Yashaswi Sharma
May 17th, 2022
ECON 499
Dr. Vincent

Report

Original Vision

My original vision for the project was to answer how, where, and when incentive compatibility and revenue maximization were optimized in RegretNet throughout its pair of allocation and payment neural networks. RegretNet is attributed to have birthed the entire field of Differentiable Economics, an offspring of the more popularly termed Computational Economics. Given the great importance accredited to RegretNet, and the significant claims to progress it has made in designing single-shot multi-agent multi-item auctions, I saw that there was a significant drought in model explainability research as to why RegretNet performs as optimally as it is claimed. In effect, I wanted to analyze, dissect , and explain why RegretNet performs the way that it does. I wanted to see which of these two properties RegretNet prioritizes first, where these properties are maintained and exploited, explore why RegretNet produces revenue maximizing and dominant strategy incentive compatible results, and draw conclusions as to what to possibly improve in analytically designing auction mechanisms that mimic RegretNet in revenue and incentive optimization.

By design, my method to achieve this goal was to partition/slice the models by layer and to observe the properties of each layer's output. Each layer in a neural network by itself represents a miniature non-linear function. My belief is that since these mini-nonlinear layer functions are progressively stacked and are trained regressively, stronger penalized terms in our objective function affect earlier layers significantly more over latter layers. For our case, given the RegretNet objective function's quadratic penalty for violating incentive compatibility constraints, my initial hypothesis was that incentive compatibility would be held first, on which revenue maximization would occur throughout the latter layers. My goal to explore whether this would be true was via analyzing pre-trained models and partitioning their layers such that each layers output could be observed.

Key Misunderstandings

One of my earlier misunderstandings, one that we had gotten a chance to briefly talk about during our meetings, was how regret could be evaluated without a notion of an alternative type. In my earlier readings of the RegretNet paper, I have come to realize, with an appreciation of my ex-post condition on this, that I had originally thought there was some method to deterministically generate what the optimal combined misreport would be given our set of true types per object per agen. This is opposed to how RegretNet actually calculates what the optimal misreport ought to be: another internal optimization protocol that uses gradient ascent to find the optimal misreport with respect to the total utility of a given agent (valuation * allocation probability - payment). With hindsight, I understand that having a deterministic method does not make sense; utility functions, although differentiable, do not produce in themselves what the misreport ought to have been given the true valuation of an agent. Throughout my entire effort in this project, I can say with certainty that this notion of a gradient ascended optimal misreport is a major concept of how RegretNet works to optimize their neural networks, and it is a key property I have familiarized myself with greatly.

Another misunderstanding that led to a major pain point throughout my entire effort in this project was the belief that the utility function was inherently easily differentiable and did not require close monitoring of how the differentiation was taking place with respect to its inputs. Through further research and analysis of the RegretNet paper in my attempt to reverse-engineer it, I realized quickly that our utility functions needed very specific gradient values. Not only did our utility function need to be differentiable with respect to a misreport (something confusing in itself), it also needed to be differentiable with respect to some parametrization (a key vagueness left in the RegretNet paper that hindered my progress in developing a flushed out objective function). Hence, our utility function needed to be properly differentiable against a neural network, a concept that I had never considered nor encountered in my entire prior experience in Machine Learning. Although neural networks are considered to be functional estimators, as discussed earlier, they are a progressive layering of several miniature nonlinear functions. Taking the gradient of the utility function with respect to a neural network model itself was a concept that took me a significant amount of time to fully grasp. However, the story does not end here as there is still a major flaw in this entire analysis: everytime the utility function gets called, we inherently saved the gradients of both neural networks when computing the gradient of the utility

function. A key implementation confusion I still have regarding this is how do we then train via PyTorch the neural networks if they store multiple sets of gradients for every time the utility function is called. Do we take the average of the gradients to update? Do we primarily use the optimizer to update model parameters via the objective function and delete all other gradients that were prior used? Although there are still many unanswered questions, I know with certainty that my time spent on this project greatly allowed me to think about these implementation details in a lot more depth, something I appreciate as I have a greater appreciation for how RegretNet is optimized.

In a similar vein, I also misunderstood how important the regret function is to RegretNet, and how its differentiation is unique. The gradient of the regret function is used to train our models as it is a major component of the gradient of our loss function. Since the regret function inherently uses the utility function (since to be able to calculate regret we need to take the difference between the optimal misreport utility and the true-report utility), explorations over the feasibility in the differentiation of the utility function bled into understanding how the regret function can be differentiated since we needed its gradients with respect to the model parameters. As mentioned earlier, differentiation on model parameters from a neural network is not easily defined since neural networks progressively process any input, hence, they do not have a clean aggregate differential on which we can take a clean gradient of the utility function and by extension the regret function.

Although my earlier misunderstandings mostly got cleared up leading to further investigations, my understanding of how the RegretNet Loss function can be implemented is still to be fully understood. My initial belief was that the RegretNet loss function could be easily differentiated by PyTorch's auto-gradient differentiation system since at first glance it is composed of differentiable operations. However, with significant closer study, I came to the understanding that the outputs of the loss function did not easily correspond to how backpropagation may work. First and foremost, the loss function is parametrized by both a lambda vector holding the lagrangian values and by the allocation and payment model since both are used to calculate utility. This naturally leads to further muddying as the RegretNet training algorithm computes regret gradients outside of the loss function, and differently defines what the loss function's gradient ought to look like. This needs to occur because RegretNet uses an augmented lagrangian system to optimize RegretNet which is a form of differentiation that

PyTorch's autograd system does not support natively. This naturally means that I have to implement a custom-loss function for RegretNet that overrides PyTorch's autograd system. As such, although I did not get a chance to flush out the objective function implementation, I do have a greater appreciation for how RegretNet is trained since the loss function is extremely unique that cannot be composed of more traditional machine learning loss functions.

Why Project Failed

This project failed on my end by two primary mistakes: I should have spent more time trying to find an original code base for RegretNet at the beginning of the semester, and I should have focused more on the theoretical deep learning components of how RegretNet is trained. I believe I would have been able to properly within time constraints complete this project if I had an existing code base off which I could work on my model explainability research. I dedicated my entire time reconstructing what RegretNet does, that I did not have any time to figure out why RegretNet does what it does.

With hindsight, the project also failed because I did not pay enough attention to how RegretNet optimized its pair of allocation and payment networks, or more broadly the theoretical machine learning side to RegretNet. I feel as though I should have spent more time understanding how RegretNet computed optimal misreports through its inner optimization problem, and how RegretNet computed the gradient of the loss function and how that gets differentiated.

My main attempt at this project failed, but my alternative attempt did bear some fruit. I have working implementations of the model (and how differentiation works with respect to the model itself and how gradient updates are to be carried out), the utility function (and how it can be differentiated with respect to both misreports and model parameters), the regret function (and how it can be differentiated with respect to the model parameters), and the data generators. What I lack is a well defined objective function that allows proper gradient updates to model weights, and a training loop where all of this can be done. With greater focus on my two primary mistakes above, I believe I could have been successful in reverse engineering RegretNet from scratch, and then analyzing its underlying properties.