Randomly Wired Networks are on the rise, have we been creating wrong Networks all along?

Mudit Verma
muditverma@asu.edu
Arizona State University
Tempe, AZ

ABSTRACT

Designing a neural network for classification tasks is experiencing a shift from manual choices to automated network generation. Surprisingly, recent works used random graph generation techniques to obtain random neural network architectures that could compete with state of the art methods for an image classification task. This motivates the use of MCMC Sampling methods, which have been extensively used for random walks on graphs & graph generation. This project aims to view and present ideas on Neural Architecture Search from the lens of MCMC Sampling. We run a Markov chain to sample random neural architectures and contrast them with other random graph generation methods. We support claims of why algorithms that have been used to generate state-of-the-art, yet, random neural networks are actually biased and how true random sampling is not the way to go. Our observations on the accuracy versus various graph metrics reveal that certain properties of the graph are desired and should not be chosen randomly, tempering the unrest in the NAS community about whether the existing contributions to intelligent searches went to vain.

KEYWORDS

Neural Architecture Search, MCMC Sampling, Random Graphs

1 NAS BACKGROUND

Deep Learning methods have been shown to be successful in numerous and diverse tasks like computer vision, language identification, text generation. [8, 11, 23] and many more [1]. This has caused a paradigm shift from feature engineering towards architecture design. Although designing architectures allows flexibility and is less demanding, with the increase in complexity of tasks, neural network design itself has become a field of research.

Neural Architecture Search as a research area started as early as in the 1980s with researchers using genetic algorithms [16] and formulating different ideas to approach the problem [10]. Since then, the field has experienced many contributions. Some of the more recent works [2, 6] started with confining the problem to some tractable search space of neural architectures in the image recognition domains or searching for specific parts of the neural network like activation functions [20]. They use Reinforcement Learning and Recurrent Neural Network controllers to obtain the neural architecture. Further, in the timeline, more global approaches were seen, which would repeat a fixed module, called cell, in different ways to obtain a final neural network [32, 33]. However, these approaches were quite slow in execution and required many different architectures to be trained before achieving a respectable accuracy score. To curb this flaw weight sharing paradigms were used like [19] - Efficient NAS or ENAS. Although better, these methods were still slower in comparison to the training of hand-crafted networks. As noted by [24], the success of ENAS, a very popular NAS technique, was a result of the search space design rather than the controller training and concluded that one-shot architecture design is an efficient alternative to architecture search by ENAS. [4] talks extensively about neural architecture design along dimensions of search space, search strategy, and performance estimation.

In the realm of One-shot architecture search, some works [3, 27]involved the use of faster search algorithms like Hill-Climbing with better child generation function which could result in obtaining a trained final architecture is similar in duration as those of hand-crafted networks. [29] lists out various one-shot techniques used for neural architecture search where in some works would pre-generate a neural architecture graph and then train it as the final architecture.

However, recently published [30] shatters most of the above works and numerous other articles which showcase a neural architecture as it's contribution to classification tasks. They use random graph generation algorithms to sample a graph and called the placement of edges in the graph as "Wiring". After converting the obtained graph to a Directed Acyclic Graph, they use their mapping function to map graph nodes to layers and edges to aggregation operation (like add outputs of multiple layers to feed to the next layer). The mapping function is fixed beforehand and does not change with the produced graph. Surprisingly, they could compete closely with the state-of-the-art accuracy values for ImageNet [21] challenge dataset, with few randomly generated neural networks. xie2019exploring makes it look as if all the intelligence put into curate hand-crafted architectures is no better than generating random architectures; however, we will discuss how hidden priors like graph density and longest path length affect accuracy.

This motivates this project and a discussion on whether the "randomly generated graphs" were truly random and question what kinds of graphs were able to get respectable accuracy scores. We will explore a bit more about connections of NAS to network science and Brain wiring to get a more concrete idea of why Wiring is essential for the image classification task.

