7th ICML Workshop on Automated Machine Learning (2020)

RicciNets: Curvature-guided Pruning of High-performance Neural Networks Using Ricci Flow

S. Glass SEG67@CAM.AC.UK
Department of Physics, University of Cambridge

S. Spasov SES88@CAM.AC.UK
Department of Computer Science and Technology, University of Cambridge

P. Liò PL219@CAM.AC.UK
Department of Computer Science and Technology, University of Cambridge

Abstract

A novel method to identify salient computational paths within randomly wired neural networks *before* training is proposed. The computational graph is pruned based on a node mass probability function defined by local graph measures and weighted by hyperparameters produced by a reinforcement learning-based controller neural network. We use the definition of Ricci curvature to remove edges of low importance before mapping the computational graph to a neural network. We show a reduction of almost 35% in the number of floating-point operations (FLOPs) per pass, with no degradation in performance. Further, our method can successfully regularize randomly wired neural networks based on purely structural properties, and also find that the favourable characteristics identified in one network generalise to other networks. The method produces networks with better performance under similar compression to those pruned by lowest-magnitude weights. To our best knowledge, this is the first work on pruning randomly wired neural networks, as well as the first to utilize the topological measure of Ricci curvature in the pruning mechanism.

1. Introduction

At birth, the construction of the most important networks is largely random and random graph modelling is heavily used in the study of the human brain (Bullmore and Sporns, 2009; Bassett and Sporns, 2017). Recent work on randomly wired neural networks has emulated this in the field of deep learning, and moves away from the wiring approach that has typically dominated NAS (Xie et al., 2019). Randomly wired networks display comparable performance to state-of-the-art architectures (eg. ResNet, DenseNet), and provide a relatively unrestricted space on which to perform further optimisation. We propose a search method that takes place within a low-dimensional search space; a pruning methodology which operates on networks produced by a successful random network generator. It is based on the discrete Ricci curvature of a graph, with estimates of a node's community, contribution to robustness and computational demand contributing to the identification of salient computational paths in the network. A curvature-guided diffusion process, Ricci flow, deforms the discrete space of the graph, and edges within the graph are removed based on their local deformation. A reinforcement learning controller parameterises the Ricci flow process. To the best of the authors' knowledge, this is the first work to take inspiration from the physics concept of space curvature deformation, in combination with reinforcement learning, to drive the process of neural network pruning.

The technique proposed has three key advantages: (1) it is a successful form of regularisation which promotes sparse network connectivity, and as a result low computational demand; (2) it operates with no degradation in baseline performance; (3) a successful hyperparameter state can be applied, without further optimisation, to similarly produced random networks. The process operates before training, saving compute during both training and inference. This is the first known work to investigate the pruning of randomly wired neural networks. Using *RicciNets*, we demonstrate novel, efficient generation of *compact* neural architectures.

2. Related Work

Ricci curvature is the definition of curvature used in Einstein's Field Equations. Loosely, it measures the deformation of a volume on the surface of a manifold relative to the volume in Euclidean space (see Appendix A). Unlike other ML works, in which continuous Ricci curvature is used to visualise high-dimensional loss landscapes (Li et al., 2018) or in novel characterisations of structural features (Chazal and Michel, 2017; Rieck et al., 2018), we use a measure of Ricci curvature within the discrete space of a graph. In order to do this, we relate Ricci curvature to optimal transport (Ollivier, 2009). Given a probability measure at each node, optimal transport can be formulated on a graph and Ricci curvature can be calculated. For a metric space (X, d) equipped with probability measure m_x for each $x \in X$, the Ollivier-Ricci curvature, κ, along the shortest path xy is given by Eq. (1), where $W(m_x, m_y)$ is the Wasserstein distance, and $d(x, y)$ is the path distance:

$$\kappa(x, y) = 1 - \frac{W(m_x, m_y)}{d(x, y)}. \tag{1}$$

Similar to Ni et al. (2019), we use a curvature-guided diffusion process, Ricci flow, to detect community structures within a network. The probability distribution used here includes further terms to estimate the computational demand of an individual node as well as its contribution to the overall network's robustness with respect to damage. In both works, the curvature evolves under discrete time intervals, Eq. (2).

