# Machine Learning for Model Driven Engineering

## SNN: A framework for Unstructured Neural Network Architectures

**Sarosh Krishan**

Universit’a degli studi dell’Aquila, Italy

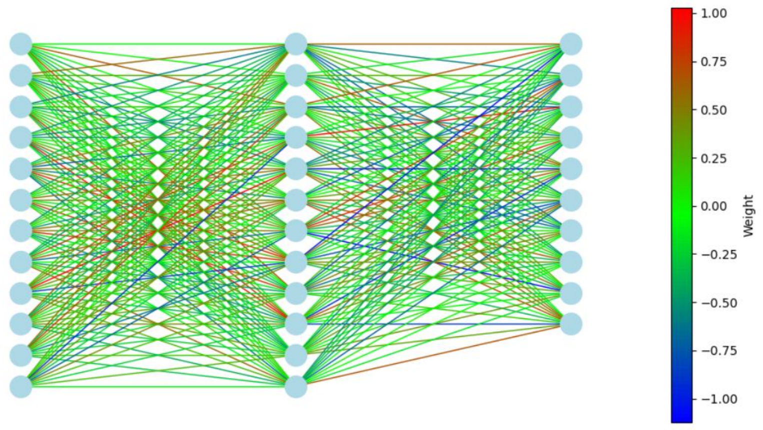
Email: sarosh.krishan@student.univaq.it

Academic year: 2024–2025

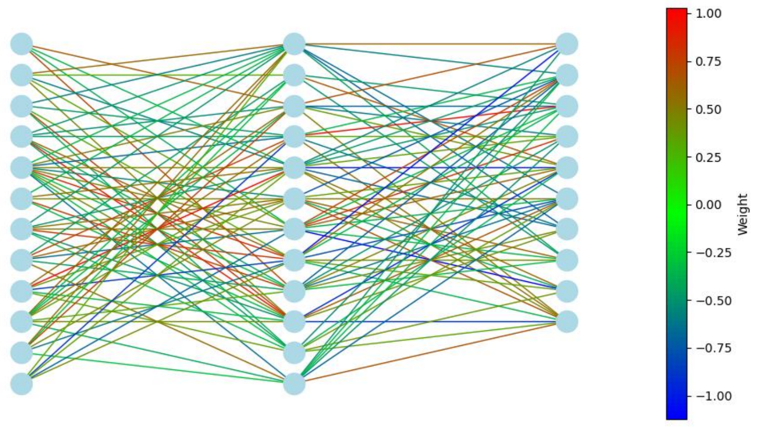
## 1. About the Topic

Traditional Neural Network Architectures, that make use of a layered approach for creating deep networks are highly effective at generalizing patterns and learning from a given dataset, however, they are not the most efficient at what they do.

For example, consider a simple 3-layered multi-layer perceptron.



When filtering out weights with an absolute value less than 0.3, you can see that a lot of the weights in the initial network weren’t that significant to the network’s overall performance. This reduction of unnecessary parameters is known as pruning.



## 1.1. Pruning

According to the paper ‘The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks’, “Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy” [1].

However, in most pruning approaches, a mask is created to zero out the weights that aren’t required instead of removing them completely. This is because there exists no framework for neural networks where the traditional ‘layers’ can be broken.

Instead of zeroing out the weights, if there was a way to remove them completely, it would be possible to reduce the space requirements of large networks by a more significant proportion.

## 1.2. Network Architecture Search (NAS)

Network Architecture Search (NAS) is a field of research that works with methods of discovering efficient neural network architectures. However, given the traditional frameworks, NAS methodologies are limited to discovering only layered architectures.

The problem with that is, if there are unstructured architectures that can perform better on certain problems, it would be impossible to detect them. Even if found, the current frameworks don’t allow for effective training for such architectures.

For example, the paper: ‘Layerless Feedforward Neural Networks with Automatic Architecture Optimization’ [2] attempts to discover neural network architectures with skip connections beyond a single layer by arranging neurons in a single array and adding $\frac{n}{2}(n+1)$ weights where every neuron can be connected to every subsequent neuron regardless of their distance. The paper had promising results but was able to conduct experiments with only basic networks as the author had to hard code everything in c++ and cuda due to a lack of frameworks that supported such architectures.

