Causality in Machine Learning: Class projects

Robert Ness

This course focuses on the application of causal inference theory in generative machine learning. The course project requires you to implement a cutting-edge causal machine learning workflow.

You are implementing your project in groups of 2-3 students. The final deliverable is a Github repository that includes notebooks with code and description, and a presentation of your results to the class.

Please choose one of the following projects.

Deep causal variational inference

This project is ideal for students interested in deep learning.

In this project, you will train a supervised variational autoencoder using Deepmind's dSprites dataset.

dSprites is a dataset of sprites, which are 2D shapes procedurally generated from 5 ground truth independent "factors." These factors are color, shape, scale, rotation, x and y positions of a sprite.

All possible combinations of these variables are present exactly once, generating N = 737280 total images.

Factors and their values:

- Shape: 3 values {square, ellipse, heart}
- Scale: 6 values linearly spaced in (0.5, 1)
- Orientation: 40 values in (0, 2π)
- Position X: 32 values in (0, 1)
- Position Y: 32 values in (0, 1)

There is a sixth factor for color, but it is white for every image in this dataset.

The purpose of this dataset was to evaluate the ability of disentanglement methods. In these methods, you treat these factors as latent and then try to "disentangle" them in the latent representation.

However, in this project, you will not treat these factors as latent, and include them as labels in the model training. Further, you will invent a causal story that relates these factors and the images in a DAG such as in the following figure:

You would implement this causal story as a structural causal model of the form:

The image variable will be a 64 x 64 array. The noise term for the image variable will be the traditional Gaussian random variable. The structural assignment g for the image variable will be the decoder.

Deliverables

- Create a DAG and an SCM articulate a causal story relating shape, orientation, scale, X, Y, and the data.
- Resample the dataset so you get a new dataset with an empirical distribution that is faithful to the DAG and is entailed by the SCM (instructor will provide guidance).
- Using Pyro, implement the causal VAE. The instructor will provide you with a notebook that implements a primitive version of the model, including code for setting up data loaders, reshaping of tensors, and the learning algorithm. The instructor will provide you with guidance on how to incrementally expand this model.
- Use the trained model to answer some counterfactual queries, for example, "given this image of a heart with this orientation, position, and scale, what would it have looked like if it were a square?"

