Non-Differentiable, Parametric Model Optimization via Causal Discovery

Peter Schaldenbrand
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
pschalde@andrew.cmu.edu

Abstract

Causal discovery algorithms found in the software suite Tetrad can be used to determine which weights in a neural network need to be changed to optimize this model. The algorithm searches to see if the outputs of hidden units of the network are causally linked to increasing the error in its output. An intervention on the incoming weights to units that are causally linked to the error can decrease the loss. In this paper, I introduce the initial results of using Tetrad to optimize a simple feed-forward neural network that classifies images. The initial results are promising, but I introduce many future experiments that could show whether this optimization method is truly useful. I also introduce an algorithm for optimizing a parametric model that contains non-differentiable, intermediate components.

1 Introduction

In 1986, Geoffrey Hinton co-authored a paper introducing the back-propagation algorithm[7]. In 2017, Hinton said he was "deeply suspicious" of this algorithm and that "my view is throw it all away and start again" [5]. The back-propagation has been used, modified, and improved extensively over the last three decades, but the improvements are small. Hinton is hinting that to make a vast improvement to neural network optimization, we need to think outside the box and come up with an entirely different method of optimization. Inspired by Hinton's quote, I introduce a new method of optimization that has the potential to be completely unrelated to back-propagation.

Tetrad[1] is a suite of causal discovery algorithms developed by the Philosophy Department at Carnegie Mellon University. Search algorithms in Tetrad can take as input a data set and determine which variables are causally linked. Even if the data set is only observational data, Tetrad can estimate causality in the data. The way that causal search algorithms such as PC[8] work is by first assuming a fully connected graph, then doing systematic independence tests to eliminate edges. The output is a sparse graph representing causal links between variables.

If we consider the hidden units of a neural network to be agents, we can use causal information to intervene on the units that are causing an error in the prediction. In order to determine which hidden units are responsible for the error in the output, we could run an experiment and tweak each weight individually. But we would also have to tweak the other weights in the network, too, to ensure that it is indeed this particular weight that is causing the error. A process like this would take an extraordinary amount of computation and is one of the reasons why algorithms like back-propagation and evolutionary algorithms are so valuable to optimization.

So, running an experiment to see which hidden units in a model are causing the error is infeasible. Causal discovery algorithms such as PC or FGES [6] attempt to find causal structure in observed

[1]http://www.phil.cmu.edu/tetrad/

data. So we can pipe a batch of training examples through the network, calculate the loss, then run FGES to determine which hidden units caused this error. Then, an intervention can be made on the hidden unit by adjusting the immediate weights preceding this unit. The value of the adjustment can be determined in a number of ways. Tetrad offers an estimation feature that will estimate the strength of the causal relationship. This can be used to scale an adjustment to the weights. Other methods of determining the adjustment value include the correlation of the hidden units output to the input, another optimizer's value, or a set value (something small such as $1 \times 10^{-}5$).

2 Related Work

I have not been able to find any prior usage of Tetrad in this way, and I have spoken with the creators of it and they are also unaware of any such use cases. I have not been able to find any papers about using conditional independence to optimize neural networks so far.

3 Algorithm

The proposed optimization algorithm works on any parametric model, F, that has intermediate output values that are affected by numeric parameters and the overall model's output has a loss function. The model has intermediate units, each with parameter(s), θ_i, for the i^{th} intermediate unit. The intermediate units operate on either input data or the output of other intermediate units depending of the structure of the model and have an output of h_i.

There exists a training data set with examples, $\mathbf{x}$, and their corresponding labels, $\mathbf{y}$, such that the model tries to minimize the difference between $F(\mathbf{x}|\boldsymbol{\theta})$ and $\mathbf{y}$, defined by some loss function. The optimal model $\hat{F}$ is found by minimizing the loss with respect to θ as seen in Equation 1.

$$\hat{F} = \min_{\theta} \left[loss \left(F \left(x | \boldsymbol{\theta} \right), y \right) \right] \tag{1}$$

The error term, E is defined as the value of the loss between the model's output, $\mathbf{o} = F(x|\boldsymbol{\theta})$, and the true values, $\mathbf{y}$.

$$\mathbf{E} = \textit{loss}(\mathbf{o}, \mathbf{y}) \tag{2}$$

Tetrad can be used to determine which parameters should be intervened upon to decrease the error in prediction. In Equation 3, the $Tetrad$ algorithm takes the intermediate units' output $\mathbf{h}$, the model's output $\mathbf{o}$, and error term(s) as input and generates a vector, $\mathbf{c}$, indicating if an intermediate unit had caused the error as well as a vector, $\boldsymbol{\alpha}$ representing the extent to which each intermediate unit caused the error.

