techniques over the GENIE3 architecture to do efficient GRN reconstruction.

2.1 GRNUlar: Unrolled model for recovering directed GRNs

We aim to leverage the supervision available to us from the gene expression data simulators. This supervision is in the form of input gene expression data and the corresponding underlying GRN. We speculate that tuning GRN recovery models under this supervision will lead us to better capture the intricacies of the real data and potentially improve upon the unsupervised methods.

We first describe our novel approach of modeling regression functions using a neural network. Based on this modeling, we motivate the use of unrolled algorithms and design its deep architecture.
2.1.2 Motivation for unrolled algorithms

Notice that there can be many neural network representations possible in Section 2.1.1 which can satisfy the Eq. 1 and can lead to different GRNs. These GRNs vary mostly in terms of the sparsity obtained and it is hard for users to manually choose them. Unrolled algorithms help resolve this problem as the sparsity related hyperparameters (e.g., the weight of the l1 norm term) can be learned from supervision.

2.1.3 Designing the unrolled algorithm

We follow similar procedure as the unrolled algorithm designed for the sparse graph recovery task as described in Shrivastava et al. (2020). Briefly outlining the approach,

1. Define a related optimization problem for the task at-hand.
2. Apply the Alternative Minimization (AM) algorithm and unroll it for certain number of iterations.
3. Replace the hyperparameters with problem dependent neural networks and treat all the unrolled iterations together as a deep model.
4. Learn this unrolled architecture under supervision by defining a direct optimization objective.

**Step 1:** We consider the following non-linear optimization objective function for the regression defined in Eq. 1 with l1 penalty

\[
\arg\min_{W} \sum_{k=1}^{M} \|X_k - f_{SV}(X_k)\|_2^2 + \rho \|\Pi_i W_k\|_1
\]  

(3)

where \(f_{SV}(X_k)\) is a neural network. For example, we can define a 2 layer neural network with ReLU non-linearity as

\[
\hat{f}(W_1, W_2)(X_k) = W_2 \cdot \text{ReLU}(W_1 \cdot X_k + b_1) + b_2
\]  

(4)

We learn the weights \(\{W_k\}\) and the biases \(\{b_i\}\) while optimizing for Eq. 3.

**Step 2:** We now apply the Alternate Minimization approach to the optimization given in Eq. 3. Since the objective above is non-linear, we will need an iterative approach to minimize it w.r.t. \(W\). However, we can make our problem easier by separating the two terms such that we can get closed form solution of the l1 penalty term. We can achieve this by introducing an additional variable \(Z\) where \(Z_{i} = \Pi_i W_k\). We then have

\[
\arg\min_{W} \sum_{k=1}^{M} \|X_k - f_{SV}(X_k)\|_2^2 + \rho \|Z\|_1
\]  

(5)

We can include the \(Z\) term in optimization and as a square penalty term

\[
\arg\min_{W, Z} \sum_{k=1}^{M} \|X_k - f_{SV}(X_k)\|_2^2 + \rho \|Z\|_1 + \frac{1}{2} \lambda \|\Pi_i W_k - Z\|_F^2
\]  

(6)

and alternately minimize \(Z\) and \(\Theta\) for \(l \in [0, L]\) iterations as,

\[
W^{(l+1)} \leftarrow \arg\min_{W} \sum_{k=1}^{M} \|X_k - f_{SV}(X_k)\|_2^2 + \frac{1}{2} \lambda \|\Pi_i W_k - Z_l\|_F^2
\]  

(7)

\[
Z_{l+1} \leftarrow \arg\min_{\mathbf{Z}} \rho \|\mathbf{Z}\|_1 + \frac{1}{2} \lambda \|\Pi_i W_k^{(l+1)} - \mathbf{Z}\|_F^2
\]  

(8)

The update of \(Z\) is of the form \(f(\mathbf{Z}) + \rho \|Z\|_1\), where \(f(\mathbf{Z})\) is a convex function. Similar to Shrivastava et al. (2020), the minimizer of this function is the proximal operator given by \(\eta_{\rho \|\cdot\|_1}() = \text{sign}() \max(|\cdot| - \rho/\lambda, 0)\).

