SIAM J. Sci. Comput.
Vol. 41, No. 5, pp. S297–S320

# MULTILEVEL ARTIFICIAL NEURAL NETWORK TRAINING FOR SPATIALLY CORRELATED LEARNING[*]

C. B. SCOTT[†] AND ERIC MJOLSNESS[‡]

**Abstract.** Multigrid modeling algorithms are a technique used to accelerate iterative method models running on a hierarchy of similar graphlike structures. We introduce and demonstrate a new method for training neural networks which uses multilevel methods. Using an objective function derived from a graph-distance metric, we perform orthogonally-constrained optimization to find optimal prolongation and restriction maps between graphs. We compare and contrast several methods for performing this numerical optimization, and additionally present some new theoretical results on upper bounds of this type of objective function. Once calculated, these optimal maps between graphs form the core of multiscale artificial neural network (MsANN) training, a new procedure we present which simultaneously trains a hierarchy of neural network models of varying spatial resolution. Parameter information is passed between members of this hierarchy according to standard coarsening and refinement schedules from the multiscale modeling literature. In our machine learning experiments, these models are able to learn faster than training at the fine scale alone, achieving a comparable level of error with fewer weight updates (by an order of magnitude).

**Key words.** multigrid methods, neural networks, classification, image analysis

**AMS subject classifications.** 46N10, 47N10, 65M55, 68T05, 82C32

**DOI.** 10.1137/18M1191506

## 1. Motivation.

Multigrid methods (or multilevel methods when the underlying graph is not a grid) comprise a modeling framework that seeks to ameliorate a core problem in iterative method models with local update rules: namely, that these models have differing rates of convergence for fine-scale and coarse-scale modes [38]. Because iteration of these models involves making updates of a given characteristic length, they are maximally efficient for propagating modes of approximately this wavelength, but ignore finer modes and are inefficient on coarser modes. Multigrid approaches gain computational benefit by addressing these multiple length-scales of behavior using multiple model resolutions, rather than attempting to address them all at the finest scale (in which the coarse modes converge slowly). These methods make use of "prolongation" and "restriction" operators to move between models in a hierarchy of scales. At each level, a "smoothing" step is performed—usually, for multilevel methods, a smoothing step consists of one pass of some iterative method for improving the model at that scale.

In this paper, we describe a novel general algorithm for applying this approach to the training of artificial neural networks (ANNs). In particular, we demonstrate the efficiency of this new method, which combines ideas from machine learning and multilevel modeling, by training several hierarchies of autoencoder networks (ANNs

[*]Received by the editors June 15, 2018; accepted for publication (in revised form) May 17, 2019; published electronically October 29, 2019.
https://doi.org/10.1137/18M1191506

**Funding:** This work was supported by U.S. National Science Foundation NRT Award 1633631, Human Frontiers Science Program grant HFSP-RGP0023/2018, U.S. National Institute on Aging grant AG059602, U.S. National Institutes of Health grant R01HD073179, USAF/DARPA FA8750-14-C-0011, and the Leverhulme Trust.

[†]Department of Computer Science, University of California, Irvine, Irvine, CA 92617 (scottcb@uci.edu).

[‡]Departments of Computer Science and Mathematics, University of California, Irvine, Irvine, CA 92617 (emj@uci.edu).

which learn a mapping from data to a lower-dimensional latent space) [16, 6]. By applying multilevel modeling methods we learn this latent representation with an order of magnitude less cost. We will make our notion of "cost" more precise in the experiments section, section 4.

## 2. Background.

**2.1. Prior work.** In this section, we discuss prior attempts to apply ideas from multigrid methods to neural network models. Broadly speaking, prior approaches to neural net multigrid can be categorized into two classes: (1) neural network models which are "structurally multigrid," i.e., are typical neural network models which make use of multiple scales of resolution, and (2) neural network training processes which are hierarchical in some way, or use a coarsening-refinement procedure as part of the training process.

In the first class are approaches [14, 20, 26, 32]. Reference [20] implements a convolutional network in which convolutions make use of a multigrid-like structure similar to a Gaussian pyramid, with the motivation that the network will learn features at multiple scales of resolution. Reference [14] defines a convolution operation, inspired by multigrid methods, that convolves at multiple levels of resolution simultaneously. Reference [32] demonstrates a recurrent neural network model which similarly operates in multiple levels of some scale space; but in this paper the scale space is a space of aggregated language models (specifically, the differing scales are different levels of generality in language models—for example, topic models are coarsest, word models are finest, with document models somewhere in between). Reference [26] applies an algebraic multigrid approach to the specific case of training relaxation- or optimization-based neural networks. Common to all four of these approaches is that they make use of a modified neural net structure while leaving the training process unchanged, except that the network accepts multiresolution inputs.

In contrast, multilevel neural network models [3, 31] in the second category present modified learning procedures which also use methodology similar to multilevel modeling. Reference [3] introduces a network which learns at coarse scales, and then gradually refines its decision making by increasing the resolution of the input space and learning "corrections" at each scale. However, that paper focuses on the capability of a particular family of basis functions for neural networks, and not on the capabilities of the multigrid approach. Reference [31] presents a reframing of the neural network training process as an evolution equation in time, and then applies a method called MGRIT (multigrid reduction in time [11]) to achieve the same results as parallelizing over many runs of training.

Our approach is fundamentally different: we use coarsened versions of the network model to make coarse updates to the weight variables of our model, followed by "smoothing steps"' in which the fine-scale weights are refined. This approach is more general than any of [14, 20, 32], since it can be applied to any feed-forward network and is not tied to a particular network structure. The approach in [31] is to parallelize the training process by reframing it as a continuous-in-time evolution equation, but it still uses the same base model and therefore only learns at one spatial scale.

Our method is both structurally multilevel and learns using a multilevel training procedure. Our hierarchical neural network architecture is the first to learn at all spatial scales simultaneously over the course of training, transitioning between neural networks of varying input resolution according to standard multigrid method schedules of coarsening and refinement. To the best of our knowledge, this represents a fully novel approach to combining the powerful data analysis of neural networks with the model acceleration of multiscale modeling.

**2.2. Outline.** Section 3 covers the mathematical theory underlying our method. We first introduce the necessary definitions, which are then used in subsection 3.2.1 to define an objective function which evaluates a map between two graphs in terms of how well it preserves the behavior of some local process operating on those graphs (interpreting the smaller of the two graphs as a coarsened version of the larger). In subsection 3.3.3 we examine some properties of this objective function, including presenting some projection matrices which are local optima for particular choices of graph structure and process. In subsections 3.4 and 3.4.2, we define the *multiscale artificial neural network (MsANN)*, a hierarchically-structured neural network model which uses these optimized projection matrices to project network parameters between levels of the hierarchy, resulting in more efficient training. In section 4, we demonstrate this efficiency by training a simple neural network model on a variety of datasets, comparing the cost of our approach to that of training only the finest network in the hierarchy. Finally, we conclude the paper by proving two novel properties of our objective function in section 5.

**3. Theory.** In this section, we first define basic terms used throughout the paper, and explain the core theory of our paper: that of optimal prolongation maps between computational processes running on graph-based data structures, and hence between graphs. In this paper we use a specific example of such a process, single-particle diffusion on graphs, to examine the behavior of these prolongation maps. Finally, we discuss numerical methods for finding (given two input graphs $G_1$ and $G_2$, and a process) prolongation and restriction maps which minimize the error of using $G_1$ as a surrogate structure for simulating the behavior of that process on $G_2$. We will define more rigorously what we mean by "process," "error," and "prolongation" in section 3.2.1.

**3.1. Definitions.** In order to describe our objective function, we must first introduce some core concepts related to minimal mappings between graphs.

- Graph lineage: A graph lineage is a sequence of graphs, indexed by $l \in \mathbb{N} = 0, 1, 2, 3 \ldots$, satisfying the following:
  - $G_0$ is the graph with one vertex and one self-loop, and;
  - successive members of the lineage grow roughly exponentially—that is, the growth rate is $O(b^{l+\epsilon})$ for some $b > 1$, $\epsilon \geq 0$, and $l > 1$.

  We introduce this term to differentiate this definition from that of a graph *family*, which is a sequence of graphs without the growth condition. Most of the graph lineages we examine in this work are structurally similar—for example, the lineage of path graphs of length $2^l$. However, we do not define this similarity in a rigorous sense, and we do not require it in the definition of a lineage.
- Graph Laplacian: We define the Laplacian matrix of a graph $G$ as $L(G) = A(G) - D(G)$, where $A(G)$ and $D(G)$ are the adjacency matrix and diagonal degree matrix of the graph, respectively. The eigenvalues of this matrix are referred to as the spectrum of $G$. See [4, 10] for more details on graph Laplacians and spectral graph theory. Our sign convention for $L$ agrees with the standard continuum Laplacian operator, $\Delta$, of a multivariate function $f$: $\Delta f = \sum_{i=1}^{n} \frac{\delta^2 f}{\delta x_i^2}$.
- Kronecker product and sum of matrices: Given a $(k \times l)$ matrix $M$, and some

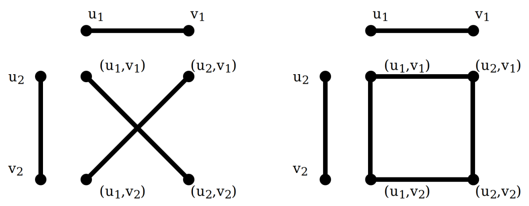

FIG. 1. *Two types of graph product: the Cross product ($G_1 \times G_2$, left) and Box product ($G_1 \square G_2$, right). For two edges $v_1 \sim u_1 \in G_1$ and $v_2 \sim u_2 \in G_2$, we illustrate the resultant edges in the set of vertices $\{(u_1, v_1), (u_2, v_1), (u_1, v_2), (u_2, v_2)\}$ in the graph product.*

other matrix $N$, the Kronecker product is the block matrix

$$M \otimes N = \begin{bmatrix} m_{11}N & \cdots & m_{1l}\mathbf{N} \\ \vdots & \ddots & \vdots \\ m_{k1}N & \cdots & m_{kl}\mathbf{N} \end{bmatrix}.$$

If $M$ and $N$ are square, their Kronecker sum is defined, and is given by

$$M \oplus N = M \otimes I_N + I_M \otimes N,$$

where we write $I_A$ to denote an identity matrix of the same size as $A$.

- Box product ($\square$) of graphs: For $G_1$ with vertex set $U = \{u_1, u_2 \ldots\}$ and $G_2$ with vertex set $V = \{v_1, v_2 \ldots\}$, $G_1 \square G_2$ is the graph with vertex set $U \times V$ and an edge between $(u_{i_1}, v_{j_1})$ and $(u_{i_2}, v_{j_2})$ when either of the following is true:
  - $i_1 = i_2$ and $v_{j_1}$ and $v_{j_2}$ are adjacent in $G_2$ or
  - $j_1 = j_2$ and $u_{i_1}$ and $u_{i_2}$ are adjacent in $G_1$.

