Mutation-based Modular Decomposition of Deep Neural Networks

Tra Reynolds
Department of Computer Science and Software Engineering
Auburn University
Auburn, United States of America
trr0026@auburn.edu

Ali Ghanbari
Department of Computer Science and Software Engineering
Auburn University
Auburn, United States of America
ghanbari@auburn.edu

***Abstract*—Prior to deep neural networks (DNNs), the decomposition of software has been an established method of replacement of insufficient programming and reuse of efficient code. Modularization of DNNs, however, is a less prominent area of study due to its increased complexity. Changes to DNNs, therefore, are difficult to implement. After alterations, DNNs are required to go through extended periods of retraining to produce a functional program once again. This paper seeks techniques of DNN modularization and the production of methods to modify DNNs without the need for arduous retraining.**

***Index Terms*—deep learning, deep neural networks, machine learning, mutation analysis, modularization**

I. Introduction

As machine learning becomes increasingly more important in society, so does the need for ways to further develop the medium. After all, machine learning is expensive, both monetarily and computationally. Moreover, the developmental process is long and arduous, with the mere derivation of an innovative idea being only the start. The most rigorous element in machine learning algorithms, often built using deep neural networks (DNNs), is its physical formation. As a DNN begins taking shape, it must go through an extended period of testing and retesting. In order to create an adequate model, this training must cover large datasets, taking up resources perhaps only to underperform in some areas and need to be entirely retrained.

This is where modular decomposition becomes a necessity. The ability to reuse and replace existing, trained DNN modules proves invaluable in creating new DNN models. By separating a DNN into segmented modules capable of a specialized task, one could combine several of these DNN modules to develop a DNN with the ability to perform multiple tasks. Most importantly, this new DNN model would not have to go through retraining to function effectively. In pursuit of this objective, some methods have been proposed, mainly those that attempt to compartmentalize DNN modules after training. Considering these methods, we developed a system of DNN decomposition using mutation analysis inside Python.

Mutation analysis is a white-box testing method often used in testing for errors, only recently being used to train artificial intelligence models. The procedure involves the randomized altering of elements within the object of testing. The resulting altered models, or mutants, are analyzed and graded on their performance. Those that demonstrate acceptable proficiency are allowed to mutate further, while those that do not are discarded. When it comes to DNN decomposition, the methodology is similar. In this instance, the neurons within the layers of the DNN are being mutated, allowing for a detailed analysis of the impact these individual neurons have on the output and producing a metric that can be used to decide how to sever them into DNN modules [4].

Before beginning modularization, it is important to define the metrics that are indicators of a successful DNN decomposition. One of those metrics is accuracy. No program is perfect, and as such, no program will always be accurate. When decomposing a DNN model, it is imperative to minimize loss of accuracy. Two other methods are cohesion and coupling, which are related. Cohesion is the overlap of weights within modules, and coupling is the overlap of weights between modules. The goal during modularization, thus, is to maximize cohesion and minimize coupling. Therefore, a highly cohesive, hardly coupled DNN module that retains the accuracy of the original monolithic DNN model is the objective when decomposing into DNN modules [3].

II. Related Work

While researching DNN decomposition, it became apparent that methods of modularization after training followed a similar sequence [1], [2]. This sequence is as followed:

A. Concern Identification (CI)

For the first step, concern identification (CI) involves discerning which neurons contribute to a given output. Large DNN models, particularly those that can produce many possible outputs, will have neurons that contribute to multiple outputs. Neurons that behave in this manner will be assigned to the output they most contribute to and later pruned from the outputs it does not.

It is worth mentioning that the removal of too many neurons can cause a neural network to be too sparsely connected, possibly resulting in decreased accuracy and/or processing speed [1]. Therefore, it is important to set a limit on the amount of neurons that can be removed from the developing DNN module.

