Extracting Compton Form Factors from Deeply Virtual Compton Scattering Data with Artificial Neural Networks

The goal of our research was to extract the Compton form factors (CFFs) ReH, ReE, and ReH~ from datasets containing variables from Deeply Virtual Compton Scattering experiments. These variables include F, phi, t, k, x_b, QQ, F1, F2, a DVCS constant, and the actual values for the three Compton form factors. Additionally, the mathematical relationship of these variables is detailed by the BHDVCS formula, which takes phi, t, k, x_b, QQ, F1, F2, ReH, ReE, ReH~, and the DVCS constant as inputs and outputs F, the cross-section value. The neural network aims to accurately produce values for ReH, ReE, and ReH~ by using values of QQ, t, k and x_b and fitting against true values of F. The variable F is spread with an error of 5%, and the goal of the extraction method is to produce Compton form factors with similar error.

Review

The use of neural networks to extract parameters from deeply virtual Compton scattering has been studied before, specifically by Krešimir Kumerički, Dieter Müller, and Andreas Schäferin in 2011. In this paper, neural networks are used to extract the value of the Compton form factor H, focusing on both the real and imaginary parts. The network takes x_b and t as inputs and produces ReH, the real part of H, and ImH, the imaginary part of H. The network uses sigmoid activation functions for inner layers and is trained on 2000 epochs using a modified back-propagation technique called *resilient back-propagation*. This study determines that the use of neural networks is comparable to other statistical methods, and provides a clean and simple extraction method for ReH and ImH. It also states that other Compton form factors may be harder to extract with neural networks, as some CFFs may be less dependent on kinematic parameters.

Methodology

In order to measure a baseline for extracting CFFs, we started by using the SciPy optimize function to fit F using least squares regression. This method produced an average error of about 8% in each of the form factors. We then designed the architecture of the network to take QQ, t, k and x_b as inputs, feed these values through multiple inner layers, and return three values for the form factors. I chose to use PyTorch since other team members were already using TensorFlow, and by using different libraries we could potentially create multiple approaches. Before adjusting hyperparameters, the PyTorch network yielded an average error of around 20%. To achieve more accurate results, I rewrote the standard means squared error loss function to incorporate the provided error in F. This error was used to divide the squared error before taking the mean so that the F values with different errors would be weighted differently in the

measurement of loss. Additionally, I ran optimization tests to tune hyperparameters for the network and found that 1 input layer to two inner layers with 100 nodes to an outer layer was the optimal architecture. A learning rate of 0.02 and 2500 epochs led to the lowest average error across all sets. Different architectures yielded varying degrees of accuracy for different data sets, yet no clear correlation was found between kinematic parameters and optimal architectures. For nodes in the inner layers, the network uses a Tanh activation function to determine weights and biases.

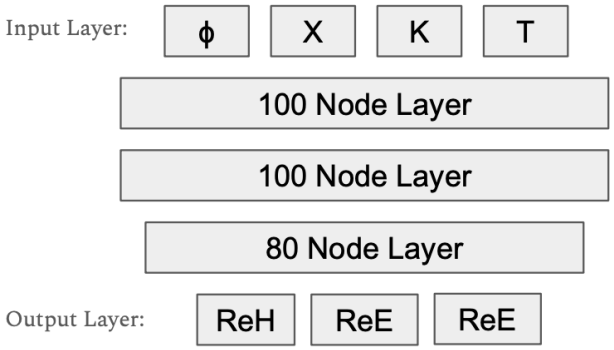

FIGURE 1. Optimal architecture for artificial neural networks to produce local fit
of Compton form factors. Derived by finding lowest average percent error in fits across all provided data sets.

Once we had a functional network, we moved to Monte Carlo replica testing to measure the variance in results produced by networks with randomized input. We created 1000 replicated datasets by randomizing F on a gaussian distribution using the values of F and errF. The mean of the extracted Compton form factors from these 1000 replicated sets would then be used to compare to the target values. The PyTorch network again produced errors of less than 10% on average, with low variance in fits. All results are listed below. All prior networks were trained on data from single sets, and our latest task was to design a network that would produce a global fit. To do so the network would be fed the four kinematic parameters from all 15 sets of the pseudo-data and produce a set of respective Compton form factors. The global fit would be run over many replicated data sets, and the mean of the produced CFFs would be compared against the target variables. Due to errors in the integration of PyTorch functions into the custom loss function, a successful global fitting PyTorch network has not been achieved.

Results

Using the SciPy optimize() function and least squares regression, the average percent errors for ReH, ReE, ReH~ in each of the 15 datasets were 8.5%, 8.7%, and 7.8% respectively. With a simple mean squared error loss function, not using errF, the average errors in ReH, ReE, ReH~ were 23.4% 16.7%, and 17.7%, respectively. When adding errF into the loss function, these errors were reduced to 7.8%, 8.2%, and 6.33%. Appendix A shows results from 100 replicas for sets 0–8. An example distribution of ReH values is given below in figure 1.