---

The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

doi: bioRxiv preprint

Fig. 2: Visualizing the GRNUlinear algorithm’s architecture.
Thus, the updates of AM algorithm are given by

\[
W^{(t+1)} \leftarrow \arg\min_{W} \frac{1}{M} \sum_{k=1}^{M} \left[ X_{O}^{k} - f_{W}(X_{P}^{k}) \right]^{2} + \frac{1}{2} \lambda \left[ \Pi_{i} |W_{i}| - Z_{i} \right]^{2}
\]

(9)

\[
Z_{i+1} \leftarrow \eta_{\lambda,i} \left( \Pi_{i} |W_{i}^{(t+1)}| \right)
\]

(10)

Step 4: We identify the proximal operator \( \eta_{\lambda,i} \) and \( \lambda \) as the hyperparameters which control the sparsity of the final graph. We can parameterize them as \( \rho_{\text{min}}, \Lambda_{0} \) respectively. These neural networks are minimalistic in design and takes the solution of the previous update to predict the next value. We will learn these problem dependent neural network using supervision. As for the Eq. (9), we optimize for the \( W_{i}^{(t)} \) by using state of the art deep learning optimizers. The corresponding values of \( Z_{i}^{(t)} \) can be obtained by plugging in the \( W_{i}^{(t)} \) in its closed form update, Eq. 10. We unroll these updates for \( L \) iterations and treat it as a highly structured deep model.

**Algorithm 1: GRNUlar-base**

Function \( \text{covTF}(X, T, \text{TF-names}) \):

\[
\Sigma_{T} \leftarrow \frac{1}{N} (X - \mu)^T (X - \mu)
\]

Select \( \Sigma_{T} \subset \Sigma_{O} \) an \( T \times O \) submatrix using the TF-names

return \( \Sigma_{T} \)

Function \( \text{fitDNN}(X, Z, \lambda, \text{TF-names}) \):

Fit \( W \) based on updated regularization terms \( \lambda, Z \)

\( X_{P}, X_{O} \leftarrow X \) (using the TF-names)

\( f_{W} \leftarrow \text{Initialize Neural Networks} \)

\( W \leftarrow \arg\min_{W} \sum_{k=1}^{M} \left[ X_{O}^{k} - f_{W}(X_{P}^{k}) \right]^{2} + \frac{1}{2} \lambda \left[ \Pi_{i} |W_{i}| - Z_{i} \right]^{2} \)

(Using standard Deep Learning optimizers for 'E1' epochs)

\( \Theta \leftarrow \Pi_{i} |W_{i}| \)

return \( \Theta \)

Function \( \text{GRNUlar-cell}(X, \Sigma_{T}, \Theta, Z, \lambda) \):

\( \lambda \leftarrow \Lambda_{0}((Z - \Theta)^{T}_{\pi}, \lambda) \)

\( \Theta \leftarrow \text{fitDNN}(X, Z, \lambda) \)

For all \( i, j \) do

\( \rho_{ij} \leftarrow \rho_{\text{min}}(\Theta_{ij}, \Sigma_{T_{ij}}, Z_{ij}) \)

\( \Sigma_{T_{ij}} \leftarrow \eta_{\lambda,ij} (\Theta_{ij}) \)

return \( \Theta, Z, \lambda \)

**Function \( \text{GRNUlar}(X) \):**

\( X^{0} \leftarrow 1 \)

\( \Sigma_{T} \leftarrow \text{covTF}(X) \)

\( Z^{0} \leftarrow \text{zeros}(T, O) \)

\( W^{0} \leftarrow \text{fitDNN}(X; Z^{0}, X^{0}) \)

\( \Theta^{0} \leftarrow \Pi_{i} |W_{i}|^{0} \)

For \( t = 0 \) to \( L - 1 \) do

\( \Theta^{t+1}, Z^{t+1} \leftarrow \text{GRNUlar-cell}(X, \Sigma_{T_{t}}, \Theta^{t}, Z^{t}, \lambda) \)

return \( \Theta_{L}, Z_{L} \)

**Algorithm 2: GRNUlar**

Function \( \text{covTF}(X, T, \text{TF-names}) \):

\( \Sigma_{T} \leftarrow \frac{1}{N} (X - \mu)^T (X - \mu) \)

Select \( \Sigma_{T} \subset \Sigma_{O} \) an \( T \times O \) submatrix using the TF-names

return \( \Sigma_{T} \)

Function \( \text{fitDNN-fast}(X, Z, W, \lambda, \text{TF-names}) \):

For \( p = 0 \) to \( P - 1 \) do

\( J_{p} \leftarrow \sum_{k=1}^{M} \left[ X_{O}^{k} - f_{W}(X_{P}^{k}) \right]^{2} + \frac{1}{2} \lambda \left[ \Pi_{i} |W_{i}| - Z_{i} \right]^{2} \)

\( W^{p+1} \leftarrow \text{optimizer-update}(W^{p}, \nabla J_{p}) \)

\( \Theta \leftarrow \Pi_{i} |W_{i}| \)

return \( W, \Theta, \lambda \)

Function \( \text{GRNUlar-cell}(X, \Sigma_{T}, \Theta, Z, \lambda) \):