2 PROPOSED WORK

(1) (Section 3) **Reading:** We study how NAS connects to Network Science which affects the architecture design choices. Further, we draw certain points from cognitive science as a motivation to decouple learning an architecture and learning network weights. The reading also explores the involvement

, ,
2019.

of MCMC with NAS and Deep Learning and concludes that [30] is most likely the only attempt at using a random graph generation strategy to obtain and train a neural network.

(2) (Section 6) **Theme:** This project critiques the idea that truly random graphs are the way to go in the field of Neural Architecture Search.
(3) (Section 4) **Implementation:** We will then propose a simple Markov chain (MC) to sample random undirected graphs. The sampled graphs will be converted to neural networks via a trivial mapping function, mapping graph nodes to layers and graph edges to aggregation operations. We obtain several other graphs via three other random graph generation models - Erdos-Renyi (ER), Barabasi-Albert (BA), and Watts-Strogatz (WS), and contrast them against the MC-graphs along with various graph metrics. The results of the experiments are compiled to reveal some interesting insights about the nature of graphs that perform well on an image classification task.
(4) (Section 7) **Future Work:** We will end the discussion with some interesting ideas involving the use of MCMC techniques in the are of NAS.

3 ESTABLISHING CONNECTIONS

The aim of this section is to study the motivations for Wiring in neural architectures from a cognitive science perspective.

3.1 NAS Connections to Brain Wiring Network Science

[31] beautifully compares artificial neural networks to humans and nature. It explains the idea of innate abilities in human babies to be able to recognize and manipulate items really quickly, which suggests that they are born with highly structured brain connectivity. This connectivity provides a scaffolding upon which rapid learning can occur. By this analogy, either the neural network, be initialized with "good" wired architectures, or these must be learned evolutionarily (transfer learning). Given the diversity, weight sharing among entirely different classification tasks have not proven to be successful, leading us to believe that neural architectures are supposed to be constructed with certain priors and exhibit certain properties. One can argue that weight initialization may also play a role, which it does to some extent, but [27] shows that their NAS architecture performance remained about the same even with different weight initialization methods. Another important point posited by [31] is that the brain wiring (innate) abilities are not completely learned during the lifetime of the animal; they are hardwired to most extents. This encourages us to assume that learning of network architecture should be partially decoupled from the learning that happens during the training of the neural network. This decoupling shows that even from the perspective of neuroscience, the network topology and Wiring is as important as it's training schedule, if not more.

[25] takes results of [30] to say that "achievable input-output functions largely over-laps for different architectures". This further suggests that works involving "intelligent" contributions by coming up with a new wired network topology would not be much different from approximation by other randomly wired networks. Although we acknowledge and appreciate the random wiring strategy to be useful, our results show that "good" networks are skewed by some properties of graphs even with repeated attempts to generate random graphs. Hence, coming up with "any" architecture will not be sufficient for obtaining a good accuracy score.

[28] explains small world networks. In network science, small world graphs are such that one can reach one any one node to any other node in at most 6 hops. Such graphs have been found in neural wiring of worms [26] motivating us to fix certain hopsize in which we can move from any one node to any other node. Hence we have the concept of components as we will discuss later.

3.2 MCMC & Deep Learning & Random Search for NAS

Markov Chains and Monte-Carlo methods have been used for various sampling tasks in Deep Learning like [9, 17]. These works have mostly been in the context of using MCMC to aid the creation of generative models. Some of the other works involve the use of MCMC to create DAGs of bayesian relationships from observed data [13, 14]

Parallelly, Markov chains have been used to perform random walks on graphs, sample random distributions, and even sample random graphs [5]. However, none of the works to the best of our knowledge relates to using a Markov Chain for sampling neural architectures. In fact, [30] is as close as research works get to employing a Markov chain for sampling neural architectures.

Random Search has been tried with Neural Architecture Search several times; however, the randomness was in either selecting operations for evolving architecture [27] or were involved the image data/ model accuracy/weights of other architectures in determining the final architecture [15, 22].

4 SETUP

In this section, we discuss the setup for the implementation part of this project. We will familiarize ourselves with three popular random graph generation algorithms and propose a Markov Chain model to generate random graphs. This will be followed by explaining fixed rules or constraints to generate a directed acyclic called NAS Graph. Finally, we will look at how this DAG is mapped to become a neural network.