$$w_{ij}^{k+1} = (1 - \kappa_{ij}^{(k)}) d^{(k)}(i, j). \tag{2}$$

Successful efforts in NAS may require months or even years of compute time. Zoph and Le (2017) demonstrated a computationally expensive process in which a RL controller parameterised a search within a high-dimensional search space. Resultant architectures match state-of-the-art performance. We optimise a combination of three hyperparameters for our search. Randomly wired networks offer a relatively unrestricted initial search space with good baseline performance. Xie et al. (2019) found a Watts-Strogatz graph generator (Watts and Strogatz, 1998) produced networks with the best performance, with $WS(K = 4, P = 0.75)$ and $N = 32$ (see Appendix B). Their simple graph-to-network mapping allowed a focus on wiring and structural features. The successful generator and straightforward mapping are both used here.

3. Method

Random computational graphs are generated using the Watts-Strogatz model. First, we use a controller neural network to predict a hyperparameter state. Second, we propose a node mass distribution based on local graph measures weighted by the predicted hyperparameters. Then, we use the process of Ricci flows to compute weights associated with each edge and prune the computational graph based on an edge threshold value. The pruned computational graph is mapped to a neural network and trained on the dataset. Accuracy and FLOPs per pass are combined to yield a reward then passed to the controller network, which is updated via a policy gradient method (see Appendix C). The code is publicly available at https://github.com/seglass5/RicciNets.

3.1 Mass Distribution

We calculate the curvature within a network using a hypothesised probability (mass) distribution as Eq. (3):

$$m_x^{\alpha,\beta,\gamma,\delta}(x_i) = \begin{cases} \alpha & \text{for } x = x_i \\ (1-\alpha)\left[\beta(\frac{1}{\text{Deg(x)}}) + \gamma(\text{Input(x)}) + \delta(\frac{\text{Output(x)}}{\text{Input(x)}})\right] & \text{for } x \in \pi(x) \\ 0 & \text{Otherwise.} \end{cases} \tag{3}$$

Input(x) defines the input degree of node x, Output(x) the output degree and Deg(x) the total degree. $\pi(x)$ defines the immediate neighbours of x. α gives the proportion of mass to remain on a node. β, γ and δ control the contribution of each of the three terms. The first term promotes a well-defined community structure, $\frac{1}{\text{Deg(x)}}$ yields a lower mass for a node with more neighbours. The second term promotes low input degree. Taking the transformation operation at a node to be of linear complexity in input degree, this approximates to promoting low computational burden at each node. The final term promotes a smaller ratio of output degree to input degree. A greater loss in accuracy is observed for removal of a node with high output degree, and an edge with low target node input degree (Xie et al., 2019). A smaller ratio of output degree to input degree is therefore taken to indicate better robustness with respect to graph damage. By requiring that the masses of a node and its neighbours sum to unity, this can be reduced to an equation in three hyperparameters.

3.2 Pruning Threshold

The weight associated to each edge in the graph is updated via Ricci flow from an initial value of zero, Eq. (2). The weights are normalised to prevent expansion to infinity and checked for convergence on each iteration. The process of Ricci flow ran for 50 iterations and typically reached convergence well within this limit. The threshold for pruning was set to the mean of all the weights in the network following Ricci flow. Hyperparameter selection can alter the skewness of the distributions of curvatures and weights, and adjusting the distribution of weights is interpreted as the controller network learning a definition of saliency; if very few paths can be considered salient, parameter prediction can lead to negative skewness in the weight distribution and more edges are removed. Similarly, if the drop in accuracy is too high for removing a group of paths, a set of hyperparameters that adjusts the mean to save this group can be learnt.

3.3 Controller

An auxiliary controller network generates hyperparameter states. The controller is implemented as a simple feed forward network. The parameter states are one-hot encoded and discretised in the range $[0, 1]$. The output parameters, $[\alpha, \beta, \delta]$ are passed to the pruning function. The controller seeks to maximise its expected reward, Eq. (4), where $J(\theta_c)$ indicates the expected reward at a parameter state θ_c. $a_{1:T}$ represents the list of actions (possible hyperparameter combinations) for T hyperparameters,

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]. \tag{4}$$

Since the reward signal is non-differentiable, we use a policy gradient method to iteratively update θ_c. We use the REINFORCE rule (Williams, 1992), Eq. (5).

