The Need for Adoption of Neural HPC (NeuHPC) in Space Sciences

Homa Karimabadi, Jason Wilkes, Aaron Roberts

Supplementary Material

1. AI and Its Impact on Scientific Research Process

The AI revolution has not only proven to be transformative technology, it has also changed the scientific research process, which in turn has contributed to its own rapid advancement. The main drivers for this rapid advancement are: i) AI is a unifying technology applicable to a wide variety of data types and applications. Similar to transfer learning where a model trained on natural images can be fine-tuned on medical images for cancer detection, there is also transfer knowledge where for example an advance in NLP modeling such as transformers in neural nets is soon propagated across other application domains such as computer vision and timeseries prediction. This creates a fast *cross-pollination* of the best ideas across different domains. ii) The availability of end-to-end AI platforms such as tensorflow, that are continually updated and professionally maintained, has been a critical enabling technology for rapid prototyping, developing, and sharing of models. iii) Unlike previous techniques, the accuracy of deep learning models does not saturate but continues to increase with more data. This has led to exponential improvements in cases where there is large available data. For example, the larger language model GPT-3 was developed less than a year after its predecessor GPT-2. It has two orders of magnitude more parameters (175 billion parameters) than GPT-2 (1.5 billion parameters) and it was trained on over 500 billion words. Assuming a person were to speak at an average rate of 150 words per minute, 500 billion words would represent over 55 million hours or more than 6300 years of continuous speech. This led to a significant increase in its efficacy, to the point where GPT-3 can write original content with equivalent fluency to that of a human (New York Times). ChatGPT, a human-like conversational bot, which is the based on GPT-3, is considered a tipping point in AI and has garnered over a million users within one week of release, the fastest technology adoption in history, by far. Similarly, AlphaGo went from beating the top player in Go, Lee Sedol, on March 2016 to AlphaZero in late 2017, which taught itself from scratch how to master the games of chess, shogi, and Go, beating a world-champion computer program in each case. iv) To keep pace, the scientific research process has improved with a more efficient feedback loop. It is now standard protocol to share all components of research including data as well as code and models developed on common AI platforms like tensorflow. This enables others to readily verify, reproduce, deploy, and build upon others' research. Further, posting of papers on arXiv allows for rapid communication and dissemination to the community at large. In addition, establishment of benchmarks and reporting results against those has provided an unbiased assessment of a given work, reducing "clutter" and making the important contributions stand out more clearly.

2. Automated Detection of Current Sheets and Measurements of their Lengths

Here, we describe the algorithm that we developed for quantifying the current sheets as observed in the 2D full PIC simulations of turbulence by Karimabadi et al. [2013]. Identification of current sheets can be thought of as a segmentation problem and one approach would be to create a large, labeled set

to train a segmentation network. Note that the size of each time slice (8192 x 16384) would require large memory for segmentation networks. There are several options: a) use of several TPUs or shared memory machines, b) down sampling of images, c) using techniques such as sliding window.

Here, we used an unsupervised approach which bypasses the labor-intensive creation of a labeled set, and it also has low memory requirements. Given our promising results, one could use the current sheets identified by our unsupervised approach to create labels which the domain experts can then correct as needed. Once a segmentation model is trained, one can then run it on a new set, identify and correct the errors and retrain. This semi-supervised approach can make the labeling process much more efficient.

Our unsupervised algorithm proceeds in several stages, none of which is particularly complex alone, but they can accomplish something non-trivial when used in tandem. Starting with the raw image data, we preprocess it as follows: we square the data, apply Gaussian blur, rescale the pixel values to integers between 0 and 255, and finally discard all pixels except the top 3% by brightness. This step is the most ad hoc of the algorithm's steps and could likely be replaced with many other methods. In practice this step was developed empirically to make the task easier for the "object detection" algorithms later in the pipeline. Squaring the image pixel by pixel before applying the Gaussian blur ensures that the noise, we add will smooth out the boundaries of the current sheets without affecting their interior. That is, regions whose brightness is on the boundary between surviving and not surviving the later thresholding step will be smoothed, while the "backbone" of each current sheet will already be sufficiently bright from the squaring step that it would not be affected by the smoothing.