\( \lambda \leftarrow \Lambda_{0}((Z - \Theta)^{T}_{\pi}, \lambda) \)

\( \Theta, W \leftarrow \text{fitDNN-fast}(X, Z, W, \lambda) \)

For all \( i, j \) do

\( \rho_{ij} \leftarrow \rho_{\text{min}}(\Theta_{ij}, \Sigma_{T_{ij}}, Z_{ij}) \)

\( Z_{ij} \leftarrow \eta_{\lambda,ij} (\Theta_{ij}) \)

return \( W, \Theta, \lambda \)

Function \( \text{goodINIT}(X, \text{TF-names}) \):

\( W \leftarrow \arg\min_{W} \sum_{k=1}^{M} \left[ X_{O}^{k} - f_{W}(X_{P}^{k}) \right]^{2} \)

(Using standard Deep Learning optimizers for 'E1' epochs)

\( \Theta \leftarrow \Pi_{i} |W_{i}| \)

return \( W, \Theta, \lambda \)

Function \( \text{GRNUlar}(X) \):

\( \Theta^{0}, W^{0} \leftarrow \text{goodINIT}(X) \)

\( \lambda^{0} \)

\( \Sigma_{T} \leftarrow \text{covTF}(X) \)

\( Z^{0} \leftarrow \text{zeros}(T, O) \)

For \( l = 0 \) to \( L - 1 \) do

\( \psi^{l+1}, \theta^{l+1}, Z^{l+1}, \lambda^{l+1} \leftarrow \text{GRNUlar-cell}(X, \Sigma_{T}, \Theta^{l}, W^{l}, Z^{l}, \lambda^{l}) \)

return \( \Theta_{L}, Z_{L} \)

2.2 Efficient GRNUlar algorithm using ‘good’ initialization

We propose an alternate ‘good’ initialization technique to reduce the runtime of Algorithm 1. We posit that, if we optimize for the \( 1^{\text{st}} \) term of Eq. 9 beforehand and obtain good initial values of \( W^{0} \), then we do not need to do minor adjustments to the \( W \) as we update \( \Theta \) and \( Z \). We then just need to unroll the optimization (the new \( \text{fitDNN-fast} \) function) for only a few iterations \( P \sim \{2, 5, 10\} \). Refer Algorithm 2 as well as Fig. 2 which pictorally shows the highly structured deep architecture of the GRNUlar algorithm. For the \( p^{\text{th}} \) unrolled iteration, we have

\[
J_{p} = \sum_{k=1}^{M} \left[ X_{O}^{k} - f_{W}(X_{P}^{k}) \right]^{2} + \frac{1}{2} \lambda \left[ \Pi_{i} |W_{i}| - Z_{i} \right]^{2}
\]

(12)

\[
\text{Loss} = \left[ \Pi_{i} |W_{i}|^{L} - W^{*} \right]^{2}_{p} + \lambda_{p} \left[ \Pi_{i} \left| W_{i}^{(L)} \right| \right] + W^{*}
\]

(11)

The ground truth \( W^{*} \in [0, 1]^{O \times T} \) matrix, where 1 indicates an edge between \((i, o)\). Algorithm 1: GRNUlar-base provides a supervised learning framework for the unrolled model directly based on the updates of the AM algorithm, Eqs. 9 & 10.

*Output* — 2020/4/23 — page 4 — #4
We compared with the GRNBOOST2, GENIE3 define a loss function which can find a desirable balance between them. For evaluation purposes of the recovered graphs, it will be very useful if we can observe that in every iteration of Algorithm 2 we optimize the unrolled parameters $\rho_{nn}, \lambda_{nn}$ (tiny neural networks) to learn the underlying graph sparsity from the supervision provided. Thus, we want to highlight that the overall training does not require much training data as well as the number of unrolled iterations. We also empirically verify that GRNUlar algorithm performs equivalent to GRNBase algorithm with significant runtime improvement.

A note on the general idea of neural network based parameterization: For the GRNUlar algorithm, we can further parameterize the optimizer update given in Eq. 13 and learn it from the supervision provided, similar to $\lambda_{nn}$. In our current implementation, we use the 'adam' optimizer. We want to highlight that our technique of parameterization in an unrolled fashion is very generic and can be used for any off-the-shelf optimizer. For instance, consider the example of parameterizing gradient descent optimizer which is realized using the $\text{nn}$ optimizer.

