Pruning Increases Orderedness in Weight-Tied Recurrent Computation

Yiding Song [*1]

Abstract

Inspired by the prevalence of recurrent circuits in biological brains, we investigate the degree to which directionality is a helpful inductive bias for artificial neural networks. Taking directionality as topologically-ordered information flow between neurons, we formalise a perceptron layer with all-to-all connections (mathematically equivalent to a weight-tied recurrent neural network) and demonstrate that directionality, a hallmark of modern feed-forward networks, can be *induced* rather than hard-wired by applying appropriate pruning techniques. Across different random seeds our pruning schemes successfully induce greater topological ordering in information flow between neurons without compromising performance, suggesting that directionality is *not* a prerequisite for learning, but may be an advantageous inductive bias discoverable by gradient descent and sparsification.

1. Introduction

Neuronal organisation in the biological brain is far less rigid than modern feedforward neural networks. Dynamic synaptic growth and pruning occur early in human Prefrontal Cortex development (Kolk & Rakic, 2022); recurrent computation has been shown to be important in macaque decision-making (Mante et al., 2013); and in contrast to the feedforward architectures of standard vision models, biological vision utilises recurrence extensively (van Bergen & Kriegeskorte, 2020).

In light of the lack of inherent directionality in many biological computations, this paper seeks to investigate the necessity of directionality (sometimes referred to in biology as *hierarchical organisation*), i.e. that information flow between neurons follows an inherent topological ordering, as a helpful inductive bias within artificial perceptron layers. Starting from an *all-to-all* connected architecture where every neuron is connected to every other, we examine whether with the right initialisation and pruning techniques, the resulting network can be finetuned via gradient descent to regain a topological ordering in information flow between neurons.

We make three main contributions within this paper: (i) Formalise the idea of a directionless 'complete perceptron' layer similar in form to weight-tied recurrent neural networks; (ii) Introduce a metric to measure topological ordering of information flow in terms of the weight matrix of the complete perceptron layer; and (iii) Investigate various initialisation and pruning methods for inducing such a topological ordering.

2. Related Work

Past works have explored how modularity and hierarchical organisation arise within information processing networks. Mengistu et al. (2016) demonstrated that networks with a connection cost tend to evolve to process information in a hierarchical way. Liu et al. (2023) used a similar idea to develop Brain-Inspired Modular Training, showing that regularising by connection cost increases modularity and interpretability in neural networks. However, these works mostly focused on feedforward architectures and did not investigate the impact of connection cost or sparsity on directionality.

Another related body of literature is architecture search via pruning (Stothers, 2019; Li et al., 2022); to the best of our knowledge, these works focus on feedforward architectures only. The Lottery Ticket Hypothesis (Frankle & Carbin, 2018; Malach et al., 2020) demonstrated that well-designed pruning can uncover high-performing subnetworks prior to training, motivating our exploration of pruning as an important factor that influences network behaviour in learning.

[*]Equal contribution [1]Harvard College, Cambridge, MA, USA. Correspondence to: Yiding Song <yidingsong@college.harvard.edu>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

Architecture-wise, the idea of building models with all-to-all neuronal connections has been found in earlier Boltzmann Machines (Ackley et al., 1985) and Hopfield Networks (Hopfield, 1982); the difference is that our analysis focuses on optimisation via gradient descent and does not minimise an explicit energy function, aligning more closely with traditional Multi-Layer Perceptrons (MLPs) than energy-based networks.

3. Methods

3.1. The Complete Perceptron Layer

To empirically verify the benefit of information directionality as an inductive bias, we adapt the standard perceptron layer into a directionless *complete perceptron layer* (analogous to the idea of a complete graph), where each neuron is connected to every other neuron within the layer. Formally, a complete perceptron layer consists of o output units, h hidden units, and i input units; we use the convention that the first o neurons are designated as output neurons, the next h hidden neurons, and the final i input neurons. We initialise a weights matrix $\mathbf{W} \in \mathbb{R}^{(o+h)\times(o+h+i)}$ (the value $\mathbf{W}_{ij}$ at row i and column j is the strength of the connection from neuron j to neuron i), a values vector $\boldsymbol{v} \in \mathbb{R}^{(o+h)}$ (initialises the value of each of the $o+h$ output and hidden neurons), and an optional bias vector $\boldsymbol{b} \in \mathbb{R}^{(o+h)}$ (the value $\boldsymbol{b}_i$ at index i is the bias of the ith neuron).

