Mathematics and Machine Learning Internship

Written Report Group 3 - Causal Deep Learning

Introduction

Causal Deep Learning - Motivation

Causal Deep Learning aims to combine statistical causal inference with the power of deep learning, creating AI capable of reasoning about cause-and-effect. Expected advantages over traditional deep learning are better generalization, higher interpretability and easier modeling of outcomes after intervention.

This large field of working with statistics and AI in a causal environment can be separated into two subfields which differ in the way they are approached.

Causality for Deep Learning uses already existing causal relationships to enhance Deep Learning Capabilities. For example the causal relationship *light switch → light* might be given to an autonomous robot which is tasked with turning on the light in an unknown room. This causal knowledge might lead the robot to search for light switches instead of all kinds of switches or just randomly trying out things.

Deep Learning for Causality tries to recover causal relationships in the first place. So when given some data we apply deep learning techniques to find these causalities which reveal some information about our data. We can then use this gathered knowledge later on. Biology and medicine are obvious fields where this approach is (trying to be) applied. For example diagnosing liver disorders[1] from certain parameters like gallstones, alcoholism and more. We also further evaluated Causal Discovery on a Protein Signaling Network[2] where the influence of proteins on each other is measured.

We focused on the second part of Causal Deep Learning and compared different algorithms beyond different datasets and created a tool which allows users to easily make these benchmarks themselves with different parameters, datasets, etc.

Before introducing our Causal Discovery Platform we will go over some basics and then take a look at two key algorithms we mainly analyzed in our project.

Directed Acyclic Graphs

A Directed Acyclic Graph (DAG) consists of vertices and directed edges with the additional constraint, that there is **no directed path** $v_1 \rightarrow v_2 \rightarrow \dots v_n \rightarrow v_1$ so no loops are present in the graph. In the scope of learning and utilizing causality, DAGs are used to represent causal relationships. Vertices represent variables, while a directed edge stands for causation, e. g. $v_1 \rightarrow v_2$ means v_1 directly influences v_2 .

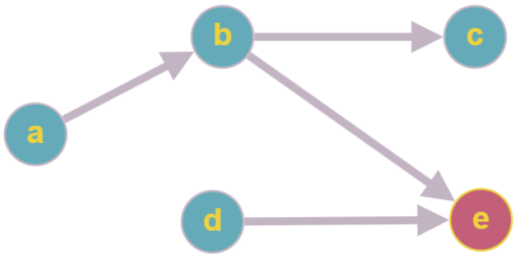
[1] Onisko et al., “A Probabilistic Causal Model for Diagnosis of Liver Disorders.”
[2] Sachs et al., “Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data.”

Structural Equation Models

Structural Equation Modeling (SEM) provides a powerful framework to model and reason about causal relationships between variables of a dataset. In SEM, causal mechanisms are encoded using systems of equations that describe how each variable is generated as a function of its direct causes and an associated exogenous noise term. The goal herein is to analyze not only observable variables but rather to learn underlying latent constructs, which have to be inferred from the observable variables.

As such SEMs embedded into neural architectures or implemented into Deep Learning allow encoding of causal structures rather than correlation, enable intervention and counterfactuals, which are important steps to move from pattern detecting models to cause-and-effect reasoning artificial intelligence.

Let's further clarify this concept by taking a look at an example. Suppose we have the following DAG:



We can see this graph is indeed a DAG and furthermore we can describe some of its properties the following way:

- Variables: $X = \{X_a, X_b, X_c, X_d, X_e\}$
- Noise: $z = \{z_a, z_b, z_c, z_d, z_e\}$
- Causal parents of X_e: $X_{pa(e)} = \{X_b, X_d\}$
- Structural Equations: $X_i = f_i(X_{pa(i)}, z_i) \qquad \forall i \in \{a, \dots, e\}$

These SEMs can be used to model the relationships in our graph and are what we try to recover in the end. In this model the value of a variable is calculated by some kind of function which takes the parents values as an input together with some unknown random noise value. Our node e could be linear dependent on its parents nodes with the linear SEM $X_e = 3 \cdot X_b + 0.3 \cdot X_d + z_e$ for example. Finding the coefficients (and the function) is essentially what we are trying to do. We therefore evaluate our DAG represented by an SEM over the given data and calculate some kind of loss to see how well our DAG fits the data.