For instance, consider the example of parameterizing gradient descent optimizer which is realized using the $\text{nn}$ optimizer. For the $\text{nn}$ optimizer, we can further parameterize the optimizer update given by

$$\lambda^{t+1} = \lambda_{nn}(\Theta^t - \alpha [Z - \Theta^t])$$

$$\lambda^{t+1} = \lambda_{nn}(\Theta^t - \alpha [Z - \Theta^t])$$

$\text{Eq.} 15$ tells us that having $\beta > 1$ will weigh recall higher than precision as it places more emphasis on the FNs. Similarly, having $\beta < 1$ will attenuate the influence of FNs and thus weigh recall lower than precision. A note on inputs to loss function: In order to ensure that the entries of $\Theta^t \in [0, 1]$ we pass it through the $\text{tanh}([60\Theta^t])$ operation which is an entry-wise absolute value followed by an entry-wise tanh function. In some cases, we also do an additional diagonal masking operation, $60\Theta^t_{nn} = \max_{\Theta^t} + 60$, so as to prevent the NaNs.

Thus, we define a loss function between the predicted $\Theta^t$ and true underlying matrix $\Theta^t$ as the combination of the MSE (or Frobenius norm) loss and the $L_2$ loss. It is often tricky to jointly optimize and balance between multiple loss functions. Taking hint from the loss balancing technique described in Rappaport et al. (2019), we introduce a balancing ratio $r = \frac{L_{mse}}{L_{true}}$ which adjusts the scales of both the losses. Note that while implementing this scaling, at every epoch, we calculate '$r$' by detaching the losses from the computational graph to keep it as a constant.

For training GRNUlar, a collection of input expression data and their corresponding ground truth GRNs can be sampled from GRN guided realistic data simulators. The loss function in Eq. 16 is designed to directly optimize the output GRN prediction of GRNUlar to the ground truth GRN connections.

We aim to optimize the loss function over the average of such data pairs so that the learned architecture is able to perform well over a family of problem instances.

2.4 GLAD model & proposed modification for TFs

The GRNUlar algorithm described above is designed to reconstruct GRNs using the TF information. But, there can be cases where we do not know the underlying TFs. Majority of the existing methods are typically designed to include TF information and their performance in terms of recovery quality and runtimes deteriorate significantly if the TFs are not provided. Our experiments corroborate these statements.

We found an alternative method to slightly mitigate above concerns. We directly use the unrolled model GLAD (Shrivastava et al., 2020) for the GRN inference problem. This model does not use the TF information for GRN recovery. Briefly, the GLAD model's architecture is based on the unrolling the iterations of an Alternate Minimization algorithm for the graphical lasso problem with applications towards sparse graph recovery. We refer the reader to the Algorithm 3 in Supplementary-A (GLAD- TF model) We modify the architecture of GLAD to include prior information about the TFs if available. We assume that all the edges in the actual GRN have at least one gene which belongs to the list of TFs. We introduce a masking operation at every step of the unrolled iterations as shown in Algorithm 2 in Supplementary-A, which eliminates all the unwanted edges that are between the non-TF nodes.

3 Experiments

3.1 Description of evaluation metrics & methods

Recovery Metrics: Following the metrics used in (Dhaichina and Sinha, 2019; Chen and Mar, 2018), we use AUROC (Area Under the Operating Characteristics) and AUPRC (Area Under the Precision Recall Curve) values for our evaluation.

Baseline methods: We compared with the GRNBOOST2, GENIE3 as they are representative of regression based methods. We used the Arborot package to run these algorithms (Moorman et al., 2019). We additionally compared with the Graphical Lasso algorithm (GLASSO) using their "scipy" implementation. We did extensive fine tuning of the hyperparameters for all the baseline methods using the training/valid data and then reported the results on the test data.

Settings of the unrolled algorithms: For the GLAD model, we used the standard initialization as recommended by the authors (refer Fig. 3). We chose the number of unrolled iterations $L = [15, 30]$. For the GRNUlar
In this subsection, we aim to conduct an exploratory study to gauge the generalization ability of the unrolled algorithms for the GRN inference experiments with varying number of the total single cells, \(M = \{100, 500, 1K, 5K, 10K\}\). We also observed that varying the number of cell types from \(C = 2\) to \(C = 10\), we see that the AUPRC values increase in general. The top panel contains the methods without the TF information and the bottom panel shows the AUPRC plots of the methods including the TF information. The unrolled algorithms in general outperform the traditional methods.

For both the unrolled methods, we chose 2 models based on AUPRC and AUROC results on the validation data. We use the scaled loss function (Eq. 16) to jointly optimize for MSE and \(F_0\) loss. The values of \(\beta\) used in our experiments were chosen from the set \([0.5, 1, 2, 5]\). We implemented the unrolled algorithms using PyTorch and were ran on Nvidia P100 GPUs.