4.1 Random Graph Generation Algorithms

We use a total of four algorithms to sample random graphs, which are, proposed Markov Chain (MC), Erdos-Renyi (ER), Barabasi-Albert (BA), and Watts-Strogatz (WS). The latter three have been used by [30] to generate random graphs.

4.1.1 ***Markov Chain***. Algorithm 1 gives the proposed Markov Chain. The operations from any state can be described as, stay, swap 2 edges, add/remove edge and add/remove a node.

It can be seen that the presented chain is both aperiodic and irreducible. That is, we have a self-loop on all nodes hence aperiodic, and also, we can move in the state space by adding or removing nodes and adding or removing edges with positive probability. Hence there must exist a stationary distribution for the presented chain. We also put a limit to the maximum and a minimum number

of nodes in the graph to be 5 to 100. If an operation is selected which violates this, the previous graph is returned.

Although a thorough investigation in mixing times and stationary distribution values is to be conducted, we can intuit that the stationary distribution will not be uniformly random. Among addNode and removeNode operations, the probability to transition to a new state will not be equal to transition back to the initial state with operation newNode. Thus, we can think of the Probability distribution not being symmetric. Nevertheless, the graphs may be biased, but do not use any knowledge of the classification task.

Algorithm 1 Markov Chain

procedure MarkovChain(G_0, k) ▷ Inital Graph and Chain Length
  while $t \neq k$ **do**
    $G \leftarrow G_0$ with prob 0.1 ▷ STAY
    $G \leftarrow$ edge Swap with prob 0.4 ▷ SWAP
    if edge swap fails **then**
      $G \leftarrow G_0$
    end if
    $G \leftarrow$ add/remove edge with prob 0.2 ▷ EDGE
    pick 2 nodes randomly, and/remove edge with equal prob
    $G \leftarrow$ add/remove node with prob 0.3 ▷ NODE

4.1.2 ***Other Random Models.*** [30] explains the ER, BA, and WS models. For our discussion, we need to know that **ER Model** produces a graph with N Nodes and an edge between two nodes is connected with probability P, independent of all other nodes and edges. The **BA Model** generates a random graph by sequentially adding new nodes. The initial state is M nodes without any edges ($1 \leq M < N$). The method sequentially adds a new node with M new edges. Any graph generated by BA(M) has exactly $M * (N - M)$ edges. Finally, in the **WS Model**, the N nodes are regularly placed in a ring, and each node is connected to its $K/2$ neighbors on both sides (K is an even number). Then, in a clockwise loop, for every node v, the edge that connects v to its clockwise i^{th} next node is rewired with probability P. "Rewiring" is defined as uniformly choosing a random node that is not v and that is not a duplicate edge. This loop is repeated $K/2$ times for $1 \leq i \leq K/2$.

ER, BA, and WS models are available in the networkx library [7].

4.2 Obtaining the Graph

The complete graph is split into components $C_1, C_2, C_3 ... C_n$ such that each component is an Undirected Sub Graph. Each component has one root node and one sink node such that if this graph were to be converted to a DAG, the first and last element of topologically sorted order would be the root and sink of this graph. These components are attached, such that the root of C_i is the sink of C_{i-1}. The root of C_0 is the input node, and the sink of C_n is the output node. Figure 1 shows one such architecture with 3 components {a,b,ou}

4.2.1 ***Undirected Sub Graph.*** We use one of the algorithms mentioned in Section 4.1. This gives us a component of the final graph.

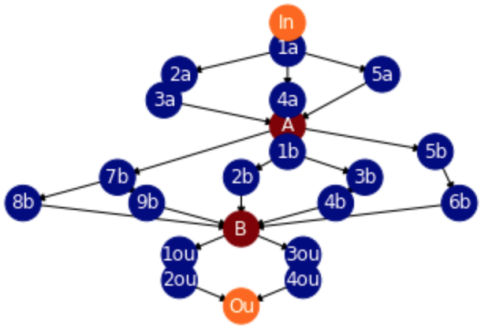

Figure 1: Best performing architecture our of all sampled graphs for all methods. This is a 3 component graph generated by MC model

Two new nodes are added to the component such that for they become the root and sink nodes. All the current nodes in the component which has no predecessor are connected to the root node (edge from root to node). All the nodes which have no successor are connected to the sink node (edge from nodes to sink).

