CONNECTIVITY, DISTRACTORS, AND THE SPEED OF LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In deep learning, neural network architectures are typically fixed at the start of training, allocating each unit a predetermined set of inputs. Since that set is fixed independently of what the unit learns, some of its input connections may end up being useless, carrying no information about the unit's target. We study how these *distractor connections* affect the speed of learning. Drawing from theory in the linear setting, we characterize how updating the weights of such connections slows learning, then investigate the extent to which this phenomenon affects neural networks. To measure this, we construct a problem in which good connectivity is known in advance, allowing good and bad connectivity patterns to be compared directly. We find that bad connectivity slows learning and is especially costly in non-stationary problems, where learning speed determines performance. Scaling network width in this regime cannot solve the problem, and beyond a point *worsens* rather than improves performance. Learning connectivity has been pursued mainly for parsimony; our results suggest that it could also lead to faster learning, further motivating the use of methods that learn connectivity.

1 INTRODUCTION

Neural networks are, by design, structures that learn many things in parallel. Networks with multiple outputs do this explicitly, though even in a network with a single output, the implicit demand of being useful to its receiving units drives most hidden units to learn something unique. In this way, each hidden and output unit can be thought of as an individual learner following the same learning process, but with different feedback signals. As the feedback signal—and hence the implicit objective—of each unit is generally unique, it is natural to expect that different units will require different input signals. Some of the input signals a unit consumes may be broadly useful to a wide number of units, while others may be useful to only a small number or even a single unit.

Despite the above, our learning algorithms do not reflect the fact that different features require different input signals. Architectures are usually predetermined and fixed, connecting every output and every hidden unit to the same set of features within a given layer, regardless of which features are useful to which outputs. Some architectural advancements like convolutional layers (LeCun et al., 1998) exploit known properties of select problems, but the hand-design approach does not work if the structure of the problem is unknown, nor can it be applied at the granularity of individual units and connections. A network will thus inevitably contain connections in which the sending unit provides little to no utility to the receiving unit; we refer to these useless connections as ***distractor connections***.

The presence of distractor connections may seem inconsequential, since their weights can simply be driven to zero—but this solution is surprisingly ineffective. So long as errors persist, the weights of a network—zeroed out distractor weights included—will be assigned credit and change continually. Because distractor connections do not carry useful predictive information by definition, updates to these weights are equivalent to injections of noise (Sutton, 1992). In linear regression with fixed features, this noise increases the error of predictions and can only be reduced by decreasing the step-size, which in turn slows learning (Widrow et al., 1976). In a neural network, distractor connections that are assigned credit will also pass a gradient backwards, potentially slowing learning via interference with representations in prior layers. Decaying the step-size can hide these costs in stationary problems, but not in non-stationary problems where there is no convergence, the target

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

never settles, and how fast the network learns determines how well it performs. Our work addresses the questions of to what extent this phenomenon affects neural networks, and how the effects scale with larger networks and more complex problems.

The core contribution of this work is to establish and explain the relationship between connectivity and the speed of learning. We begin with a linear example in Section 2, where the simplicity of the learning dynamics allows us to fully explain the mechanisms by which distractor connections slow learning. Extending these findings to larger non-linear networks requires a setting in which we know which connections are useful and which are distractors. Lacking an algorithm that learns good connectivity, we instead construct a problem in which good connectivity is known in advance (Section 3). This problem lets us test whether the linear findings carry over to non-linear networks, and how the effects scale with the size of the problem and network (Section 4). We find that they do carry over, and that the effects only worsen with scale: in the worst case, scaling up a naively connected network reduces average performance rather than improving it.

2 Distractor Connections

In the prior section, we discussed how naive connectivity can result in distractor connections; this section explains how they slow learning. While our primary objective is to understand the role distractor connections play in multi-layer non-linear neural networks, the complexity of such systems makes this understanding more difficult to attain. To simplify the problem, we first consider only a single unit and its inputs, where existing theory on linear learning allows us to make precise claims about the mechanism that slows learning. When we turn to neural networks in later sections, that mechanism supplies intuition for what we observe.

2.1 Distractor Inputs

Consider a linear unit with N inputs x_t and weights w_t, trained by the Least Mean Squares (LMS) rule (Widrow & Stearns, 1985), $w_{t+1} = w_t + \alpha\,(y_t - w_t^\top x_t)\,x_t$, to minimize mean squared error. Its target y_t is a linear function of $n \leq N$ *useful* inputs, plus noise. The remaining $N - n$ *distractor* inputs are independent of the target, so their optimal weights are zero. The optimal weight $w^*_{i,t}$ of each useful input drifts by a random walk, $w^*_{i,t+1} = w^*_{i,t} + \epsilon_{i,t}$ with $\mathrm{Var}(\epsilon_{i,t}) = d_i^2$, making it a non-stationary problem. Studying a non-stationary problem like this is important because our goal is to measure the speed of learning. In a stationary problem, a learner with a decaying step-size eventually reaches the optimal solution, and the cost of slow learning is not visible in the convergent error. When the target keeps moving, the speed at which the learner tracks it directly determines its error.