Because a complete perceptron layer is all-to-all connected, in order for its weights to learn complex, non-linear relationships, we run it for multiple iterations, replacing spatial depth with temporal depth. Note that the input neurons are clamped and their values remain unchanged. The computation for a complete perceptron layer is given in Algorithm 1. At a high level, after initialising $\mathbf{W}$, $\boldsymbol{v}$, and $\boldsymbol{b}$, we evolve the hidden state $\boldsymbol{s}$ for a fixed number of iterations T, with:

$$\boldsymbol{s}^{(t+1)} = \sigma\left(\left[\boldsymbol{s}^{(t)}\ \boldsymbol{x}\right]\mathbf{W}^\top + \boldsymbol{b}\right), \qquad t = 0, \ldots, T-1, \tag{1}$$

where $\boldsymbol{s}^{(t)}$ is the hidden state at timestep t, $[\cdot\ \cdot]$ denotes concatenation along the feature axis (as opposed to the batch axis). Unless otherwise stated, σ is the sigmoid function and the initial state $s^{(0)}$ is sampled from a random normal distribution. After T iterations, the layer outputs $\boldsymbol{y} = \boldsymbol{s}^{(T)}_{[:,\,1:o]}$, the values of the first o neurons along each feature axis.

3.2. Connections to Existing Architectures

Equation (1) is in fact equivalent to a recurrent neural network with constant inputs and weights across layers:

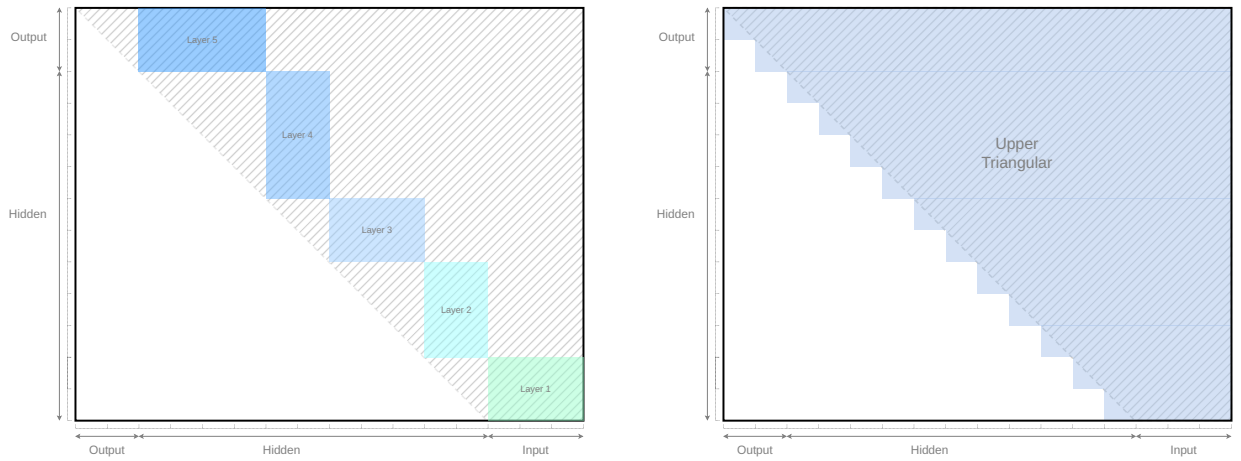
$$\boldsymbol{s}^{(t+1)} = \sigma\left(\boldsymbol{s}^{(t)}\mathbf{W_s}^\top + \boldsymbol{x}\mathbf{W_x}^\top + \boldsymbol{b}\right) \tag{2}$$

where $\mathbf{W_s} = \mathbf{W}_{[:,1:o+h]}$ are the first $o+h$ columns of $\mathbf{W}$ and $\mathbf{W_x} = \mathbf{W}_{[:,o+h+1:o+h+i]}$ are the last i columns of $\mathbf{W}$, such that $\mathbf{W} = [\mathbf{W_s} \mid \mathbf{W_x}]$. This lends a nice perspective to interpret complete perceptron layers as weight-tied recurrent networks with a constant input, grounding our results in the context of a more commonly studied architecture.