4.2.2 ***Obtaining a DAG for Undirected Subgraph.*** A simple yet efficient way of converting an undirected graph to a directed acyclic graph is to impose a strict total order on the edge directions. When enumerating all the edges of the obtained undirected graph, we can set our total order condition to be the lexographic precedence of node names. Nodes are named by numbers, followed by their component labels.

4.3 Mapping Nodes to Layers & Operations

Once we have a DAG, it is quite trivial to convert to a neural network. We can assume all the nodes to be layers and edges to be aggregators. A layer is a combination of a relu, conv (convolution) layer, batchNorm sublayers. The number of channels in convolution layer is dependent upon the input image dimension and is manually set so that there are no discrepancies in the network dimensions.

All the internal nodes in a component (other than root and sink) have the same inChannels and outChannels. inChannels and outChannels are changes at the root/sink nodes of any/all of the components. The internal nodes have padding.

For example, in Figure 1, node 'In' is the input node, and root of component 'a'. 'A' is the sink of component 'a'. {1a,2a,3a,4a,4a,} are internal nodes of component 'a'. All internal nodes have the same inChannels and outChannels for the conv sublayer. The outChannels of root node, say, 'B' matches all its immediate children inChannels, and the inchannel of the root node, again, 'B' must match the outchannels of its predecessors. There may be a MaxPool layer at the sink nodes, except the sink of the last component. Finally, there is a Flatten layer attached to 'Ou' which linearizes the 'Ou' layer and connects to a fullyConnected layer with outputs = number of classes in the classification task.

5 EXPERIMENTS

This section explains the different experiments conducted on NAS Graphs. We set certain parameters to gain insights about the effect of those parameters on the final testing accuracy of the trained neural network.

The code for this is available on github [1] which describes the exact configurations used. We have the following parameters : *epochs*, *batchsize*, *testbatchsize*, *learningrate*, *gamma*, *seed* which are common regardless of the graph generation algorithm. The optimzer for all neural networks is AdaDelta and Loss function is negative log likelihood implementations of PyTorch 1.3.1 [18].

The number of components is a common parameter [30] and is set to 3 for all the models. 3 has been set because the input image is 28x28 of MNIST in our experiments versus high dimensional Imagenet images used in .

Each algorithm has its own sets of parameters. MC has chain run length, ER has edge probability and number of Nodes, BA has the number of edges and max number of Nodes, WS has the number of nodes, and the neighbor size, probability of rewiring.

To obtain a spectrum of different type of graphs we change the number of nodes parameter for ER, BA and WS from 5, 10, 20 and generate graphs with different p/m/k (p from 0.2 to 0.8, m from 2 to 8, k from 4 to 7) values to obtain 12 graphs for each for these algorithms and over 20 graphs from MC model. Hence we train and test and showcase our results on MNIST dataset having 60,000 images with over 56 sampled graphs.

6 RESULTS

In this section, we will analyze the results of experiments conducted from the perspective of critiquing the idea posited by [30] that random graph generation is the panacea for neural architecture search problems.

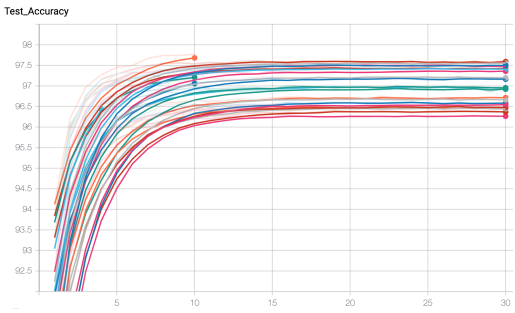

Figure 2: Test Accuracy for MC Models

Figures 4,5,6 are enough to showcase our observations for graph metrics. Figures 2 and 3 present the test accuracies for MC and ER/BA/WS generated models respectively. Interested readers may refer to attached appendix which shows plots with other node degree metrics like mean, median, variance and total count of nodes. Each spot in the plots 4,5,6 is a random graph trained for 30 epochs and other specified hyper parameters. Color of the spot determines the algorithm used to obtain that graph. For plots 2 and 3, all the lines are training curves versus epochs for each of the graphs. Colors indicate distinct graph models.