Understanding why distractor inputs slow learning requires considering the step-size parameter of the LMS update rule. The step-size controls the tradeoff between adaptability and sensitivity to noise; increasing the step-size allows the learner to adapt faster at the cost of fitting to more gradient noise, while decreasing the step-size has the opposite effect. The optimal step-size for a problem balances this tradeoff by minimizing the sum of costs from both sources—the cost from *lag* (i.e., slow adaptation) and the cost from *gradient noise*. The answer to our question lies in how distractors shift these costs.

Both costs have closed forms. Under the standard analysis of LMS with a drifting target (Widrow et al., 1976; Widrow & Stearns, 1985), the steady-state excess errors from gradient noise and from lag are

$$\frac{\alpha}{2}\,\xi_{\min}\left(\sigma_1^2 + \cdots + \sigma_N^2\right) \qquad \text{and} \qquad \frac{d_1^2 + d_2^2 + \cdots + d_n^2}{2\alpha}, \tag{1}$$

where $\xi_{\min}$ is the irreducible error of the best linear predictor, σ_i^2 is the variance of input i, and d_i^2 is the per-step variance of the drift of input i's optimal weight. The gradient-noise cost grows linearly with the step-size, summing over the variance of all N inputs, because the weights for all inputs, distractors included, are perturbed by noise on every update. The lag cost shrinks as $1/\alpha$ and counts only the n useful weights because only they have a target to track. The optimal weight of each distractor is fixed at zero and so does not drift. Because one cost grows with the step-size while the other shrinks, raising the step-size eventually costs more in gradient noise than it saves in

lag, and the optimal step-size occurs at that crossover point:

$$\alpha^* = \sqrt{\frac{d_1^2 + d_2^2 + \cdots + d_n^2}{\xi_{\min}\left(\sigma_1^2 + \cdots + \sigma_N^2\right)}}. \tag{2}$$

A distractor adds its variance to the gradient-noise cost and nothing to the lag cost, so it moves that point to a smaller step-size. This is visible in Equation 2, where distractor inputs appear only in the denominator through their variances. In effect, distractors increase the cost of fitting to gradient noise, which requires a smaller step-size, which results in slower learning.

2.2 Distractor Connections

The effect of distractors in the linear setting is clear, but how do these findings translate to multi-layer, non-linear networks? We can gain insight into this by considering each hidden unit, together with its incoming connections, as a linear learning system of its own. A hidden unit's pre-activation is a weighted sum of its inputs, and the update to each of those weights takes the same form as the LMS update because both are instances of stochastic gradient descent:

$$w_{t+1} = w_t - \alpha\, \delta_t x_t, \tag{3}$$

where x_t is the input carried by the weight and δ_t is the error at the receiving unit. What differs in neural networks is where these signals come from. Under LMS, δ_t is computed from a target given directly to the unit and x_t is drawn from a fixed distribution, whereas in a neural network both are produced by other units that are themselves learning. The equations of the prior subsection therefore no longer hold, but the tradeoff they describe carries over.

Applied to a neural network, the tradeoff predicts that distractor connections slow learning just as distractor inputs do, and the structure of the network suggests the slowdown could be worse. Distractor connections are still perturbed on every update, and the remedy of lowering the step-size to average out noise remains at odds with the fact that higher step-sizes reduce the cost of lag. Unlike distractor inputs, however, their effects are not confined to the unit that receives them. The noise a distractor connection injects into its receiving unit is passed forward to every unit downstream, and the gradients it carries backward can interfere with the representations learned upstream (Fahlman & Lebiere, 1989; Yu et al., 2020). We hypothesize these effects will worsen the slowdown caused by distractor connections, but the linear analysis cannot predict the size of either effect. This leads us to study the problem empirically in the remainder of this work.

3 The Multi-MNIST Problem

To explore the effect of distractor connections empirically, we propose the multi-MNIST problem, a variant of MNIST (LeCun et al., 1998) that includes multiple subproblems per sample. The MNIST training set consists of 60,000 examples, each a pair of a 28×28 grayscale image of a handwritten digit (flattened to a 784-dimensional vector and converted to a float scale of $[0, 1]$) and its corresponding label in $\{0, \ldots, 9\}$. The multi-MNIST problem extends this by combining n image-label pairs into a single sample. Each sample is constructed by independently drawing n MNIST examples and concatenating their images and labels under a shared ordering, yielding an input of dimension $784n$ and a target of dimension $10n$. We refer to a single instance of the problem as having n subproblems. The loss is computed by independently applying softmax cross-entropy to each of the n examples, then summing the resulting n values. We mainly consider a non-stationary version of the problem in which, at a fixed interval, one of the n slots is selected uniformly at random, and the label mapping for that subproblem is replaced with a random permutation of $\{0, \ldots, 9\}$ (Elsayed & Mahmood, 2023).

