Graph Prolongation Convolutional Networks: Explicitly Multiscale Machine Learning on Graphs, with Applications to Modeling of Biological Systems

Cory B. Scott ¹  Eric Mjolsness ¹

Abstract
We define a novel type of ensemble Graph Convolutional Network (GCN) model. Using optimized linear projection operators to map between spatial scales of graph, this ensemble model learns to aggregate information from each scale for its final prediction. We calculate these linear projection operators as the infima of an objective function relating the structure matrices used for each GCN. Equipped with these projections, our model (a Graph Prolongation-Convolutional Network) outperforms other GCN ensemble models at predicting the potential energy of monomer subunits in a coarse-grained mechanochemical simulation of microtubule bending. We demonstrate these performance gains by measuring an estimate of the FLOPs spent to train each model, as well as wall-clock time. Because our model learns at multiple scales, it is possible to train at each scale according to a predetermined schedule of coarse vs. fine training. We examine several such schedules adapted from the Algebraic Multigrid (AMG) literature, and quantify the computational benefit of each. Finally, we demonstrate how under certain assumptions, our graph prolongation layers may be decomposed into a matrix outer product of smaller GCN operations.

1. Introduction
1.1. Convolution and Graph Convolution
Recent successes of deep learning have demonstrated that the inductive bias of Convolutional Neural Networks (CNNs) makes them extremely efficient for analyzing data with an inherent grid structure, such as images or video.

1Department of Computer Science, University of California Irvine, Irvine, California, USA. Correspondence to: Cory B. Scott <scottcb@uci.edu>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML).

In particular, many applications use these models to make per-node (per-pixel) predictions over grid graphs: examples include image segmentation, optical flow prediction, anticipating motion of objects in a scene, and facial detection/identification. Further work applies these methods to emulate physical models, by discretizing the input domain. Computational Fluid Dynamics and other scientific tasks featuring PDEs or ODEs on a domain discretized by a rectangular lattice have seen recent breakthroughs applying machine learning models, like CNNs to handle data which is structured this way. These models learn a set of local filters whose size is much smaller than the size of the domain - these filters may then be applied simultaneously across the entire domain, leveraging the fact that at a given scale the local behavior of the neighborhood around a pixel (voxel) is likely to be similar at all grid points.

Graph Convolutional Networks (GCNs) are a natural extension of the above idea of image ‘filters’ to arbitrary graphs rather than nD grids, which may be more suitable in some scientific contexts. Intuitively, GCNs replace the image filtering operation of CNNs with repeated passes of: 1) aggregation of information between nodes according to some structure matrix 2) nonlinear processing of data at each node. We refer the reader to a recent survey by Bacciu et al (2019) for a more complete exploration of the taxonomy graph neural networks.

1.2. Microtubules
As an example of a dataset whose underlying graph is not a grid, we consider a coarse-grained simulation of a microtubule. Microtubules (MTs) are self-assembling nanostructures which are ubiquitous in living cells. MTs play important structural roles during cell division, cell growth, and separation of chromosomes (in eukaryotic cells) (Chakrabortty et al., 2018). Microtubules are comprised of a lattice structure of two conformations (α and β) of tubulin. Free-floating tubulin monomers associate energetically into dimer subunits, which then associate head-to-tail to form long chain-like complexes called protofilaments. Protofilaments as-
sociate side-to side in a sheet; at some critical number of protofilaments (which varies between species and cell type) the sheet wraps closed to form a repeating helical lattice with a seam. See (Pampaloni & Florin, 2008), Page 303, Figure 1. Key properties of microtubules are:

**Dynamic instability:** microtubules grow from one end by attracting free-floating tubulin monomers (VanBuren et al., 2005). Microtubules can spontaneously enter a “catastrophe” phase, in which they rapidly unravel, but can also “rescue” themselves from the catastrophe state and resume growth (Gardner et al., 2013; Shaw et al., 2003).

**Interactions:** Microtubules interact with one another: they can dynamically avoid one another during the growth phase, or collide and bundle up, or collide and enter catastrophe (Tindemans et al., 2014). The exact mechanism governing these interactions is an area of current research.

**Structural strength:** microtubules are very stiff, with a Young’s Modulus estimated at $\approx 1\text{GPa}$ for some cases (Pampaloni & Florin, 2008). This stiffness is thought to play a role in reinforcing cell walls (Kis et al., 2002).