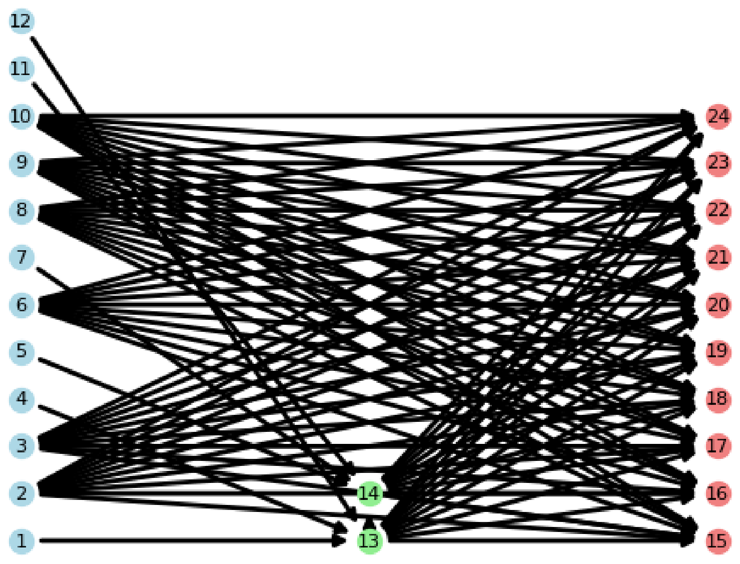
Other areas such as **Reinforcement Learning can** also makes use of unstructured NN architectures, for example the Neuro-evolution of augmenting topologies (NEAT) algorithm trains generations of NN models, where the best performing member of each generation is chosen to reproduce with the use of evolution algorithms. Removing the limitation of layered architectures can help NEAT create more complex models that can solve problems more effectively.

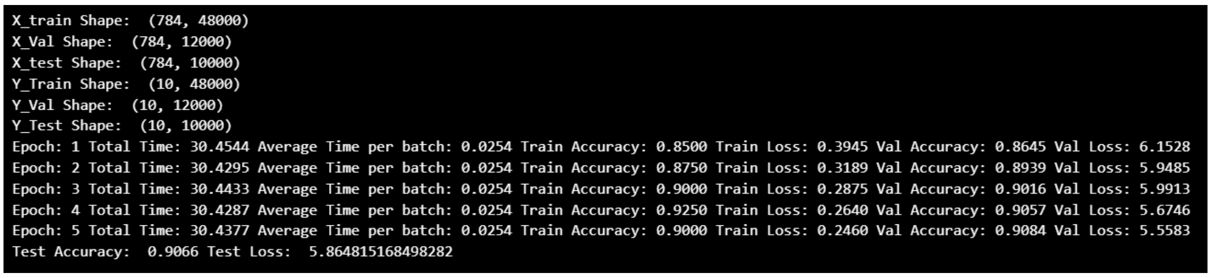
SNN seeks to create a training pipeline for such unstructured Neural Networks, so the potential of artificial intelligence can be explored beyond the limitations of layered architectures. SNN provides the following benefits over traditional neural network architectures:

- Getting better results since unstructured architectures add more nonlinearity to the NN models.
- Getting the same results as layered architectures with less data.
- Getting the same results as layered architectures with smaller architectures (as presented in many pruning techniques)
- Further efficiency in memory utilization after pruning.
- Better inference speeds at deployment after pruning.
- More freedom in the field of network architecture search (NAS).

## 2. The current framework

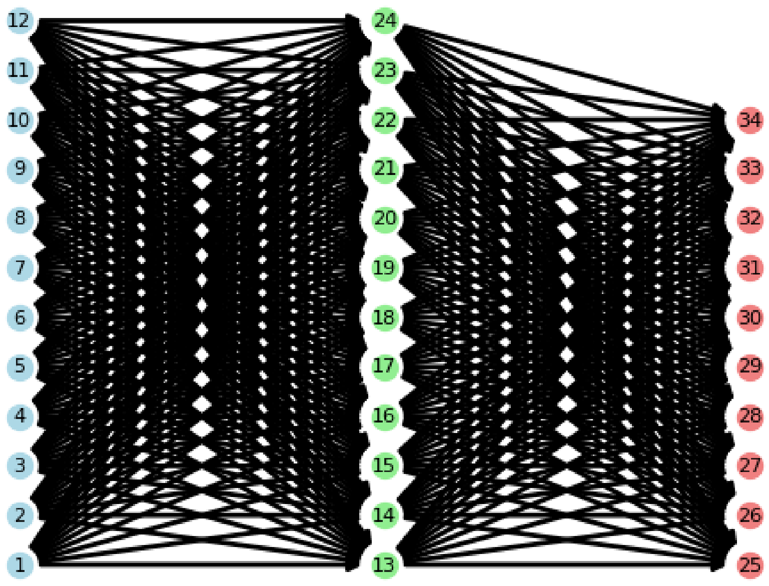
The current framework splits all neurons into individual perceptrons that train as independent nodes. For example, the framework can train networks such as the following:

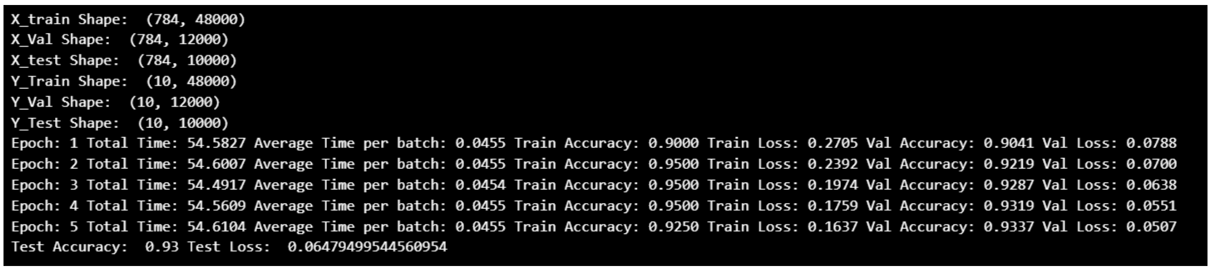




```
X_train Shape:  (784, 48000)
X_Val Shape:  (784, 12000)
X_test Shape:  (784, 10000)
Y_Train Shape:  (10, 48000)
Y_Val Shape:  (10, 12000)
Y_Test Shape:  (10, 10000)
Epoch: 1 Total Time: 30.4544 Average Time per batch: 0.0254 Train Accuracy: 0.8500 Train Loss: 0.3945 Val Accuracy: 0.8645 Val Loss: 6.1528
Epoch: 2 Total Time: 30.4295 Average Time per batch: 0.0254 Train Accuracy: 0.8750 Train Loss: 0.3189 Val Accuracy: 0.8939 Val Loss: 5.9485
Epoch: 3 Total Time: 30.4433 Average Time per batch: 0.0254 Train Accuracy: 0.9000 Train Loss: 0.2875 Val Accuracy: 0.9016 Val Loss: 5.9913
Epoch: 4 Total Time: 30.4287 Average Time per batch: 0.0254 Train Accuracy: 0.9250 Train Loss: 0.2640 Val Accuracy: 0.9057 Val Loss: 5.6746
Epoch: 5 Total Time: 30.4377 Average Time per batch: 0.0254 Train Accuracy: 0.9000 Train Loss: 0.2460 Val Accuracy: 0.9084 Val Loss: 5.5583
Test Accuracy:  0.9066 Test Loss:  5.864815168498282
```

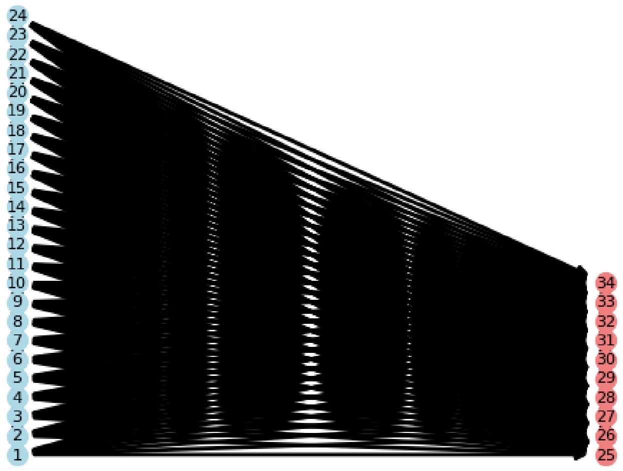
Additionally, following are the results of a layered architecture trained within the SNN framework.

