Feature Interaction In Interpretability:A Case For Explaining Ad-Recommendation Systems Via Neural Interaction Interaction Detection

Michael Tsang, Dehua Cheng, Hanpeng Liu,Xue Feng, Eric Zhou, and Yan Liu
Department of Computer Science, University of Southern California
{tsangm,hanpengl,yanliu.cs@usc.edu}, {dehuacheng,xfeng,hanningz,@fb.com}

Reproducibility Summary

*Template and style guide to ML Reproducibility Challenge 2020. The following section of Reproducibility Summary is **mandatory**. This summary **must fit** in the first page, no exception will be allowed. When submitting your report in OpenReview, copy the entire summary and paste it in the abstract input field, where the sections must be separated with a blank line.*

Scope of Reproducibility Summary

I was recommended by supervisor to take a look at this paper. The topic of this paper is quite interesting. There are many applications of machine learning models when it comes to creating recommendation systems. These models can range from recommending songs on Spotify to add to a playlist or YouTube suggesting which videos it thinks you will like based on what you have previously watched. The same is true of recommendation systems that choose targeted ads for each user online. These are examples of black box recommendation machine learning models. This paper devises a system that will be able to interpret features interactions from a recommendation model and encode these interactions into a new recommendation model where the input and output models are both black box models.

The paper can be found at https://openreview.net/pdf?id=BkgnhTEtDS

The associated repository can be found at https://github.com/mtsang/interaction_interpretability

My fork with all my changes: https://github.com/AidanPolese/interaction_interpretability/

On my fork there is still a big chance that things are still buggy up to where I was halted as I made so many changes between a very large amount of executions.

0.1 Model descriptions

As previously stated, the input and output models that are used in this paper are black box, so there isn't much to be commented on in the model sense. However, there are two major processes used to create the output model and I think those are worth covering. The processes are called MADEX and GLIDER.

MADEX stands for "Model-Agnostic Dependency Explainer" which sounds pretty complicated. However, to put it simply, MADEX takes in the black box recommendation system and their associated dataset and outputs the top detected feature interactions.

The next really important piece of the puzzle is GLIDER. GLIDER stands for "Global Interaction Detection and Encoding for Recommendation". This one is fairly complex, but it is essentially running MADEX over different batches of data to see which features interactions are globally the most important feature interactions of the recommendation system. It is a way to both enhance a black box recommendation system but isolate it and how it works.

Methodology

The general idea of this paper is the improve the recommendation accuracy of black box recommendation systems. The two main systems that are MADEX and GLIDER, they are summarized above. The paper provides code that will allow for a person to reproduce their results using MADEX and then GLIDER. Specifically, it provides instructions on how

Submitted to ML Reproducibility Challenge 2020. Do not distribute.

to use and demo the MADEX algorithm. After MADEX is demo'd and understood, then the user is walked through how GLIDER works. The process goes over how to prepare the same data they used, train the autoint recommendation model that is demo'd in their paper, and then execute the GLIDER algorithm to improve the output of the autoint enhanced recommendation system. The main end goal is to be able to replicate their results when using the autoint model as input with the criteo dataset. It is essentially recreating one vertical slice of their research.

Reproducing this project was quite tricky considering the resources that were used by the authors and what I had access to. The author provided source code that is meant to be used to recreate this paper, but this is a red herring of sorts. The basic skeleton the code is present but there are many pieces of the code that are either extremely buggy or are not present at all. This meant that there was a lot of bug fixing and code hunting that I needed to do. The multitude of issues with the provided code will be elaborated on lower down in this paper.

This paper was based around the use of CUDA technology on Nvidia GPUs and an incredibly powerful CPU on a linux machine. The authors themselves had access to a 32-CPU Intel Xeon E5-2640 v2 @ 2.00GHz server with 2 Nvidia 1080Ti GPUs. Personally, I have access to a MacBook Pro with a 2.6 GHz 6-Core Intel Core i7 and an AMD Radeon Pro 5300M 4 GB and Intel UHD Graphics 630 1536 MB. Very big difference in power level. This meant that I was tasked to see if this paper was possible to reproduce using an Ubuntu Virtual Machine powered by VirtualBox without access to CUDA technology. What's more is that if I did have an Nvidia GPU, Type 2 Virtual Machines like VirtualBox generally don't have GPU pass through so it doesn't really matter if I use a type 1 or type 2 VM. I also do not have access to a Cloud VM as they tend to be quite expensive and do not have the funds to rent one.

Results

For a person with a limited knowledge of machine learning practices and having this course be my only exposure to the subject, along with a huge difference in computing power and resources available, this paper is borderline impossible to accurately and feasibly reproduce. I was able to reproduce about three quarters of this paper until there was a combination of computing resource issues as well as limited knowledge for debugging. I was able to get similar results up until this point, however, a full conclusion of if this paper can be reproduced by the average student with an average laptop remains to be seen. I am quite disappointed that a main sticking point was hardware, but also that I lack specific knowledge of the code. There were also issues regarding library usage and the authors personal files that were also used to be able to progress past this point.