$$Tetrad(\mathbf{h}, \mathbf{o}, \mathbf{E}) = \mathbf{c}, \boldsymbol{\alpha} \tag{3}$$

$$c_i = \mathbb{1} \left[\ h_i \ \ \textit{caused} \ \ E \ \right] \tag{4}$$

Theoretically, α_i should be zero if c_i is zero, though, in practice this is not generally true since a trace amount of causality can often be estimated between any two variables.

We can now define a scheme to update the parameters of F for time $t+1$ as seen in Equation 5. The new parameter $\theta_{i,t+1}$ is an adjustment of its original value if it caused the error in the prediction. λ is a scaling factor to ensure that α and the magnitude of θ correspond.

$$\theta_{i,t+1} = \theta_{i,t} + \lambda \cdot c_i \cdot \alpha_i \cdot \mathbb{1} \left[\alpha_i > 0 \right] \tag{5}$$

There is a possibility that an α value is less than zero. This is the case in which the intermediate unit is causing a lower error. In this scenario, the parameter should not be intervened upon, hence the necessity of the last term in Equation 5, $\mathbb{1} \left[\alpha_i > 0 \right]$.

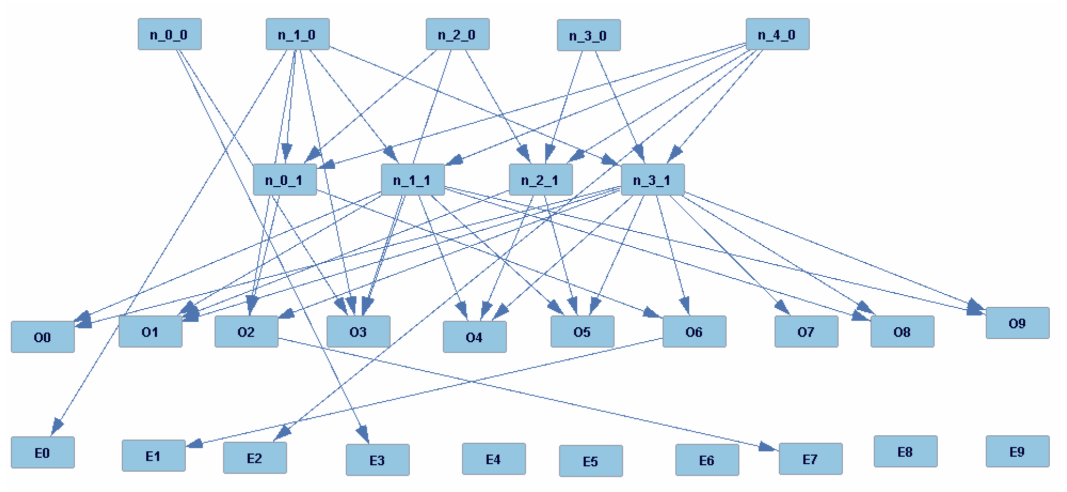

Figure 1: Graph displaying the causal links between the hidden units (n_[i]_h), outputs (O[i]), and classification errors (E[i]) of a neural network.

4 Methods

Tetrad operates on tab-delimited files where the top row is the variable names and the subsequent rows are variable values. The search algorithms in Tetrad use the values of each column to do independence tests and discover a causal model. I create a data set similar to this for use in optimizing a neural network. To create this data set in the context of this paper, a batch of samples is fed through the network. Each hidden unit gets a column with their output values as well as columns for each of the error terms.

I add constraints to the search to ensure that only plausible causal models are searched for. In Tetrad, this is called adding knowledge and is usually related to time. The constraints that I add ensure that the hidden units in the n^{th} hidden layer cannot be causally linked to the $n - 1^{th}$ hidden layer in that particular direction. In other words, the only possible causal edges are directed from earlier layers to later layers.

An example of a model showing the causal links between the units in a neural network is shown in Figure 1. There are many ways to decide which units to intervene upon. For example, you could update the weights leading into a hidden unit if:

- The hidden unit has a direct, causal link to the error.
- There exists a path of causality from the hidden unit to the error.
- The hidden unit has a causal effect on any other variable.