Challenges

One of the major challenges in causal discovery is that the search space grows super-exponentially with the number of variables (vertices in our graph). Traditionally algorithms went with a score-based approach where some kind of scoring function is used to

find fitting DAGs. The search space can be somewhat narrowed down through local heuristics or greedy techniques, but this does not solve this problem overall in a fast and reliable manner. In general it is proven that learning bayesian networks from data is an NP-hard problem[3].

Therefore it is interesting to use continuous optimization techniques on this problem. Recent advancements have motivated a different formulation which has sparked a lot of new algorithms with increasingly better performance. This has motivated us to take a look at these algorithms to analyze, understand and compare them.

Algorithms

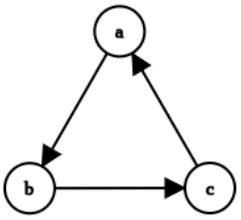
We identified multiple algorithms which leverage deep learning to discover causal relationships from given data. We focused on the DAGs with NO TEARS[4] algorithm which was one of the first approaches to reformulate the causal structure learning problem as a continuous optimization problem.
However this algorithm assumes a linear dependency among the data variables which leads to lower performance in nonlinear relations which can often be found in real-world scenarios. Take for example the relationship between the amount of fertilizer given to a plant and its crop yield. While adding fertilizer leads to higher crop yields in the beginning, adding more and more fertilizer will not have the same production gain due to nutrient saturation of the plants and adding too much might even lead to oversaturation and have negative effects on the yield.

NOTEARS - linear

The main achievement of the paper that proposes the NOTEARS algorithm is to characterize acyclicity of a graph in a different way. Let's take a look at the following examples to get a good understanding of the key findings:

Suppose we have a graph A and its corresponding binary adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

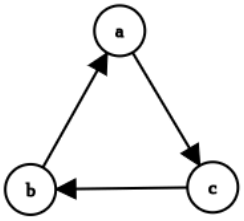


Note: for the sake of simplicity and readability A will refer to both the graph and the adjacency matrix based on the context

If we multiply the adjacency matrix by itself we get the following result:

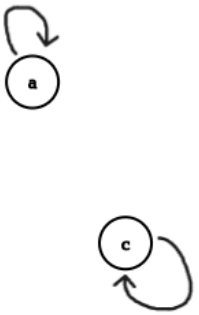
[3] Chickering et al., "Large-Sample Learning of Bayesian Networks Is NP-Hard."
[4] Zheng et al., "DAGs with NO TEARS."

$$A^2 = A \cdot A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



Let's do this once more:

$$A^3 = A^2 \cdot A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



We can quickly see that after the first multiplication A^2 represents our initial graph after two steps. That is from every node we follow exactly 2 edges and see which node we end in. Therefore there is an edge $b \rightarrow a$ in A^2 because the path $b \rightarrow c \rightarrow a$ exists in our initial graph A.

This can be generalized to all directed graphs such that A^k represents the graph A after exactly k steps.

Furthermore by taking a look at A^3 we recognize that after 3 steps we always end up in the vertex we started in. This shows us there exists a cycle in A, which can also be seen by taking a look at the diagonal entries in the adjacency matrix A^3. Checking for the cyclicity of our graph was easy for us by just taking a look at the graph in the beginning, but this might not be the case for large scale graphs.