Nevertheless, we choose to define the network using Equation (1) because it is easier to interpret for the questions posed in this paper. Note that at each timestep, each hidden unit i performs a weighted sum (followed by some activation function) over its inputs, with the weights defined by row i in the weights matrix $\mathbf{W}$. Therefore, if the weights of a complete perceptron layer were to simulate a perfectly feedforward MLP when executed, then $\mathbf{W}$ would consist of rectangular blocks above the main diagonal (see Figure 1a). Each block corresponds to the weights of one layer in the MLP attending to the outputs of the previous layer; the first layer only attends to the inputs. In the case of Figure 1a, if the complete perceptron layer were to be computed over 5 iterations, it would simulate the 5-layer MLP whose layer weights are contained in each block.

No such block structure emerged from the gradient descent and pruning methods experimented in this paper. Therefore we generalise the notion of layer-wise feedforward directionality to a *topological ordering* of information flow between neurons: that is, earlier neurons only feed information into later neurons, and information does not flow back. On the weights matrix, this corresponds to an upper triangular matrix,[1] because later neurons are allowed to attend to the outputs of earlier ones, but not vice versa. Such a weights matrix shares a similar pattern of information flow to an MLP with only one neuron in each layer and many weighted, feedforward residual connections.

[1] Here we will allow values on the diagonal, as we choose to permit a neuron to communicate with itself.



(a) Weights perfectly simulating a normal multilayer perceptron.

(b) Weights with perfect topological ordering.

Figure 1. Visualisation of the weights matrix of a complete perceptron layer for two different cases: perfect simulation of a standard multi-layer perceptron and perfect topological ordering.

3.3. Orderedness

Because the indexing of the hidden units within a weights matrix is arbitrary, we cannot simply measure the 'orderedness' of a weights matrix by comparing it with its upper-triangular equivalent; we must take into account all the possible indexing of the hidden units. Therefore, to establish a good metric for the *orderedness* of a network, we wish to find the permutation of hidden units such that least of the weights are included in the lower triangular matrix. This reduces 'back-flow' of information and minimises the weights that would need to be discarded to achieve perfect topological ordering.

More formally, denote by Π the set of all permutations on the order of the hidden neurons on the weight matrix $\mathbf{W}$. Define by $L(\mathbf{W})$ the sum of the lower-triangular values in matrix $\mathbf{W}$ (excluding the diagonal), and by $S(\mathbf{W})$ the sum of all the values in matrix $\mathbf{W}$. Additionally, let $\mathbf{W}^{\text{abs}}$ take the absolute value of all elements in matrix $\mathbf{W}$. Then, the *orderedness* O of the weight matrix $\mathbf{W}$ is defined as:

$$O(\mathbf{W}) = 1 - \left(\min_{\pi \in \Pi} \frac{L(\pi(\mathbf{W}^{\text{abs}}))}{S(\pi(\mathbf{W}^{\text{abs}}))} \right) \tag{3}$$

The metric $O(\mathbf{W})$ measures the proportion of weights that flow forward in the complete perceptron layer when the hidden units are organised in such as way as to minimise feedback edges. This quantity is 1 in the limit of a perfectly topologically-ordered feedforward network.

3.4. Initialisation and Pruning

We evaluate the emergence of orderedness under a set of initialisation schemes for the weights and values, as well as a suite of pruning operations that act *directly* on the weight matrix W during training. The full quantitative effects are reported in Table 1; the details of each method are explained in the Appendix (Section B).