### 3.2 Evaluating GRN inference methods on synthetic data

In this subsection, we aim to conduct an exploratory study to gauge the generalization ability of the unrolled algorithms for the GRN inference task using the synthetic data simulator. We compare all the algorithms on various simulator settings and list our key observations.

**Graphs and expression data for training**: To provide supervision for the unrolled algorithms we use the SERGIO simulator. To create random directed graphs (GRNs) we first decide on some number of TFs or master regulators. Then, we randomly add edges between the TF and the other nodes based on the sparsity requirements. Also, we randomly add some edges between the TF themselves but exclude the self-regulation edges. We further add minimal number of edges to maintain connectivity of graph. We want to highlight that we do not need many graphs to train our experiments were chosen from the set \([0.5, 1, 2, 5]\). We implemented the unrolled algorithms using PyTorch and were ran on Nvidia P100 GPUs.

We simulate cells from multiple steady states. When simulating data with no technical noise (what we refer to as clean data), we set the following parameters: sampling-state=15 (determines the number of steps required to decide the master regulators’ basal production cell rate for all genes) and \(\text{noise-type} = \text{dpd}\) (the type of intrinsic noise which is Dual Production Decay noise, which is the most complex out of all noise-types). We simulate cells from different types of biological processes and gene-expression levels with various amount of intrinsic and technical noise. We simulate cells from multiple steady states. When simulating data with no technical noise (what we refer to as clean data), we set the following parameters: sampling-state=15 (determines the number of steps for simulations for each steady state), noise-param=0.1 (controls the amount of intrinsic noise); noise-type=dpd (the type of intrinsic noise is Dual Production Decay noise, which is the most complex out of all types provided). We set genes’ decay parameter to 1. The parameters required to decide the master regulators’ basal production cell rate for all cell types - low expression range of production cell rate \(\sim U[0.2, 0.5]\) and high expression range of cell rate \(\sim U[0.7, 1]\). We chose \(K\) as the maximum interaction strength between master regulators and target genes. Positive strength values indicate activating interactions and negative indicates repressive interactions and ±1 signs
GRNUlar: Unrolled algorithm for GRN reconstruction

Fig. 7: (Noisy settings, dropout percentile=82%) We report the average results over 15 test graphs in the noisy settings. GLAD is the best performing method in absence of TFs. GRNUlar gives notable AUROC values and it outperforms other methods, which are randomly assigned. When adding technical noise, we add the dropout events which are considered to be a major source of technical noise in real data. Parameters which control the amount of dropouts include shape (which we set to 20) and percentile, which we vary among the values $q \in \{25, 50, 75\}$. Larger $q$ corresponds to higher technical noise. All other parameters are set to default values.

For experiments in this subsection, we found that GRNB2-TF consistently outperforms GENIE3 method and hence we only show GRNB2-TF results in the plots. Also, each data point in the plots represent its value along with the standard deviation over the test graphs.

3.2.1 Clean: simulated data with no technical noise

The 'clean' gene expression data from SERGIO follows all the underlying kinetic equations but excludes all the external technical noises. We can consider this data as being recorded with no technical errors. Fig. 4 compares different methods on their AUROC performance on varying number of cells and number of cell types. For GRNUlar model we chose 2 layer NN with P=50, H=40, 60, 100. L=15, vary $\beta = \{2, 5\}$ in the loss function and we selected the best model based on the validation results.

Fig. 8: Heatmap of AUROC and AUPRC values from the real data from the BEELINE framework by Patapaa et al. (2020). We ran all the methods including the TF information. [S][N][C] represent the ground truth networks [String-network][Non-Specific-ChIP-seq-network][Cell-type-specific-ChIP-seq-network] respectively. Data of the species [m] mouse and [h] human were used. GRNUlar performs better than the other algorithms in both the metrics.

To perform this study, we make use of the realistic data sets provided by the SERGIO simulator. They provide three scRNA-Seq datasets DS1, DS2 & DS3 which are generated from input GRNs with 100, 400 & 1200 genes respectively. These networks were sampled from real regulatory networks of E.coli and S.cerevisae. For each dataset, the settings are: number of cell types C=9; total number of single cells M=2700, and there are 300 cells per cell type. Each data set was synthesized in 15 replicates by re-executing SERGIO with identical parameters multiple times. The parameters were configured such that the statistical properties of these synthetic dataset are comparable to the mouse brain in Genes et al. (2015).