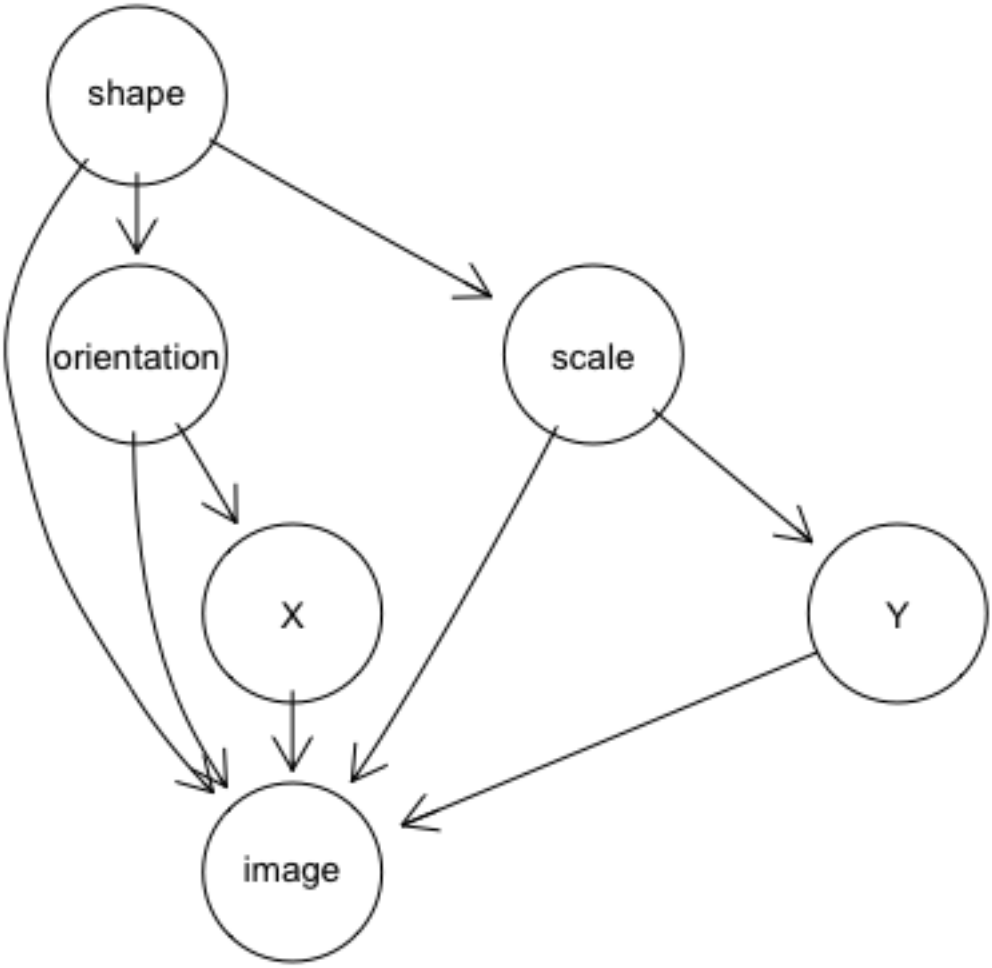

Figure 1: vae_dag

$$
\begin{aligned}
\text{shape} &= f_{\text{shape}}(N_{\text{shape}}) \\
\text{orientation} &= f_{\text{orientation}}(\text{shape}, N_{\text{orientation}}) \\
\text{scale} &= f_{\text{scale}}(\text{shape}, N_{\text{scale}}) \\
\text{X} &= f_{\text{X}}(\text{orientation}, N_{\text{X}}) \\
\text{Y} &= f_{\text{Y}}(\text{scale}, N_{\text{Y}}) \\
\text{image} &= g(\text{orientation}, \text{scale}, \text{X}, \text{Y}, \text{image}, N_{\text{image}})
\end{aligned}
$$

Figure 2: vae_scm

Counterfactual policy evaluation with Open AI gym

This project is ideal for students who are interested in reinforcement learning.

In this project, you will take an environment from OpenAI gym and demonstrate a counterfactual evaluation use case.

Deliverables

- Select an environment with a discrete state and action space (e.g., the frozen lake models)
- Reimplement the transition function in this environment as a structural causal model. Use the technique described in class and the reference below.
- Create two policies for this environment. You can train them using a standard RL algorithm. Alternatively, you can make the policy up, though it should still be a sensible policy; for example in the frozen lake environment, you could make up a policy that always tries to take a step that minimizes the distance to the goal but still tries to avoid holes.
- Pretend this policy was in production, and generate logs.
- With the second policy, use counterfactual evaluation with the structural causal model to evaluate how well the second policy would have performed had it been in production. Compare the counterfactual expectation of reward, to the expected reward of the in-production model.

Counterfactual Off-Policy Evaluation with Gumbel-Max Structural Causal Models

Deconfounding film predictions

This project is ideal for people who want experience using a "deep" latent variable model in Pyro to solve a practical causal inference question. Most of what you have done so far in Pyro has not taken advantage of PyTorch's ability to work with vectors and models with many parameters (like deep neural nets). This project demonstrates the power of causal modeling with tensors.

In this paper, you will implement a method described in two papers by Yixin Wang et al.; The Blessings of Multiple Causes and The Deconfounded Recommender: A Causal Inference Approach to Recommendation. In these papers, the authors propose a deconfounding technique using a class of models called probabilistic factor models. The core of the approach in each paper is the same, though this description focuses on the problem of predicting box office revenue, described in the first paper. The second paper describes a recommendation system application and is a good choice because it has a clearer standard of evaluation.

When you read these papers, you will see that they are premised on the potential outcomes framework, which is different from the approach we've taken in class. Below I provide some guidance for how to implement things our way.

In the box-office prediction problem, your goal is to predict box-office revenue given which actors will be in a film. A naive approach would be just to train a predictive model, such as a neural net or linear regression, mapping an actor vector to a revenue outcome. However, if the goal is to use this model to choose a set of actors for a film, then this is an intervention problem – instead of `condition({"actor": "Brad Pitt"}`, you `do({"actor": "Brad Pitt"})` when you make this decision.

Your goal therefore is to find $E(Y|do(A_j = 1)) - E(Y|do(A_j = 0))$ for a given actor (or for multiple actors). However, there are unobserved confounders. For example, Samantha Bond played Ms. Moneypenny in several James Bond films, all of which made large amounts of money. However, it would be silly to suggest that she has a large causal effect on box office revenue and that she should be added to a new film. We need a model that deconfounds things like whether or not a film is a Bond film.

The paper proposes the following technique to deconfounding this prediction. Let Y be revenue and $A_1 ... A_m$ be binary variables for the presence or absence of m actors in the database.