With these findings we can conclude the following formula:

$$A \in DAG \Leftrightarrow tr\left(\sum_{k=1}^{\infty} A^k\right) = 0$$

Where $tr(A)$ stands for the $trace$ of A which sums the diagonal entries of the matrix. This means we can multiply A by itself for an infinite amount of times, sum all of the products and then evaluate its trace to check whether our matrix is acyclic or not / is a DAG or not.

Let's extend the formula bit:

$$A \in DAG \Leftrightarrow tr\left(\sum_{k=0}^{\infty} A^k\right) = d$$

We are now starting at index 0 instead of 1 which we have to account for by adding d (the amount of vertices) at the other side. That is because $A^0 = I$ and $tr(I) = d$ (with I being the identity matrix).

One major problem is that as k grows, our sum becomes larger and larger. Also there is currently no way to handle the infinite sum computationally. We therefore want to include some type of reweighting such that our sum approaches a limit at some point. We choose to reweigh our formula by dividing it by $k!$:

$$A \in DAG \Leftrightarrow tr\left(\sum_{k=0}^{\infty} \frac{A^k}{k!}\right) - d = 0$$

Not only does this guarantee that our sum approaches a limit at some point (faculties grow faster than polynomials) it also gives us the possibility to recharacterize our formula by introducing the *matrix exponential* e :

$$A \in DAG \Leftrightarrow tr(e^A) - d = 0$$

Now we are almost there with just one little change we need to make: going from binary adjacency matrices to continuous ones (also called weighted adjacency matrices). This change is motivated by the fact that we want to perform some optimization (searching for a minimum) on this formula instead of just checking the DAG search space which was the common approach beforehand.
Therefore we need our function to be smooth to be able to compute its derivatives at any given point. This does not work with binary adjacency matrices.

With continuous adjacency matrices now in play we also allow negative edge weights which leads to problems when calculating the matrix exponential (keep in mind we saw the exponential as resembling our initial matrix after k steps. Negative edges destroy this property). However there is a simple fix for this problem: we just multiply each entry by itself resulting in only positive edges. This is also referred to as the *Hadamard product* $\circ$.

Now our final formula h looks like this:

$$h(A) = tr\left(e^{A \circ A}\right) - d$$

where $tr(A)$ is the $trace\ of\ A$ (sum of diagonal entries), d is the amount of vertices of our graph A, $\circ$ is the *Hadamard product* and e the matrix exponential.

As explained above this formula represents our initial matrix A after any amount of steps (weighted as the steps get larger). We then take a look at the diagonals to see if for a certain amount of steps we can walk in a loop to check for cyclicity. If and only if A is free of cycles then $h(A) = 0$. This is the main result of the DAGs with NO TEARS paper:

$$A \in DAGs \Leftrightarrow h(A) = 0$$

This smooth and continuous function can now be used for a constrained optimization. In the paper augmented Lagrangian is used which essentially introduces a growing penalty for h which allows cyclic versions of our graph in early steps of optimization but punishes them harder later on. The closer h is to zero the closer our estimated graph is to a DAG.

However in optimization we will almost never get h to equal zero because a certain cyclic property might fight the data better or we are stuck in a local minima of some sort. This means to get a DAG from this algorithm we need to cut edges below a certain threshold to achieve acyclicity. In the paper the authors choose a value of 0.3 which was chosen experimentally. *(Note: The other Causal Deep Learning group from the MPI Mathematics and Machine Learning Internship has focused a bit more on choosing a fitting cutoff value)*

NOTEARS - nonlinear

For now we assumed that the dependencies in our data are some kind of linear relationships. However in reality this does not have to be the case (take fertilizer vs crop yield from above as example). We therefore want to generalize the approach to nonlinear SEMs as well. Our structural equations now look like this:

$$X_i = f_i\left(X_{pa(i)},\, z_i\right) \quad \forall i \in \{1,\, \ldots,\, n\}$$

where f_i equals any kind of differentiable function. Now if we try to apply the previous approach we quickly run into the problem of how to define our matrix A. We can still use an adjacency matrix to note dependency of variables but not in which way they depend on each other (what is f)
This extension of A to some other kind of matrix which can then be used again with the formula h we constructed above to characterize acyclicity is one of the major aspects in the second paper we analyzed: Learning Sparse Nonparametric DAGs[5]. For the sake of simplicity we will not go into too much detail here but just try to give the reader a general idea on what the paper does and how it extends the linear case to nonlinear ones.