B. Tangling Identification (TI)

Though the proceeding DNN module now only contains neurons that have an impact on a given output, a problem arises from its overspecialization. As the DNN module becomes more specialized, it also becomes more predisposed towards a given output. This output is favored regardless of the input [1]. To avoid this outcome, it is important to add back in some of the less relevant neurons. The neurons added back into the module need to be a mix of those that still contained some level of contribution to the desired output and those that did not. This procedure allows for the DNN module to have some method of determining an output that is counter to the output the majority of the neurons most contribute to identifying [1].

C. Concern Modularization (CM)

This final step is where modularization occurs. To do so, the neurons that do not contribute to the positive result within the last hidden DNN layer are removed. The noncontributing neurons in the output layer are then abstracted. This technique is known as channeling and concludes the modularization of a DNN model [1].

D. Special Case: CNN Models

The concern modularization step, however, is not the same for all DNN models. There are DNN models that require a different sequence of steps due to their increased complexity, most notably convolutional neural networks (CNNs). CNN models are a type of deep neural network that is most associated with image classification. For example, a DNN model that identifies the presence of something within an image would most likely be a trained CNN model. To decompose a CNN model, to start, a channeling technique is used, like previously described. Then, the process backtracks through the contributing and noncontributing neurons in the previous layers, removing the neurons that only participate in identifying the noncontributing output. Backtracking through the convolution layer, however, is not feasible because in a convolution layer, the inputs cannot be directly mapped with the outputs. To remove irrelevant neurons from the convolution layer, we store the position of the neurons in each sliding window (which is a segment of an image that is currently being analyzed) with the neurons in the output during the forward pass, and remove the neurons during the backward pass [2]. The product is a decomposed CNN model.

E. Special Case: Modularizing while Training

While researching methods of the modularization of DNN models, a unique technique was found. Rather than attempting to decompose a trained DNN model, this method shows that a DNN model can be trained to easily decompose later. This process is done by training the DNN model to have high cohesion and low coupling while training it to have a certain functionality [3].

Just like in modularization after training, modularization while training starts with finding the contributing neurons to the desired output. This process is similar to the way previously described. The next step is the evaluation of cohesion and coupling, as these are the metrics used to define how easily the module will decompose [3]. This is done using an application of the Jaccard Index, which is as followed:

$$Jaccard(U,V) = \frac{|U \cap V|}{|U \cup V|} \tag{1}$$

Coupling is calculated by averaging of the Jaccard Indexes of every neuron in the module. A value of 1 would indicate the neurons are the same, with a 0 indicating complete uniqueness. Cohesion is calculated by averaging the Jaccard Indexes between modules. Gradient descent is a method of optimizing both of these metrics as the module trains for its accuracy as well. This method calls for gradual directional steps towards a desired output. Should the modularization changes move in an undesired direction, the DNN model will revert to the previous state [3]. With this step finished, modularization can now occur, similarly to the methods explained earlier.

III. Methodology

With preliminaries addressed, the project that is this paper's focus is the use of mutation analysis to assist in the decomposition of DNN models. The name of this application is INCITE, named after its method of inciting neurons using mutations [5].

A. Density-based Spatial Clustering of Applications with Noise (DBSCAN)

To execute this process, it is a necessity to implement the concept of density-based spatial clustering of applications with noise (DBSCAN). DBSCAN is a method of data clustering by grouping data points together by their metric proximity and relative majority, excluding outliers or small groupings of outliers.

The desire to apply DBSCAN comes from the aspiration to quicken the processing time of the technique. Before starting, INCITE used the top-performing neurons as the basis for the mutants. Instead, using the top-performing cluster of neurons, and ignoring outliers, should result in an accelerated increase in performance. This approach focuses on improving the greater performance of the larger DNN model, rather than the outliers. Implementing DBSCAN into Python involves the importing of scikit-learn, which is a Python module already used within the program. A problem arose, however, when implementing DBSCAN. The outlying neurons' performance vastly trumped the performance of the other neurons. The result is the clustering of all remaining neurons except for the outliers, which is not ideal. The settings required to guarantee the removal of outliers resulted in fewer clusters than desired. It became apparent that data normalization would become a necessity.