The next step involves application of edge detection. In this step we use the Canny edge detection algorithm to transform the bright regions in each backbone to a thin sheet of pixels outlining the perimeter of those regions. The purpose of this algorithm will become clear below since it is being used as a sort of preprocessing step for the clustering algorithm that follows. In the clustering stage, we use DBSCAN algorithm that works by grouping points within a certain spatial radius, optionally with a lower bound on the minimum number of points required to be a cluster (rather than being assigned to the "noise" cluster, where all outliers are sent). The edge detection step above was performed with DBSCAN in mind. DBSCAN can only threshold based on the number of points in a point cloud. It has no concept of the "length" of the point clouds it builds up by connecting these points, except to the extent that adjacent points are no more than epsilon distance apart.

Consider now the problem of detecting a long thin current sheet, while ignoring other structures like circular blobs whose radius may be larger than the minimum thickness of our thin current sheet. To illustrate, imagine we have a circular blob with a 20-pixel radius, which we would like to ignore, and a current sheet 100 pixels long and 5 pixels wide, which we would like to detect. The circular blob will have approximately $\pi * r^2 = 1256$ white pixels in its interior, while the current sheet has only 500 pixels. No simple threshold will allow us to count the smaller point cloud as a "cluster," while claiming the larger one is "noise."

This was the motivation for applying edge detection. The result of edge detection is to turn each of these point clouds from an "area" to a "perimeter." To detect an object's edges is to discard its interior.

Now, after transforming these objects from areas to perimeters, our circular blob has $2\pi r = 125$ pixels inside it, while our current sheet has 210. What is important here is not the specific magnitudes, but the fact that the sort order has now changed. Before edge detection, the circular blob (considered as a point cloud) had more points than the current sheet. After edge detection, the reverse is true. To put it differently, ideally what we would like to do here is something like DBSCAN with a "minimum length" argument, that kills off "short" clusters and preserves "long" ones. That is, we would like to say, "current sheets whose length is less than L will be called noise, while those longer than L will be signal."

At this stage of our algorithm, we do not yet have any geometric notions like "length" available to us. All we have is sets of 0-dimensional points in space. Since we are forced to work with point clouds, edge detection offers a good approximation of the geometric threshold we desire. Transforming areas to edges transforms our point clouds so that the attribute we care about (their length) will be easier to detect using standard DBSCAN with a simple threshold.

At this point we have effectively performed object detection (a set of current sheets, each represented as a point cloud) and our objects are now spatially separated from one another by a distance no less than our DBSCAN threshold. We can now use off-the-shelf geometry libraries to turn our point clouds into polygons, and begin to measure arc lengths, convex hulls, or other geometric properties.

There is a subtlety regarding the measurement of length/perimeter. Imagine we are interested in measuring the length of a coastline which has fractal structure or even a simpler case of a very jagged contour around a circle. If we measure the length of the contour following each of the fine scale jagged points, the length will be much larger than if we take the radius of the circle and calculate the length of the perimeter from the radius $2\pi r$ (which is analogous to what Minkowski functionals do). In case of turbulence, both types of measurements would be interesting. If one could do simulations at realistic mass ratio and high ratio of electron plasma frequency to electron cyclotron frequency, there would be very short scale fluctuations and it is conceivable that the current sheets would have fractal structure. If so, measurements of “length” accounting for the jaggedness of the current sheets could lead to large scales, and possibly even MHD scales. However, the physical implications of such “length” in terms of macroscopic properties of turbulence such as dissipation is less clear. Here, we focus on measurement of length while ignoring any jaggedness. A useful technique for this purpose is the convex hull which we used.

The complete code including the code to generate the videos can be found at https://github.com/homakar9/NeuHPC/tree/main/turbulence.

The two parameters in the algorithm are the threshold and the radius in pixels for DBSCAN. Videos using different thresholds can be found at https://github.com/homakar9/NeuHPC/tree/main/turbulence/movies.

We have also included movie of the plasma mixing (https://github.com/homakar9/NeuHPC/tree/main/turbulence/mixing) which is less stochastic than the evolution of current sheets. It would be interesting to derive equations for the temporal evolution of mixing which is relevant to the problem of transport at the magnetopause.