1. Fit a probabilistic model M that assumes that the As have a common latent cause Z.
2. Verify this is a good model using a posterior predictive check (you won't have to do this).
3. Augment the data by using the model to estimate $\hat{Z} = E_M(Z|A_1, ..., A_J)$, a vector of predictive values for Z.
4. Estimate the intervention distribution by adjusting for $\hat{Z}$: $P(Y|do(A_j = 1)) = \sum_{\hat{Z}=\hat{z}} P(Y|do(A_j = 1), \hat{Z} = \hat{z})P(\hat{Z} = \hat{z})$.
5. Predict causal effects in terms of box office revenue using $E(Y|do(A_j = 1)) - E(Y|do(A_j = 0))$.

How do we model this?

Data

Use the TMDB 5000 Movie Dataset dataset, as in the paper, with log revenue as the prediction target. There are multiple variables you can use as causes, but you should stick to actors. I suggest starting by taking a small set of actors, and subsetting the data to films where at least one of these actors is present, then gradually expanding the set of actors and the data subset. Note that in this approach there should not be any edges between actor-causes.

Intuition

The intuition for how the technique works is that $\hat{Z}$ behaves as a propensity score, i.e. a statistic that renders the A_j independent of unobserved confounders. By construction, it renders all A_j conditionally independent of one another. If it accomplishes this, then conditioning on $\hat{Z}$ blocks all backdoor paths of dependence between the $A_j $'s through latent confounders. In other words, $\hat{Z}$ is a proxy for unobserved latent confounders. It seems like magic, but the approach succeeds by relying on the multiplicity of actor-causes to even estimate $\hat{Z}$; in this respect, it is an example of a statistic that is not identifiable with univariate data but is with multivariate data. It is not a free lunch, the trade-off is that you reduce confounder bias but causal effect estimates have more variance.

Model

We will use a generative modeling approach slightly different than that described in the paper, and more similar to the above variational autoencoder project described above. The forward model of the data should look like the following (very rough) pseudocode:

```
def model(A_vals, Y_vals):
    # gamma, alpha are hyperparameters
    theta = sample(p(·|gamma))
    for each i
        Zi = sample(p(·|alpha))
        for each j in J # J causes
            Aij = sample(p(·|Z[i], theta[j]), obs=A_vals[i, j])
        Yi ~ sample(p(·|Ai1 ,.., AiJ, Zi), obs=Y_vals[i])
```

Deliverables

The professor has a working example fit on a small subset of the full data. Your job will be to expand it to a larger dataset.

- Implement the predictive model M in Pyro. Provide a well-documented repo and reproducible notebooks
- Estimate causal effects for each actor $E_M(Y|do(A_j = 1)) - E_M(Y|do(A_j = 0))$ both by adjusting for Z and by do-calculus. Compare the two outcomes and make sure they align.

- Implement a comparison predictive model M_2 and estimate the effects for each actor $E_{M_2}(Y|A_j = 1) - E_{M_2}(Y|A_j = 0)$. This should be a simple model with the same structural form as the `Yi = sample(p(·|Ai1 ,.., AiJ, Zi), obs=Y_vals[i])` line in the original model.
- Rank actors by the biggest difference between the two models. Sanity check your results against the original authors' results.
- Add genre into your model, and answer questions where you have do(genre = scifi) or other.
- BONUS: Covert you model to an SCM using the method described in class, and answer a counterfactual question like, "How much more would movie X have made if they cast A instead of B"?

Causal agent models

In this project, you will implement core agent models (e.g. bandits, Markov decision processes, etc.). However, your agent models will feature key elements of causal decision making.

Deliverables

This will be just like implementing a bandit algorithm. The causal elements of this implementation are detailed in design documentation prepared by a past student. If interested, talk to the professor.

Experiments with Omega.jl

Omega.jl is a probabilistic programming package in Julia that has some abstractions for causal inference. In this project, you will create examples using SCMs for counterfactual inference. You will generate examples of basic counterfactual reasoning. You will also create examples that show how to infer the counterfactual entities learned in class (effect of treatment on the treated, probability of necessity and sufficiency, etc.). Your ultimate deliverable will be a tutorial featuring these examples, as well as an analysis of the pros and cons of using Omega for causal reasoning.

Examples of counterfactual reasoning with Omega do not exist outside of a few toy examples in the documentation. Further, the Omega is pretty new, so debugging could be a challenge. For these reasons, this project is ideal for people who enjoy thinking about programming idioms and aren't afraid to dive into the library's codebase when difficulties arise.

DIY causal modeling language

In this project, you build your own prototype causal modeling language using Haskell, Scala, Scheme, R, or whatever you prefer. This project is suitable for students already familiar with design and implementation of programming languages. Here are some reference readings:

Practical Probabilistic Programming with Monads

Write your own general-purpose monadic probabilistic programming language from scratch in 50 lines of (Scala) code

A Language for Counterfactual Generative Models

If interested, ask the professor for more relevant readings.