Empirically, we found that simply training the complete perceptron layer on synthetic tasks is not sufficient to induce greater orderedness. Inspired by the work of Mengistu et al. (2016); Liu et al. (2023) investigating the relationship between connection cost and modularity, we were motivated to see whether enforcing sparsity could encourage the model to be more economical with its high-magnitude weights, and whether that would lead to increased orderedness for efficient computation.

4. Experiments

Because of the limitations of compute and the low scalability of the complete perceptron layer, we chose to use very simple tasks for our experiments. Nevertheless, the point of interest here is the learning dynamics of these complete networks. We used two datasets for our experiments: XOR and Sine. For the XOR task, the input is two binary numbers and the

Table 1. Quantitative effects of various initialisation and pruning methods on orderedness. All data is reported averaged across 10 seeds, with standard deviation after the ± sign.
(a) Effect of various initialisation methods on the change in orderedness pre- and post-training (ΔO). By default, random normal initialisation is applied on the weights matrix and values vector. Initial orderedness pre-training (O) is provided for context.

Initialisation	O **(Untrained)**	ΔO **(XOR)**	ΔO **(Sine)**
Default	0.672 ± 0.050	0.032 ± 0.047	0.023 ± 0.033
Random Uniform ($\mathbf{W}$)	0.647 ± 0.045	0.075 ± 0.057	0.076 ± 0.039
Zeros ($\boldsymbol{v}$)	0.680 ± 0.070	0.049 ± 0.078	-0.004 ± 0.030

(b) Effect of various pruning methods on change in orderedness pre- and post-training (ΔO). The change in orderedness without training is provided as a control. We prune the weights matrix only.

Pruning (on $\mathbf{W}$)	ΔO **(Untrained)**	ΔO **(XOR)**	ΔO **(Sine)**
Default: none	0.000 ± 0.000	0.032 ± 0.047	0.023 ± 0.033
Random (p=0.5)	0.170 ± 0.062	0.243 ± 0.082	0.104 ± 0.047
Top-K (k=0.5)	0.039 ± 0.032	0.116 ± 0.060	0.086 ± 0.037
Dyn. Top-K (k=0.5)	0.038 ± 0.025	0.149 ± 0.073	0.086 ± 0.037
Tril-damp (f=0.8)	0.328 ± 0.050	0.280 ± 0.045	0.343 ± 0.026
Dyn. Tril-damp (f=0.8)	0.315 ± 0.048	0.280 ± 0.045	0.343 ± 0.026

output is the binary XOR of the two inputs. For the Sine task, the input is two real numbers $a, b, \in [0, 3]$ and the output is $(\sin(a) + \sin(b))/2$ (we chose the range so that all outputs are positive, in order to suit the Sigmoid activation function).

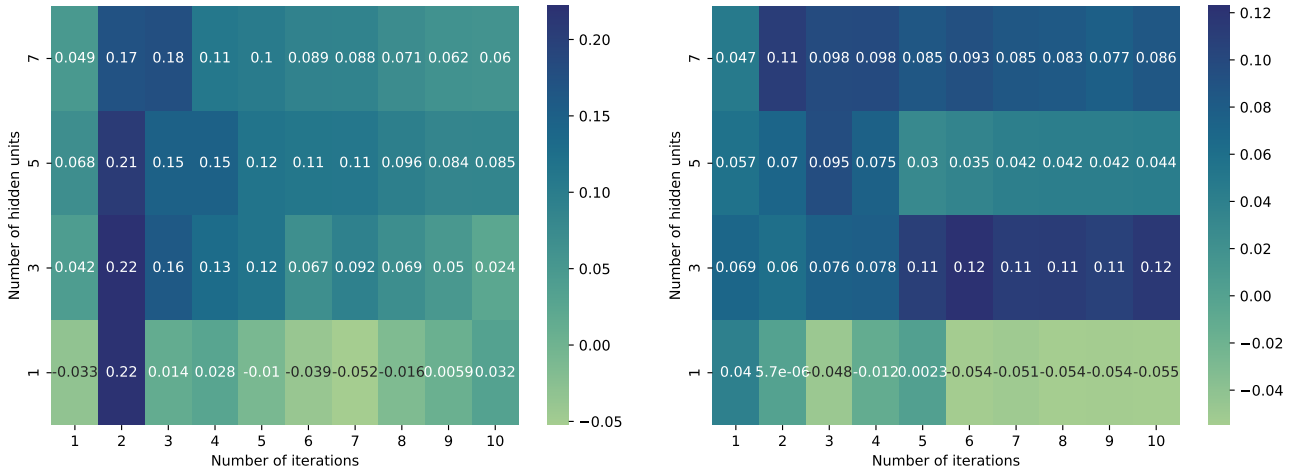
The standard hyperparameters used for the complete perceptron layer in the default control setup are presented in the Appendix (Section C). To investigate the emergence of orderedness beyond the default control setup, we varied the way we initialised values and weights and experimented with different pruning methods. Additionally, we tried to establish a relationship between the number of hidden units within the complete layer, the number of iterations we evolve it for, and the resulting orderedness of the weights. For pruning, we also investigated how the desired sparsity levels affected orderedness.