  This may be rephrased in terms of the Kronecker sum $\oplus$ of the two matrices:

$$A(G_1 \square G_2) = A(G_1) \oplus A(G_2) = A(G_1) \otimes I_{|G_2|} + I_{|G_1|} \otimes A(G_2). \quad (3.1)$$

- Cross product ($\times$) of graphs: For $G_1$ with vertex set $U = \{u_1, u_2 \ldots\}$ and $G_2$ with vertex set $V = \{v_1, v_2 \ldots\}$, $G_1 \times G_2$ is the graph with vertex set $U \times V$ and an edge between $(u_{i_1}, v_{j_1})$ and $(u_{i_2}, v_{j_2})$ when both of the following are true:
  - $u_{i_1}$ and $u_{i_2}$ are adjacent in $G_1$ and
  - $v_{j_1}$ and $v_{j_2}$ are adjacent in $G_2$.

  We include the standard pictorial illustration of the difference between these two graph products in Figure 1.
- Grid graph: a grid graph (called a lattice graph or Hamming graph in some texts [7]) is the distance-regular graph given by the box product of path graphs $P_{a_1}, P_{a_1}, \ldots, P_{a_k}$ (yielding a grid with aperiodic boundary conditions) or by a similar list of cycle graphs (yielding a grid with periodic boundary conditions).
- Prolongation map: A prolongation map between two graphs $G_1$ and $G_2$ of sizes $n_1$ and $n_2$, with $n_2 \geq n_1$, is an $n_2 \times n_1$ matrix of real numbers which is an optimum of the objective function of (3.3) (possibly subject to some set of constraints $C(P)$).
- Eigenvalue matching: Given two matrices $A_1$ and $A_2$, and lists of their eigenvalues $\{\lambda^{(1)}_1, \lambda^{(1)}_2, \ldots, \lambda^{(1)}_{n_1}\}$ and $\{\lambda^{(2)}_1, \lambda^{(2)}_2, \ldots, \lambda^{(2)}_{n_2}\}$, with $n_2 \geq n_1$, we

define the *minimal eigenvalue matching* $m^*(A_1, A_2)$ as the matrix which is the solution of the following constrained optimization problem:

$$m^*(A_1, A_2) = \arg\inf_M \sum_{i=1}^{n_2}\sum_{j=1}^{n_1} M_{i,j}(\lambda_j^{(1)} - \lambda_i^{(2)})^2$$
$$\text{subject to} \quad \left(M \in \{0,1\}^{n_2\times n_1}\right) \wedge \left(\sum_{i=1}^{n_2} M_{i,j} = 1\right) \wedge \left(\sum_{j=1}^{n_1} M_{i,j} \le 1\right). \tag{3.2}$$

In the case of eigenvalues with multiplicity $> 1$, there may not be one unique such matrix, in which case we distinguish matrices with identical cost by the lexicographical ordering of their occupied indices and take $m^*(A_1, A_2)$ as the first of those with minimal cost. This matching problem is well studied and efficient algorithms for solving it exist; we use a Python language implementation [8] of a 1957 algorithm due to Munkres [27]. Additionally, given a way to enumerate the minimal-cost matchings found as solutions to this eigenvalue matching problem, we can perform combinatorial optimization with respect to some other objective function $g$, in order to find optima of $g(P)$ subject to the constraint that $P$ is a minimal matching.

### 3.2. Optimal prolongation maps between graphs.

**3.2.1. Our objective function.** Given two graphs $G_1$ and $G_2$, we find the optimal prolongation map between them as follows: We first calculate the graph Laplacians $L_1$ and $L_2$, as well as pairwise vertex Manhattan distance matrices (i.e., the matrix with $T_{i,j}$ the minimal number of graph edges between vertices $i$ and $j$ in the graph), $T_1$ and $T_2$, of each graph. Calculating these matrices may not be trivial for arbitrary dense graphs; for example, calculating the pairwise Manhattan distance of a graph with $m$ edges on $n$ vertices can be accomplished in $O(m + n\log n)$ by the Fibonacci heap version of Dijkstra's algorithm [13]. Additionally, in section 3.3 we discuss an optimization procedure which requires computing the eigenvalues of $L_i$ (which are referred to as the *spectrum* of $G_i$). Computing graph spectra is a well-studied problem; we direct the reader to [9, 29]. In practice, all of the graph spectra computed for experiments in this paper took a negligible amount of time ($< 1$s) on a modern consumer-grade laptop using the scipy.linalg package [19], which in turn uses LAPACK routines for Schur decomposition of the matrix [2]. The optimal map is defined as $P$ which minimizes the matrix function

$$\begin{aligned} &\inf_{P|C(P),\alpha>0,\beta>0} E(P) \\ &= \inf_{P|C(P),\alpha>0,\beta>0} \left[(1-s)\left|\left|\frac{1}{\sqrt{\alpha}}PL_1 - \sqrt{\alpha}L_2P\right|\right|_F^2 \quad \text{“Diffusion term”} \right. \\ &\qquad \left. + s\left|\left|\frac{1}{\sqrt{\beta}}PT_1 - \sqrt{\beta}T_2P\right|\right|_F^2\right] \quad \text{“Locality term,”}^1 \end{aligned} \tag{3.3}$$

where $||\cdot||_F$ is the Frobenius norm, and $C(P)$ is a set of constraints on $P$ (in particular, we require $P^TP = I_{n_1}$, but could also impose other restrictions such as sparsity,

[1]By this we mean the notion that neighborhoods of $G_1$ should be mapped to neighborhoods of $G_2$ and vice versa.

regularity, and/or bandedness). The manifold of real-valued orthogonal $n_2 \times n_1$ matrices with $n_1 \leq n_2$ is known as the Stiefel manifold; minimization constrained to this manifold is a well-studied problem [30, 34]. This optimization problem can be thought of as measuring the agreement between processes on each graph, as mapped through $P$. The expression $PX_1 - X_2P$ compares the end result of

1. advancing process $X_2$ forward in time on $G_2$ and then using $P$ to interpolate vertex states to the smaller graph, to
2. interpolating the initial state (the all-ones vector) using $P$ and then advancing process $X_1$ on $G_1$.

Strictly speaking, the above interpretation of our objective function does not apply to the Manhattan distance matrix T of a graph, since $T$ is not a valid time evolution operator and thus is not a valid choice for $X$. However, the objective function term containing $T$ may still be interpreted as comparing travel distance in one graph to travel distance in the other, that is, we are implicitly comparing the similarity of two ways of measuring the distance of two nodes $v_k$ and $v_l$ in $G_1$:

1. the Manhattan distance, as defined above and
2. $\sum_{i=1}^{n_2} \sum_{j=1}^{n_2} p_{ik} d_{G_2}(u_i, u_j) p_{jl}$, a sum of path distances in $G_2$ weighted by how strongly $v_k$ and $v_l$ are connected, through $P$, to the endpoints of those paths, $u_i$ and $u_j$.

Parameters $\alpha$ and $\beta$ are rescaling parameters to compensate for different graph sizes; in other words, $P$ must only ensure that processes 1 and 2 above agree up to some multiplicative constant. In operator theory terminology, the Laplacian is a time evolution operator for the single particle diffusion equation: $L_i = A(G_i) - \text{diag}(1 \cdot A(G_i))$. This operator evolves the probability distribution of states of a single-particle diffusion process on a graph $G_i$ (but other processes could be used—for example, a chemical reaction network or multiple-particle diffusion). The process $L$ defines a probability-conserving master equation of nonequilibrium statistical mechanics $dp/dt = L \cdot p$ which has formal solution $p(t) = \exp(tL) \cdot p(0)$. Premultiplication by the prolongation matrix $P$ is clearly a linear operator, i.e., linear transformation from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_2}$. Thus, we are requiring $P$ which minimizes the degree to which the operator diagram

$$\begin{tikzcd} L_1 \arrow[r, "\Delta t"] \arrow[d, "P"] & {L_1}' \arrow[d, "P"] \\ L_2 \arrow[r, "\Delta t"] & {L_2}' \end{tikzcd} \qquad \text{(Diagram 1)}$$

fails to commute. $\Delta t$, of course, refers to advancement in time; see [18, Figure 1], for a more complete version of this commutative diagram for model reduction.

We thus include in our objective function terms with (1) graph diffusion and (2) graph locality as the underlying process matrices ($T$, the Manhattan distance matrix, cannot be considered a time evolution operator because it is not probability-preserving). Parameter $s$ adjusts the relative strength of these terms to each other; so we may find "fully diffuse" $P$ when $s = 0$ and "fully local" $P$ when $s = 1$. Figure 2 illustrates this tradeoff for an example prolongation problem on a pair of grid graphs, including the transition from a global optimum of the diffusion term to a global optimum of the locality term. In each case, we only require $P$ to map these processes into one another up to a multiplicative constant: $\alpha$ for the diffusion term and $\beta$ for the locality term. Exhaustive grid search over $\alpha$ and $\beta$ for a variety of prolongations

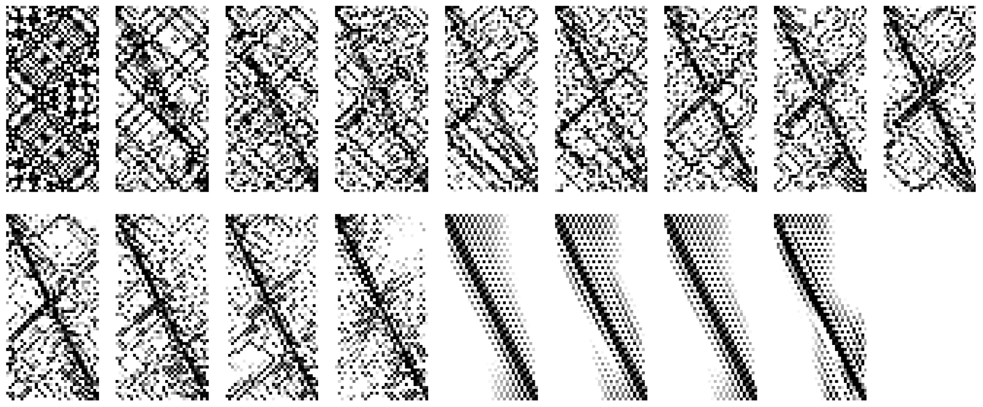

Fig. 2. *Several solutions of our objective function found by PyManOpt as* $s$*, the relative weight of the two terms of our objective function, is tuned from* 0 *(fully diffuse, top left) to* 1 *(fully local, bottom right). Within each subplot, grayscale indicates the magnitude of matrix entries. Note that the* $P$ *matrices found with* $s = 0$ *do not appear to be structured in a way which respects the locality of the original graphs, whereas the matrices with* $s = 1$ *do.*

between (a) one-dimensional (1D) grid graphs and (b) two-dimensional (2D) grid graphs of varying sizes has suggested that for prolongation problems where the $G_i$ are both paths or both grids, the best values (up to the resolution of our search, $10^{-6}$) for these parameters are $\alpha = 1.0$ and $\beta = n_1/n_2$. However, we do not expect this scaling law to hold for general graphs.