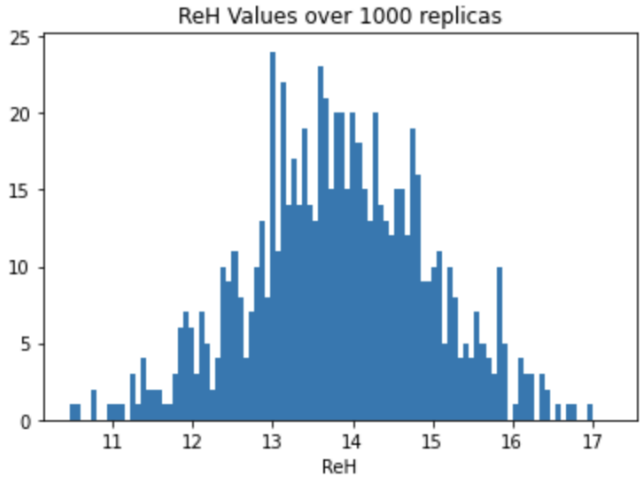

FIGURE 2. Frequency of ReH values over 1000 replicas of a single set.

Conclusion

The fit results from the replica method yielded positive results for the extraction of the Compton form factors with average percent errors in fit lower than those of the baseline fit. The current architecture of 4 layers with 100 nodes, a learning rate of 0.2, and 2500 produces the lowest average error in fits for local fits. With an average error of 7.4% across the three Compton form factors, we are close to the 5% spread of the simulated data, meaning our approach is quite effective. Due to exploding gradients that allow a network to escape local minima and maximize the probability of finding the global minimum in loss, I am confident that these network results are optimal for the given architecture. It is still unclear which properties of certain sets allow for a more accurate extraction of ReH, ReE, and ReH~ , and a preprocessing method to determine optimal architectures may improve results. With a low sigma in Monte Carlo replica testing, these results are proven to be reproducible and precise. In both the baseline least squares regression fit and the final artificial neural network fit, ReH~ seems to be the form factor that is most accurately extracted, with ReE as the least accurate.

Appendix A

Set 0
True ReH: 13.06
Mean: 13.73 (5.18% Error)
Sigma: 1.101786

True ReE: -53.06
Mean: -56.64 (6.75% Error)
Sigma: 5.643754

True ReHT: 7.25
Mean: 6.80 (6.29% Error)
Sigma: 0.636681

Set 1
True ReH: 12.55
Mean: 11.69 (6.93% Error)
Sigma: 2.682727

True ReE: -52.55
Mean: -47.55 (9.52% Error)
Sigma: 14.102813

True ReHT: 6.97
Mean: 7.55 (8.22% Error)
Sigma: 1.296748

Set 2
True ReH: 7.22
Mean: 6.54 (9.46% Error)
Sigma: 2.630998

True ReE: -47.22
Mean: -43.22 (8.48% Error)
Sigma: 15.091114

True ReHT: 4.01
Mean: 4.52 (12.66% Error)
Sigma: 0.957782

Set 3
True ReH: 7.65
Mean : 7.43 (2.86% Error)
Sigma: 2.579284

True ReE: -47.65
Mean: -46.67 (2.06% Error)
Sigma : 13.014765

True ReHT: 4.25
Mean: 4.12 (3.01% Error)
Sigma: 1.042199

Set4
True ReH: 12.55
Mean: 11.59 (7.65% Error)
Sigma: 1.231093

True ReE: -52.55
Mean: -47.44 (9.73% Error)
Sigma: 6.315369

True ReHT: 6.97
Mean: 7.39 (6.00% Error)
Sigma: 0.719295

Set5
True ReH: 7.22
Mean: 6.69 (7.36% Error)
Sigma: 1.834488

True ReE: -47.22
Mean: -44.13 (6.56% Error)
Sigma: 10.422852

True ReHT: 4.01
Mean: 4.23 (5.37% Error)
Sigma: 0.633418

Set6
True ReH: 11.74
Mean: 10.19 (13.21% Error)
Sigma: 3.240054

True ReE: -51.74
Mean: -42.97 (16.95% Error)
Sigma: 18.437086

True ReHT: 6.52
Mean: 6.94 (6.36% Error)
Sigma: 1.594173

Set7
True ReH: 7.65
Mean: 8.13 (6.27% Error)
Sigma: 2.261585

True ReE: -47.65
Mean: -50.18 (5.31% Error)
Sigma: 11.376090

True ReHT: 4.25
Mean: 4.47 (5.21% Error)
Sigma: 1.527616

Set8
True ReH: 12.55
Mean: 9.88 (21.33% Error)
Sigma: 0.625082

True ReE: -52.55
Mean: -38.62 (26.51% Error)
Sigma: 3.199023

True ReHT: 6.97
Mean: 6.81 (2.40% Error)
Sigma: 0.823573