5. Results and Discussion

A graph of the training losses of the complete perceptron layer on both tasks is shown in Appendix (Section D), presented in comparison to the training losses of traditional MLPs. In both cases, loss descends smoothly despite the absence of pre-imposed directionality, establishing the proof-of-concept that a complete perceptron architecture is trainable.

Pruning Correlates with Orderedness. Table 1 displays the resulting orderedness of different initialisation and pruning techniques. Initialisation does not appear to play a huge role in increasing orderedness, whilst various pruning methods see more success. It is expected that Tril-Damp methods (which encourage an upper-triangular weights matrix) lead to a large increase in the orderedness of networks with their explicit inductive bias. More surprising is that the two variants of Top-K pruning, which select for weights with the largest absolute value and are agnostic to orderedness, lead to an increase in orderedness without the explicit need for any other inductive bias. The much lower distribution of values of ΔO from an untrained network indicates that this observation is unlikely to be due to the nature of Top-K sparsification itself, suggesting that under correct pruning techniques, orderedness can be induced naturally. Random pruning works surprisingly well, but this is more so an artefact of the mathematical definition of orderedness, because even on an untrained network, random pruning leads to an increase in orderedness.

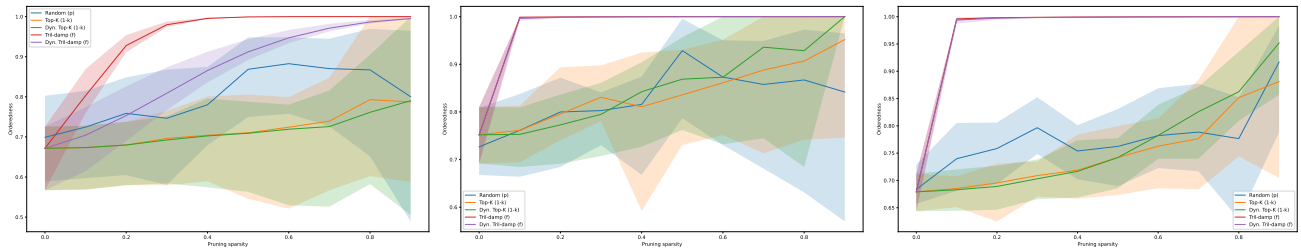
Effect of Hidden State Size and Evolution Iterations. As the number of hidden units is increased, the expressive potential of the complete perceptron layer also increases. Therefore, it may be expected that more complicated layering structures can emerge from such networks, provided that the number of evolution iterations also increases. However, as shown in Figure 2, this simple prediction is not the case. Based on the data, the relationship between the number of hidden states, evolution iterations, and orderedness is still unclear. However, one striking feature that emerges is the 'spike' of orderedness when the network is evolved for 2 iterations only in the XOR task, across all different numbers of hidden units. One possible explanation is that a 2-iteration complete perceptron is structurally highly directional: in the first iteration, the values of the inputs propagate throughout the network; and in the next iteration, the output is extracted. Therefore, any



(a) Mean ΔO for the XOR task.