### 3.3. Numerical optimization of P matrices.

**3.3.1. Minimization method.** We tried various publicly available optimization codes to find optima of our objective function. Unless otherwise noted, all $P$ matrices found via optimization were found using PyManOpt [33], a Python language package for manifold-constrained optimization. In our experience, this package outperformed other numerical methods codes such as constrained Nelder–Mead (as implemented in Mathematica 11.3 [37] or SciPy [19]), gradient descent with projection back to the constraint manifold, or the orthogonally-constrained optimization code of [36]. More details on our comparison of these software packages may be found in the section “Comparison of numerical methods” of the Supplementary Material accompanying this paper.

**3.3.2. Initialization.** We initialize our minimization with an upper-bound solution given by the Munkres minimum-cost matching algorithm; the initial $P$ is $m^*(L_1, L_2)$ as defined in (3.2), i.e., the binary matrix where an entry $P_{(i,j)}$ is 1 if the pair $(i, j)$ is one of the minimal-cost pairs selected by the minimum-cost assignment algorithm, and 0 otherwise. While this solution is, strictly speaking, minimizing the error associated with mapping the spectrum of one graph into the spectrum of the other (rather than actually mapping a process running on one graph into a process on the other) we found it to be a reasonable initialization, outperforming both random restarts and initialization with the appropriately sized block matrix $\binom{I}{0}$. As detailed further in section 5, the $P$ found as a solution to this matching problem provides an upper bound for the full orthogonality-constrained optimization problem.

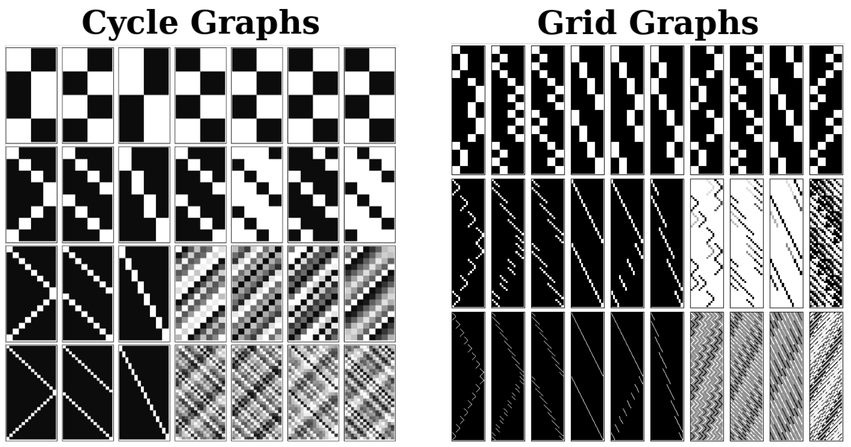

FIG. 3. *Examples of $P$ matrices for cycle graph (left) and grid graph (right) prolongation problems of various sizes, which can be generated by closed-form representations dependent on problem size. Within each of the top and bottom plots, columns represent a series of matrices each generated by a particular numerical recipe, with rows representing increasing sizes of prolongation problem. Each matrix plot is a plot of the absolute value of matrix cell values. These closed-form representations were initially found as local minima of our objective function on small problems and then generalized to closed-form representations. For the "Cycle Graphs" section, the prolongation problems were between cycle graphs of sizes $n_1 = 2, 4, 8, 16$ and $n_2 = 2 * n_1$. Columns 1–3 were solutions found with $s = 1$ (fully local), and the rest were found with $s = 0$ (fully diffuse). For the "Grid Graphs" section, the prolongation problems were between grids of size $(n_1, n_1)$ to grids of size $(2n_1, 2n_1)$ for $n_1$ in $4, 8, 16$. Columns 1–6 are fully local and columns 7–10 are fully diffuse, respectively. As in Figure 2, grayscale values indicate the magnitude of each matrix entry.*

**3.3.3. Precomputing $P$ matrices.** For some structured graph lineages it may be possible to derive formulaic expressions for optimal $P$ and $\alpha$, as a function of the lineage index. For example, during our experiments we discovered species of $P$ which are local minima of prolongation between path graphs, cycle graphs, and grid graphs. A set of these outputs is shown in Figure 2. They feature various diagonal patterns as naturally idealized in Figure 3. These idealized versions of these patterns are all also empirical local minima of our optimization procedure, for $s = 0$ or $s = 1$, as indicated. Each column of Figure 3 provides a regular family of $P$ structures for use in our subsequent experiments in section 4. We have additionally derived closed-form expressions for global minima of the diffusion term of our objective function for some graph families (cycle graphs and grid graphs with periodic boundary conditions). Proof of the optimality of these solutions may be found in the supplementary materials which accompany this paper. However, in practice these global minima are nonlocal (in the sense that they are not close to optimizing the locality term) and thus may not preserve learned spatial rules between weights in levels of our hierarchy.

Examples of these formulaic $P$ matrices can be seen in Figure 3. Each column of that figure shows increasing sizes of $P$ generated by closed-form solutions which were initially found by solving smaller prolongation problems (for various graph pairs and choices of $s$) and generalizing the solution to higher $n$. Many of these examples are similar to what a human being would design as interpolation matrices between cycles and periodic grids. However, (a) they are valid local optima found by our optimization code and (b) our approach generalizes to processes running on more complicated or nonregular graphs, for which there may not be an obvious a priori choice of prolongation operator.

We highlight the best of these multiple species of closed-form solution, for both cycle graphs and grid graphs. The interpolation matrix-like $P$ seen in the third column of the "Cycle Graphs" section, or the sixth column of the "Grid Graphs" section of Figure 3, were the local optima with lowest objective function value (with $s = 1$, i.e., they are fully local). As the best optima found by our method(s), these matrices were our choice for line graph and grid graph prolongation operators in our neural network experiments, detailed in section 4. We reiterate that in those experiments we do not find the $P$ matrices via any optimization method—since the neural networks in question have layer sizes of order $10^3$, finding the prolongation matrices from scratch may be computationally difficult. Instead, we use the solutions found on smaller problems as a recipe for generating prolongation matrices of the proper size.

Furthermore, given two graph lineages $G_1^{(1)}, G_1^{(2)}, G_1^{(3)}, \ldots$ and $G_2^{(1)}, G_2^{(2)}, G_2^{(3)}, \ldots$, and sequences of optimal matrices $P_1^{(1)}, P_1^{(2)}, P_1^{(3)}, \ldots$ and $P_2^{(1)}, P_2^{(2)}, P_2^{(3)}, \ldots$, mapping between successive members of each, we can construct $P$ which are related to the optima for prolonging between members of a new graph lineage which is comprised of the levelwise graph box product of the two sequences. We show in section 5.2, Corollary 5.2 conditions under which the value of the objective function at $P_{\text{box}}^{(i)} = P_1^{(i)} \otimes P_2^{(i)}$ is an upper bound of the optimal value for prolongations between members of the lineage $G_1^{(1)} \Box G_2^{(1)}, G_1^{(2)} \Box G_2^{(2)}, G_1^{(3)} \Box G_2^{(3)}, \ldots$. We leave open the question of whether such formulaic $P$ exist for other families of structured graphs (complete graphs, $k$-partite graphs, etc.). Even in cases where formulaic $P$ are not known, the computational cost of numerically optimizing over $P$ may be amortized, in the sense that once a $P$-map is calculated, it may be used in many different hierarchical neural networks or indeed many different multiscale models.

**3.4. Multiscale artificial neural network algorithm.** In this section we describe the MsANN training procedure, both in prose and in pseudocode (Algorithm 3.1). Let $\mathcal{M}_0 \ldots \mathcal{M}_L$ be a sequence of neural network models with identical "aspect ratios" (meaning the sizes of each layer relative to other layers in the same model) but differing input resolution, so that $\mathcal{M}_0$ operates at the finest scale and $\mathcal{M}_L$ at the coarsest. For each model $\mathcal{M}_l$, let $\theta_0^{(l)}, \theta_1^{(l)}, \ldots, \theta_{n_{\text{vars}}-1}^{(l)}$ be a list of the $n_{\text{vars}}$ network parameters (each in matrix or vector form) in some canonical order which is maintained across all scales. Let the symbol $\mathcal{P}_j^{(l)}$ represent either of the following:

- If the network parameters $\theta_j^{(i)}$ at levels $i = 0, \ldots, L$ are weight matrices between layers $m_1$ and $m_2$ of each hierarchy, then $\mathcal{P}_j^{(l)}$ represents a pair of matrices $(P_{\text{input}_j}^{(l)}, P_{\text{output}_j}^{(l)})$, such that
  - $P_{\text{input}_j}^{(l)}$ prolongs or restricts between possible values of nodes in layer $m_1$ of model $\mathcal{M}_l$, and values of nodes in layer $m_1$ of model $\mathcal{M}_{l+1}$.
  - $P_{\text{output}_j}^{(l)}$ does the same for possible values of nodes in layer $m_2$ of each model.
- If the network parameters $\theta_j^{(i)}$ at levels $i = 0, \ldots, L$ are bias vectors which are added to layer $m$ of each hierarchy, then $\mathcal{P}_j^{(l)}$ represents a single $P_j^{(l)}$ which prolongs or restricts between possible values of nodes in layer $m$ of model $\mathcal{M}_l$, and values of nodes in layer $m$ of model $\mathcal{M}_{l+1}$.

As a concrete example, for a hierarchy of single-layer networks $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2$, each with one weight matrix $W^{(l)}$ and one bias vector $b^{(l)}$, we could have $\theta_0^{(l)} = W^{(l)}, \theta_1^{(l)} = b^{(l)}$ for each $\mathcal{M}_l$. $\mathcal{P}_0^{(0)}$ would represent a pair of matrices which map between the space of possible values of $W^{(0)}$ and the space of possible values of $W^{(1)}$ in a manner detailed in the next section. On the other hand, $\mathcal{P}_1^{(0)}$ would represent a single matrix which maps between $b^{(0)}$ and $b^{(1)}$. Similarly, $\mathcal{P}_0^{(1)}$ would map between $W^{(1)}$ and $W^{(2)}$, and $\mathcal{P}_1^{(1)}$ between $b^{(1)}$ and $b^{(2)}$. In section 3.4.2, we describe a general procedure for training such a hierarchy according to standard multilevel modeling schedules of refinement and coarsening, with the result that the finest network, informed by the weights of all coarser networks, requires fewer training examples.

**3.4.1. Weight prolongation and restriction operators.** In this section we introduce the prolongation and restriction operators for neural network weight and bias optimization variables in matrix or vector form, respectively.