Let's talk about some mathematical prerequisites needed for the extension in an intuitive way. We want to find out in which way a certain vertex is influenced by its parents through a function. This can also be thought of in a different way: how does changing the value of a parent node influence my child node?

When talking about the changes of functions derivatives are usually not far. Furthermore we want to find out how a single vertex influences another single vertex. This is where partial derivatives come into play. Let's say we want to check how much X_j depends on X_k. We can characterize X_j by its function f_j plus some noise (which we do not care about here). So now the question becomes how does X_k influence f_j ? That is, how does the value of our function change when a certain input changes? This is exactly what the partial derivative of f_j with respect to X_k calculates, denoted by $\partial_k f_j$.

Now that we have our characterization of change we need to somehow measure it in order to see if changes are small, big or not present at all. So the key aspects that our tool of measurement (denoted by $||\circ||$) should fulfill are the following:

[5] Zheng et al., "Learning Sparse Nonparametric DAGs."

1. If X_j does not depend on X_k then $\left\|\partial_k f_j\right\| = 0$
2. Vice-versa if X_j does depend on X_k then $\left\|\partial_k f_j\right\| > 0$
3. If X_j depends on X_k more than on X_l then $\left\|\partial_k f_j\right\| \geq \left\|\partial_l f_j\right\|$

All these properties are fulfilled by the *L2-norm* for functions denoted by $||\circ||_{L^2}$. This norm works similar to the *L2-norm* ($||\circ||_{l^2}$) for vectors also known as *Euclidean length*. Calculating the Euclidean length of a vector (a, b) is done by taking the root of the sum of the squares of the entries: $||(a, b)||_{l^2} = \sqrt{a^2 + b^2}$.

This can be applied to functions as well to measure the total magnitude of a function on an interval $[a, b]$. We just exchange the sum for an integral (which is somewhat similar):

$$||f||_{L^2} := \sqrt{\int_a^b f(x)^2 \, dx}$$

We can now combine both the quantification of change through partial derivatives and the measurement of said changes by the *L2-norm* into a matrix B:

$$[B]_{kj} := \left\|\partial_k f_j\right\|_{L^2}$$

So the entry $[B]_{2,5}$ is a numerical value which measures the influence of the second vertex on the fifth vertex without assuming a linear or some other kind of relationship. This change in the meaning of our matrix allows us to now apply the same kind of characterization for acyclicity (the function h defined above) to nonlinear cases. h is still smooth and continuous, keeping all its wanted properties.

It is also interesting to think what happens when f_j is linear for $\forall j \in \{1, \ldots, d\}$. In this case taking the partial derivative results in the corresponding coefficient which overall just equals the usual weighted adjacency matrix. Therefore this new characterization is truly a generalization of the linear case.

There are a few more important steps described in the paper which are needed to get the algorithm fully working. One of these is how to approximate the functions which are describing the relationship. The authors are proposing two methods: using a Multilayer perceptron (MLP) and Orthogonal basis expansions. These are trying to find the underlying ground truth function by analyzing the given data. Afterwards Dual Ascension is applied again and the problem can be optimized once more.