(b) Mean ΔO for the Sine task.

Figure 2. Change in orderedness of the weights matrix after training (ΔO) as a function of both hidden units and number of iterations for the XOR and Sine tasks. Here ΔO is reported as the mean across 10 seeds. Dynamic Top-K (with $k = 0.5$) was used for pruning.



(a) Relationship between pruning sparsity and orderedness (untrained).

(b) Relationship between pruning sparsity and orderedness (XOR).

(c) Relationship between pruning sparsity and orderedness (Sine).

Figure 3. The relationship between target pruning sparsity and orderedness for the XOR and Sine tasks (the measure of increasing sparsity in terms of pruning coefficients is enclosed by brackets in the legend). The equivalent plots for an untrained network are shown for control. Note that since the pruning logic was evolved for only 10 steps on the untrained network, Tril-damping methods were less efficient at damping lower-triangular values at the same sparsity level than the networks trained on XOR and Sine tasks; this difference is therefore not indicative of anything else other than our experimental setup.

feedback from one hidden unit to another is meaningless, because it either does not utilise the inputs or will never be fed to the output units; at least 3 iterations are required for the input to propagate into the hidden units, be fed back amongst its hidden neurons, and subsequently be used to compute the output.

Sparsity and Orderedness. Figure 3 gives a closer analysis of how pruning correlates with orderedness. There does appear to be some trend for both tasks, where orderedness increases with sparsity for Top-K methods on top of the levels predicted by the untrained control. However, the variance of the data is too large for any substantial conclusions to be drawn. More theoretical and empirical work is required to understand precisely how sparsity affects orderedness.

6. Conclusion

We introduce the complete perceptron layer and *orderedness* as a metric for directionality in neural network computation. Through experiments on the complete perceptron layer with two synthetic tasks, we show that orderedness can be induced rather than hard-wired by applying appropriate pruning techniques in conjunction with gradient descent.

Software and Data

The code to reproduce this work is available at the following GitHub repository: https://github.com/PerceptronV/orderedness-by-pruning.

Acknowledgements

We are grateful to Kazuki Irie and Hanming Ye for helpful, spontaneous discussions. We also thank Gabriel Kreiman and Giordano Ramos-Traslosheros for their comments and, respectively, teaching and TA-ing the course that inspired this work.

References

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1): 147–169, 1985.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kolk, S. M. and Rakic, P. Development of prefrontal cortex. *Neuropsychopharmacology*, 47(1):41–57, 2022.

Li, Y., Zhao, P., Yuan, G., Lin, X., Wang, Y., and Chen, X. Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization. *arXiv preprint arXiv:2206.01198*, 2022.

Liu, Z., Gan, E., and Tegmark, M. Seeing is believing: Brain-inspired modular training for mechanistic interpretability. *Entropy*, 26(1):41, 2023.

Malach, E., Yehudai, G., Shalev-Schwartz, S., and Shamir, O. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pp. 6682–6691. PMLR, 2020.

Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78–84, 2013.

Mengistu, H., Huizinga, J., Mouret, J.-B., and Clune, J. The evolutionary origins of hierarchy. *PLoS computational biology*, 12(6):e1004829, 2016.

Stothers, D. B. Turing's child machine: A deep learning model of neural development. 2019.

van Bergen, R. S. and Kriegeskorte, N. Going in circles is the way forward: the role of recurrence in visual inference. *Current Opinion in Neurobiology*, 65:176–193, 2020.