For a 2D matrix of weights $W$, define

$$\begin{aligned} \mathrm{Pro}_{\mathcal{P}} \circ W \equiv \mathrm{Pro}_{(P_{\text{input}},P_{\text{output}})} \circ W &\equiv P_{\text{input}} W P_{\text{output}}^T, \\ \mathrm{Res}_{\mathcal{P}} \circ W \equiv \mathrm{Res}_{(P_{\text{input}},P_{\text{output}})} \circ W &\equiv P_{\text{input}}^T W P_{\text{output}}, \end{aligned} \tag{3.4}$$

where $P_{\text{input}}$ and $P_{\text{output}}$ are each prolongation maps between graphs which respect the structure of the spaces of inputs and outputs of $W$, i.e., whose structure is similar to the structure of correlations in that space. Further research is necessary to make this notion more precise. In our experiments on autoencoder networks in section 4, we use example problems with an obvious choice of graph to use. In these 1D and 2D machine vision tasks, where we expect each pixel to be highly correlated with the activity of its immediate neighbors in the grid, 1D and 2D grids are clear choices of graphs for our prolongtion matrix calculation. Other choices may lead to similar results; for instance, we speculate that since neural network weight matrices may be interpreted as the weights of a multipartite graph of connected neurons in the network, these graphs could be an alternate choice of structure to prolong/restrict between. We leave for future work the development of automatic methods for determining these structures.

Note that the Pro and Res linear operators satisfy $\mathrm{Res}_{\mathcal{P}} \circ \mathrm{Pro}_{\mathcal{P}} = I$, the identity operator, so $\mathrm{Pro}_{\mathcal{P}} \circ \mathrm{Res}_{\mathcal{P}}$ is a projection operator.

For a 1D matrix of biases $b$, define

$$\begin{aligned} \mathrm{Pro}_{\mathcal{P}} \circ b &= P \cdot b, \\ \mathrm{Res}_{\mathcal{P}} \circ b &= P^T \cdot b, \end{aligned} \tag{3.5}$$

where, as before, we require that $P$ be a prolongation matrix between graphs which are appropriate for the dynamics of the network layer where $b$ is applied. Again $\mathrm{Res}_{\mathcal{P}} \circ \mathrm{Pro}_{\mathcal{P}} = I$.

Given such a hierarchy of models $\mathcal{M}_0 \dots \mathcal{M}_L$, and appropriate Pro and Res operators as defined above, we define an *MsANN* to be a neural network model with the same layer and parameter dimensions as the largest model in the hierarchy, where each layer parameter $\Theta_j$ is given by a sum of prolonged weight matrices from level $j$ of each of the models defined above:

$$\Theta_j = \theta_j^{(0)} + \mathrm{Pro}_{1\to 0} \circ \theta_j^{(1)} + \mathrm{Pro}_{2\to 0} \circ \theta_j^{(2)} \dots \mathrm{Pro}_{L\to 0} \circ \theta_j^{(L)}. \tag{3.6}$$

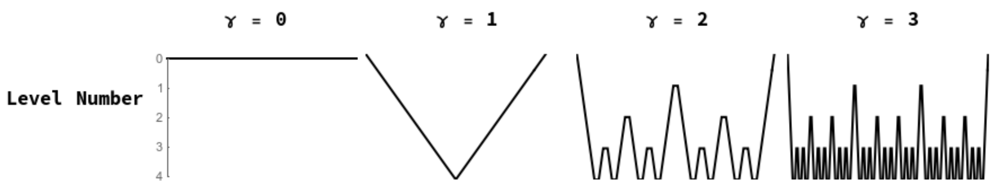

FIG. 4. *Visits to models in a hierarchy of neural networks realized by several values of the recursion frequency parameter $\gamma$. The $\gamma = 1$ case and the $\gamma = 2$ case are referred to as "V-cycles" and "W-cycles," respectively. Each time the multilevel training procedure visits a level, it performs some number, k, of smoothing steps (i.e., gradient descent at that resolution) at that model.*

Here we are using $\text{Pro}_{k\to 0}$ as a shorthand to indicate composed prolongation from model $k$ to model 0, so if $\theta_j^{(i)}$ are weight variables, we have (by (3.4))

$$\begin{aligned} \Theta_j = \theta_j^{(0)} &+ P^{(0)}_{\text{input}_j}\theta_j^{(1)}\left(P^{(0)}_{\text{output}_j}\right)^T \\ &+ P^{(0)}_{\text{input}_j}P^{(1)}_{\text{input}_j}\theta_j^{(2)}\left(P^{(1)}_{\text{output}_j}\right)^T\left(P^{(0)}_{\text{output}_j}\right)^T \\ &+ \cdots + \left(P^{(0)}_{\text{input}_j}\dots P^{(L-1)}_{\text{input}_j}\theta_j^{(L)}\left(P^{(L-1)}_{\text{output}_j}\right)^T\dots\left(P^{(0)}_{\text{output}_j}\right)^T\right), \end{aligned} \tag{3.7}$$

and if $\theta_j^{(i)}$ are bias variables, we have (by (3.5))

$$\Theta_j = \theta_j^{(0)} + P^{(0)}_{\text{bias}_j}\theta_j^{(1)} + P^{(0)}_{\text{bias}_j}P^{(1)}_{\text{bias}_j}\theta_j^{(2)} + \cdots + \left(P^{(0)}_{\text{bias}_j}P^{(1)}_{\text{bias}_j}\dots P^{(L-1)}_{\text{bias}_j}\theta_j^{(L)}\right). \tag{3.8}$$

We note that matrix products such as $P^{(0)}_{\text{input}_j}\dots P^{(k)}_{\text{input}_j}$ need only be computed once during model construction.

**3.4.2. Multiscale artificial neural network training.** The MsANN algorithm is defined in terms of a recursive "cycle" that is analogous to one epoch of default neural network training. Starting with $\mathcal{M}_0$ (i.e., the finest model in the hierarchy), we call the routine MsANNCycle(0), which is defined recursively. At any level $l$, MsANNCycle trains the network at level $l$ for $k$ batches of training examples, recurses by calling MsANNCycle($l + 1$), and then returns to train for $k$ further batches at level $l$. The number of calls to MsANNCycle($l + 1$) inside each call to MsANNCycle($l$) is given by a parameter $\gamma$.

This is followed by additional training at the refined scale; this process is normally [35] referred to by the multigrid methods community as "restriction" and "prolongation" followed by "smoothing." The multigrid methods community additionally has special names for this type of recursive refining procedure with $\gamma = 1$ ("V-Cycles") and $\gamma = 2$ ("W-Cycles"). See Figure 4 for an illustration of these contraction and refinement schedules. In our numerical experiments below, we examine the effect of this parameter on multigrid network training.

Neural network training with gradient descent requires computing the gradient of the error $E$ between the network output and target with regard to the network parameters. This gradient is computed by taking a vector of error for the nodes in the output layer, and *backpropagating* that error backward through the network layer by layer to compute the individual weight matrix and bias vector gradients. An individual network weight or bias term $w$ is then adjusted using gradient descent, i.e., the new value $w'$ is given by $w' = w - \eta\frac{dE}{dw}$, where $\eta$ is a learning rate or step size. Several techniques can be used to dynamically change learning rate during model training—we refer the reader to [5] for a description of these techniques and backpropagation in general.

Our construction of the MsANN model above did not make use of the Res (restriction) operator—we show here how this operator is used to compute the gradient of the coarsened variables in the hierarchy. This can be thought of as continuing the process of backpropagation through the Pro operator. For these calculations we assume $\Theta_j$ is a weight matrix, and derive the gradient for a particular $\theta_j^{(k)}$. For notational simplicity we rename these matrices $W$ and $V$, respectively. We also collapse the matrix products

$$P^{(\text{input})} = P^{(0)}_{\text{input}_j} P^{(1)}_{\text{input}_j} \cdots P^{(k)}_{\text{input}_j}, \tag{3.9}$$

$$\left(P^{(\text{output})}\right)^T = \left(P^{(L-1)}_{\text{output}_j}\right)^T \left(P^{(L-2)}_{\text{output}_j}\right)^T \cdots \left(P^{(0)}_{\text{output}_j}\right)^T. \tag{3.10}$$

Let $\frac{dE}{dW}$ be a matrix where $\left(\frac{dE}{dW}\right)_{mn} = \frac{dE}{dw_{mn}}$, calculated via backpropagation as described above. Then, for some $m, n$,

$$\begin{aligned} \frac{dw_{mn}}{dv_{kl}} &= \frac{d}{dv_{kl}}(\cdots + \text{Pro} \circ V + \cdots)_{mn} \\ &= \frac{d}{dv_{kl}}(\cdots + \text{Pro}_{k\to 0} \circ V + \cdots)_{mn} = \frac{d}{dv_{kl}}(\text{Pro}_{k\to 0} \circ V)_{mn} \\ &= \frac{d}{dv_{kl}}\left(P^{(\text{input})} V \left(P^{(\text{output})}\right)^T\right)_{mn} = \frac{d}{dv_{kl}}\left(\sum_{a,b} p^{(\text{input})}_{ma} v_{ab} p^{(\text{output})}_{nb}\right) \\ &= \left(p^{(\text{input})}_{mk} p^{(\text{output})}_{nl}\right). \end{aligned} \tag{3.11}$$

Then,

$$\begin{aligned} \frac{dE}{dv_{kl}} &= \sum_{m,n} \frac{dE}{dw_{mn}} \frac{dw_{mn}}{dv_{kl}} \\ &= \sum_{m,n} \frac{dE}{dw_{mn}} p^{(\text{input})}_{mk} p^{(\text{output})}_{nl} \\ &= \left(\left(P^{(\text{input})}\right)^T \frac{dE}{dW} P^{(\text{output})}\right)_{kl}, \end{aligned} \tag{3.12}$$

and so

$$\frac{dE}{dV} = \left(P^{(\text{input})}\right)^T \frac{dE}{dW} P^{(\text{output})},$$

and, therefore, finally

$$\frac{dE}{dV} = \text{Res}_{0\to k} \circ \frac{dE}{dW}, \tag{3.13}$$

where Res is as in (3.4).

We also note here that our code implementation of this procedure does not make explicit use of the Res operator; instead, we use the automatic differentiation capability of Tensorflow [1] to compute this restricted gradient. This is necessary because

**Algorithm 3.1.** One "cycle" of the MsANN procedure.

```
Procedure MsANNCycle($l$):
    Train model $\mathcal{M}_l$ for $k$ batches, where each consists of the following:
        1. Feed examples through the network in feed-forward mode.
        2. Compute error $E$ between network output and target.
        3. Use the classical backpropagation algorithm to compute the gradient of
           top-level parameter $\Theta_j$ w.r.t. this error.
        4. Use the appropriate Res operations to compute the gradient of $E$ w.r.t.
           the parameters in $\mathcal{M}_l$, as described in (3.13).
    if max_depth has not been reached then
        for $1 \leq i \leq \gamma$ do
            MsANNCycle($l + 1$);
            Train model $\mathcal{M}_l$ for $k$ batches, as above
        end
    end
```

data is supplied to the model, and error is calculated, at the finest scale only. Hence we calculate the gradient at this scale and restrict it to the coarser layers of the model. It may be possible to feed coarsened data through only the coarser layers of the model, eliminating the need for computing the gradient at the finest scale, but we do not explore this method in this paper.