B. Data Normalization

Data normalization involves the redefining of metrics in order to better organize results. This process is crucial when attempting to analyze data. Because of the existence of large outliers, the data must be normalized so that it can be better analyzed. The technique used in this paper is the normalize function, which like DBSCAN, is also included in scikit-learn. Since the performance of neurons is already organized in an array, this method of normalization is preferred. Afterwards, the array of performance scores is now easier to analyze, and DBSCAN now clusters that data more appropriately.

IV. Results

The result of the decomposition of DNN models using mutation analysis shows that the loss of accuracy is minimal. This outcome is true for simple DNN models, CNN models, and regressions models. Additionally, this modularization method achieves the high coupling, low cohesion principles previously discussed across the three model varieties as well [5].

Prior to data normalization, the contribution scores of a neuron layer were 3410, 1238, 809, 303, and 248. Attempting to apply a reasonable DBSCAN function to these scores would always result in the inclusion of the final four contribution scores in one cluster. Considering the distance from neuron 2 to neuron 5, this is not ideal. After data normalization, the contribution scores are 0.672, 0.531, 0.476, 0.181, and 0.087. Applying a DBSCAN function to these normalized contribution scores within a distance of 0.12 results in neuron 1 being an outlier, neurons 2 and 3 in one cluster, and neurons 4 and 5 being in a different cluster. This outcome is a far more reasonable clustering of the data.

V. Challenges

Throughout the course of this project, many challenges have appeared. Research into the topic of DNN decomposition is newly emerging and intellectually challenging. Materials regarding the subject are dense and far outside the scope of prior knowledge and personal educational findings. Required programming skills also became an obvious area of insufficiency. The modification of INCITE to select the top-performing cluster of neurons was not achieved. The modules used within Python, though a familiar programming language, are largely untested by an inexperienced author, resulting in a halting of crucial progress. Supplementation of experience and knowledge would result in a more complete project.

VI. Conclusion

This paper attempts to show the myriad of ways the modularization of DNN models is possible. It also seeks to identify the interworkings of these systems. The specific method by which this paper achieved this objective is via mutation analysis, which is also a concept in its infancy, as is modularization while training and DNN decomposition as a whole. Previously a concept reserved for software, DNNs will have to find efficient manners of modularization should their presence continue its current expansion velocity. The computational and resource cost will be too great should the unfortunate yet inevitable bugs and inaccuracies occur. Applications of DNNs, such as self-driving cars, medical analysis, and criminal investigation techniques cannot afford the delays in patches should full retraining of models be required. Even for DNN purposes with less pressing aims and fewer fatal consequences, economic concerns are still valid, and the desire to reduce such costs is no less practical. To this end, DNN decomposition will need to improve; though, having observed the rapid advancements in the field, it is with unwavering certainty that those researching and developing DNN modularization will produce stellar achievements.

Acknowledgements

This work has been supported by the Distributed Research Experiences for Undergraduates (DREU) program by CRA. I would like to thank Dr. Ghanbari for his guidance and mentorship. All errors are mine.

References

[1] Rangeet Pan and Hridesh Rajan. 2020. On Decomposing a Deep Neural Network into Modules. In Proceedings of The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

[2] Rangeet Pan and Hridesh Rajan. 2022. Decomposing convolutional neural networks into reusable and replaceable modules. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). Association for Computing Machinery, New York, NY, USA, 524–535. https://doi.org/10.1145/3510003.3510051

[3] B. Qi, H. Sun, H. Zhang, R. Zhao and X. Gao, "Modularizing While Training: A New Paradigm for Modularizing DNN Models," in 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 2024 pp. 353-364. doi: 10.1145/3597503.3608135

[4] L. Ma, et al., "DeepMutation: Mutation Testing of Deep Learning Systems," in 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), Memphis, TN, USA, 2018 pp. 100-111. doi: 10.1109/ISSRE.2018.00021

[5] A. Ghanbari, "Decomposition of Deep Neural Networks into Modules via Mutation Analysis," Symposium on Software Testing and Analysis (ISSTA 2024). ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3650212.3680390