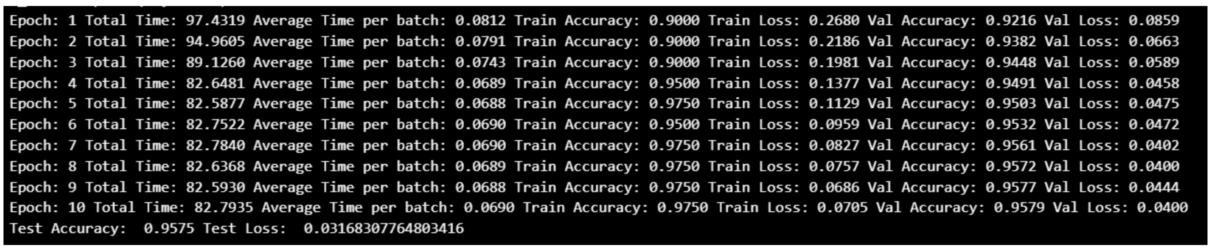




```
X_train Shape:  (784, 48000)
X_Val Shape:  (784, 12000)
X_test Shape:  (784, 10000)
Y_Train Shape:  (10, 48000)
Y_Val Shape:  (10, 12000)
Y_Test Shape:  (10, 10000)
Epoch: 1 Total Time: 54.5827 Average Time per batch: 0.0455 Train Accuracy: 0.9000 Train Loss: 0.2705 Val Accuracy: 0.9041 Val Loss: 0.0788
Epoch: 2 Total Time: 54.6007 Average Time per batch: 0.0455 Train Accuracy: 0.9500 Train Loss: 0.2392 Val Accuracy: 0.9219 Val Loss: 0.0700
Epoch: 3 Total Time: 54.4917 Average Time per batch: 0.0454 Train Accuracy: 0.9500 Train Loss: 0.1974 Val Accuracy: 0.9287 Val Loss: 0.0638
Epoch: 4 Total Time: 54.5609 Average Time per batch: 0.0455 Train Accuracy: 0.9500 Train Loss: 0.1759 Val Accuracy: 0.9319 Val Loss: 0.0551
Epoch: 5 Total Time: 54.6104 Average Time per batch: 0.0455 Train Accuracy: 0.9250 Train Loss: 0.1637 Val Accuracy: 0.9337 Val Loss: 0.0507
Test Accuracy:  0.93 Test Loss:  0.06479499544560954
```

Compared to this, a network that has every node (n) connected to subsequent (n – 1) nodes was also trained. For example, these are the results of a fully connected unstructured network with the same number of nodes as the layered architecture.





```
Epoch: 1 Total Time: 97.4319 Average Time per batch: 0.0812 Train Accuracy: 0.9000 Train Loss: 0.2680 Val Accuracy: 0.9216 Val Loss: 0.0859
Epoch: 2 Total Time: 94.9605 Average Time per batch: 0.0791 Train Accuracy: 0.9000 Train Loss: 0.2186 Val Accuracy: 0.9382 Val Loss: 0.0663
Epoch: 3 Total Time: 89.1260 Average Time per batch: 0.0743 Train Accuracy: 0.9000 Train Loss: 0.1981 Val Accuracy: 0.9448 Val Loss: 0.0589
Epoch: 4 Total Time: 82.6481 Average Time per batch: 0.0689 Train Accuracy: 0.9500 Train Loss: 0.1377 Val Accuracy: 0.9491 Val Loss: 0.0458
Epoch: 5 Total Time: 82.5877 Average Time per batch: 0.0688 Train Accuracy: 0.9750 Train Loss: 0.1129 Val Accuracy: 0.9503 Val Loss: 0.0475
Epoch: 6 Total Time: 82.7522 Average Time per batch: 0.0690 Train Accuracy: 0.9500 Train Loss: 0.0959 Val Accuracy: 0.9532 Val Loss: 0.0472
Epoch: 7 Total Time: 82.7840 Average Time per batch: 0.0690 Train Accuracy: 0.9750 Train Loss: 0.0827 Val Accuracy: 0.9561 Val Loss: 0.0402
Epoch: 8 Total Time: 82.6368 Average Time per batch: 0.0689 Train Accuracy: 0.9750 Train Loss: 0.0757 Val Accuracy: 0.9572 Val Loss: 0.0400
Epoch: 9 Total Time: 82.5930 Average Time per batch: 0.0688 Train Accuracy: 0.9750 Train Loss: 0.0686 Val Accuracy: 0.9577 Val Loss: 0.0444
Epoch: 10 Total Time: 82.7935 Average Time per batch: 0.0690 Train Accuracy: 0.9750 Train Loss: 0.0705 Val Accuracy: 0.9579 Val Loss: 0.0400
Test Accuracy:  0.9575 Test Loss:  0.03168307764803416
```

The accuracy of the unstructured network improves by 3% using the same number of nodes. Even though the unstructured network takes more time, the results highlight the potential of such networks in network architecture search (NAS).