In addition to having multiple heuristics for deciding which hidden unit to intervene upon, there is a question of what value to change the weights to. Similar to back-propagation, in this paper I adjust the weights by a certain amount. There are four weight adjustment amounts I consider in this paper:

- Use a set value (small $\sim 1e - 5$) and use the sign of an existing optimizer or correlation value to know if it's a positive or negative adjustment.
- Use the value that an existing optimizer such as Adam or Stochastic Gradient Descent computes.
- Use a correlation value or another simple measure of the relationship between the hidden unit's output and the error terms.
- Use Tetrad's causal strength estimator feature.

In this paper, I use a batch of training examples from the Cifar10[3] or MNIST [4] data sets to create the data to feed into Tetrad. Using the causal model produced from Tetrad's search algorithms, I look for hidden units to intervene upon if a causal path exists from that hidden unit to the error in the prediction. I experiment with either using a set value for the weight adjustment in the intervention or the value that the Adam[2] optimizer would have used.

5 Technical Challenges

I am using PyTorch for the neural network implementation in this paper. Tetrad is written in Java and this has posed a great technical challenge. The context switch between Python and Java at the end of each batch is taxing on the hardware of my machine leading to an epoch duration of 5 to 15 minutes for networks with only 10-60 hidden units. This is in contrast to only 12 seconds per epoch for the standard neural network implementation.

Geneticists use Tetrad and other causal modelling software. Genetics data can be extremely wide but perhaps only a few samples. This is similar to the data in this paper since a neural network can have hundreds or thousands of hidden units but the batch size might not be very large. I will look into the tools that geneticists use in hope that their causal search algorithms are faster than the current process I'm using.

The Python implementation of Tetrad does not currently include the estimator feature. The estimator will determine the strength and sign of a causal link between variables. This is useful since nodes that cause the error to decrease do not need to be intervened upon, but currently I cannot distinguish the nodes that decrease the error from the nodes that increase it.

6 Results

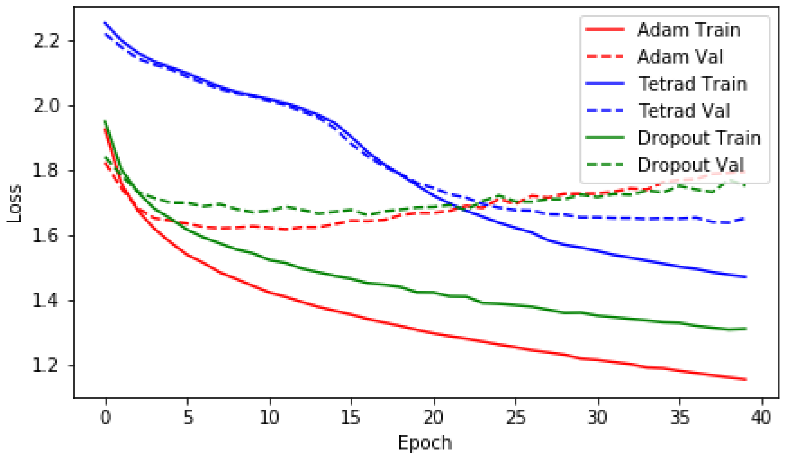

Figure 2: Training behavior for three Cifar10 image classifier networks with two hidden layers with 60 units in each layer. The dropout model had two additional dropout layers. Both the Adam and Dropout models used Adam as an optimizer, while the Tetrad model used the Adam weight adjustment on hidden units that Tetrad determined caused the error.

With the technical challenges detailed above, experimenting with the different combinations of methods was very difficult due to how long it took to train a network using Tetrad. In order to test if the Tetrad network was able to achieve as good of results as another optimizer, I created a model with two hidden layers with 60 hidden units in each. The model had 10 outputs since it was predicting the classes of the Cifar10[3] data set. The images in this data set were converted to grey scale to reduce dimensionality. One copy of this model used Adam to optimize it and another used Tetrad. The Tetrad version used the Adam values to intervene upon the network, so in order to test this against a normal network with dropout, I added a copy of this model that had dropout layers. The training behavior can be seen in Figure 2.