The multi-MNIST problem is designed to satisfy several desiderata that make it well suited for evaluating the impact of connectivity on learning speed. Most importantly, good and bad connectivity patterns are known in advance by construction. Because the n MNIST examples in each sample are drawn independently, a network of n independent sub-networks should perform at least as well as a densely connected network with the same number of parameters. This allows us to test the effects of good connectivity without first requiring a method that finds good connectivity.

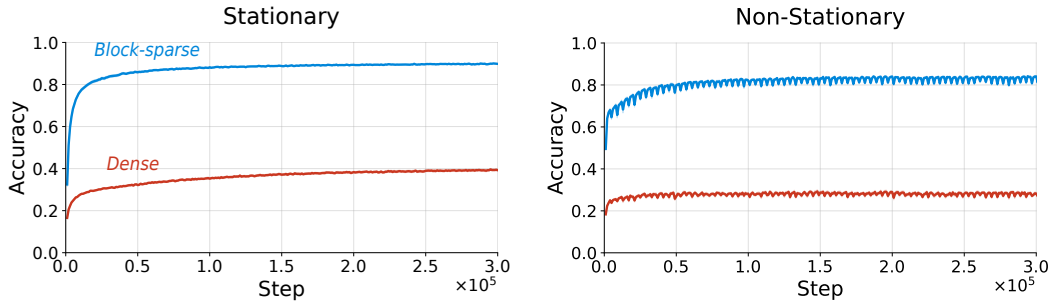

Figure 1: Mean accuracy across 30 runs for block-sparse and dense networks. The left multi-MNIST problem is stationary, and the right non-stationary with a label permutation every 800 steps. The 95% confidence interval regions are too narrow to see.

Another desirable property of the problem is that the ideal connectivity has a mix of shared and independent structure. The ten outputs within a subproblem all depend on the same image and can benefit from shared representations, whereas connections across subproblems carry no useful signal and can only act as distractors. While our primary focus is on the effect of distractor connections, knowing where representations should be shared is also an important part of how connectivity affects learning speed. The presence of both makes the problem a more holistic test for algorithms that aim to learn connectivity.

A final desirable property is that the number of subproblems n provides a clean scaling axis. Varying n changes the input and output dimensionality without changing the difficulty of any individual subproblem, which lets us study how the effects of connectivity scale. Together, these properties make the multi-MNIST problem an ideal setting for testing the effects of connectivity on learning speed, and for benchmarking algorithms that learn connectivity.

4 Experiments

The experiments in this section are aimed at understanding how the connectivity of a neural network, and the distractor connections it creates, affects its ability to learn. All experiments compare two-layer networks with leaky ReLUs (Maas et al., 2013) that, unless stated otherwise, vary only in connectivity pattern and parameter count, trained by stochastic gradient descent with single-sample updates. Step-sizes are swept over powers of two and selected per method, separately for each experiment. We primarily compare two connectivity patterns: block-sparse and dense.

Dense networks connect every input to every hidden unit and every hidden unit to every output, so each hidden unit touches every subproblem. In contrast, block-sparse networks partition hidden units evenly across subproblems, connecting each hidden unit only to the inputs and outputs of its assigned subproblem. Dense connectivity is disadvantageous in the multi-MNIST problem because there is no advantage to mixing information across independent subproblems, but hidden units in a dense network are forced to do so anyway. A hidden unit can avoid this mixing by zeroing weights to all subproblems but one, but doing so results in distractor connections. In Section 2 we hypothesized that this will result in slower learning, and that multi-layer networks will exacerbate the problem. We use this setup to test that hypothesis.

4.1 The Impact of Connectivity

In our first set of experiments, we test the hypothesis that a network with dense connectivity will learn slower than a network with block-sparse connectivity, even if both are capable of converging to similar levels of accuracy. We compare the performance of comparably sized dense and block-sparse networks in stationary and non-stationary versions of the multi-MNIST problem with 4 subproblems. Both networks have approximately 19,000 parameters, and label permutations in the non-stationary problem occur every 800 steps. In the results plotted in Figure 1, we observe that the block-sparse network achieved significantly higher accuracy than the dense network.