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}[\nabla_{\theta_c} log(P(a_t|a_{(t-1):1}; \theta_c))R]. \tag{5}$$

An empirical approximation of the above quantity is given in Eq. (6). m is the number of parameter states sampled in one batch by the controller. The reward that the network in the k^{th} parameter state achieves after training is R_k,

$$\frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} [\nabla_{\theta_c} log(P(a_t|a_{(t-1):1}; \theta_c))R_k]. \tag{6}$$

The reward used to update policy is given in Eq. (7). The top one accuracy, A, is regularised by the FLOPs per pass of the network F using a regularisation parameter μ.

$$J(\theta_c) = A - \mu \frac{F}{F_{baseline}}. \tag{7}$$

Episode rewards are discounted according to Eq. (8), where $v(\theta_c)$ is the episode reward for a state θ_c, γ a discount parameter, and k the number of iterations within an episode. This encourages prolonged episodes.

$$v(\theta_c) = \sum_{k=0}^{N} \gamma^k J(\theta_{c+k}). \tag{8}$$

4. Experiments and Results

Experimentation is based on the classification of images from the CIFAR-10 dataset, with 50,000 training images and 10,000 test images. The images are batched in groups of 64. Each network is trained for 4 epochs, and the policy gradient controller ran over 20 episodes, with episodes batched in pairs to update policy.

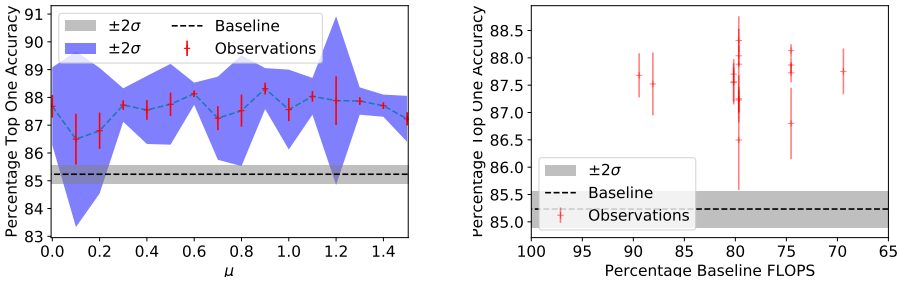
4.1 Evaluation Procedure

Network performance is evaluated using the top-one accuracy and the number of FLOPs per pass of the network produced. Performance is measured in relation to a baseline set by

an unpruned network produced using the same generator parameters. To assess the importance of considering the topology of randomly wired networks in the pruning procedure, we compare *RicciNets* against pruning weights by lowest magnitude (Zhu and Gupta, 2017). The combination of hyperparameters learnt for a given graph is applied to other graphs of the same random graph generator. We note that while this implementation of randomly wired neural networks yielded an accuracy of $91.5 \pm 0.2\%$ on CIFAR-10 over 100 training epochs, we only report results achieved after 4 epochs owing to resource constraints.

4.2 Results

Fig. 1 (a) shows the variation in top-one accuracy of the resultant networks for regularisation parameter μ in the range [0, 1.5]. The methodology produces better-than-baseline performance under compression across the range of μ, with a small variance in top one accuracy ($\sigma^2 = 0.21$). (b) shows the top-one accuracy of architectures against the FLOPs per pass expressed as a percentage of baseline. All networks produced operated using $68 - 91\%$ of the baseline FLOPs per pass. For small networks, more severe compression would result in a chain-like structure and a sharp drop off in accuracy, which would be discouraged by the controller (see Appendix D). An exploratory step carried out with probability p within the controller, or further fine-tuning of the policy gradient network could result in a larger range of compression. *RicciNets* demonstrates a restricted range of compression when compared to pruning via lowest-magnitude weights. Within this range, however, the networks produced by *RicciNets* demonstrate better performance than those pruned by weight, Table 1. Future work includes incorporating more control over the level of compression via alternative ways to regularize the reward objective.



(a) Accuracy against μ. Pruned networks display better-than-baseline performance, with almost 3% increase in accuracy on baseline when averaged across the range of μ.