A. Algorithmic Specification of a Complete Perceptron Layer

Algorithm 1 Forward Computation of a complete perceptron layer

Require: Input $\boldsymbol{x} \in \mathbb{R}^{B \times i}$, parameters $(\mathbf{W}, \boldsymbol{v}, \boldsymbol{b})$, iterations T, element-wise activation function σ

$\boldsymbol{s} \leftarrow \text{repeat}(\boldsymbol{v}, B)$ {stack one copy of $\boldsymbol{v}$ for each batch}
for $t = 1$ to T **do**
  $\boldsymbol{h} \leftarrow [\boldsymbol{s}\ \boldsymbol{x}]$ {append clamped input}
  $\boldsymbol{s} \leftarrow \sigma\big(\boldsymbol{h}\,\mathbf{W}^\top + \boldsymbol{b}\big)$ {let information propagate through network}
end for
Return $\boldsymbol{s}_{[:,1:o]}$ {first o units constitute the layer output}

B. Initialisation and Pruning Strategies

Initialisation schemes. *Random Normal* samples elements from $\mathcal{N}(0, 1)$; it is the default initialisation method for the weights matrix $\mathbf{W}$ and values vector $\boldsymbol{v}$. *Random Uniform* samples elements from $\mathcal{U}(0, 1)$; it helps establish whether the lack of negative initialisation affects orderedness. *Zeros* sets every element to be 0.

Pruning operations. *Random Prune* independently zeros each weight with probability p. *Top-K Prune* keeps the top-k fraction of weights by absolute value and zeros the rest. *Dynamic Top-K* does not aggressively select for the top-k fraction right from the start, but instead begins by keeping a much larger fraction k' of the weights, with k' gradually approaching k as training progresses. The function we use is the following:

$$k'(x) = 1 - \big(1 - k\big)\sin^4\Big(\tfrac{\pi}{2}x\Big), \quad x \in [0, 1], \tag{4}$$

where x is the training progress. *Tril-Damping* subtracts a fixed fraction of each value from the lower triangle of the weight matrix:

$$\mathbf{W} \leftarrow \mathbf{W} - f\,\mathrm{tril}(\mathbf{W}, -1) \tag{5}$$

where f is a factor between 0 and 1. *Dynamic Tril-Damping* follows a similar pattern to Dynamic Top-K pruning, but instead it starts with a small factor f' so that the lower triangle is not aggressively dampened at the start of training, with f' approaching f as training progresses. The function we use is the following:

$$f'(x) = f\,\sin^4\Big(\tfrac{\pi}{2}x\Big), \quad x \in [0, 1]. \tag{6}$$

C. Hyperparameters

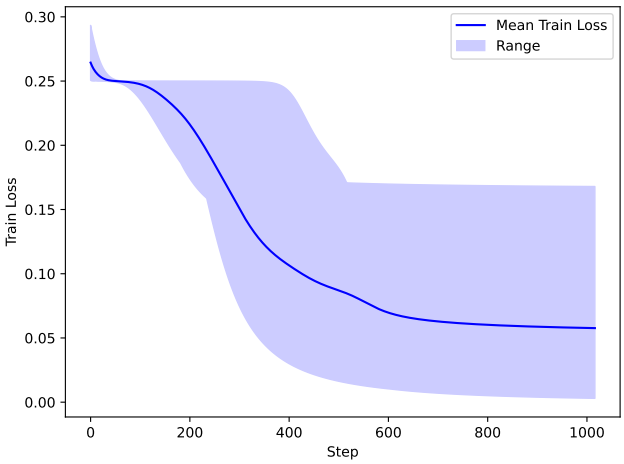
For the XOR task, by default we used a complete layer with 5 hidden units, an evolution time of 3 iterations, a batch size of 4, and trained the model for 1000 steps (which was found to be sufficient during initial hyperparameter tuning); for the Sine task, the default was 10 hidden units, an evolution time of 3 iterations, a batch size of 10, and training time of 600 steps. For results on an untrained model, we used the same defaults as the XOR task but only evolved our pruning logic for 10 steps, so for low values of f, Tril-damping methods were less efficient at inducing orderedness (as can be seen in Figure 3a). Across all tasks, the sigmoid activation function was applied and random uniform distribution was applied on the weights matrix and values vector; bias was not used. In all experiments, we employed the Mean Squared Error loss function with the Adam optimiser (Kingma, 2014) at a learning rate of 0.01, and ran each setup with 10 different seeds for reliability.