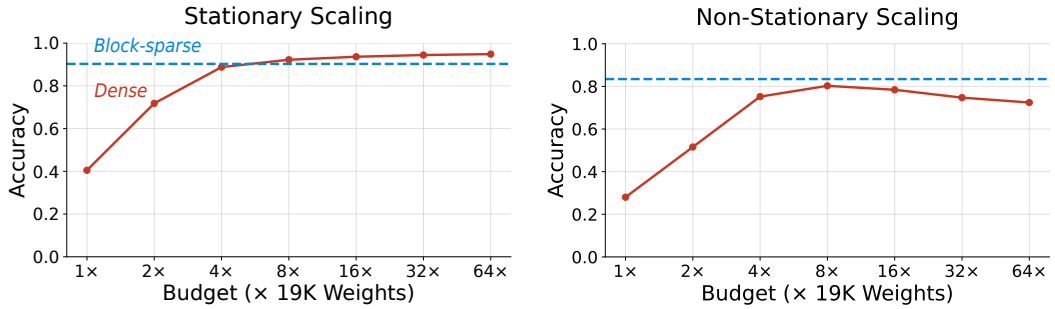

Figure 2: Accuracy over the last 10% of steps, averaged over 30 runs per point for dense networks of sizes ranging from 19,000 to 1.2M parameters. Performance of a 19,000-parameter block-sparse network is provided as reference. The left multi-MNIST problem is stationary, and the right non-stationary with a label permutation every 800 steps.

The fact that this performance gap exists even in the stationary version of the problem suggests that the cause is a lack of representational capacity rather than the existence of distractor connections. A dense network has more connections per hidden unit, so it has fewer hidden units than a block-sparse network even if their parameter counts are nearly equivalent. If we want to fairly compare the learning speed of both networks, we need to ensure that the dense network can represent at least all of the solutions that the block-sparse network can. We can achieve this by simply increasing the width of the dense network.

Reusing the same experimental setup, we train dense networks with sizes ranging from 19,000 to 1.2 million parameters. We plot the steady-state accuracy of a dense network across 7 different parameter budgets on the stationary and non-stationary versions of the multi-MNIST problem in Figure 2, and compare against an unchanged, 19,000-parameter block-sparse network. With 64 times the budget of the block-sparse network, the dense network is given a significant advantage. In the stationary problem, the dense network matches the block-sparse network at four times the parameter count, as expected, and continues to improve as capacity grows. In the non-stationary problem, however, the dense network never matches the 19,000-parameter block-sparse network, and beyond 152,000 parameters, performance even *degrades*. Why?

We previously hypothesized that the dense network would learn slower due to the existence of distractor connections and interference across subproblems, but if that is the case, why would performance not suffer in both stationary and non-stationary problems? The answer lies in what our performance metric measures. We measure average accuracy over the final 10% of time steps, but this captures different things in the two settings. In the stationary problem, given enough updates, the average is dominated by convergent performance, which a sufficiently large network can reach regardless of learning speed. In the non-stationary problem, the constant change prevents convergence, and the average instead reflects how quickly the network adapts (i.e., the speed of learning). Dense connectivity therefore has an effect in both settings, but those effects are visible only when the metric is sensitive to learning speed. While the effects of distractor connections can be scaled away in stationary problems, they cannot be avoided in non-stationary problems.

4.2 Isolating the Cause

Our results so far are consistent with the hypothesis that distractor connections slow learning, but they do not rule out other plausible explanations. The experiments in this subsection aim to eliminate these alternatives and confirm the mechanism responsible for slower learning.

One alternative is that the gap between dense and block-sparse networks stems from differences in their loss landscapes. Block-sparse connectivity restricts the network to a subset of solutions, removing many poor ones from consideration. Perhaps a dense network could, in principle, learn as quickly as a block-sparse network, but simply never finds a representation that lets it do so. To test this, we train a block-sparse network until its performance stabilizes, then add the connections needed to make it dense and continue training. The new weights are initialized to zero so that the network's predictions are unchanged at the moment of transition, as is common in methods

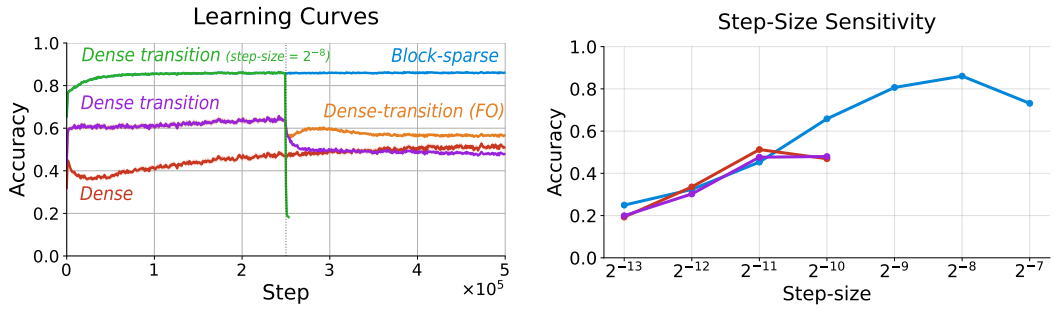

Figure 3: The left figure depicts the average accuracy per step, and the right figure the average accuracy over the final 10% of training steps. Both plots average over 30 runs per method. The problem is non-stationary multi-MNIST with 16 subproblems and a label permutation every 125 steps. Dense and dense-transition methods diverged with step-sizes greater than 2^{-10}. A 95% confidence interval is too narrow to see.