## 4. Machine learning experiments.

### 4.1. Preliminaries.

We present four experiments using this multiscale neural network method. All of the experiments below demonstrate that our multigrid method outperforms default training (i.e., training only the finest-scale network), in terms of the number of training examples (summed over all scales) needed to reach a particular mean-squared error (MSE) value. We perform two experiments with synthetic machine vision tasks, as well as two experiments with benchmark image datasets for machine learning. While all of the examples presented here are autoencoder networks (networks whose training task is to reproduce their input at the output layer, while passing through a bottleneck layer or layers), we do not mean to imply that MsANN techniques are constrained to autoencoder networks. All network training uses the standard backpropagation algorithm to compute training gradients, and this is the expected application domain of our method. Autoencoding image data is a good choice of machine learning task for our experiments for two main reasons. First, autoencoders are symmetric and learn to reproduce their input at their output. Other Machine Learning (ML) models (for instance, neural networks for classification) have output whose nodes are not spatially correlated, and it is not yet clear if our approach will generalize to this type of model. Second, since the single- and double-object machine vision tasks operate on synthetic data, we can easily generate an arbitrary number of samples from the data distribution, which was useful in the early development of this procedure. Our initial successes on this synthetic data led us to try the same task with a standard benchmark real-world dataset. For each experiment, we use the following measure of computational cost to compare relative performance. Let $|\mathcal{M}|$ be the number of trainable parameters in model $\mathcal{M}$. We compute the cost of a training step of the weights in model $\mathcal{M}_k$ using a batch of size $b$ as $\frac{|\mathcal{M}_k|}{|\mathcal{M}_0|}b$. The total cost $C(t)$ of training at step $t$ is the sum of this cost over all training steps thus far at all scales. This cost is motivated by the fact that the number of multiply operations for backpropagation is $O(nm)$ in the total number of network parameters $m$ and training examples $n$, so we are adding up the relative cost of using a batch of size $b$ to adjust the weights in model $\mathcal{M}_k$, as compared to the cost of using that same batch to adjust the weights in $\mathcal{M}_0$.

**4.2. Simple machine vision task.** As an initial experiment in the capabilities of hierarchical neural networks, we first try two simple examples: finding lower-dimensional representation of two artificial datasets. In both cases, we generate synthetic data by uniformly sampling from

1. the set of binary-valued vectors with one "object" comprising a contiguous set of pixels one-eighth as long as the entire vector set to 1, and the rest zero, and
2. the set of vectors with two such nonoverlapping objects.

In each case, the number of possible unique data vectors is quite low: for inputs of size 1024, we have $1024 - 128 = 896$ such vectors. Thus, for both of the synthetic datasets we add binary noise to each vector, where each "pixel" of the input has an independent chance of firing spuriously with $p = 0.05$. This noise in included only in the input vector, making these networks *denoising autoencoders*: models whose task is to remove noise from an input image.

**4.2.1. Single-object autoencoder.** We first test the performance of this procedure on a simple machine vision task. The neural networks in our hierarchy of models each have layer size specification (in number of units) $[2^n, 2^{n-2}, 2^{n-3}, 2^{n-2}, 2^n]$ for $n$ in $\{10, \dots, 6\}$, with a bias term at each layer and sigmoid logistic activation. We present the network with binary vectors which are 0 everywhere except for a contiguous segment of indices of length $2^{n-3}$ which are set to 1, with added binary noise as described above. The objective function to minimize is the MSE between the input and output layers. Each model in the hierarchy is trained using AdamOptimizer [21] in Tensorflow [1], with learning rate $\alpha = 0.0005$.

The results of this experiment are plotted in Figure 5 and summarized in Table 1. We perform multiple runs of the entire training procedure with differing values of $k$ (the number of smoothing steps), $\gamma$ (the multigrid cycle parameter), and $L$ (depth of hierarchy). Notably, nearly all multigrid schedules demonstrate performance gains over the default network (i.e., the network which trains only at the $l = 0$ scale), with more improvement for higher values of $k$, $L$, and $\gamma$. The hierarchy which learned most rapidly was the deepest model ($L = 6$) with $k = 4$ and $\gamma = 3$. Those multigrid models which did not improve over the default network were only slightly more computationally expensive per unit of accuracy than their default counterparts, and the multigrid models which did improve, improved significantly.

**4.2.2. Double-object autoencoder.** We repeat the above experiment with a slightly more difficult machine vision task—the network must learn to denoise an image with two (nonoverlapping) "objects" in the visual field. We use the same network structure and training procedure, and note that we see again (plotted in Figure 5 and summarized in Table 2) that the hierarchical model is more efficient, reaching lower error in the same amount of computational cost $C(t)$. The multigrid neural networks again typically learn much more rapidly than the nonmultigrid models.

**4.3. MNIST.** To supplement the above synthetic experiments with one using real-world data, we perform the same experiment with an autoencoder for the MNIST handwritten digit dataset [24, 25]. In this case, rather than the usual MNIST classifi-

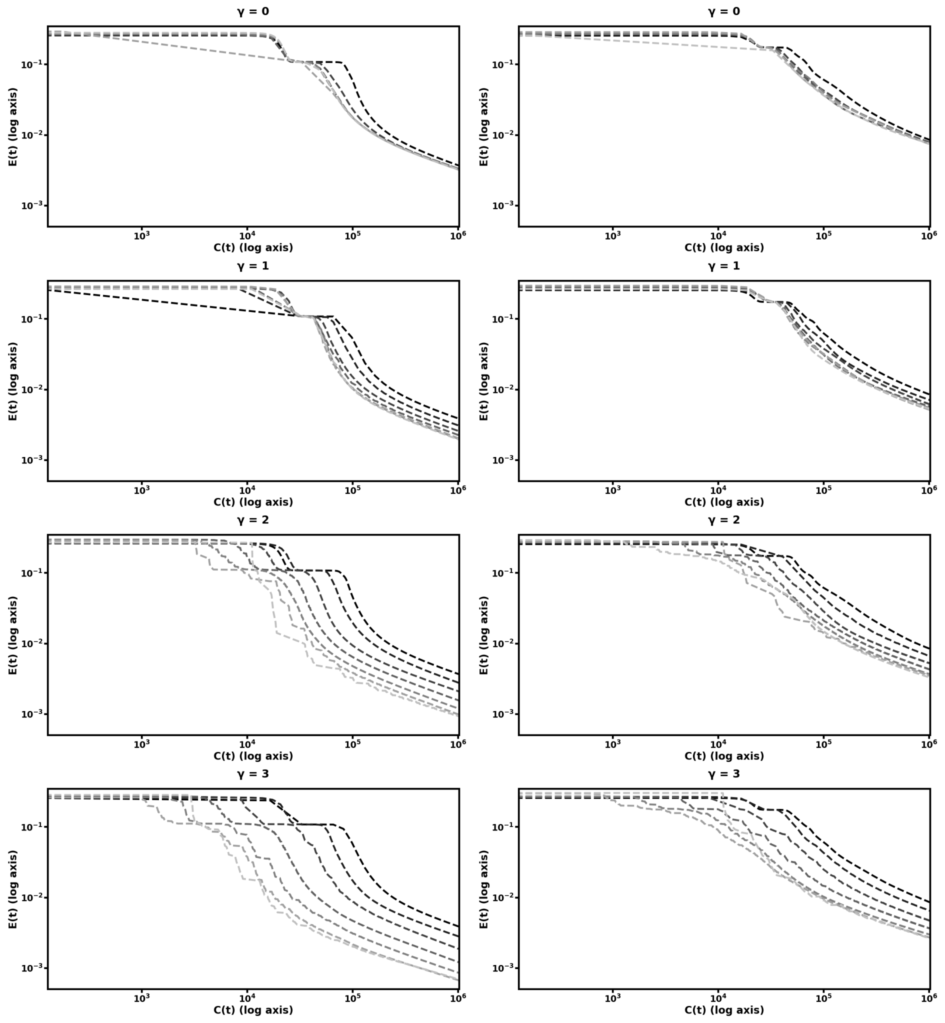

Fig. 5. *Log-log plots of accuracy $E(t)$ as a function of training cost $C(t)$ attained by a variety of hierarchical neural networks training on a simple machine vision task, demonstrating that deeper hierarchies with more mutligrid behavior learn more rapidly. Plots are ordered from top to bottom in increasing depth of recursion parameter $\gamma$; left plots are the single-object experiments and right plots are the double-object experiments. Within each plot, different curves represent different values of the depth of hierarchy, from $L = 6$ (lightest) to $L = 0$ (darkest). Each line is the best run for that pair $(L, \gamma)$ over all choices of $k$ (number of smoothing steps at each level) in* $\{1, 2, 4, 8, 16, 32, 64, 128\}$.

cation task, we use an autoencoder to map the MNIST images into a lower-dimensional ($d = 128$) space with good reconstruction. We use the same network structure as in the 1D vision example; also as in that example, each network in the hierarchy is constructed of fully connected layers with bias terms and sigmoid activation, and smoothing steps are performed with RMSProp [15] with learning rate 0.0005. The only difference is that in this example we do not add noise to the input images, since the dataset is larger by two orders of magnitude.

In this experiment, we see (in Figure 6 and Table 3) similar improvement in efficiency. Table 3 summarizes these results: the best multilevel models learned more rapidly and achieved lower error than their single-level counterparts, whereas the worst

Table 1

*Best performance (on validation dataset for the one-object autoencoding task) by any combination of parameters in our sweep over values for $\gamma$ (recursion constant), $L$ (depth of network), and $k$ (number of batches processed at each visit to each level). We report the final MSE for both the best and worst combination of these parameters, as well as for default training. We also report the best combination of parameters. The second row indicates the cost $C(t)$ necessary to train each model to $\frac{1}{10}$ of the error at which it began. The best MsANN network reaches this threshhold in an order of magnitude less cost, and its final error is roughly half that of the default model, demonstrating clear improvement over training without multigrid.*

| | Best MsANN | Worst MsANN | Default | Best MsANN params |
|---|---|---|---|---|
| Final MSE | $6.612 \times 10^{-4}$ | $4.431 \times 10^{-3}$ | $3.654 \times 10^{-3}$ | $(\gamma = 3, L = 5, k = 004)$ |
| Cost to $\frac{1}{10}$ MSE | $7.342 \times 10^{3}$ | $1.640 \times 10^{5}$ | $1.266 \times 10^{5}$ | $(\gamma = 3, L = 6, k = 004)$ |

Table 2

*Best performance (on validation dataset for the two-object autoencoding task). Again the MsANN network demonstrates performance and accuracy gains over neural network training alone. See Table* 1.

| | Best MsANN | Worst MsANN | Default | Best MsANN params |
|---|---|---|---|---|
| Final MSE | $2.576 \times 10^{-3}$ | $8.998 \times 10^{-3}$ | $8.816 \times 10^{-3}$ | $(\gamma = 3, L = 6, k = 002)$ |
| Cost to $\frac{1}{10}$ MSE | $2.433 \times 10^{4}$ | $2.623 \times 10^{5}$ | $2.216 \times 10^{5}$ | $(\gamma = 3, L = 6, k = 016)$ |

multilevel models performed on par with the default model. Because the MNIST data is comprised of 2D images, we tried using $P$ matrices which were the optima of prolongation problems between grids of the appropriate sizes, in addition to the same 1D $P$ used in the prior two experiments. The difference in performance between these two choices of underlying structure for the prolongation maps can be seen in Figure 6. With either approach, we see similar results to the synthetic data experiment, in that more training steps at the coarser layers results in improved learning performance of the finer networks in the hierarchy. However, the matrices optimized for 2D prolongation perform marginally better than their 1D cousins—in particular, the multigrid hierarchy with 2D prolongations took 60% of the computational cost to reduce its error to $\frac{1}{10}$ of its original value, as compared to the 1D version. We explore the effect of varying the strategy used to pick $P$ in subsection 4.4.