[1] https://github.com/famishedrover/MCMC-NAS

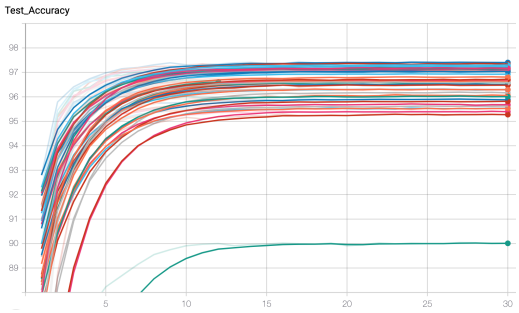

Figure 3: Test Accuracy for ER,BA,WS Models

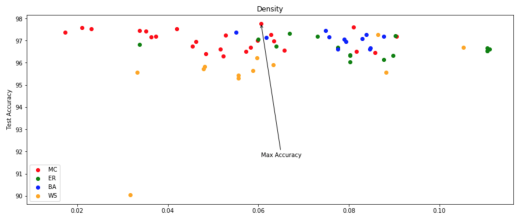

Figure 4: Density

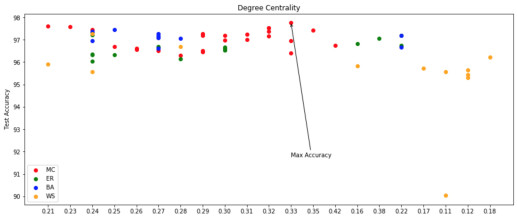

Figure 5: Degree Centrality

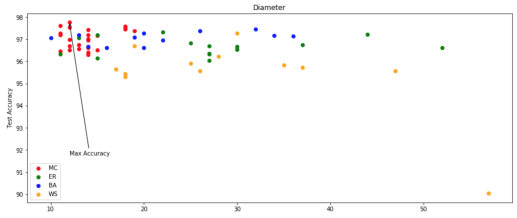

Figure 6: Diameter

6.1 Other Works

Although the primary aim of this work is not to obtain state-of-the-art accuracy, however for completeness, we will mention the result of other works as well. Results on MNIST by one of the one-shot NAS works [27] is 0.28% error on test-set. Additionally, [12] is a non-NAS method which achieves 0.18% error.

6.2 Observations

The following observations can be drawn from the plots :

(1) Maximum accuracy achieved by any graph is 97.6%, the architecture for which is given in Figure 1.
(2) Figure 4 shows that the density of the generated model is spread throughout; however, if one were to imagine a curve joining all the highest points for all densities, the accuracies decline from just over 97% to 96%. Moreover, the max accuracy graph is at the center of this plot. MC models are found to be better than other generated graphs (even though parameters like the probability of an edge between two nodes was varied from 0.2 to 0.8).
(3) Figure 5 shows an even spread of graphs by this metric, yet as the degree measure increases, the accuracy falls.
(4) Figure 6 shows that high performing graphs are skewed towards shorter diameter graphs. Diameters for DAGs is the longest path length (from root to leaf). Therefore we can find longer neural networks usually generated by ER/WS models to perform worse. Between BA and MC models, BA is more evenly spread, while MC clusters around small length graphs.
(5) Other plots from the attached Appendix, like node degree variance, shows that MC models are randomly sampled as graphs of all variance can be seen.
(6) Figure 7 in the attached Appendix shows that graphs generated by other methods (ER/WS/BA) with similar max degree nodes to the best performing network (by MC) perform worse. While MC models have varying max node degree values, perform comparatively to the max performing graph.
(7) Test Accuracy curves of these graphs in Figures 2 and 3 shows that MC models lie in the accuracy range of 96.5% to 97.5% whereas other methods have a much larger range from 95% to 97%.