that grow new connections (e.g., Bellec et al., 2018; Evci et al., 2020). We call this method *dense-transition*, and compare it, along with a variant introduced below, to standard dense and block-sparse networks in Figure 3. These experiments use a multi-MNIST problem with 16 subproblems and a label permutation every 125 steps. The left plot shows learning curves, and the right plot shows sensitivity to the step-size.

We can draw several conclusions from the learning curves alone. With the same step-size as the block-sparse network, the dense-transition network (green curve) diverges shortly after the new connections are added. Even at its own best step-size (purple curve), its performance quickly degrades and eventually drops below that of the standard dense network. Because the dense-transition network starts from the block-sparse network's solution and still cannot maintain its performance, the loss landscape cannot explain the gap. These results also confirm that distractor connections play a major role in the dense network's inferior performance, as the performance of the dense-transition method drops as soon as the additional connections are added, despite each being initialized to zero.

That the dense-transition network diverges at the block-sparse network's step-size points to another pattern worth examining: the relationship between connectivity and the optimal step-size. As the sensitivity curves show, the dense and dense-transition networks both perform best at a step-size of 2^{-11}, whereas the block-sparse network continues to improve with higher step-sizes up to its optimum of 2^{-8}. This matches our prediction from Section 2 that networks with more distractors have smaller optimal step-sizes.

With the role of distractor connections established, we can now examine how they affect learning in neural networks. In the linear setting, a distractor input slows learning by injecting noise into the output. In a neural network, a distractor connection also passes gradients backward, which can carry noise into earlier layers and cause interference. To separate these effects, we introduce *forward-only dense-transition* (*dense-transition (FO)*; orange curve), which differs from dense-transition only in that gradients are not passed backward through the added connections. This prevents the new connections from causing interference upstream, so any remaining drop in performance is due only to the noise they inject in the forward pass. Comparing the forward-only network (orange) to the full dense-transition network (purple) shows that both effects play a role, each accounting for roughly half of the drop in performance.

4.3 Scaling the Problem and Network

Having established the mechanism by which bad connectivity slows learning, we now turn to the magnitude of the effect. The cost of bad connectivity was already noticeable at four subproblems with relatively small networks—how does the cost scale as the network and problem sizes grow? We sweep block-sparse and dense networks across multi-MNIST problems with 2 to 32 subproblems and network sizes from 3,000 to 13M parameters. The rate of label permutation scales with the number of subproblems so that each subproblem's labels are permuted on average every 2000 steps. Results are shown in Figure 4.

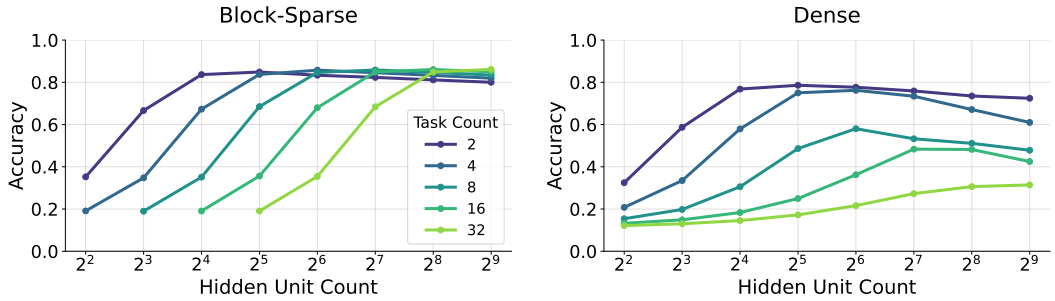

Figure 4: Accuracy over the last 10% of training steps, averaged over 5 runs per point, for block-sparse and dense networks ranging in size from 4 to 512 hidden units. Step-sizes are tuned separately for each point. The problem is non-stationary multi-MNIST with a variable number of subproblems. The label permutation frequency is scaled so that each subproblem experiences a single permutation every 2000 steps on average.

The same patterns from our previous experiments appear here, but the cost of dense connectivity grows sharply as the number of subproblems increases. At every subproblem count, no dense network matched the peak accuracy of the block-sparse network, even though a dense network has n times as many parameters as a block-sparse network of the same width. The block-sparse network retains near-peak performance across a much wider range of sizes, though with fewer subproblems it too shows mild degradation past a certain scale. The magnitude of the gap is the most striking finding. At 32 subproblems, a block-sparse network with roughly 200,000 parameters reaches around 85% accuracy, while the dense network never surpasses 32%, even with 64 times as many parameters. These results echo the same conclusion as our prior scaling experiments, that a network cannot scale its way out of bad credit assignment.