We define the our training and testing settings such that there are considerable differences between the training and testing datasets. We use all of the DS1, DS2 & DS3 datasets for testing, and only the DS1 dataset for training. The major similarities and differences between the training and testing data are:

- **Similarities:** (1) The SERGIO settings for the train data are sampled from similar ranges as that of DS1. Specifically, the parameters like production cell rates, decays, noise-param and interaction strength. (2) We train on data with no dropouts as opposed to 82% dropout percentile in the case of the DS datasets. We thus, keep the dropout information hidden from the models.
- **Differences:** (1) The underlying GRNs are completely different in terms of sparsity and connection patterns. (2) We train on data with no dropouts as opposed to 82% dropout percentile in the case of the DS datasets. We thus, keep the dropout information hidden from the models. (3) The datasets DS2, DS3 are completely different from train data & DS1 in terms of the underlying GRN as well as the corresponding SERGIO parameters are sampled from different range of values. For details, refer to Table1 in Dhaebenia and Sinha (2019) & supplementary Tables S1, S3.
terms related to ESC cell differentiation and cell fate towards endodermal
improved performance over TFs in DS1/DS2/DS3 are 10/37/127 respectively. So, we chose
our strategy to choose the dimensions, all test settings. We can further improve the results by tuning the SERGIO
function for these experiments with a single hidden layer (2006). We can observe that the unrolled algorithm
gives slightly
the 21 data pairs. In general for real data, we observe very low AUPRC.
GRNUlar in general performs better than other methods in the noisy settings. We can improve its performance further, by training using multiple data simulators
and by better finetuning of simulators to resemble the real data.

3.4 Real single cell RNA-Seq datasets
In this section, we evaluate the methods on seven datasets from five experiments which include human mature hepatocytes (hHP) (Camp et al., 2017), human embryonic stem cells (hESC) (Chu et al., 2016), mouse embryonic stem cells (mESC) (Hayashi et al., 2018), mouse dendritic cells (mDC) (Shalek et al., 2014), and three lineages of mouse hematopoietic stem cells (Nestorova et al., 2016): erythroid lineage (mHSC-E), granulocyte-macrophage lineage (mHSC-GM) and lymphoid lineage (mHSC-L). These are the same datasets used in Pratapa et al. (2020) and we use their corresponding ground-truth networks for our experiments as well. For each dataset there are three versions of ground-truth networks: cell-type-specific ChIP-seq, nonspecific ChIP-seq and functional interaction networks collected from STRING. We then have in all different data pairs, different types of expression data evaluated against 3 different types of ground truth.
Preprocessing: For each gene expression data and its corresponding network, we first sort all the genes according to their variance and select the top 500 varying genes. We also have access to a list of known TFs. We only consider all the TFs whose variance had p-value at most 0.01. Now, we find the intersection between the top 500 varying genes and all the TFs to find the subset of genes which act as the TF, see Table 3 in Supplementary-B. Then, we select the sub-graph of top 500 varying genes from the underlying GRN as our ground truth for evaluation.
Training details: We train on the expression data which is similar to the expected biological processes
activities among the genes related to the expected biological processes compared to all the genes selected by variation. We then set the threshold for interaction score as 0.22, and obtained the network shown in Fig. 11. In this network, the TFs SOX7, SOX17, MTF2, GATA6 and CITED2 are known TFs in either stem cell differentiation or embryo development; NOTCH1 and RBPI are TFs in the NOTCH pathway which controls cell fate specification (www.genecards.org). The TFs with highest interaction scores are highly relevant TFs for the cells under study.

We now show how our predicted interactions may bring new biological insights. For instance, we noticed that one of the target genes of SOX7 with strong interaction is CFC1. From ChIP-Seq experiments (the [Cell-type-specific-ChIP-seq] ground truth network mentioned previously), SOX2 is a TF for CFC1. However, in our prediction results, we predicted SOX7 and SOX17 as the TFs for CFC1. We note that the dataset consists of ESC cells differentiating into primitive endoderm cells, and SOX2 is a key TF in mouse ESCs governing the pluripotency of the cells (Masui et al., 2007). As the cells differentiate, the pluripotency goes down, so the SOX2 function may also decrease. To verify this, we use the pseudotome of the cells obtained from (Pratapa et al., 2020), which was inferred with Slingshot (Street et al., 2018), and visualize the gene-expression levels of CFC1, SOX2, SOX7 and SOX17 (Fig. 10). For readability we plot the actual gene-expression levels cell by cell only for CFC1, and for the SOX TFs we plot the fitted lines of their expression levels obtained using LOESS regression. The dashed lines represent the standard deviation. We see that indeed the SOX2 expression decreases along the pseudotime, and the expression levels of CFC1, SOX7 and SOX17 increase. The fitted lines of SOX7 and SOX17 show that they are much better predictors for the expression of CFC1 than SOX2. Indeed, it is discussed that SOX7 and SOX17 are highly related members of the SOX family and their high expression in ESCs are correlated with a downregulation of the pluripotency and an upregulation of the primitive endoderm-associated program (Sarkar and Hochedlinger, 2013). This example showcases how we can use predicted regulatory networks to find regulatory pathways for a specific biological program. Some of these may already have evidence in literature but some may be new and our prediction can be used to provide hypothesis for further experimental validation.
and other genes. In cases where the TF information is available, we use a multi-task learning framework to model the regression between TF architecture and its inductive bias from the regression based formulation in the data. We also show the superior performance of a recently developed existing methods under various settings of both simulated data and real data.