In this work we introduce a model which learns to reproduce the dynamics of a graph signal (defined as an association of each node in the network with a vector of discrete or real-valued labels) at multiple scales of graph resolution. We apply this model framework to predict the potential energy of each tubulin monomer in a mecha-nochemical simulation of a microtubule.

### 1.3. Simulation of MTs and Prior Work

Non-continuum, non-event-based simulation of large molecules is typically done by representing some molecular subunit as a particle/rigid body, and then defining rules for how these subunits interact energetically. Molecular Dynamics (MD) simulation is an expansive area of study and a detailed overview is beyond the scope of this paper. We instead proceed to describe in general terms some basic ideas relevant to the numerical simulation detailed in Section 3.1. MD simulations proceed from initial conditions by computing the forces acting on each particle (according to the potential energy interactions and any external forces, as required), determining their instantaneous velocities and acceleration accordingly, and then moving each particle by the distance it would move (given its velocity) for some small timestep. Many variations of this basic idea exist. The software we use for our MD simulations, LAMMPS (Plimpton, 1993) allows for many different types of update step: we use Verlet integration (updating particle position according to the central difference approximation of acceleration (Verlet, 1967)) and Langevin dynamics (modeling the behavior of a viscous surrounding solvent implicitly (Schneider & Stoll, 1978)). We also elect to use the microcanonical ensemble (NVE) - meaning that the update steps of the system maintain the total number of particles, the volume of the system, and the total energy (kinetic + potential). For more details of our simulation, see Section 3.1 and the source code, available in the Supplementary Material accompanying this paper. Independent of implementation details, a common component of many experiments in computational molecular dynamics is the prediction of the potential energy associated with a particular conformation of some molecular structure. Understanding the energetic behavior of a complex molecule yields insights into its macro-scale behavior: for instance, the problem of protein folding can be understood as seeking a lower-energy configuration. In this work, we apply graph convolutional networks, trained via a method we introduce, to predict these energy values for a section of microtubule.

### 1.4. Mathematical Background and Notation

**Definitions:** For all basic terms (graph, edge, vertex, degree) we use standard definitions. We use the notation $\{x_i\}_{i=a}^b$ to represent the sequence of $x_i$ indexed by the integers $a, a+1, a+2, \ldots, b$.

**Graph Laplacian:** The graph Laplacian is the matrix given by $L(G) = A(G) - \text{diag}(A(G) \cdot 1)$ where $A(G)$ is the adjacency matrix of $G$, and $1$ is an appropriately sized vector of $1$s. The graph Laplacian is given by some authors as the opposite sign.

**Graph Diffusion Distance (GDD):** Given two graphs $G_1$ and $G_2$, with $|G_1| \leq |G_2|$ the Graph Diffusion Distance $D(G_1, G_2)$ is given by:

$$
D(G_1, G_2) = \inf_{P \in C(P)} \|PL(G_1) - L(G_2)P\|_F, \tag{1}
$$

where $C(P)$ represents some set of constraints on $P$, and $\| \cdot \|_F$ represents the Frobenius norm. We take $C(P)$ to be orthogonality: $P^T P = I$. Note that since in general $P$ is a rectangular matrix, it may not be the case that $PP^T = I$. Our recent work (Author & Author, 2019) has examined variants of this distance measure, and techniques for efficiently calculating this distance. Detailing these is outside of the scope of this paper; all $P$ matrices detailed in this work were calculated using the constrained optimization package Pymanopt (Townsend et al., 2016).

**Prolongation matrix:** we use the term “prolongation matrix” to refer to a matrix which is the optimum of the minimization given in the definition of the GDD.

### 2. Model Architecture

The model we propose is an ensemble of GCNs at multiple scales, with optimized projection matrices performing the mapping in between scales (i.e. between ensemble members). More formally, Let $\{G_i\}_{i=1}^k$ represent a sequence of graphs with $|G_1| \geq |G_2| \ldots \geq |G_k|$, and let $\{Z_i = z(G_i)\}_{i=1}^k$ be their structure matrices (for some
chosen method $z$ of calculating the structure matrix given the graph). In all experiments in this paper, we take $z(G) = L(G)$, the graph Laplacian, as previously defined.

In an ensemble of Graph Convolutional Networks, let $\theta^{(i)}_l = \{W^{(i)}_l, b^{(i)}_l\}$ represent the parameters (filter matrix and bias vector) in layer $l$ of the $i$th network.