4.4 Learning Connectivity

Block-sparse connectivity works well because we designed it with knowledge of the problem's structure, but such knowledge is rarely available. If a good connectivity structure cannot be known in advance, could a network learn it from experience?

We test two methods that learn connectivity: Deep Rewiring (DEEP R) (Bellec et al., 2018) and Sparse Evolutionary Training (SET) (Mocanu et al., 2018). Both initialize a network with random sparse connectivity, then prune and grow connections while keeping the number of connections fixed. DEEP R prunes a weight when it changes sign, and SET prunes the weights with the smallest magnitudes at a fixed interval. Both grow new connections at random. We compare these methods to three baselines with static connectivity: dense, block-sparse, and a *static sparse* network initialized in the same way as SET. All networks have approximately 203,000 parameters and are trained on multi-MNIST with 16 subproblems and a label permutation every 125 steps. Hyperparameters for each method are listed in Appendix A.

The learning curves in Figure 5 show that both DEEP R and SET improve over the static sparse network, but close less than half of the gap to the block-sparse network. These methods leave significant room for improvement, and the path purity of their networks suggests they are even further from good connectivity than accuracy alone indicates. We define the path purity of an output as the fraction of its paths to the inputs that lead to inputs from the same subproblem, and the path purity of a network as the average over its outputs. A block-sparse network has a path purity of 1 because it completely separates the subproblems, whereas dense and randomly connected networks have an expected path purity of $1/n$, because each output has an equal expected number of paths to the inputs of every subproblem. Surprisingly, neither method ever surpasses a path purity of 0.08, compared to 0.0625 for random connectivity. Both are barely uncovering the structure of the problem.

Why would methods designed to learn connectivity fail to do so in a problem with such clear-cut independent structure? We identify two likely causes.

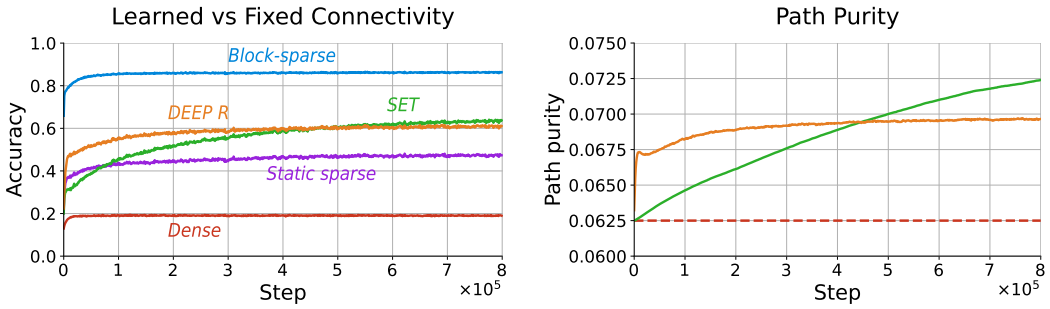

Figure 5: The left figure depicts the average accuracy per step, and the right figure the path purity of SET (green) and DEEP R (orange). The dashed line marks the expected path purity of dense and randomly connected networks ($1/16$). Both plots average over 30 runs per method. The problem is non-stationary multi-MNIST with 16 subproblems and a label permutation every 125 steps. All networks have approximately 203,000 parameters, with hyperparameters tuned separately for each method (Appendix A).

First, both SET and DEEP R decide which connections to keep based on their current weights. But a connection and its weight encode different things; the connection encodes the existence of a relationship between two units, while the weight encodes the nature of that relationship. In a stationary problem, relationships never change, so a weight near zero means no relationship exists. In a non-stationary problem, it may only mean the relationship is changing. Multi-MNIST is a problem where which inputs are related to which outputs never changes, but both methods fail to exploit this structure, pruning connections whose weights only transiently cross zero.

Second, the search over connections itself introduces distractor connections. New connections are added at random, so many will be distractors. Even if they are eventually pruned, they cause the same problems as any other distractor while they exist. As long as the search continues, the methods will continue to produce new distractor connections. Methods that grow connections more selectively, such as RigL (Evci et al., 2020) and SNFS (Dettmers & Zettlemoyer, 2019), may add fewer distractors, but they often require statistics about every possible connection, giving up some of the benefits of sparsity.

5 Related Work

Aside from the connectivity learning methods tested in Section 4.4, the works closest to ours are those of Martin & Modayil (2021) and Modayil & Abbas (2023). Both propose methods for learning the network structure of reinforcement learning agents that maintain many auxiliary predictions. Learning structure in this way outperforms fixed architectures and lets the agents scale to large observation spaces whose structure is not known in advance. Multi-MNIST is inspired by the Multi-Catch environment of Modayil & Abbas (2023), which combines independent problems so that the right structure is known in advance.