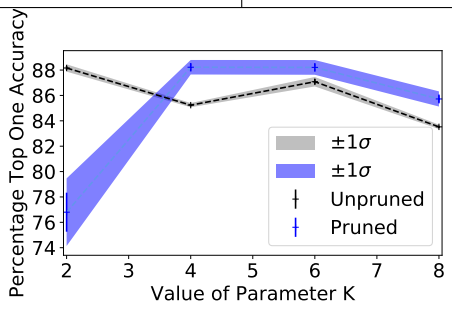
(b) Accuracy gainst FLOPs per pass of pruned network as a percentage of baseline. Overlapping errorbars appear darker.

Figure 1: The pruned networks show no degradation in performance under compression.

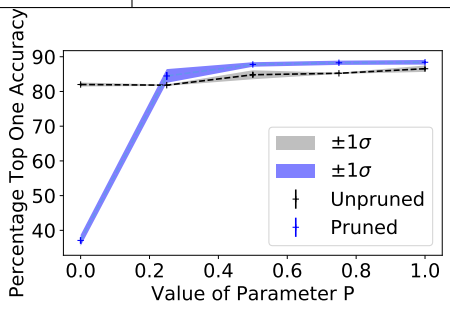
The combination of hyperparameters that produced the greatest accuracy using $WS(4, 0.75)$ generalised to other Watts-Strogatz graphs. Pruned networks generated with different K and P displayed an increase in performance and moderate compression, Fig. 2. *RicciNets* maintained the salient computational paths identified in the learnt case.

Table 1: Average top one accuracy and average percentage baseline weights remaining after pruning for *RicciNets*, pruning via lowest magnitude weights and baseline. Average taken within the $40-50\%$ range of weights remaining.

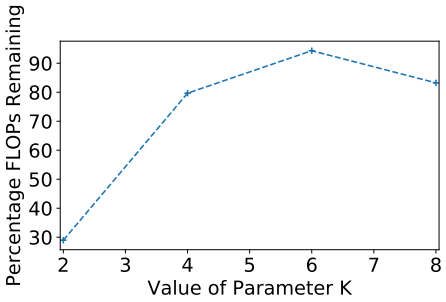
Pruning	**Top One Accuracy (%)**	**Weights Remaining (%)**
RicciNets	87.59 ± 0.11	41.90 ± 0.47
Lowest Magnitude	84.77 ± 0.55	45.00 ± 3.54
Baseline	85.23 ± 0.09	100.00



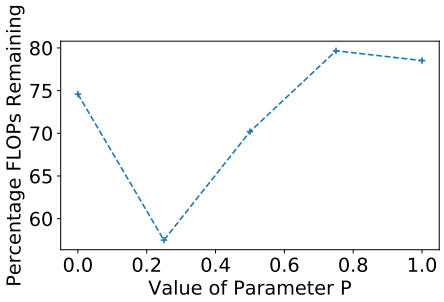
(a) Top one accuracy against the value of parameter K in $WS(K,P)$, with $P = 0.75$ and $N = 32$.



(b) Top one accuracy against the value of parameter P in $WS(K,P)$, with $K = 4$ and $N = 32$.



(c) FLOPs per pass through the network against parameter value K in $WS(K,P)$, with $P = 0.75$ and $N = 32$.



(d) FLOPs per pass through the network against parameter value P in $WS(K,P)$, with $K = 4$ and $N = 32$.

Figure 2: Pruning WS graphs without specific optimisation showed no drop in performance.

5. Conclusions

Our model combines the principle of curvature with ML to carry out neural architecture search. It successfully identifies salient computational paths, and demonstrates a reduction in computational cost for no degradation in baseline performance. It outperforms pruning via lowest-magnitude weights on randomly wired neural networks. A combination of hyperparameters learnt with a given network generalises to others from the same generator with no specific optimisation, offering compression for no drop in performance. The results obtained suggest a successful novel methodology for compact NAS, and are the first on the compression dynamics of randomly wired neural networks. Future work will develop more comparative methods against other pruning procedures, and investigate off-policy controller algorithms.

References

Danielle S Bassett and Olaf Sporns. Network neuroscience. *Nature neuroscience*, 20(3):353, 2017.

Dan Knopf Bennett Chow. *The Ricci Flow: An Introduction (Mathematical Surveys and Monographs)*. Mathematical Surveys and Monographs. American Mathematical Society, 2004. ISBN 0821835157,9780821835159.