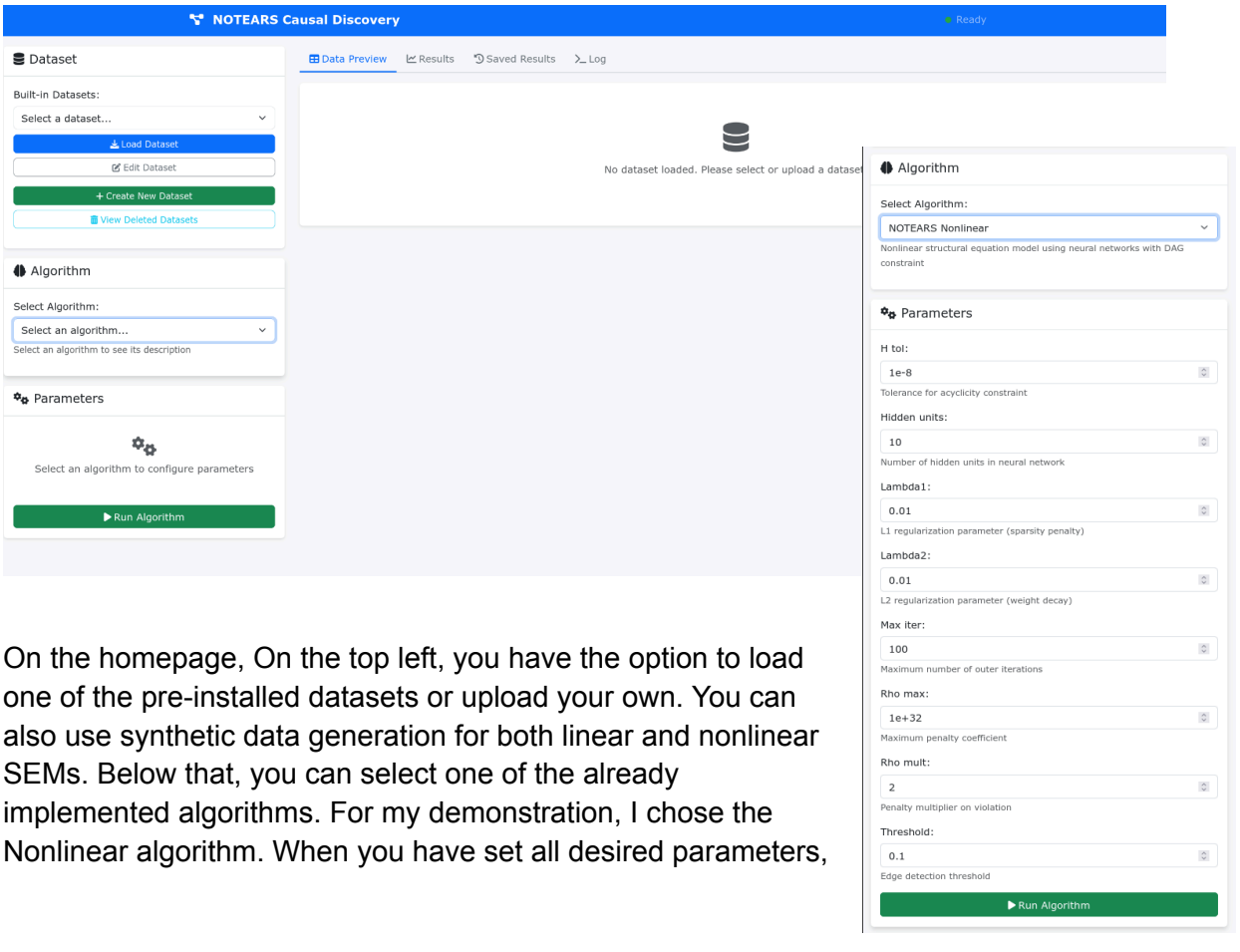
Causal Discovery Platform

Overview

After we acquired the fundamentals of Causal Deep Learning by reading the papers, our next step was to test the discussed algorithms ourselves. For this, we first adopted the code from the GitHub repository for the "DAGs with No Tears" paper one-to-one and ran it with test datasets. In doing so, we found that it is difficult to replicate the results achieved in the paper, and we had to run the algorithm very frequently with different parameters. Managing and evaluating the many results quickly became very confusing. Therefore, we decided to build a graphical web-based platform with which we can easily and clearly test different algorithms and evaluate the results. In the following section, I will demonstrate how to use the tool.