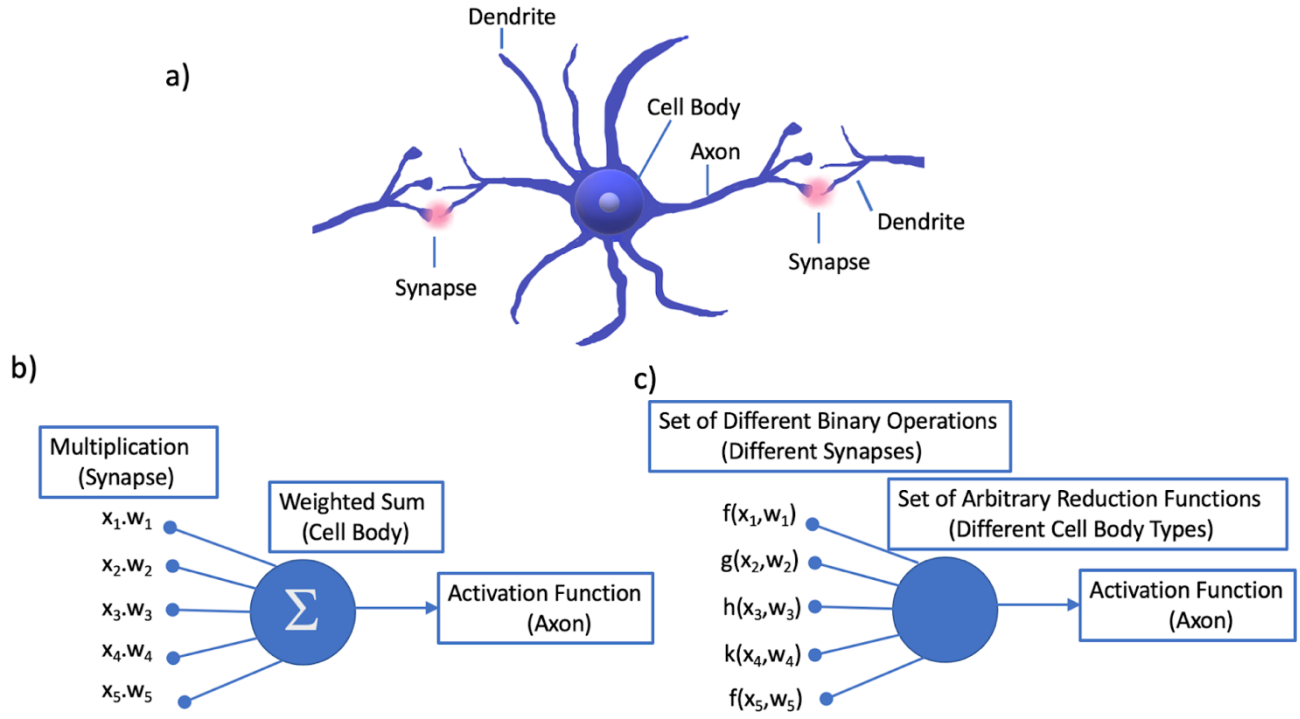
3. A New Type of Neural Net for Derivation of $1/r^2$

The standard artificial neural nets were inspired by their biological counterparts, but the focus has been on improved results in practical applications rather than mimicking the brain's design closely. While many ANN architectures have been considered (e.g., recurrent, LSTM, CNN, FFN), most share the same design of artificial neurons (Figure 1). In contrast, the human nervous system contains a wide variety of neurons (e.g., Granule, Purkinje, Pyramidal, Renshaw, Spindle, among others) consisting of different types of synapses and cell body types. While some of these cell types have rough parallels in modern machine learning models (e.g., Spindle cells may be likened to "skip connections"), most of these computational units have no obvious parallel in the models we build in machine learning.

It is reasonable to assume that one reason for the presence of such a wide variety of neurons in the brain is to maximize efficiency of computations for the wide variety of objectives that it is tasked to handle. Adjusting weights is not a cost-free operation for the brain. Changing synaptic conductances long term requires gene expression and protein synthesis in the postsynaptic cell. Having multiple routes to computing the same function allows the brain to take a path of least resistance, thus disturbing as few previously learned associations as possible. In this sense, we can think of the brain striking a balance between accuracy and efficiency and it accomplishes that through the existence of a variety of neurons.

Unlike most applications of ANN where accuracy is the main objective, the problem of symbolic regression introduces a second objective of parsimony which can be thought of as a proxy for computational efficiency. As such, and carrying the analogy to the human nervous system, we have devised an ANN where the network has access to different types of neurons consisting of different synapses and cell body types (Fig. 1).