D. Training Loss of MLP and Complete Perceptron Layers

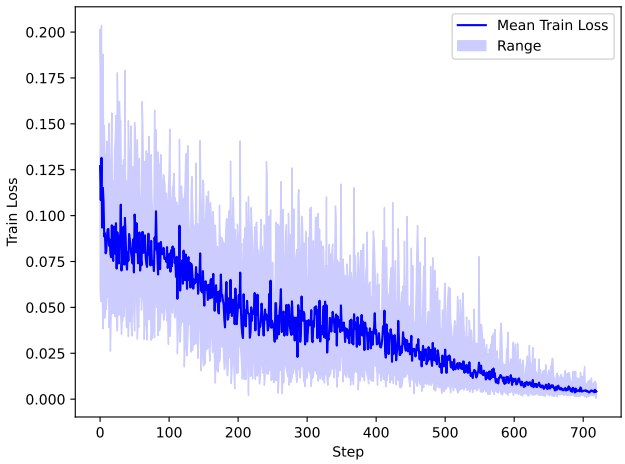
The following plots show the training losses of the complete perceptron layer and a traditional MLP on the XOR and Sine tasks. It confirms that the complete perceptron layer, even when pruned during training, is capable of learning.

For the MLP, we used one hidden layer with 2 units on the XOR task, and one hidden layer with 10 units on the Sine task. Bias, Sigmoid activation, and early stopping (after training loss decreases past 10^{-3}) were used for both tasks. The batch size, optimiser, and learning rate are the same as those of the default complete perceptron layer settings.

For the complete perceptron layers, default settings are used with DynamicTopK pruning (k=0.5).

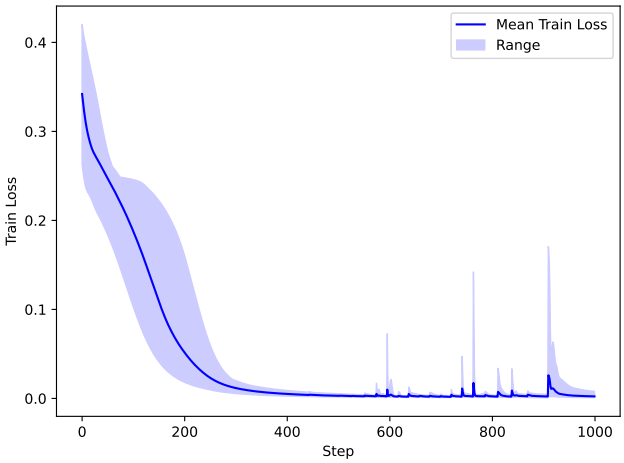


(a) Training loss of a normal MLP for the XOR task.

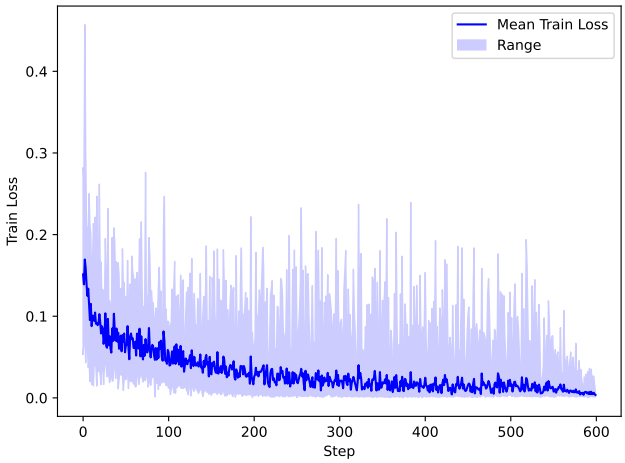


(b) Training loss of a normal MLP for the Sine task.

Figure 4. Training loss of a normal MLP for the XOR and Sine tasks.



(a) Training loss of a complete perceptron layer for the XOR task; default settings are used with DynamicTopK pruning ($k = 0.5$).



(b) Training loss of a complete perceptron layer for the Sine task; default settings are used with DynamicTopK pruning ($k = 0.5$).

Figure 5. Training loss of a complete perceptron layer for the XOR and Sine tasks.