Usage

The ones who want to try the tool themselves, can check out our GitHub repository (https://github.com/moritzfUniLE/MathAndML). There, the installation and functionality are described in detail in the README. Nevertheless, I also will present the most important features here.

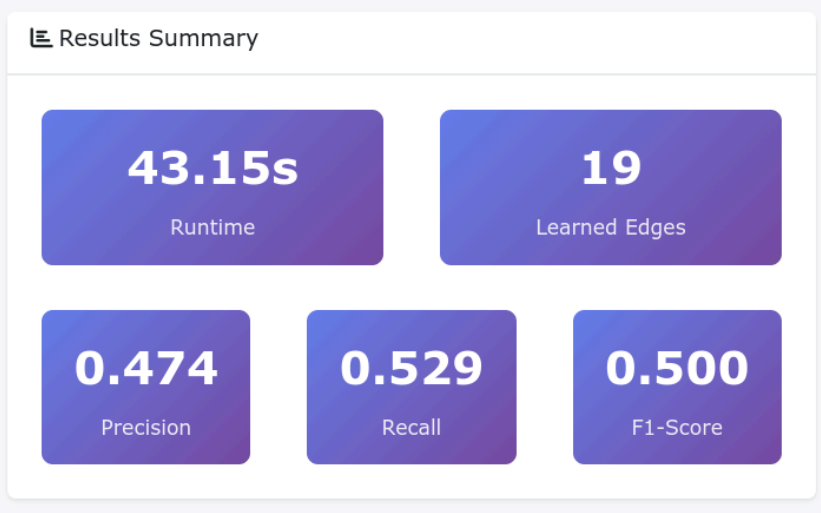


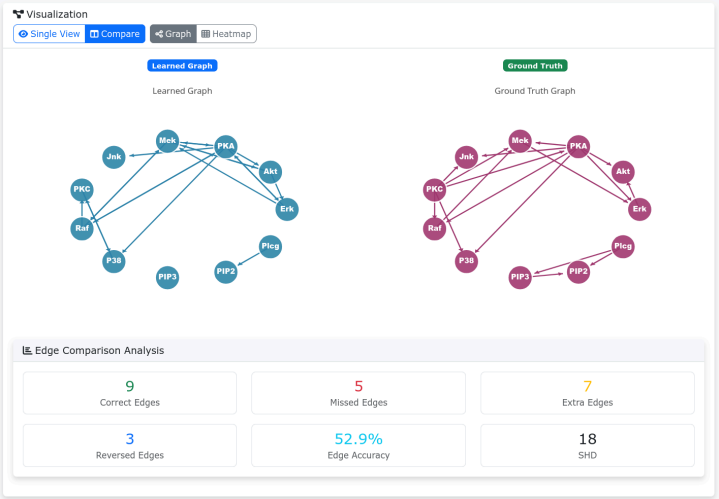
On the homepage, On the top left, you have the option to load one of the pre-installed datasets or upload your own. You can also use synthetic data generation for both linear and nonlinear SEMs. Below that, you can select one of the already implemented algorithms. For my demonstration, I chose the Nonlinear algorithm. When you have set all desired parameters,