The code and its application to the $1/r^2$ problem can be found here https://github.com/homakar9/NeuHPC/tree/main/newton.
It would be interesting to explore the mathematical properties of such a network (e.g., stability, convergence, training efficiency). Our goal here was to highlight the fact that there is significant flexibility in the design of neural networks, both in terms of architecture (e.g., CNN, LSTM) but also in its building blocks (perceptron).



Supplementary Figure 1. a) A neuron consists of dendrites, cell body, axon and synapse. Dendrites are responsible for receiving input from other neurons and axon is responsible for transmission from one to the other. At synapse, the connecting structure between two neurons, electrical signals are modulated in various amounts. The cell body processes the electrical signals, and the axon has the information stored in the form of an action potential. An actual neuron fires an output signal only when the total strength of the input signals exceeds a certain threshold. b) A perceptron can be thought of as having a synapse, a cell body and axon. Standard ANNs are based on a single type of synapse (the inputs and weights interact with multiplication) and cell body type (weighted inputs are aggregated with "reduce sum"). The activation function plays the role of an axon and can be chosen from a variety of functions such as ReLU, *sigmoid*, *tanh*, among others. c) In our proposed ANN, there are different types of neurons, each with different types of synapses (input and weights interact with arbitrary binary operations) and cell body types (weighted inputs are aggregated with arbitrary reduction functions).

4. Auto-differentiation

Given the importance of auto-differentiation in HPC, here we discuss it in more details. Auto-differentiation provides an accurate and efficient calculation of derivatives which is relied on for training neural networks.

There is no formal requirement for a neural net or its layers to be differentiable. Examples of non-differentiable neural nets include derivative-free neural nets (e.g., Aly et al., 2019). As Aly et al. state, the non-differentiability can arise "where the neural network or task is non-differentiable, does not have a representative loss function or the derivatives are uninformative in guiding the optimizer". It is also easy to see this point if one insists on using argmax which is not differentiable. There are several different types of derivative-free optimization algorithms that can be used with neural networks, including evolutionary algorithms, Monte Carlo, and particle swarm optimization. These algorithms

work by iteratively searching for the optimal weights for the model through a process of trial and error, without the use of derivatives.

There is, however, a great advantage in being able to do the optimization/training using backpropagation and the chain rule. Therefore, all the components of standard neural nets such as max pooling, activation functions, loss function, etc. are *chosen* to be differentiable almost everywhere. The term differentiable "almost everywhere" refers to the function being differentiable everywhere except on a set of measure zero. An example is the activation function ReLU which is continuous but differentiable in all its domain except at the origin x=0. Despite not being differentiable (everywhere), the fact that it is differentiable "almost everywhere" enables calculation of gradients using subderivatives in backpropagation algorithm. See for example Lee et al. (2020) where they discuss the correctness of auto-differentiation for non-differentiable functions.

Note that it is possible to have a non-learnable layer that is not differentiable, but the learnable layers need to be differentiable almost everywhere to be able to take advantage of the backpropagation and chain rule. And this is the key. The important point is not that the neural network is differentiable but that each step is and that there is a machinery in place to compute it efficiently. In other words, the standard neural nets are put together to be trainable using backpropagation and the chain rule, but this feature is not an inherent property of neural nets in general.

From a practical standpoint, the real value is in the "*auto"* part of auto-differentiation. The chain rule (and "the pieces being differentiable") were always used, before auto-differentiation, but by "hand". Chain rule allowed researchers to train networks with backpropagation, but it placed the burden of correct implementation of tensor calculus on the software developer, a painstaking and error prone requirement that prevented many technically competent developers from being practically able to participate in deep learning. The "auto" in auto-differentiation was made possible by the advent of frameworks for symbolic computation using a computational graph. *The knowledge of a computational graph is what permits symbolic, rather than numerical, differentiation*. In symbolic differentiation, the graph would know that the derivative of *sine* is a *cosine* and uses it to evaluate the derivative. To highlight the symbolic nature of auto-differentiation, we have constructed a version of auto-differentiation along with some examples in https://github.com/homakar9/NeuHPC/tree/main/newton. While there is a subtle difference between auto-differentiation and symbolic differentiation, it is not of practical relevance, and we will not discuss it here.