6.3 Comments

Why Random Wiring is not the solution ? These experiments clearly indicate that the neural networks which do perform really well are likely to be shorter, have high degree centrality (middle is high in the plot), and are neither too dense or too sparse. These properties can be observed for all better performing neural networks regardless of their generation algorithm, indicating the "goodness" of networks. claims of random neural networks being the next steps in the field of Neural Architecture Search breaks down. Since true random graph sampler would generate graphs which on expectation lie at the center of all of these plots, and it is conspicuous that "well-performing graphs" do not cluster around the center. [27] is a one-shot NAS technique that uses hill-climbing search over NAS space using morphism operators to evolve their network. Hence their networks are dependent upon accuracy they achieve, unlike in random wiring methods. Moreover, they achieve an error of 0.28% on MNIST, which is much better than ours. This is a typical case of partial decoupling between network architecture and network learning, as in Section 3.1.

Are these results any good? The results of this paper and [30] indicates that logical explanations given in many of the works which claimed to have come up with a better architecture just by rewiring the network edges are incorrect. Moreover, we claim that the "goodness" of architecture depends upon the graph properties and the input size. We say this, since agrees that one-shot methods tend to over-parameterized their networks, and given the diversity in examples of individual classes, it becomes difficult for the neural network to overfit/memoize the data. In doing so, they tend to generate humongous neural networks with billions of parameters; however, we see that in our experiments, it is the more compact networks that performed better.

7 FUTURE WORK

One of the interesting works by [5] attempts to generate a connected random graph conditioned on an ensemble of connected random graphs using a Metropolis-Hastings framework. A work involving collecting and generating different "good" performing architectures can be collected, and then leveraging the above work, random graphs may be sampled off of the created sample. It would be similar to bootstrapping weights or "transfer" learning of architectures via MCMC.

In the current setup, we can create another layer of Markov Chain, deciding the type of the layer at each node. A major hurdle would be to sample such that the chosen parameters of the selected layers for the node are dimensionally compatible.

8 CONCLUSION

In this paper, we present evidence from the cognitive science and NAS community to gauge the importance of Wiring (& random Wiring) in neural architectures. We critique some of the ideas presented by [30] since they show one of the first works in experimenting with randomly sampled neural architectures for image classification tasks. We perform our experiments on the MNIST dataset with a proposed Markov Chain and other graph generation algorithms like Erdos-Reyni, Barabasi-Albert, and Watts-Strogatz and realize that their priors have an impact on test accuracy. Further, we find that in all of the generated random graphs, the "well" performing graphs had a pattern associated with them conditioned on specific graph metrics.

We conclude that not all randomly generated graphs are equal for a specific problem statement. The fact that the results in this paper and [30] are competitive with other better models shows that much the architecture design does not depend upon the image pixels in the classification task, but actually on the image dimensions. To be precise, the architecture design should depend upon the least number of dimensions covering most of the explained variances.

REFERENCES

[1] Mahbubul Alam, Manar D Samad, Lasitha Vidyaratne, Alexander Glandon, and Khan M Iftekharuddin. 2019. Survey on Deep Neural Networks in Speech and Vision Systems. *arXiv preprint arXiv:1908.07656* (2019).
[2] Dipankar Dasgupta and Douglas R McGregor. 1992. Designing application-specific neural networks using the structured genetic algorithm. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, 87–96.
[3] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. 2017. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528* (2017).
[4] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).
[5] Caitlin Gray, Lewis Mitchell, and Matthew Roughan. 2018. Generating connected random graphs. *arXiv preprint arXiv:1806.11276* (2018).

[6] Stephen Grossberg and Nestor A Schmajuk. 1987. Neural dynamics of attentionally modulated Pavlovian conditioning: Conditioned reinforcement, inhibition, and opponent processing. *Psychobiology* 15, 3 (1987), 195–240.

[7] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[10] J Stephen Judd. 1990. *Neural network design and the complexity of learning.* MIT press.

[11] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3128–3137.