Simon Brendle. *Ricci Flow and the Sphere Theorem*. Graduate Studies in Mathematics 111. American Mathematical Society, 2010. ISBN 0821849387, 9780821849385.

Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, Mar 2009. ISSN 1471-0048. doi: 10.1038/nrn2575. URL https://doi.org/10.1038/nrn2575.

Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: fundamental and practical aspects for data scientists. *arXiv preprint arXiv:1710.04019*, 2017.

Anshul Choudhary, John F. Lindner, Elliott G. Holliday, Scott T. Miller, Sudeshna Sinha, and William L. Ditto. Physics-enhanced neural networks learn order and chaos. *Phys. Rev. E*, 101:062207, Jun 2020. doi: 10.1103/PhysRevE.101.062207. URL https://link.aps.org/doi/10.1103/PhysRevE.101.062207.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

Carlo Sinestrari Gang Tian (auth.) Riccardo Benedetti Carlo Mantegazza (eds.) Michel Boileau, Gerard Besson. *Ricci Flow and Geometric Applications: Cetraro, Italy 2010*. Lecture Notes in Mathematics 2166. Springer International Publishing, 1 edition, 2016. ISBN 978-3-319-42350-0,978-3-319-42351-7.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Chien-Chun Ni, Yu-Yao Lin, Feng Luo, and Jie Gao. Community detection on networks with ricci flow. *Scientific Reports*, 9(1):9984, 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-46380-9. URL https://doi.org/10.1038/s41598-019-46380-9.

Yann Ollivier. Ricci curvature of markov chains on metric spaces. *Journal of Functional Analysis*, 256(3):810 – 864, 2009. ISSN 0022-1236. doi: https://doi.org/10.1016/j.jfa.2008.11.001. URL http://www.sciencedirect.com/science/article/pii/S002212360800493X.

Bastian Rieck, Matteo Togninalli, Christian Bock, Michael Moor, Max Horn, Thomas Gumbsch, and Karsten Borgwardt. Neural persistence: A complexity measure for deep neural networks using algebraic topology. *arXiv preprint arXiv:1812.09764*, 2018.

Sebastien Boucksom Philippe Eyssidieux Vincent Guedj (eds.) Sébastien Boucksom, Philippe Eyssidieux (auth.). *An Introduction to the Kähler-Ricci Flow*. Lecture Notes in Mathematics 2086. Springer International Publishing, 1 edition, 2013. ISBN 978-3-319-00818-9,978-3-319-00819-6.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

Xianfeng David Gu (auth.) Wei Zeng. *Ricci Flow for Shape Analysis and Surface Registration: Theories, Algorithms and Applications*. SpringerBriefs in Mathematics. Springer-Verlag New York, 1 edition, 2013. ISBN 978-1-4614-8780-7,978-1-4614-8781-4.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1284–1293, 2019.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. Technical report, Google, 2017. URL `https://arxiv.org/abs/1710.01878`.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017. URL `https://arxiv.org/abs/1611.01578`.

Appendix A. Ricci Curvature

The notion of curvature was introduced by Gauss and Riemann over 190 years ago; it is a measure of how space is curved at a point in that space. Given an n-dimensional manifold, which we define as a space that locally looks n-dimensional, we may form a Riemannian metric; this assigns each tangent space of the manifold a Euclidean metric, which in turn gives the "standard" distance between any two vectors in the space. A manifold together with its corresponding Riemannian metrics forms a Riemannian manifold (Ni et al., 2019).

For a surface S, the Gaussian map from S to the unit sphere sends a point on S to the unit normal vector of S at p, a point on the unit sphere. The Gaussian curvature of the surface at a point p is the Jacobian of the Gaussian map at p, the signed area distortion of the Gaussian map at p. Hence, the plane has zero curvature, the sphere has positive curvature, and the hyperboloid of one sheet has negative curvature. The curvature depends only on the induced Riemannian metric on the surface and does not depend on how the surface is embedded in space.