In auto-differentiation, derivatives are calculated using a computational graph where each node in the graph represents an elementary operation (such as addition or multiplication) and each edge represents the flow of data (such as input values or intermediate results) between the nodes. To calculate a derivative using auto-differentiation, you start by constructing a computational graph that represents the function you want to differentiate. Then, you use the chain rule to calculate the derivative of the function by traversing the graph in a reverse direction (from the output node to the input node) and accumulating the products of the derivatives of the intermediate operations with the incoming edges. For this symbolic differentiation to be possible, one must use the functions native within a framework (e.g., TensorFlow) so their symbolic derivatives would also be known. For example, we cannot differentiate through *numpy.log* as TensorFlow would need to use its own version of *log*.

In summary, while there is no requirement for neural nets to be differentiable (e.g., Aly et al., 2019), standard neural nets are constructed so that they can be trained using backpropagation and chain rule.

This requires all learnable layers and components of the neural net (e.g., loss function, max pooling, activation functions, etc.) to be once differentiable almost everywhere. Frameworks like TensorFlow and PyTorch automate this operation, enabling the "auto" in auto-differentiation, making efficient computation of derivatives readily accessible.

5. Deep Learning

The common perception that a deep neural network is simply one that contains more than three hidden layers has led to confusion among non-experts. This definition, while not entirely inaccurate, fails to encompass the full scope of deep learning and its advancements. For example, to imply that a neural network with four hidden layers constructed twenty years ago constitutes deep learning is a misnomer. This misunderstanding of the true definition and capabilities of deep learning can cause confusion and detract from the significance of recent advancements in the field.

The term "deep" in deep learning refers to the number of layers in a neural network, with deep learning networks typically comprising dozens or even hundreds of layers. However, it is essential to note that the number of layers alone does not define deep learning. In contrast to simple neural nets, deep learning models possess the capability of *automatically extracting features* from raw data, enabling them to make predictions or decisions. This ability enables the accuracy of deep learning models to not saturate but continue to increase with more data and makes them suitable for a wide range of tasks such as image classification, speech recognition, and natural language processing, among others.

Recent advances in the architecture of neural networks, such as convolution nets, attention mechanisms, and transformers, along with more effective training approaches and techniques to improve generalization, such as dropout and regularization, have enabled the scaling of deep learning models to enormous sizes and trained on Big Data. For example, GPT-3, one of the largest deep learning models to date, has over 175 billion parameters.

While deep learning-based models have achieved superhuman capabilities in individual tasks such as vision and games like Go, and human-like conversation as in ChatGPT, the path to strong AI, intelligent machines that are indistinguishable from the human mind (self-aware rather than mimicking), or even its possibility is subject of considerable debate.

One of the remarkable features of the human brain is its power efficiency. The brain is estimated to have 85-100 billion neurons, with 100-1000 trillion connections, and a memory storage capacity of 2.5 petabytes. Recent research suggests that the neurons in the brain may be computational units on their own. Beniaguev et al. (2021) discovered that cortical neurons can be approximated by a deep neural network (DNN) with 5-8 layers, indicating that the brain's computational capabilities may be greater than previously thought. Despite this, the brain operates on a mere 20 watts of power, equivalent to the energy consumption of a small lightbulb.

In contrast, deep neural networks exhibit low energy efficiency. For example, GPT-3 was trained using ~10,000 GPUs, each consuming 300 watts. As AI models become increasingly complex and larger, the demand for computing resources to process them increases exponentially. An analysis by OpenAI found that since 2012, the amount of computational power used in the largest AI training runs has been growing exponentially at a 3.4-month doubling time. This is in stark contrast to Moore's Law, which had a 2-year doubling period. The significant difference in power usage between the brain and deep neural networks highlights a major limitation in the current design of neural networks. However, it also

suggests that it may be possible to develop artificial intelligence models that do not require exponentially growing data sets or excessive energy consumption.

References:

Aly, A., Guadagni, G. and Dugan, J.B., 2019, October. Derivative-free optimization of neural networks using local search. In 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) (pp. 0293-0299). IEEE.

Beniaguev, David, Idan Segev, and Michael London. "Single cortical neurons as deep artificial neural networks." *Neuron* 109, no. 17 (2021): 2727-2739.

Karimabadi, H., Roytershteyn, V., Wan, M., Matthaeus, W. H., Daughton, W., Wu, P., ... & Nakamura, T. K. M. (2013). Coherent structures, intermittent turbulence, and dissipation in high-temperature plasmas. Physics of Plasmas, 20(1), 012303.

Lee, W., Yu, H., Rival, X. and Yang, H., 2020. On correctness of automatic differentiation for non-differentiable functions. *Advances in Neural Information Processing Systems*, *33*, pp.6719-6730.