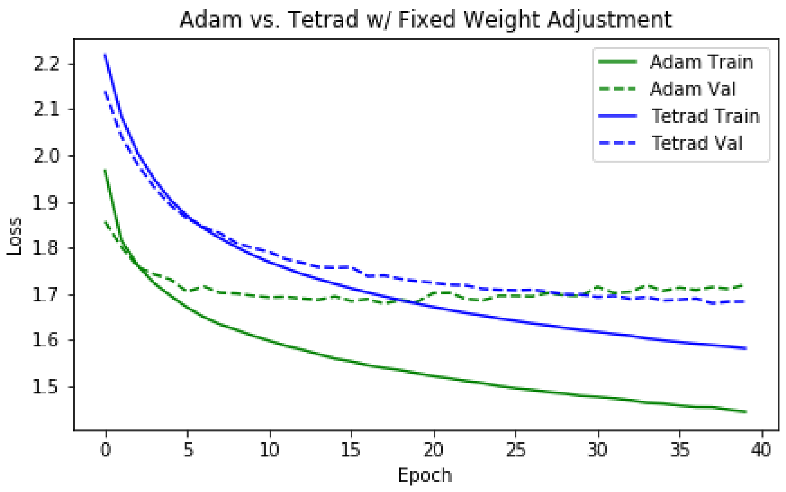

Figure 3: Training behavior of an Adam optimized network and a Tetrad optimized network where the weights are adjusted by a fixed amount. This network had two hidden layers with 30 units in each and classified Cifar10 images.

All three models in Figure 2 were able to over-fit, and each reached similar minimum loss values on the validation set: Adam = 1.617, Tetrad = 1.632, and Dropout = 1.661. This corresponds to approximately 40% accuracy on the validation set.

The Tetrad model in Figure 2 used the Adam value for intervention. To test if the Tetrad optimizer worked even without this, I created a model that used a set, small value of $1e-5$ to adjust the weights. The Adam value was necessary to determine the sign of this small adjustment value, though. The training behavior is shown in Figure 3. Both the Adam optimized model and the Tetrad model had similar minimum validation losses at 1.678 and 1.672 respectively.

The Tetrad models only update hidden units that have a path of causality to the error terms. In other methods of optimization, all the weights are adjusted at the end of every batch (though dropout adds an exception to this). During the training of one of these Tetrad neural networks, the median percentage of time a weight was updated in each epoch was 7%. The max percentage of occasions that a hidden unit was updated was 58% and some hidden units were not updated at all.

7 Discussion

The results in this paper, as seen in Figures 2 and 3, demonstrate that Tetrad is able to optimize a neural network with similar prediction accuracy as the same network with a state of the art optimizer such as Adam[2]. Currently though, the technical challenges of using Tetrad are preventing it from becoming a practical optimizer.

Upon reaching a minimum validation loss value, the Tetrad optimized network had updated its weights 1/14th the amount of times a normal optimizer updated its weights. On the other hand, the Tetrad network took 4 to 5 times as many epochs to converge. These results suggest that even state of the art optimizers like Adam perhaps don't need to update every weight upon every batch, and techniques such as dropout[9] verify this. As seen in Figure 2, the Tetrad model was able to be optimized with less weight updates than the model with Adam and dropout.

Moving forward with this work, I would like to answer the following research questions:

- Could Tetrad optimize a network faster than other algorithms? Either in terms of time or number of epochs.
- Could a Tetrad optimized network reach a more optimal solution than other optimizers?
- Would a Tetrad optimized network generalize better to held out data?

Once I find solutions for the technical challenges outlined in this paper, I will run experiments to find answers to these questions. In this paper, Tetrad was only used to optimize small networks with only dense layers, so in future work I will try far larger networks such as ResNet[1] and layer types such as convolutions and long short-term memory units.

I will also try experimenting with the benefits of this optimizer that does not require that the network is differentiable using the algorithm proposed in this paper. One particular experiment would be to generate MIDI files. MIDI is a protocol of music that is a list of notes/pitches, instruments, and other aspects of music. We can easily convert MIDI to WAV audio but this process is non-differentiable. I would like to create a GAN that generates MIDI files in the style of true WAV music files. The discriminator would discriminate between the WAV from the generated MIDI files and true WAV files. The error could be used to optimize the network using Tetrad since back-propagation would not work since a step in the model is non-differentiable.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[3] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[4] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[5] Steve LeVine. Artificial intelligence pioneer says we need to start over. [Online; posted 15-September-2017].

[6] Joseph Ramsey, Madelyn Glymour, Ruben Sanchez-Romero, and Clark Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International Journal of Data Science and Analytics*, 3(2):121–129, Mar 2017.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[8] Peter Spirtes, Clark Glymour, and Richard Scheines. Causation, prediction, and search. 2000.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.