**4.4. Experiments of choice of *P*.** To further explore the role of the structure of $P$ in these machine learning models, we compare the performance of several MsANN models with $P$ generated according to various strategies. Our initial experiment on the MNIST dataset used the exact same hierarchical network structure and prolongation/restriction operators as the example with 1D data, and yielded marginal computational benefit. We were thus motivated to try this learning task with prolongations which are designed for 2D grid-based model architectures, as well as trying unstructured (random orthogonal) matrices as a baseline. More precisely, our 1D experiments used $P$ matrices resembling those in column 3 of the "Cycle Graphs" section of Figure 3. We instead, for the MNIST task, used $P$ matrices like those in column 6 of the "Grid Graphs" section of the same figure. In Figure 7, we illustrate the difference in these choices for the MNIST training task, with the same choice of multigrid training parameters: $(L = 6, \gamma = 3, k = 1)$. We compare the following strategies for generating $P$:

1. as local optima of a prolongation problem between 1D grids, with periodic

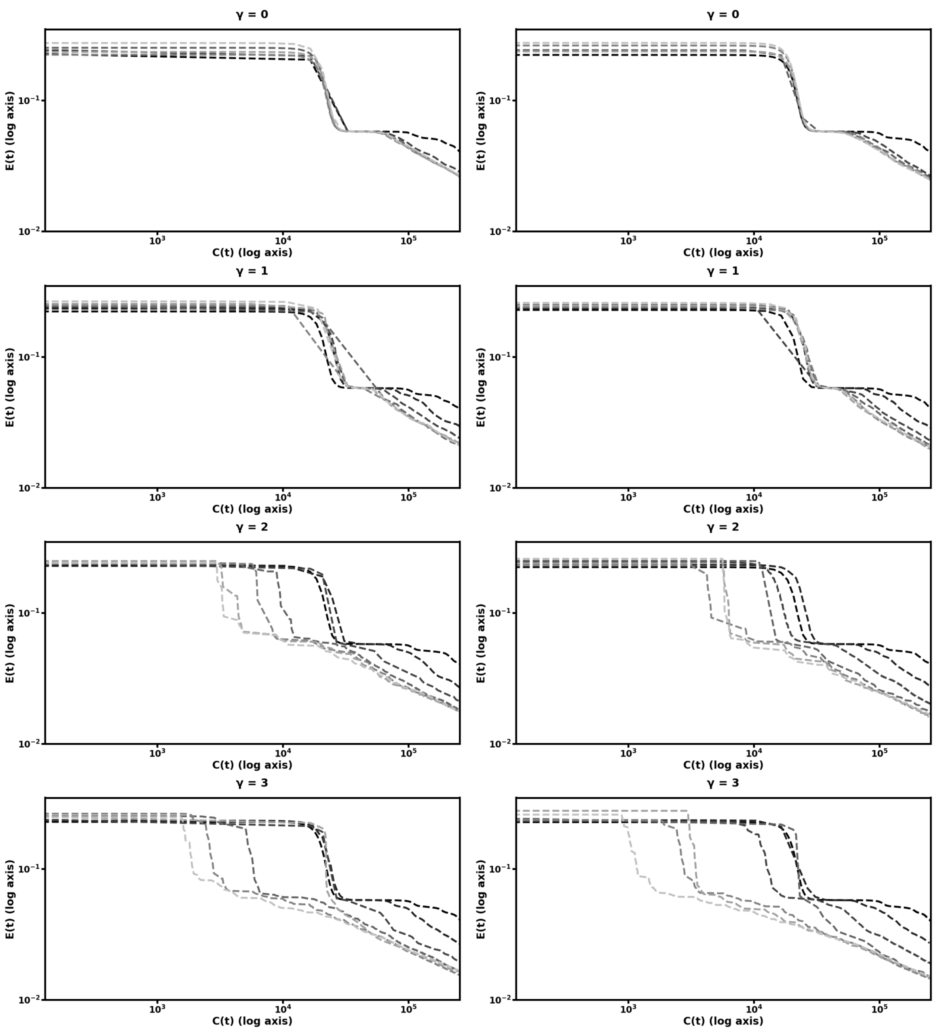

FIG. 6. *Log-log plots of MSE $E(t)$ on MNIST autoencoding task as a function of computational cost $C(t)$; the left plots represent multigrid performed with path graph prolongations for each layer while the right plots used grid-based prolongation. While both approaches show gains over default learning in both speed of learning and final error value, the one which respects the spatial structure of the input data improves more rapidly. Subplot explanations are the same as in Figure* 5.

boundary conditions;

2. as local optima of a prolongation problem between 2D grids, with periodic boundary conditions;
3. as in 2, but shuffled along the first index of the array.

Strategy 3 was chosen to provide the same degree of connectivity between each coarse variable and its related fine variables as strategy 2, but in random order, i.e., connected in a way which is unrelated to the 2D correlation between neighboring pixels. We see in Figure 7 that the two strategies utilizing local optima outperform both the randomized strategy and the default training (training only the finest scale). Furthermore, strategy 2 outperforms strategy 1, although the latter eventually catches up at the end of training, when coarse-scale weight training has diminishing marginal

Table 3

*Best performance (on validation dataset for the MNIST autoencoding task). See Table* 1. *Upper section represents scores attained by an MsANN with path-based prolongation, lower section represents grid-based prolongation. Entries marked N/A did not reach* $\frac{1}{10}$ *of their initial error during training.*

| Path-based $P$ Matrices | | | | |
|---|---|---|---|---|
| | Best MsANN | Worst MsANN | Default | Best MsANN params |
| Final MSE | $1.547 \times 10^{-2}$ | $4.605 \times 10^{-2}$ | $4.171 \times 10^{-2}$ | $(\gamma = 3, L = 4, k = 008)$ |
| Cost to $\frac{1}{10}$ MSE | $7.207 \times 10^{4}$ | N/A | N/A | $(\gamma = 3, L = 5, k = 032)$ |
| **Grid-Based $P$ Matrices** | | | | |
| | Best MsANN | Worst MsANN | Default | Best MsANN params |
| Final MSE | $1.436 \times 10^{-2}$ | $4.620 \times 10^{-2}$ | $4.132 \times 10^{-2}$ | $(\gamma = 3, L = 4, k = 002)$ |
| Cost to $\frac{1}{10}$ MSE | $5.095 \times 10^{4}$ | N/A | N/A | $(\gamma = 3, L = 6, k = 128)$ |

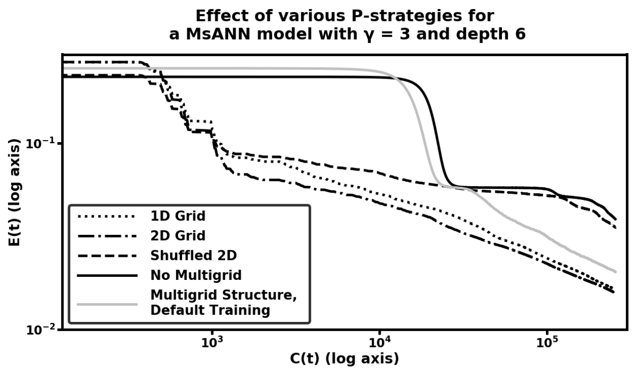

Fig. 7. *Comparison of several choices of Pro and Res operators for a multiscale neural network training experiment on MNIST data. Two choices for P which are local optima of prolongation problems demonstrate more efficient training than default, while two strategies perform worse: multigrid training with random P matrices, and training all varibles in the hierarchy simultaneously.*

returns. The random strategy is initially on par with the two optimized ones (we speculate that this is due to the ability to affect many fine-scale variables at once, even in random order, which may make the gradient direction easier to travel), but eventually falls behind, at times being less efficient than default training. We leave for further work the question of whether there are choices of prolongation problems whose solutions are even more efficient prolongation/restriction operators for this machine learning task. We also compare all of the preceeding models to a model which has the same structure as a MsANN model (a hierarchy of coarsened variables with Pro and Res operators between them), but which was trained by training all variables in the model simultaneously. This model performs on par with the default model, illustrating the need for the multilevel training schedule dictated by the choice of $\gamma$.

**4.5. Summary.** We see uniform improvement (as the parameters $L$ and $\gamma$ are increased) in the rate of neural network learning when models are stacked in the type of multiscale hierarchy we define in (3.4) and (3.5), despite the diversity of machine learning tasks we examine. Furthermore, this improvement is marked: the hierarchical models both learn more rapidly than training without multigrid and have final error lower than the default model. In many of our test cases, the hierarchical models reached the same level of MSE as the default in more than an order of magnitude fewer training examples, and continued to improve, surpassing the final level of error reached by the default network. Even in the worst case, our hierarchical model structure performed on par with neural networks which did not incorporate our weight prolongation and restriction operators. We leave the question of finding optimal $(L, \gamma, k)$ for future work—see section 6 for further discussion. Finally, we note that the model(s) in the experiments presented in section 4.2 were essentially the same MsANN models (same set of $L, \gamma, k$, and same set of $P$ matrices), and showed similar performance gains on two different machine vision problems, indicating that it may be possible to develop general MsANN model-creation procedures that are applicable to a variety of problems (rather than needing to be hand-tuned).

## 5. Upper bounds for diffusion term.

In this section, we consider two theoretical concerns:

1. invariance in Frobenius norm of diffusion term solutions under transformation to a spectral basis; and
2. decoupling a prolongation problem between graph products into a sum of prolongation problems of the two sets of graph factors.

We will here rely heavily on various properties of the Kronecker sum and product of matrices which may be found in [17, section 11.4].