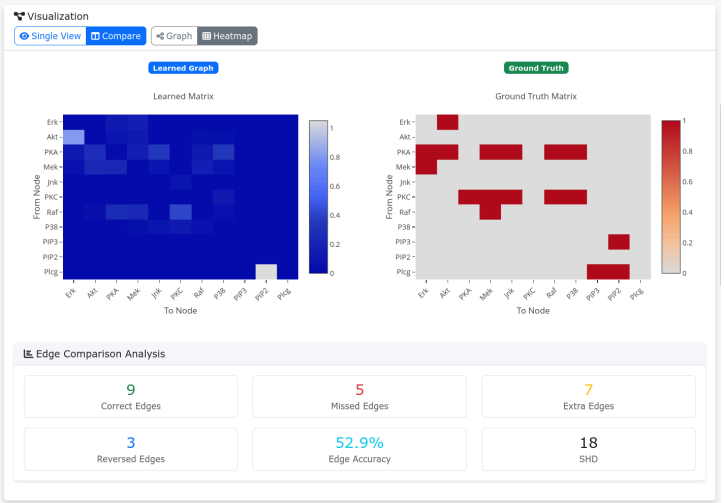
you can start the algorithm and monitor the individual iterations in the Log tab.

```
[18:20:59] Starting NOTEARS Nonlinear...
[18:20:59] Parameters: lambda1=0.01, lambda2=0.01, max_iter=100, hidden_units=10, h_tol=1e-08, rho_max=1e+32, rho_mult=2, threshold=0.1
[18:21:05] Iter 1: h=0.45049953, ρ=1.00e+0
[18:21:08] Iter 2: h=0.31138325, ρ=2.00e+0
[18:21:11] Iter 3: h=0.18346977, ρ=4.00e+0
[18:21:14] Iter 4: h=0.11505318, ρ=8.00e+0
[18:21:15] Iter 5: h=0.08096313, ρ=1.60e+1
```

After the algorithm has completed successfully, you are automatically redirected to the Results page, where your results are compared with the ground truth of the dataset (if available). Also various evaluation metrics are displayed. Additionally, you receive a graphical representation of the learned edges and can compare these with the ground truth graph.







This will sum up our short demo of the tool. Have fun trying it out yourself :).

Summary

In this report we tried to give the reader a general idea of what Causal Deep Learning is about. What kind of formal mechanisms are used to represent causal dependencies and how recent advancements in the reformulation of the DAG learning problem has sparked new algorithms which leverage continuous optimization techniques to get faster & better results.

Our own contribution was building the Causal Discovery Platform, a plug-and-play web application which can be used for evaluating DAG estimation algorithms on any kind of dataset while also providing evaluation and visualization of the results.

What worked & what didn’t

This project was a great opportunity to dive into the realm of Causal Deep Learning and also get some idea of scientific work. Reading and understanding the paper was an unknown challenge for us in the beginning and we struggled to really get a good grasp on the topic. However after a few weeks of carefully analyzing, identifying different resources and sharing our ideas and questions we were able to understand the mathematical background on the presented algorithms and also finally apply them.

Things worked well with the NOTEARS - linear algorithm due to it being pretty much off the shelf and not having too many parameters to tweak. Our results were comparable to the ones presented by the authors in their paper which gave us some confidence moving on.

However when we moved on to the NOTEARS - nonlinear version we were not able to reproduce results even close to the paper. The algorithm runtime was kind of bad which prevented some kind of hyperparameter tuning with our hardware. We felt the algorithm’s results heavily relied on choosing said hyperparameters and therefore we did not want to carry out some benchmarking with possibly corrupted results. This was one of the main motivations to create the Causal Discovery Platform: allowing the user to tweak parameters and benchmark algorithms on datasets by themselves.

For future projects a more even workflow might lead to better results. We had intense weeks where we made lots of progress and other weeks where we did not work on the project at all. While this is most likely a general problem of students when being in university, taking a balanced approach probably ensures success in the long run.

Overall it was a really great project in which we very happily participated in. A lot of trust was put into us but without leaving us behind in any sort. Dia was always open for questions and we really learned a lot even beyond the project's task.

References

- https://sites.google.com/view/causalityanddeeplearning/start
- https://arxiv.org/abs/1803.01422
- https://proceedings.mlr.press/v108/zheng20a/zheng20a.pdf
- https://www.science.org/doi/full/10.1126/science.1105809
- https://github.com/xunzheng/notears