Belilovsky, E., Kastner, K., Varoquaux, G., and Blaschko, M. B. (2017). Learning GRN-guided simulators like SERGIO, and demonstrate the application of these simulators in training deep learning models apart from benchmarking algorithms were ran on GPUs (NVIDIA P100s) while the traditional methods were ran on CPU having a single node with 28 cores.

3.5 Runtimes of different methods

Tables 1 & 2 show the inference time required for different methods with the TF information included. We run different methods on different platforms and hence comparing them directly is not fair. Although, we include them to give an idea of the runtimes to the reader. The traditional methods take multiple hours to run in absence the TF information.

4 Conclusions and Discussions

We present a novel unrolled algorithm GRNUlar, for the inference of gene regulatory networks from scRNA-seq data. The GRNUlar model’s deep architecture takes its inductive bias from the regression based formulation of the GRN recovery problem. We make use of a neural network in a multi-task learning framework to model the regression between TF and other genes. In cases where the TF information is available, we show that GRNUlar consistently performs better than representative existing methods under various settings of both simulated data and real experimental data, especially in the more realistic case of added technical noise. The deep learning framework accommodates the nonlinearity of the regulatory relationships and provides tolerance to the technical noise in the data. We also show the superior performance of a recently developed unrolled algorithm GLAD in absence of TFs.

The methods we propose are the first supervised deep learning algorithms for GRN inference from scRNA-seq data like clustering and trajectory inference (Luecken and Theis, 2019), by using the available realistic simulators for scRNA-Seq data.

**Table 1. Inference runtimes for the GRNUlar model with 2 layer NN, as we vary the hidden layer dimensions $H_z$. The time is reported for one complete forward pass (good$\text{BNN}$ and fit$\text{DNN}$-fast) for $D=1200$ genes graph. Other relevant parameters were $P=5$, $L=15$, $\text{BN}$ epochs $E=100$.**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (secs)</td>
<td>180</td>
<td>612</td>
<td>2020</td>
<td>0.79</td>
<td>1.33</td>
</tr>
</tbody>
</table>

**Table 2. Inference times for different methods on $D=2000$ genes graph. The unraveled algorithms were ran on GPUs (NVIDIA P100s) while the traditional methods were run on CPU having a single node with 28 cores.**

Acknowledgements

We thank Adriya Pratapa and Prof. T. M. Muraki for sharing the gold standard networks for real data used in their paper Pratapa et al. (2020).

References


Theis, J. (2019), by using the available realistic simulators for scRNA-Seq data (Zhong et al., 2019; Dibariun and Sinha, 2019).

The methods we propose are the first supervised deep learning algorithms for GRN inference from scRNA-seq data. Our models learn the characteristics of the underlying GRNs through the simulated data from GRN-guided simulators like SERGIO, and demonstrate the application of these simulators in training deep learning models apart from benchmarking computational methods. Similar techniques can be investigated, not only for the task of GRN inference, but also for other analysis tasks for scRNA-Seq data like clustering and trajectory inference (Luecken and Theis, 2019), by using the available realistic simulators for scRNA-Seq data (Zhong et al., 2019; Dibariun and Sinha, 2019).

Acknowledgements

We thank Adriya Pratapa and Prof. T. M. Muraki for sharing the gold standard networks for real data used in their paper Pratapa et al. (2020).
**Supplementary Material**

**A: GLAD and proposed modification**

The GLAD work formulates the sparse graph recovery problem (undirected graphical model) for the GRN recovery as fitting a multivariate Gaussian distribution on the input gene expression data with a $L_1$ normalization term. It is based on the unrolling the iterations of an Alternate Minimization algorithm for the graphical lasso problem with applications towards sparse graph recovery, refer Algorithm 3.