### 5.1. Invariance of objective function evaluation of *P* under eigenspace transformation.

For the purpose of the calculations in this section, we restrict ourselves to the "diffusion" term of the objective function of (3.3) (the term which coerces two diffusion processes to agree), which we will write as

$$D_{P,\alpha}(G_1, G_2) = \left|\left| \frac{1}{\sqrt{\alpha}} P L_1 - \sqrt{\alpha} L_2 P \right|\right|_F . \tag{5.1}$$

Because $L_1$ and $L_2$ are each real and symmetric, they may both be diagonalized as $L_i = U_i \Lambda_i U_i^T$, where $U_i$ is a rotation matrix and $\Lambda_i$ is a diagonal matrix with the eigenvalues of $L_i$ on the diagonal. Substituting into (5.1), and letting $\tilde{P} = U_2^T P U_1$, we have

$$\begin{aligned} D_{P,\alpha}(G_1, G_2) &= \left|\left| \frac{1}{\sqrt{\alpha}} P L_1 - \sqrt{\alpha} L_2 P \right|\right|_F \\ &= \left|\left| \frac{1}{\sqrt{\alpha}} P U_1 \Lambda_1 U_1^T - \sqrt{\alpha} U_2 \Lambda_2 U_2^T P \right|\right|_F \\ &= \left|\left| \frac{1}{\sqrt{\alpha}} \left(U_2^T P U_1\right) \Lambda_1 - \sqrt{\alpha} \Lambda_2 \left(U_2^T P U_1\right) \right|\right|_F \\ &= \left|\left| \frac{1}{\sqrt{\alpha}} \tilde{P} \Lambda_1 - \sqrt{\alpha} \Lambda_2 \tilde{P} \right|\right|_F , \end{aligned} \tag{5.2}$$

where $\tilde{P}$ is an orthogonal matrix $\tilde{P}^T \tilde{P} = I$ if and only if $P$ is. Since the Frobenius norm is invariant under multiplication by rotation matrices, (5.2) is a reformulation of our original Laplacian matrix objective function in terms of the spectra of the two graphs. Optimization of this modified form of the objective function subject to orthogonality constraints on $P$ is upper-bounded by optimization over matchings of eigenvalues: for any fixed $\alpha$ the eigenvalue-matching problem has the same objective function, but our optimization is over all real-valued orthogonal $P$. The orthogonality constraint is a relaxed version of the constraints on matching problems (see (3.2)) discussed in subsection 3.1, since matching matrices M are also orthogonal ($M^T M = I$). Many algorithms exist for solving the inner partial and 0-1 constrained minimum-cost assignment problems, such as the Munkres algorithm [27] (also in subsection 3.1).

We note three corollaries of the above argument. Namely, because the Frobenius norm is invariant under the mapping to and from eigenspace:

1. optimal or near-optimal $\tilde{P}$ in eigenvalue-space maintain their optimality through the mapping $U_2 \cdot U_1^T$ back to graph-space;
2. solutions which are within $\epsilon$ of the optimum in $\tilde{P}$-space are also within $\epsilon$ of the optimum in $P$-space; and
3. more precisely, if they exist, zero-cost eigenvalue matchings correspond exactly with zero-cost $P$.

A natural next question would be why it might be worthwhile to work in the original graph-space, rather than always optimizing this simpler eigenvalue-matching problem instead? In many cases (path graphs, cycle graphs) the spectrum of a member $G_l$ of a graph lineage is a subset of that of $G_{l+1}$, guaranteeing that zero-cost eigenvalue matchings (and thus, by the argument above, prolongations with zero diffusion cost) exist. However, when this is not the case, the above argument only upper bounds the true distance, since the matching problem constraints are more strict. Thus, numerical optimization over $P$, with orthogonality constraints only, may find a better bound on $D^{P,\alpha}(G_l, G_{l+1})$.

**5.2. Decomposing graph product prolongations.** We next consider the problem of finding optimal prolongations between two graphs $\mathbf{G}_\square^{(1)} = G_1^{(1)} \square G_1^{(2)}$ and $\mathbf{G}_\square^{(2)} = G_2^{(1)} \square G_2^{(2)}$ when optimal prolongations are known between $G_1^{(1)}$ and $G_2^{(1)}$, and $G_1^{(2)}$ and $G_2^{(2)}$. We show that under some reasonable assumptions, these two prolongation optimizations decouple—we may thus solve them separately and combine the solutions to obtain the optimal prolongations between the two product graphs.

From the definition of graph box product, we have

$$\begin{aligned}
L_\square^{(j)} &= L(G_1^{(j)} \square G_2^{(j)}) \\
&= A(G_1^{(j)} \square G_2^{(j)}) - D(G_1^{(j)} \square G_2^{(j)}) \\
&= \left(A(G_1^{(j)}) \otimes I_2^{(j)} + I_1^{(j)} \otimes A(G_2^{(j)})\right) - \left(D(G_1^{(j)}) \otimes I_2^{(j)} + I_1^{(j)} \otimes D(G_2^{(j)})\right) \\
&= \left(A(G_1^{(j)}) \otimes I_2^{(j)} - D(G_1^{(j)}) \otimes I_2^{(j)}\right) - \left(I_1^{(j)} \otimes A(G_2^{(j)}) - I_1^{(j)} \otimes D(G_2^{(j)})\right) \\
&= (L_1^{(j)} \otimes I_2^{(j)}) + (I_1^{(j)} \otimes L_2^{(j)}) \\
&= L(G_1^{(j)}) \oplus L(G_2^{(j)}),
\end{aligned}$$

where $\oplus$ is the Kronecker sum of matrices as previously defined. See [12, Item 3.4] for more details on Laplacians of graph products. We calculate

$$\begin{aligned}
D^{P,\alpha}\left(G_\square^{(1)}, G_\square^{(2)}\right) &= \left|\left| \frac{1}{\sqrt{\alpha}} P L_\square^{(1)} - \sqrt{\alpha} L_\square^{(2)} P \right|\right|_F \\
&= \left|\left| \frac{1}{\sqrt{\alpha}} P \left( \left(L_1^{(1)} \otimes I_2^{(1)}\right) + \left(I_1^{(1)} \otimes L_2^{(1)}\right)\right) \right.\right. \\
&\qquad \left.\left. -\sqrt{\alpha} \left( \left(L_1^{(2)} \otimes I_2^{(2)}\right) + \left(I_1^{(2)} \otimes L_2^{(2)}\right)\right) P \right|\right|_F
\end{aligned}$$

$$= \left|\left|\left(\frac{1}{\sqrt{\alpha}}P\left(L_1^{(1)}\otimes I_2^{(1)}\right)-\sqrt{\alpha}\left(L_1^{(2)}\otimes I_2^{(2)}\right)P\right)\right.\right.$$
$$\left.\left. + \left(\frac{1}{\sqrt{\alpha}}P\left(I_1^{(1)}\otimes L_2^{(1)}\right)-\sqrt{\alpha}\left(I_1^{(2)}\otimes L_2^{(2)}\right)P\right)\right|\right|_F .$$

Now we try out the assumption that $P = P_1 \otimes P_2$, which restricts the search space over $P$ and may increase the objective function:

$$\begin{aligned}
D^{P,\alpha}\left(G_{\Box}^{(1)}, G_{\Box}^{(2)}\right) &= \left|\left|\left[\frac{1}{\sqrt{\alpha}}\left(P_1\otimes P_2\right)\left(L_1^{(1)}\otimes I_2^{(1)}\right)\right.\right.\right. \\
&\qquad \left. - \sqrt{\alpha}\left(L_1^{(2)}\otimes I_2^{(2)}\right)\left(P_1\otimes P_2\right)\right] \\
&\quad + \left[\frac{1}{\sqrt{\alpha}}\left(P_1\otimes P_2\right)\left(I_1^{(1)}\otimes L_2^{(1)}\right)\right. \\
&\qquad \left.\left.\left. -\sqrt{\alpha}\left(I_1^{(2)}\otimes L_2^{(2)}\right)\left(P_1\otimes P_2\right)\right]\right|\right|_F \\
&= \left|\left|\left(\frac{1}{\sqrt{\alpha}}\left(P_1L_1^{(1)}\otimes P_2\right)-\sqrt{\alpha}\left(L_1^{(2)}P_1\otimes P_2\right)\right)\right.\right. \\
&\quad \left.\left. + \left(\frac{1}{\sqrt{\alpha}}\left(P_1\otimes P_2L_2^{(1)}\right)-\sqrt{\alpha}\left(P_1\otimes L_2^{(2)}P_2\right)\right)\right|\right|_F \\
&= \left|\left|\left(\left(\frac{1}{\sqrt{\alpha}}P_1L_1^{(1)}-\sqrt{\alpha}L_1^{(2)}P_1\right)\otimes P_2\right)\right.\right. \\
&\quad \left.\left. + \left(P_1\otimes\left(\frac{1}{\sqrt{\alpha}}P_2L_2^{(1)}-\sqrt{\alpha}L_2^{(2)}P_2\right)\right)\right|\right|_F .
\end{aligned}$$

Since $||A+B||_F \le ||A||_F + ||B||_F$,

$$\begin{aligned}
&\le \left|\left|\left(\left(\frac{1}{\sqrt{\alpha}}P_1L_1^{(1)}-\sqrt{\alpha}L_1^{(2)}P_1\right)\otimes P_2\right)\right|\right| \\
&\quad + \left|\left|\left(P_1\otimes\left(\frac{1}{\sqrt{\alpha}}P_2L_2^{(1)}-\sqrt{\alpha}L_2^{(2)}P_2\right)\right)\right|\right|_F \\
&= \left|\left|\frac{1}{\sqrt{\alpha}}P_1L_1^{(1)}-\sqrt{\alpha}L_1^{(2)}P_1\right|\right|_F ||P_2||_F \\
&\quad + ||P_1||_F \left|\left|\frac{1}{\sqrt{\alpha}}P_2L_2^{(1)}-\sqrt{\alpha}L_2^{(2)}P_2\right|\right|_F .
\end{aligned}$$

Thus assuming $P = P_1 \otimes P_2$

$$\begin{aligned}
D^{P,\alpha}\left(G_{\Box}^{(1)}, G_{\Box}^{(2)}\right) &\le ||P_2||_F\, D_{\alpha,P_1}\left(G_1^{(1)}, G_1^{(2)}\right) \\
&\quad + ||P_1||_F\, D_{\alpha,P_2}\left(G_2^{(1)}, G_2^{(2)}\right),
\end{aligned}$$

which is a weighted sum of objectives of the optimizations for prolongation from $G_1^{(1)}$ to $G_1^{(2)}$ and $G_2^{(1)}$ to $G_2^{(2)}$. Recall that our original constraint on $P$ was that $P^TP = I$; since $P = P_1 \otimes P_2$ this is equivalent (by a property of the Kronecker product; see Corollary 13.8 in [22]) to the coupled constraints on $P_1$ and $P_2$:

$$\left({P_1}^TP_1 = \frac{1}{\eta}I_1^{(1)}\right) \qquad \wedge \qquad \left({P_2}^TP_2 = \eta I_2^{(1)}\right) \tag{5.3}$$

for some $\eta \in \mathbb{R}$. For any $P_1, P_2$ which obey (5.3), we may rescale them by $\eta$ to make them orthogonal without changing the value of the objective, so we take $\eta = 1$ in subsequent calculations. Noting that $||A||_F = \sqrt{\text{Tr}(A^T A)}$, we see that

$$||P_1||_F = \sqrt{\text{Tr}(I_1^{(1)})} = \sqrt{n_1^{(1)}} \quad \text{and similarly} \quad ||P_2||_F = \sqrt{n_2^{(1)}}.$$

Thus, we have proven the following theorem.