We follow the GCN formulation given by Kipf and Welling (2016). Assuming an input tensor $X$ of dimensions $n \times F$ (where $n$ is the number of nodes in the graph and $F$ is the dimension of the label at each node), we inductively define the layerwise update rules for a graph convolutional network $GCN \left( Z_i, X, \{\theta^{(i)}_l\}_{l=1}^{m} \right)$ as:

$$X_0 = X$$

$$X_m = g_m \left( Z_i X_{m-1} W^{(i)}_m + b^{(i)}_m \right),$$

where $g_m$ is the activation function of the $m$th layer.

When $i = j - 1$, let $P_{i,j}$ be an optimal (in either the sense of Graph Diffusion Distance, or in the sense we detail in section 3.3) prolongation matrix from $L(G_j)$ to $L(G_j)$, i.e. $P_{i,j} = \arg \inf_{P|P|} \| PL(G_j) - L(G_j) P \|_F$. Then, for $i < j - 1$, let $P_{i,j}$ be shorthand for the matrix product $P_{i,i+1} P_{i+1,i+2} \ldots P_{j-1,j}$. For example, $P_{1,4} = P_{1,2} P_{2,3} P_{3,4}$.

Our multiscale ensemble model is then constructed as:

$$GCN \left( \{Z_i\}_{i=1}^{k}, X, \{\theta^{(i)}_l\}_{l=1}^{m}, \{P_{i,i+1}\}_{i=1}^{k-1} \right)$$

$$= GCN \left( Z_1, X, \{\theta^{(1)}_l\}_{l=1}^{m}, \{P_{1,i+1}\}_{i=1}^{k-1} \right)$$

$$+ \sum_{i=2}^{k} P_{i+1} GCN \left( Z_i, P_{i+1}^{T} X, \{\theta^{(i)}_l\}_{l=1}^{m} \right)$$

This model architecture is illustrated in Figure 1. When the $P$ matrices are constant/fixed, we will refer to this model as a GPCN, for Graph Prolongation-Convolutional Network. However, we find in our experiments in Section 3.3 that validation error is further reduced when the $P$ operators are tuned during the same gradient update step which updates the filter weights, which we refer to as an “adaptive” GPCN or A-GPCN. We explain our method for choosing $Z_i$ and optimizing $P$ matrices in Section 3.3.

3. Numerical Experiments

3.1. Dataset

In this Section we detail the process for generating the simulated microtubule data for comparison of our model with other GCN ensemble models. Our microtubule structure has 13 protofilaments (each 48 tubulin monomers long). As in a biological microtubule, each tubulin monomer is offset (along the axis parallel to the protofilaments) from its neighbors in adjacent protofilaments, resulting in a helical structure with a pitch of 3 tubulin units. We refer to this pitch as the “offset” in Section 3.2. Each monomer subunit (624 total) is represented as a point mass of 50 Dalton ($8.30 \times 10^{-12} \text{kg}$). The diameter of the whole structure is 26nm, and the length is $\approx 260$nm. The model itself was constructed using Molttemplate (Jewett et al., 2013), a tool for constructing large regular molecules to be used in LAMMPS simulations.

For this model, we define energetic interactions for angles and associations only. No steric or dihedral interactions were used: for dihedrals, this was because the lattice structure of the tube meant any set of four molecules contributed to multiple, contradictory dihedral interactions.

Interaction energy of an association $b$ was calculated using the “harmonic” bond style in LAMMPS, i.e. $E(b) = k(b - b_0)^2$, where $b_0$ is the resting length and $k$ is the strength of that interaction. The energy of an angle $\phi$ was similarly calculated using the “harmonic” angle style, i.e. $E(\phi) = k(\phi - \phi_0)^2$, where $\phi_0$ is the resting angle and $k$ is again the interaction strength. The resting lengths and angles for all energetic interactions were calculated using the resting geometry of our microtubule graph $G_m$; a LAMMPS script was used to print the value of every angle interaction in the model, and these were collected and grouped based on value (all $153^\circ$ angles, all $102^\circ$ angles, etc). Each strength parameter was varied over the values in $\{1, 3, 6, 1.0, 1.3, 1.6, 1.9, 2.0\}$, producing $8^6$ parameter combinations. Langevin dynamics were used, but with small temperature, to ensure stability and emphasize mechanical interactions. See Table 1 and Figure 3 for details on each strength parameter.

---

1 Other GCN research uses powers of the Laplacian, the normalized Laplacian, the symmetric normalized laplacian, etc. Comparison of these structure matrices is out of scope of this paper.