## 3. Optimization

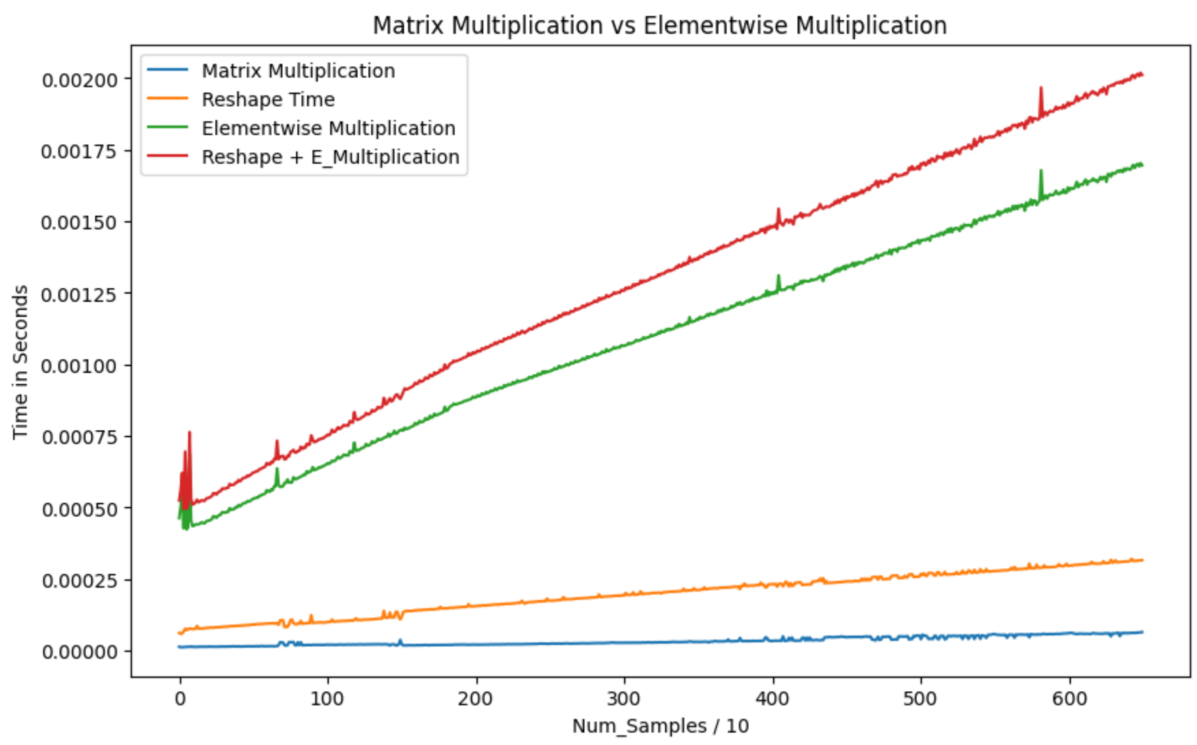
Although powerful, the SNN framework is highly unoptimized in its current state. It lacks parallelization techniques by grouping neurons into pseudo-layers that can train in parallel, potentially improving the time complexity of the algorithm. A theoretical arrangement of pseudo-layers is discussed below.

When calculating the output using forward propagation, a few things need to be considered:

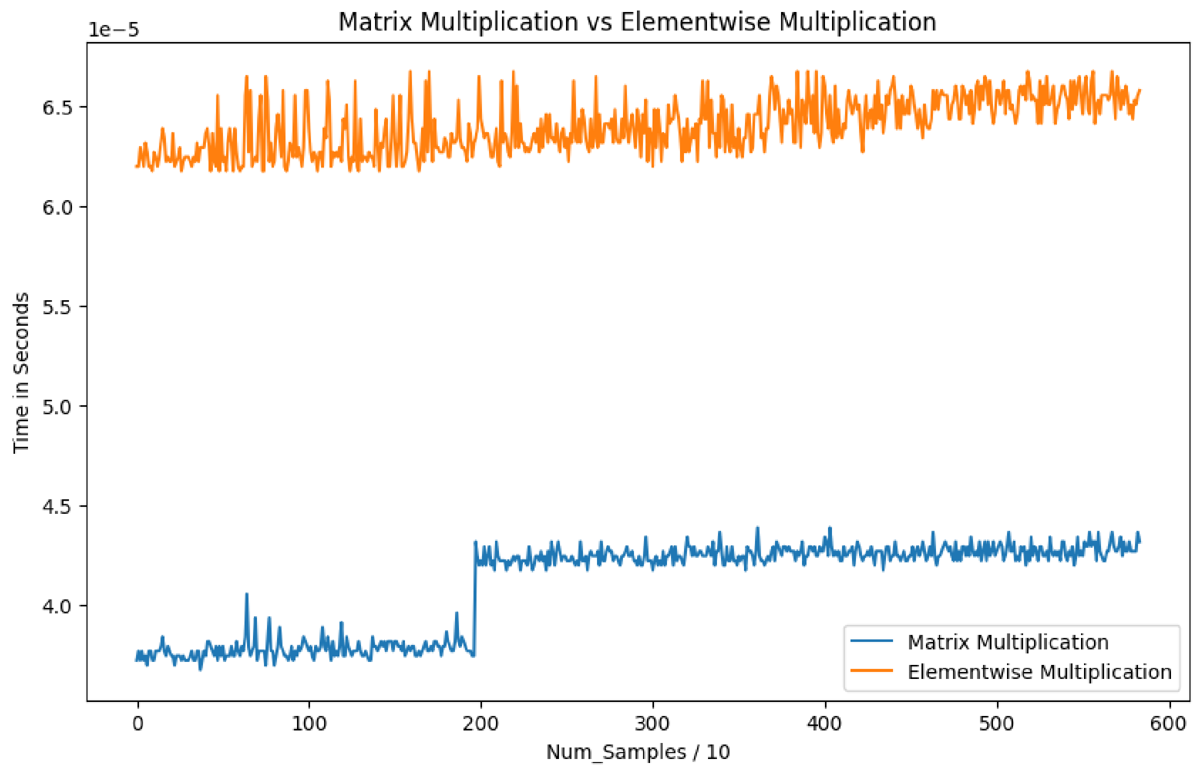
A. Neurons need to be designated as input, intermediate and output neurons.
B. The input neurons shouldn't be required to have weights connected to every input feature.
C. Neurons independent of each other can be propagated in parallel. To do this, once the architecture is defined. A function can locate all the independent neurons and arrange them in 'pseudo layers'.
D. When performing matrix multiplication in layered architecture, the input for each neuron in a layer is assumed to be the same, thus allowing for the input matrix to be an nxm matrix where n is the number of input features, and m is the number of samples. However, when grouping independent neurons in unstructured architectures, the inputs for each neuron can be different in both value and size. Thus, to calculate the output of 'pseudo layers', an elementwise multiplication approach can be taken to matrix multiplication.
E. For backprop, similar considerations need to be made.

In the matrix multiplication, instead of a 2D n x m matrix, the input can be represented as a 3D $n \times l \times m$ matrix where n is the number of neurons in the pseudo layer, l is the number of input features for the largest input (the rest of the inputs are padded with zeros to complete the matrix) and m is the number of samples in the dataset.

Both these algorithms have a theoretical time complexity of O(n). However, when tested using the numpy library in python, the new approach performed much worse on the CPU. This was potentially since numpy is a highly optimized library, compared to the hard coded approach.



However, when tested on a Tesla V100-PCIE-16GB GPU to perform the same functions using the cupy library, the performance of both approaches was nearly identical, with the hard coded approach being only microseconds slower than the built in cp.dot() function.



These results are highly promising as they show the potential for unstructured NN models to be similar to layered structures in computation time, while retaining all their other benefits.

## 4. Future Work

In its current state, the SNN framework can train any structured or unstructured multi-layer perceptron architecture. However, to make it useful, several key improvements are needed:

1. **Support for convolutional networks, RNNs and Transformers.** In the world of deep learning, a framework is fundamentally incomplete unless it can train architectures containing convolutional blocks and RNN components such as LSTMs and GRUs. Additionally, with the advent of transformers, they have become a staple of a useful deep learning framework.
2. **Optimization using Pseudo-Layers.** Regardless of its capability, SNN is significantly slower to execute than tensorflow or Keras, making any useful benefits it can provide meaningless.
3. **Pruning functionalities.** Although the network supports pruned architectures, it provides no functionality to perform pruning on existing architectures.
4. **A separate methodology for Dense-Unstructured Architectures.** Since Dense-Unstructured networks follow a coherent standard, there can be a separate functionality for training them, making them faster than ordinary unstructured architectures.

Overall, SNN can provide a more complete picture of our understanding of deep learning and can help develop more powerful and robust neural networks.

## 5. References

1. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks
2. Layerless Feedforward Neural Networks with Automatic Architecture Optimization