Riemann generalised Gaussian curvature to higher dimensions. For a Riemannian manifold *(M,g)*, the sectional curvature assigns each 2-dimensional linear subspace P in the tangent space of M at p a scalar, the Riemannian sectional curvature. The scalar is equal to the curvature of the image of P under the exponential map. A positively curved space tends to have small diameter and is geometrically crowded; a sphere, for example. Conversely, a negatively curved space is geometrically spreading out.

The Ricci curvature assigns each unit tangent vector v at a point p a scalar which is the average of the sectional curvatures of planes containing v.

There have been various approaches to generalize the concept of curvature to non-manifold spaces. Here, we look to assign curvature to a graph, *G(V, E, w)*, with vertices V, edges E and edge weights w. Ollivier-Ricci curvature Ollivier (2009) relates Ricci curvature to optimal transport, allowing a mapping to discrete spaces. Given a probability measure at each point, optimal transport can be formulated on general metric spaces and may be used to define Ricci curvature on a network with edge weights and probability measures at each vertex.

Appendix B. Watts-Strogatz Random Graph Generator

The Watts-Strogatz method demonstrated the most success within Xie et al. (2019). This operates by first placing N nodes regularly in a ring, with each node connected to its $K/2$ neighbours on both sides, where 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 $1 \leq i \leq K/2$. K and P are the only two parameters of the Watts-Strogatz model. Any graph generated by a state (K, P) has exactly $N \cdot K$ edges. (K, P) only covers a small subset of all possible N-node graphs, and a different subset than that covered by other random graph generators with equal N. Random graph generators present a relatively unrestricted initial search space, but a prior is introduced in the choice of random graph generator. Watts-Strogatz graphs display small

world properties; the typical distance between randomly chosen nodes is proportional to $log(N)$.

Appendix C. Method Overview

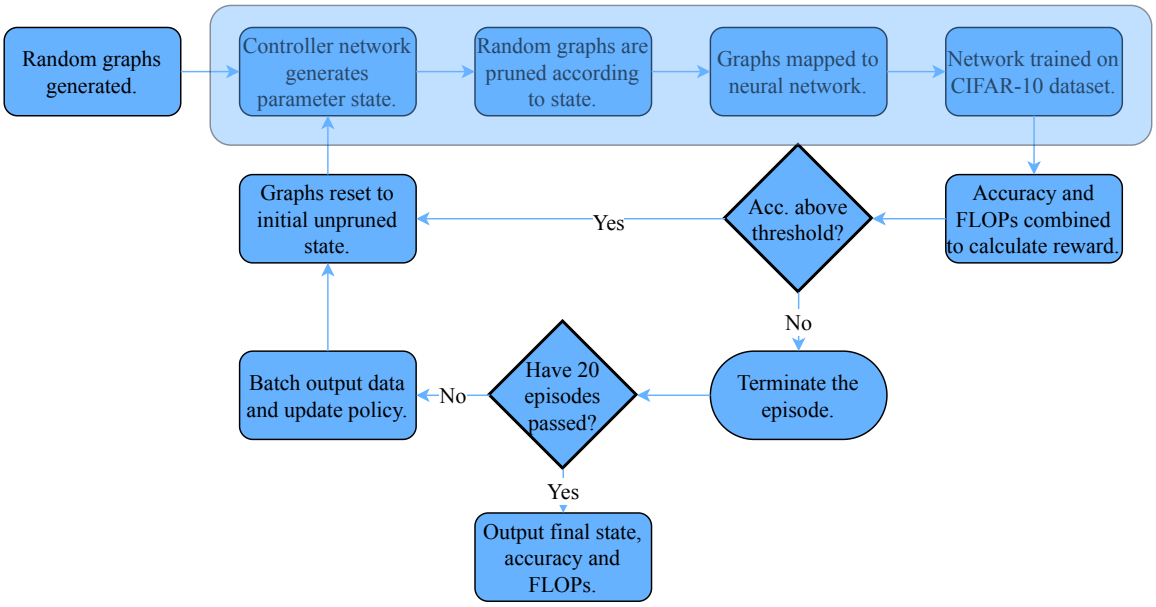


Figure 3: An overview of the methodology. Pruning takes place before the graph-to-network mapping. The pale blue box indicates a single step within a policy gradient episode.