Beyond changing connectivity, we would expect several common methods, including batching, momentum, and gradient orthogonalization, to reduce the cost of distractor connections. We do not test them here, however, because none addresses the root cause of the problem. Batching reduces the variance of updates, which should help average the noisy updates to distractor connections toward zero. But it does so at the cost of delaying updates and computing them on outdated data, which is an undesirable trade-off when learning speed matters. Momentum, as used in optimizers like Adam (Kingma & Ba, 2015), similarly averages gradients over time, but only over a short window; with the common setting of $\beta = 0.9$, it averages over roughly ten steps, which limits how much noise it can remove. Gradient orthogonalization methods like PCGrad (Yu et al., 2020) reduce interference between tasks by removing the parts of their gradients that conflict. This can work in multi-MNIST, where the task structure is known, but is not generally applicable beyond that. It also targets interference between tasks rather than the noise that distractor connections inject.

6 Discussion & Future Work

Our work tells a clear story about how connectivity can affect the speed of learning. Each unit in a network can generally benefit from a different set of inputs that cannot be known in advance. Ignoring this and connecting every unit to all possible inputs leads to distractor connections, which inject noise into downstream activations and backpropagate gradients that interfere with representations in earlier layers; these effects slow learning.

When the ideal connectivity cannot be known in advance or changes over time, learning connectivity seems like a natural solution. The methods we tested, SET and DEEP R, do help, but they close less than half of the gap to block-sparse connectivity. Like most connectivity learning methods, they were motivated by parameter efficiency rather than faster learning (Hoefler et al., 2021). The relationship between connectivity and learning speed has not been widely documented, so it is not surprising that these methods do not address it. By providing new insight into this relationship and introducing a methodology for evaluating connectivity learning algorithms with the multi-MNIST problem, we hope that this work will contribute to further progress on algorithms for learning connectivity.

Even if connectivity learning were solved, however, part of the issue would remain. Fully eliminating the problem requires addressing it at its root: credit assignment. It is not the existence of distractor connections that slows learning but the credit assigned to them; removing a connection is just one way of ensuring it is assigned no credit. Distractor connections are a special case of a broader phenomenon in which weights that do not need to change are incorrectly assigned credit and changed. A weight that has reached its optimal value, for example, needs no further change even though it is nonzero, yet gradient noise perturbs it just as it perturbs distractor connections. More generally, the ideal amount of change per weight lies on a spectrum, and no change is just one end of the spectrum. Learning how to assign credit beyond standard backpropagation could therefore address the distractor connection issue more directly than learning connectivity.

Step-size optimization methods are one existing approach in this direction (Degris et al., 2024). Algorithms like IDBD (Sutton, 1992), Autostep (Mahmood et al., 2012), and MetaOptimize (Sharifnassab et al., 2025) control how much each weight changes per update, offering even finer-grained control over how credit is assigned than changing connectivity. Existing methods do not modulate how much gradient is propagated backward, but this would be an interesting extension to this line of work. In principle, they can learn to dampen updates to weights whose gradients are dominated by noise, which for a distractor connection has an effect similar to removing it entirely. Success with per-weight step-size optimization in deep networks has been limited to date, but the underlying mechanism of changing step-sizes is well-suited to the problem.

We expect that the best solution will be a method that combines both connectivity learning and step-size optimization. The addition and removal of connections can be used to test different connectivity patterns while remaining within a fixed budget, and step-size optimization can ensure new, unproven weights do not have an outsized impact on the network's ability to learn by keeping their step-sizes small. Perhaps future work can extend the principle further, learning even more of the network's design and further reducing the inductive biases that must be specified by a human designer.

AI use statement

We used generative AI tools to implement the SET and DEEP R methods (Section 4.4), and did not use them for any other task requiring disclosure. Additionally, we used them to write experiment infrastructure and plotting code, to suggest edits to text written by the authors, and to identify a small number of related works. All AI-assisted work was reviewed by the authors; all code was tested for correctness by the first author, and the SET and DEEP R implementations were checked against reference implementations. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

The supplementary material contains all code needed to reproduce every experiment and figure in this paper, along with a README that explains how to run each experiment and generate each figure. The multi-MNIST problem is described in Section 3 and the experimental setup in Section 4,

with the number of runs and problem settings for each experiment given in the figure captions. Appendix A lists the hyperparameters searched and selected for the connectivity learning methods, and the sweep configurations for all experiments are included with the code.

REFERENCES

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Thomas Degris, Khurram Javed, Arsalan Sharifnassab, Yuxin Liu, and Richard S. Sutton. Step-size optimization for continual learning. *CoRR*, abs/2401.17401, 2024.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840, 2019.