What was easy

I am very tempted to say nothing but there were a couple easy things. The first one being using the first available part of the code, MADEX.

This is where the first hurdle for using the provided code. The file that contains all the required libraries and library version requirements are actually incorrect and have some bugs that inhibit the MADEX process. Specifically, there are library version bugs that caused the jupyter notebook code to not execute properly. Fixing this wasn't super difficult as it just involved looking though patch notes to find the closest version where the libraries that caused issues had these bugs fixed and were still compatible with each other. There were also a few extensions for jupyter that were not stated to be used that had to be installed but this wasn't difficult to fix. There were also issues with how python was pickling variables that needed debugging. So, after all this mild debugging was done, I made a few argument changes to the PyTorch inputs in the notebook and a few internal scripts and was able to get MADEX to run on a CPU only Ubuntu VM. Running MADEX after the environment pieces that were left out were fixed wasn't too difficult and I was able to replicate the MADEX results almost one to one.

What was difficult

The MADEX bugs have been covered above as they are not that difficult to fix but the GLIDER section is a different beast.

There are tons of missing code files for this project. There are 10+ files that are missing from this repository that are necessary for this project to even begin to run, but more on that in a minute.

The first issue is preparing the dataset. The dataset used in this project is called the "Criteo Display Advertising Challenge Dataset". This dataset was first released in 2014 and is no longer available for download. This is no surprise as it is 7 years old. This resulted in a hunt for this dataset. I was unable to find the full dataset after searching high and low, but I was able to find a sample of the dataset containing 10,000 samples. This data was taken from the autoint repository, https://github.com/shichence/AutoInt. This recommendation system repository contains this excerpt of the

dataset. The reason I was able to find this dataset is because autoint is actually one of the blackbox recommendation systems that is used for this paper to be improved upon.

This led into step A which is the data preparation section. This section wasn't super hard to follow as most things in this section worked, with the minor change of a few internal variables had different names which needed to be fixed.

Now comes the scary piece, Step B, Global Interaction Detection. This involves the training of a baseline Autoint model. There were library inconsistencies which need debugging. This came down to features that were supposed to be present which were bugged within the library itself. This required library debugging things like scikit-image and its compatibilities. This is also a little scary because this project indicated that it required CUDA to run. This section had a flag set to use the GPU, which I do not have, so it's output is a little shaky. Also, there was an issue with pathing. The code indicates a path with inconsistent casing with the actual project and the path is also minorly incorrect. This whole process is just covered in bugs.

And then it was time to detect global interactions. This file called detect_global_interactions.py has so many issues which made this process incredibly frustrating. I am convinced that this code was uploaded and not even used for the actual paper, if that wasn't evident from all the bugs and library version inconsistencies. There are tons of import statements referring to 10+ files that don't exist in this repo at all. I had to search all over for these files. I had to crawl through the repositories of the authors to find these files. They ended up being in a completely different project from the user mtsang, https://github.com/mtsang/neural-interaction-detection. The files are in this repo which took much too long to find. Then I had to change all the paths to these files for it to play nicely with the main script. This was a lot of debugging and searching for files. There are also type problems which I needed to resolve for input arguments.

What's more is that feeding specific arguments into —par_batch_size gave errors on input no matter what was entered and only the default argument would work. Also, the —save_id given does not work, so I had to again use the default argument. However, this default argument will prove problematic as this default argument and default output do not match up with the default arguments of step C. This required more fiddling around with paths and variable names.

Also, we are not done with the issues present in step B still. The examples save path does not match up with the folder structure of this project, so it needed to be changed, then the default input for the –data_file in step C is also incorrect and needed to be changed.

There is also an issue in step B part 2 which would give inconsistent output, not in a learning sense but in an error sense. Sometimes it will give multi-layer perceptron errors and other times it will work just fine. However, no matter on the output it gave, error or not, the files that it is supposed to generate for step C are either not present or the shape is incorrect. This issue is what halted my progress. This is where my debugging came to an end. I was unable to see if this was a hardware limitation or something deeper in the code. It had hit the point where chaos of the code became overwhelming to a point where it could no longer be feasibly debugged during this global interaction step leading into generating cross features. I looked into the code but I must admit that my knowledge of this subject is holding me back considering the insane complexity of both the algebra and code structure of included and excluded and then imported files.

1 Discussion

So, I believe it is safe to say that this repository was not actually used when conducting the research in the paper. I'm sure some of it was but a heavily altered version that could possibly contain even more files and have more things in the software development environment that can't be found conventionally. Only being able to get through part of this paper with extremely heavy debugging is disappointing for me. I'm sure the intentions of the writers were positive, and they did not intend to release something so broken. It was likely a case of "it works on my machine" and not remembering to include everything. A very relatable problem I must say. However, it did make this paper replication task almost impossible for a person of my knowledge and skill level. It might even be true of people who are much smarter than I am because there could be more files or internal issues that are beyond debugging, it's really difficult to say.

In concept, this paper is very cool. Being able to improve on black box systems is very mystifying in concept. Look at all the necessary algebra and the algorithms, it begins to make sense in concept. However, in practice, the implementation is a monster.