**Algorithm 3: GLAD**

Function `GLADcell($\vec{S}$, $\Theta$, $Z$, $\lambda$):

\[
\begin{align*}
\lambda & \leftarrow \Lambda_{nn}(\|Z - \Theta\|^2 + \lambda) \\
Y & \leftarrow \lambda^{-1}S - Z \\
\Theta & \leftarrow \tfrac{1}{2}(-Y + \sqrt{Y^\top Y + 4I})
\end{align*}
\]

For all $i, j$ do

\[
\rho_{ij} \leftarrow \rho_{nn}(\Theta_{ij}, S_{ij}, Z_{ij})
\]

$Z_{ij} \leftarrow \eta_{nn}(\Theta_{ij})$

return $\Theta, Z, \lambda$

Function `GLAD($\vec{S}$):

\[
\eta_0 \leftarrow (\vec{S} + tI)^{-1}, \lambda_0 \leftarrow 1
\]

For $k = 0$ to $K - 1$ do

$\Theta_k, Z_{k+1}, \lambda_{k+1} \leftarrow GLADcell(\vec{S}, \Theta_k, Z_k, \lambda_k)$

return $\Theta_K, Z_K$

Function `GLAD-TF($\vec{S}$):

\[
\Theta_0 \leftarrow (\vec{S} + tI)^{-1}, \lambda_0 \leftarrow 1
\]

For $k = 0$ to $K - 1$ do

$\Theta_k, Z_{k+1}, \lambda_{k+1} \leftarrow GLADcell-TF(\vec{S}, \Theta_k, Z_k, \lambda_k)$

return $\Theta_K, Z_K$

Function `GLADcell-TF($\vec{S}$, $\Theta$, $Z$, $\lambda$, $\Lambda_n$):

\[
\begin{align*}
\lambda & \leftarrow \Lambda_{nn}(\|Z - \Theta\|^2 + \lambda) \\
Y & \leftarrow \lambda^{-1}S - Z \\
\Theta & \leftarrow \tfrac{1}{2}(-Y + \sqrt{Y^\top Y + 4I})
\end{align*}
\]

For all $i, j$ do

\[
\rho_{ij} \leftarrow \rho_{nn}(\Theta_{ij}, S_{ij}, Z_{ij})
\]

$Z_{ij} \leftarrow \eta_{nn}(\Theta_{ij})$

$Z \leftarrow \text{mask-TF}(Z)$

return $\Theta, Z, \lambda$

**GLAD-TF model**: We modify the architecture of GLAD to include prior information about the TFs if available. We assume that all the edges in the actual GRN have at least one gene which belongs to the list of TFs. We introduce a masking operation at every step of the unrolled iterations as shown in Algorithm 4, which eliminates all the unwanted edges that are between the non-TF nodes.

**B: Details of preprocessing data from BEELINE framework**

Preprocessing the real data: For each gene expression data and its corresponding network, we do the following preprocessing to prepare our data. We first sort all the genes according to their variance and select the top 500 varying genes. We also have access to a list of known TFs. We only consider all the TFs whose variance had p-value at most 0.01. Now, we find the intersection between the top 500 varying genes and all the TFs to find the subset of genes which act as the TF. Table 3 in Supplementary-B.

We then select the sub-graph of top 500 varying genes from the underlying GRN as our ground truth for evaluation.

**Algorithm 4: GLAD-TF**

Function `mask-TF($\Theta$, TF-names):

\[
\begin{align*}
\text{mask} = &\text{initialize adjacency matrix with zeros} \\
\text{for } n \text{ in TF-names do} &\text{mask}[n, :] = 1
\end{align*}
\]

return $\Theta_M * \text{mask}$

Function `GLADcell1-TF($\vec{S}$, $\Theta$, $Z$, $\lambda$):

\[
\begin{align*}
\lambda & \leftarrow \Lambda_{nn}(\|Z - \Theta\|^2 + \lambda) \\
Y & \leftarrow \lambda^{-1}S - Z \\
\Theta & \leftarrow \tfrac{1}{2}(-Y + \sqrt{Y^\top Y + 4I})
\end{align*}
\]

For all $i, j$ do

\[
\rho_{ij} \leftarrow \rho_{nn}(\Theta_{ij}, S_{ij}, Z_{ij})
\]

$Z_{ij} \leftarrow \eta_{nn}(\Theta_{ij})$

$Z \rightarrow \text{mask-TF}(Z)$

return $\Theta, Z, \lambda$

Function `GLAD-TF($\vec{S}$):

\[
\begin{align*}
\eta_0 & \rightarrow (\vec{S} + tI)^{-1}, \lambda_0 \leftarrow 1 \\
\text{for } k = 0 \text{ to } K - 1 \text{ do} &\Theta_k, Z_{k+1}, \lambda_{k+1} \leftarrow GLADcell-TF(\vec{S}, \Theta_k, Z_k, \lambda_k)
\end{align*}
\]

return $\Theta_K, Z_K$

---

**Table 3. Details of expression data from the BEELINE framework.** To total number of genes for each data is 500 (highest varying genes).