[12] Kamran Kowsari, Mojtaba Heidarysafa, Donald E Brown, Kiana Jafari Meimandi, and Laura E Barnes. 2018. Rmdl: Random multimodel deep learning for classification. In *Proceedings of the 2nd International Conference on Information System and Data Mining*. ACM, 19–28.

[13] Gilles Kratzer and Reinhard Furrer. 2019. Is a single unique Bayesian network enough to accurately represent your data? *arXiv preprint arXiv:1902.06641* (2019).

[14] Gilles Kratzer and Reinhard Furrer. 2019. *mcmcabn: a structural MCMC sampler for DAGs learned from observed systemic datasets.* https://CRAN.R-project.org/package=mcmcabn R package version 0.1.

[15] Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638* (2019).

[16] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. 1989. Designing Neural Networks using Genetic Algorithms.. In *ICGA*, Vol. 89. 379–384.

[17] Simon Osindero and Geoffrey E Hinton. 2008. Modeling image patches with a directed hierarchy of Markov random fields. In *Advances in neural information processing systems*. 1121–1128.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[19] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[20] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[22] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. 2019. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142* (2019).

[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[24] Prabhant Singh, Tobias Jacobs, Sebastien Nicolas, and Mischa Schmidt. 2019. A Study of the Learning Progress in Neural Architecture Search Techniques. *arXiv preprint arXiv:1906.07590* (2019).

[25] Fabian H Sinz, Xaq Pitkow, Jacob Reimer, Matthias Bethge, and Andreas S Tolias. 2019. Engineering a less artificial intelligence. *Neuron* 103, 6 (2019), 967–979.

[26] Lav R Varshney, Beth L Chen, Eric Paniagua, David H Hall, and Dmitri B Chklovskii. 2011. Structural properties of the Caenorhabditis elegans neuronal network. *PLoS computational biology* 7, 2 (2011), e1001066.

[27] Mudit Verma, Pradyumna Sinha, Karan Goyal, Apoorva Verma, and Seba Susan. 2019. A Novel Framework for Neural Architecture Search in the Hill Climbing Domain. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. IEEE, 1–8.

[28] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440.

[29] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A Survey on Neural Architecture Search. *arXiv preprint arXiv:1905.01392* (2019).

[30] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. 2019. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569* (2019).

[31] Anthony M Zador. 2019. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications* 10, 1 (2019), 1–7.

[32] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2423–2432.

[33] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.

APPENDIX

Additional Plots

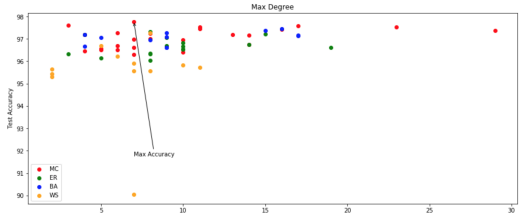

Figure 7: Max Node Degree

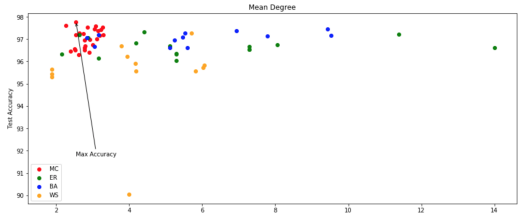

Figure 8: Mean Node Degree

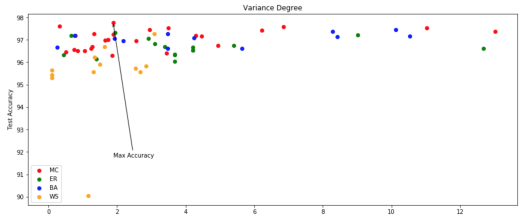

Figure 9: Node Degree Variance

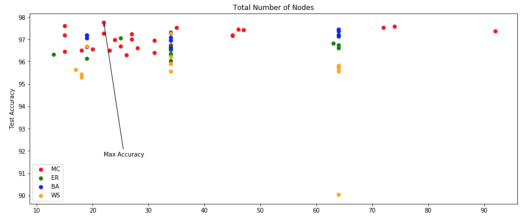

Figure 10: By Total Number of Nodes