Theorem 5.1. *Assuming that $P$ decomposes as $P = P_1 \otimes P_2$, the diffusion distance $D_{P,\alpha}(G_\square^{(1)}, G_\square^{(2)})$ between $G_\square^{(1)}$ and $G_\square^{(2)}$ is bounded above by the strictly monotonically increasing function of the two distances $D_{P_1,\alpha}$ and $D_{P_2,\alpha}$:*

$$\mathcal{F}(D_{P_1,\alpha}, D_{P_2,\alpha}) = \sqrt{n_2^{(1)}} D_{P_1,\alpha} + \sqrt{n_1^{(1)}} D_{P_2,\alpha}.$$

*Namely,*

$$D_{P,\alpha}\left(G_\square^{(1)}, G_\square^{(2)}\right) \leq \mathcal{F}\left(D_{P_1,\alpha}\left(G_1^{(1)}, G_1^{(2)}\right), D_{P_2,\alpha}\left(G_2^{(1)}, G_2^{(2)}\right)\right).$$

Thus, the original optimization over the product graphs decouples into separate optimizations over the two sets of factors, constrained to have the same value of $\alpha$. Additionally, since the requirement that $P = P_1 \otimes P_2$ is an additional constraint, we have the following corollary.

Corollary 5.2. *If $(\alpha_1, P_1)$ and $(\alpha_2, P_2)$, subject to orthogonality constraints, are optima of $D_{\alpha,P}(G_1^{(1)}, G_1^{(1)})$ and $D_{\alpha,P}(G_2^{(1)}, G_2^{(1)})$, and furthermore, if $\alpha_1 = \alpha_2$, then the value of $D_{P,\alpha}(G_1^{(1)} \square G_2^{(1)}, G_1^{(2)} \square G_2^{(2)})$ for an optimal $P$ is bounded above by $D_{P_1 \otimes P_2, \alpha_1}(G_1^{(1)} \square G_2^{(1)}, G_1^{(2)} \square G_2^{(2)})$.*

This upper bound on the original objective function is a monotonically increasing function of the objectives for the two smaller problems. A consequence of this upper bound is that if $D_{P_1,\alpha}(G_1^{(1)}, G_1^{(2)}) \leq \epsilon_1$ and $D_{P_2,\alpha}(G_2^{(1)}, G_2^{(2)}) \leq \epsilon_2$, then the composite solution $P_1 \otimes P_2$ must have $D_{P_1 \otimes P_2,\alpha}(G_\square^{(1)}, G_\square^{(2)}) \leq \epsilon = (\sqrt{n_1} + \sqrt{n_2}) \max(\epsilon_1, \epsilon_2)$. Thus if both of these distances are arbitrarily small, then the composite distance must also be small. Furthermore, if only one of these is small, so that $D_{P_1,\alpha}(G_1^{(1)}, G_1^{(2)}) \approx 0$ or $D_{P_2,\alpha}(G_2^{(1)}, G_2^{(2)}) \approx 0$, then $D_{P_1 \otimes P_2,\alpha} \approx D_{P_2,\alpha}$ or $D_{P_1 \otimes P_2,\alpha} \approx D_{P_1,\alpha}$, respectively.

We have experimentally found that many families of graphs do not require scaling between the two diffusion processes: the optimal $(\alpha, P)$ pair has $\alpha = 1$. In particular, prolongation between path (cycle) graphs of size $n$ and size $2n$ always have $\alpha_{\text{optimal}} = 1$, since the spectrum of the former graph is a subset of that of the larger—therefore, there is a matching solution of cost 0 which by the argument above can be mapped to a graph-space $P$ with objective function value 0 (we prove this in section SM1 of the Supplementary Material to this paper). In this case, the two terms of the upper bound are totally decoupled and may each be optimized separately (whereas in the form given above, they both depend on an $\alpha$).

## 6. Conclusion and future work.

We have introduced a novel method for multiscale modeling which relies on a novel prolongation and restriction operator to move between models in a hierarchy. These prolongation and restriction operators are the optima of an objective function we introduce which is a natural distance metric on graphs and graph lineages. We prove several important properties of this objective function, including an upper bound which allows us to decouple a difficult optimization into two smaller optimization problems under certain circumstances.

Additionally, we demonstrate an algorithm which makes use of such $P$ and $R$ operators to simultaneously train models in a hierarchy of neural networks (specifically, autoencoder neural networks). This multiscale artificial neural network (MsANN) approach statistically outperforms training only at the finest scale, achieving lower error than the default model and also reaching the default model's best performance in an order of magnitude fewer training examples. While in our experiments we saw uniform improvement as the parameters $\gamma$, $k$, and $L$ were increased (meaning that the hierarchy is deeper, and the model spends more relative time training at the coarser scales), this may not always be the case, and we leave the question of finding optimal settings of these parameters for future work.

Future work will also focus on investigating the properties of the distance metric on graphs, and the use of those properties in graph lineage, as well as modifying the MsANN algorithm to perform the same type of hierarchical learning on more complicated ANN models, such as Convolutional Neural Networks [23], as well as nonautoencoding tasks, for example, classification.

**Acknowledgment.** The authors wish to thank the Sainsbury Laboratory, Cambridge University for their hospitality.

## REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: A system for large-scale machine learning*, in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16, 2016, pp. 265–283.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999, https://doi.org/10.1137/1.9780898719604.

[3] B. R. Bakshi and G. Stephanopoulos, *Wave-net: A multiresolution, hierarchical neural network with localized learning*, AIChE J., 39 (1993), pp. 57–81.

[4] M. Belkin and P. Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, 2002, pp. 585–591.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*, Inf. Sci. Stat., Springer, New York, 2006.

[6] H. Bourlard and Y. Kamp, *Auto-association by multilayer perceptrons and singular value decomposition*, Biol. Cybernet., 59 (1988), pp. 291–294.

[7] A. E. Brouwer and W. H. Haemers, *Distance-regular graphs*, in Spectra of Graphs, Springer, New York, 2012, pp. 177–185.

[8] B. M. Clapper, *Munkres Implementation for Python*, 2008–, http://software.clapper.org/munkres/; accessed June 10, 2018.

[9] D. Cohen-Steiner, W. Kong, C. Sohler, and G. Valiant, *Approximating the spectrum of a graph*, in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, New York, 2018, pp. 1263–1271.

[10] D. M. Cvetkovic, P. Rowlinson, and S. Simic, *An Introduction to the Theory of Graph Spectra*, Cambridge University Press, Cambridge, UK, 2010.

[11] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. C635–C661, https://doi.org/10.1137/130944230.

[12] M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Math. J., 23 (1973), pp. 298–305.

[13] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.

[14] E. M. Grais, H. Wierstorf, D. Ward, and M. D. Plumbley, *Multi-Resolution Fully Convolutional Neural Networks for Monaural Audio Source Separation*, preprint, https://arxiv.org/abs/1710.11473, 2017.

[15] G. Hinton, N. Srivastava, and K. Swersky, *Neural Networks for Machine Learning: Lecture 6a, Overview of Mini-batch Gradient Descent*, https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[16] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science, 313 (2006), pp. 504–507.

[17] L. Hogben, *Handbook of Linear Algebra*, CRC Press, Boca Raton, FL, 2006.

[18] T. Johnson, T. Bartol, T. Sejnowski, and E. Mjolsness, *Model reduction for stochastic CaMKII reaction kinetics in synapses by graph-constrained correlation dynamics*, Phys. Biol., 12 (2015), 045005.

[19] E. Jones, T. Oliphant, P. Peterson, et al., *SciPy: Open Source Scientific Tools for Python*, 2001–, http://www.scipy.org/; accessed June 10, 2018.

[20] T.-W. Ke, M. Maire, and S. X. Yu, *Multigrid Neural Architectures*, preprint, https://arxiv.org/abs/1611.07661, 2016.

[21] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, preprint, https://arxiv.org/abs/1412.6980, 2014.

[22] A. J. Laub, *Matrix Analysis for Scientists and Engineers*, SIAM, Philadelphia, 2005.

[23] Y. LeCun, Y. Bengio, and G. Hinton, *Deep Learning*, Nature, 521 (2015), pp. 436–444.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE, 86 (1998), pp. 2278–2324.

[25] Y. LeCun, C. Cortes, and C. Burges, *MNIST Handwritten Digit Database*, AT&T Labs, http://yann.lecun.com/exdb/mnist, 2010; accessed June 10, 2018.

[26] E. Mjolsness, C. D. Garrett, and W. L. Miranker, *Multiscale optimization in neural nets*, IEEE Trans. Neural Netw. Learn. Syst., 2 (1991), pp. 263–274.

[27] J. Munkres, *Algorithms for the assignment and transportation problems*, J. Soc. Indust. Appl. Math., 5 (1957), pp. 32–38, https://doi.org/10.1137/0105003.

[28] J. A. Nelder and R. Mead, *A simplex method for function minimization*, Comput. J., 7 (1965), pp. 308–313.

[29] V. Y. Pan and Z. Q. Chen, *The complexity of the matrix eigenproblem*, in Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 507–516.

[30] T. Rapcsák, *On minimization on Stiefel manifolds*, European J. Oper. Res., 143 (2002), pp. 365–376.

[31] J. B. Schroder, *Parallelizing Over Artificial Neural Network Training Runs with Multigrid*, preprint, https://arxiv.org/abs/1708.02276, 2017.

[32] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. C. Courville, *Multiresolution recurrent neural networks: An application to dialogue response generation*, in AAAI Conference on Artificial Intelligence (AAAI-17), Menlo Park, CA, 2017, pp. 3288–3294.

[33] J. Townsend, N. Koep, and S. Weichwald, *Pymanopt: A Python Toolbox for Optimization on Manifolds using Automatic Differentiation*, preprint, https://arxiv.org/abs/1603.03236, 2016.

[34] P. Turaga, A. Veeraraghavan, and R. Chellappa, *Statistical analysis on Stiefel and Grassmann manifolds with applications in computer vision*, in IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2008, pp. 1–8.

[35] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[36] Z. Wen and W. Yin, *A feasible method for optimization with orthogonality constraints*, Math. Program., 142 (2013), pp. 397–434.

[37] Wolfram Research, Inc., *Mathematica, Version* 11.3, Champaign, IL, 2018.

[38] H. Yserentant, *Old and new convergence proofs for multigrid methods*, Acta Numer., 2 (1993), pp. 285–326.