Appendix D. Extensive Pruning

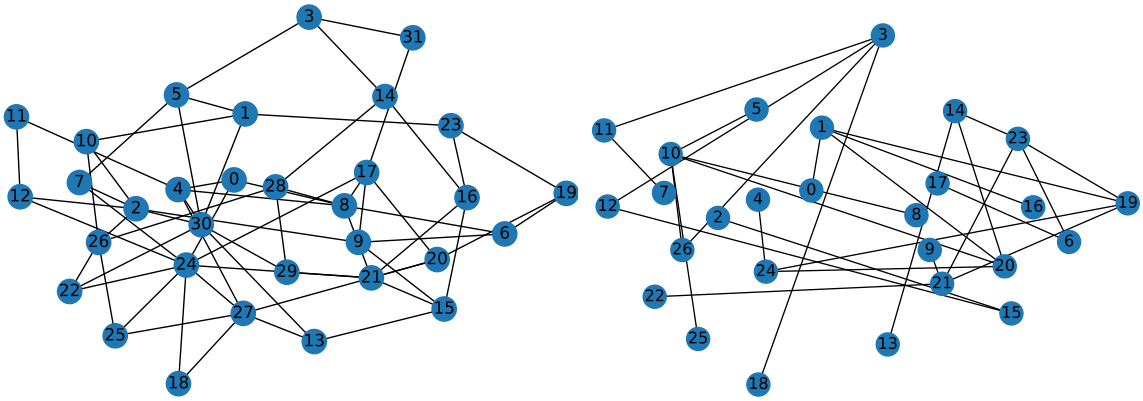


Figure 4: The unpruned state of a graph, $WS(4, 0.75)$ with $N = 32$, left, and the same graph following pruning under a selected combination of hyperparameters, right. We observe increased sparsity whilst still retaining some clustering and skipped connections. Since both nodes and paths are removed, the node labels do not carry over from the unpruned to the pruned state, and are shown here for the purposes of information flow; in both cases data is carried in the direction of increasing node label.

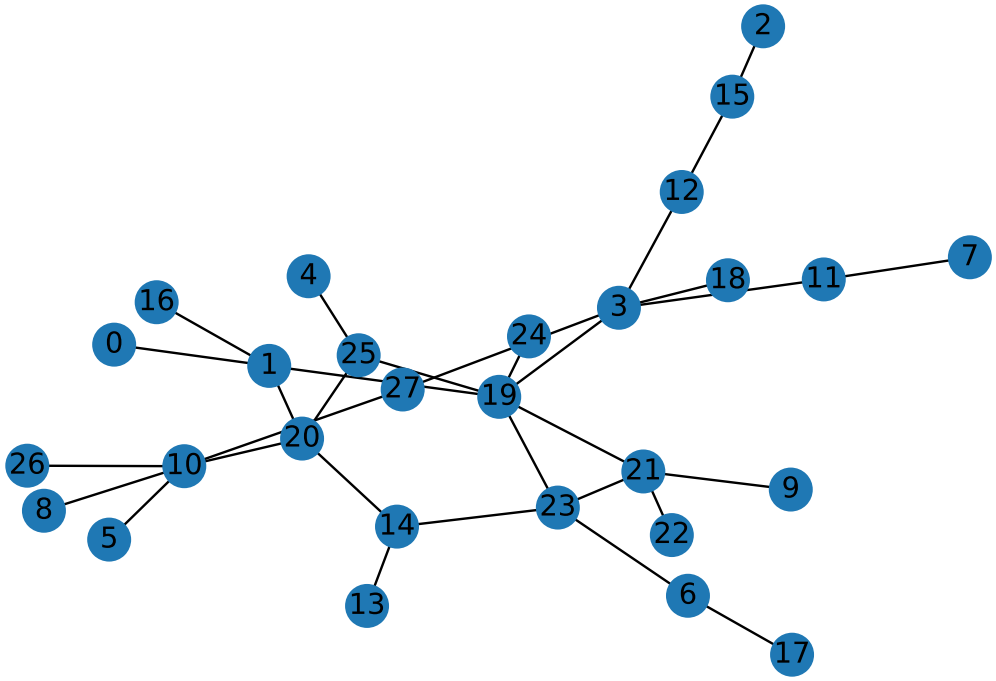


Figure 5: Sparser, chain-like architectures typically give a lower top one accuracy and so are discouraged by the controller network.