Mohamed Elsayed and A. Rupam Mahmood. Utility-based perturbed gradient descent: An optimizer for continual learning. *CoRR*, abs/2302.03281, 2023.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, pp. 2943–2952, 2020.

Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 524–532, 1989.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

Ashique Rupam Mahmood, Richard S. Sutton, Thomas Degris, and Patrick M. Pilarski. Tuning-free step-size adaptation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2121–2124, 2012.

John D. Martin and Joseph Modayil. Adapting the function approximation architecture in online reinforcement learning. *CoRR*, abs/2106.09776, 2021.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018.

Joseph Modayil and Zaheer Abbas. Towards model-free RL algorithms that scale well with unstructured data. *CoRR*, abs/2311.02215, 2023.

Arsalan Sharifnassab, Saber Salehkaleybar, and Richard S. Sutton. MetaOptimize: A framework for optimizing step sizes and other meta-parameters. In *International Conference on Machine Learning (ICML)*, pp. 54300–54325, 2025.

Richard S. Sutton. Adapting bias by gradient descent: An incremental version of Delta-Bar-Delta. In *National Conference on Artificial Intelligence (AAAI)*, pp. 171–176, 1992.

Bernard Widrow and Samuel D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

Bernard Widrow, John M. McCool, Michael G. Larimore, and C. Richard Johnson, Jr. Stationary and nonstationary learning characteristics of the LMS adaptive filter. *Proceedings of the IEEE*, 64(8):1151–1162, 1976.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

A HYPERPARAMETERS FOR LEARNING CONNECTIVITY

This appendix lists the hyperparameters for the experiment in Section 4.4. For each method, we ran two grid searches, using the first to narrow the range of values considered in the second. The tables below list the values tried across both searches, with the selected values, those with the highest average accuracy over the final 10% of training steps, shown in bold. Values marked *fixed* were not searched and, unless noted otherwise, follow the published method, and values marked *derived* are computed from other hyperparameters. All methods share a budget of 203,264 connections, the number used by a block-sparse network with 16 hidden units per subproblem.

Table 1: SET hyperparameters.

Hyperparameter	Description	Values
Hidden units	Width of the hidden layer.	64, 128, **256**, 512, 768, 1024
Step-size	Step-size of SGD.	2^{-10}, 2^{-9}, $\mathbf{2^{-8}}$, 2^{-7}
Rewiring interval	Steps between prune-and-grow events.	125, 250, 500, **1000**, 2000, 4000
Rewiring rate	Fraction of connections rewired per step.	$\mathbf{5 \times 10^{-6}}$ (fixed)
ζ	Fraction of connections pruned per rewiring event.	$\mathbf{5 \times 10^{-3}}$ (derived)
Pruning criterion	Which connections are pruned.	**Smallest magnitude** (fixed)
Initial connectivity	Random topology used at initialization.	**Erdős–Rényi** (fixed)
Weight initialization	Initial weight distribution.	**Glorot** (fixed)

Table 2: DEEP R hyperparameters.

Hyperparameter	Description	Values
Hidden units	Width of the hidden layer.	256, 512, **768**
Step-size	Step-size of SGD.	2^{-10}, 2^{-9}, 2^{-8}, $\mathbf{2^{-7}}$
L1 coefficient	Strength of the L1 pull towards zero.	0, 10^{-6}, 10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}, 10^{-2}
Noise ratio	Langevin noise scale relative to the L1 pull.	0, 0.3, 1.0, **3.0**
Temperature	Langevin temperature.	$\mathbf{3.5 \times 10^{-10}}$ (derived)
Reactivation period	Steps between reactivation events.	**1** (fixed)
Rewire output layer	Whether output connections are rewired.	**Yes** (fixed)
Reactivation magnitude	Initial magnitude of reactivated weights.	$\mathbf{10^{-12}}$ (fixed)
Initial connectivity	Random topology used at initialization.	**Uniform** (fixed)
Output density ratio	Output-layer density relative to the hidden layer, reduced from the published 30 to fit the budget.	**10** (fixed)
Weight initialization	Initial weight distribution.	**Scaled normal** (fixed)

Table 3: Baseline hyperparameters.

Method	Hyperparameter	Description	Values
Static sparse	Hidden units	Width of the hidden layer.	256, 512, **768**, 1024
	Step-size	Step-size of SGD.	2^{-10}, 2^{-9}, 2^{-8}, $\mathbf{2^{-7}}$
Block-sparse	Hidden units	16 per subproblem.	**256** (fixed)
	Step-size	Step-size of SGD.	2^{-10}, 2^{-9}, $\mathbf{2^{-8}}$, 2^{-7}
Dense	Hidden units	Width allowed by the budget.	**16** (derived)
	Step-size	Step-size of SGD.	2^{-12}, 2^{-11}, $\mathbf{2^{-10}}$, 2^{-9}