2 Association and angle constraints were sufficient to replicate the bending resistance behavior of microtubules. We hope to run a similar experiment using higher-order particle interactions (which may be more biologically plausible), in future work.
GNU Parallel (Tange, 2011) was used to run a simulation for each combination of interaction parameters, using the particle dynamics simulation engine LAMMPS. In each simulation, we clamp the first two rings of tubulin monomers (nodes 1-26) in place, and apply force (in the negative y direction) to the final two rings of monomers (nodes 599-624). This force starts at 0 and ramps up during the first 128000 timesteps (one step = 0.5 ns) to its maximum value of $3 \times 10^{-15}$N. Once maximum force is reached, the simulation runs for 256000 additional timesteps, which in our experience was long enough for all particles to come to rest. See Figure 2 for an illustration (visualized with Ovito (Stukowski, 2010)) of the potential energy per-particle at the final frame of a typical simulation run. Every $K = 32000$ timesteps, we save the following for every particle: the position $x, y, z$; components of velocity $v_x, v_y, v_z$; components of force $F_x, F_y, F_z$; and the potential energy of the particle $E$. The dataset is then a concatenation of the 12 saved frames from every simulation run, comprising all combinations of input parameter values, where for each frame we have: $x_t$, the input graph signal, a $624 \times 13$ matrix holding the position, velocity, and force on each particle, as well as values of the five interaction coefficients; and $y_t$, the output graph signal, a $624 \times 1$ matrix holding the potential energy calculated for each particle.

During training, after a training/validation split, we normalize the data by taking the mean and standard deviation of the $N_{\text{train}} \times 624 \times 13$ input and $N_{\text{train}} \times 624 \times 1$ output tensors along their first axis. Each data tensor is then reduced by the mean and divided by the standard deviation so that all $624 \times 13$ inputs to the network have zero mean and unit standard deviation. We normalize using the training data only.

![Figure 2. Microtubule model under bending load. Color of each particle indicates the sum of that particle’s share of all of the energetic interactions in which it participates. This view is of the clamped end; the other end, out of view, has a constant force applied.](image)

### 3.2. Graph Coarsening

In this Section we outline a procedure for determining the coarsened structure matrices to use in the hierarchy of GCN models comprising a GPCN. We use our microtubule graph as an example. In this case, we have two a-priori guidelines for producing the reduced-order graphs: 1) the reduced models should still be a tube and 2) it makes sense from a biological point of view to coarsen by combining the $\alpha$-$\beta$ pairs into single subunits. Given these restrictions, we can explore the space of coarsened graphs and find the coarse graph which is nearest to our original graph (under the GDD).

Our microtubule model is a tube of length 48 units, 13 units per complete “turn”, and with the seam offset by three units. We generalize this notion as follows: Let $p$ be the offset, and $k$ be the number of monomers in one turn of the tube, and $n$ the number of turns of a tube graph $G_{\text{Tube}}(n,k,p)$. The graph used in our simulation is thus $G_{\text{int}} = G_{\text{Tube}}(48,13,3)$. We pick the medium scale model $G_{\text{inter}}$ to be $G_{\text{Tube}}(24,13,1)$, as this is the result of combining each $\alpha$-$\beta$ pair of tubulin monomer units in the fine scale, into one tubulin dimer unit in the medium scale. We pick the coarsest graph $G_{\text{coarse}}$ by searching over possible offset tube graphs. Namely, we vary $k \in \{3, 4, \ldots, 12\}$ and $p \in \{0, 1, 2, 3\}$, and compute the optimal $P^*$ and its associated distance $D(G_{\text{Tube}}(24,k,p), G_{\text{int}}|P = P^*)$. Figure 4 shows the distance between $G_{\text{int}}$ and various other tube graphs as parameters $p$ and $k$ are varied. The nearest $G_{\text{Tube}}(24,k,p)$ to $G_{\text{int}}$ is that with $p = 0$ and $k = 3$. Note that Figure 4 has two columns for each value of $k$: these represent the coarse edges along the seam having weight (relative to the other edges) 1 (marked with an $S$) or having weight 2 (no $S$). This is motivated by the fact that our initial condensing of each dimer pair condensed pairs of seam edges into single edges.

![Figure 3. Microtubule model structure. Red spheres represent $\alpha$-tubulin; purple spheres represent $\beta$-tubulin. Highlighted atoms at center are labelled to show example energetic interactions: each type of interaction indicated in Table 1 (using the particle labels in this image) is applied everywhere in the model where that arrangement of particle and association types occurs in that position.](image)

### 3.3. Comparison to Other GCN Ensemble Models

In this experiment we demonstrate the efficiency advantages of our model by comparing our approach to other ensemble Graph Convolutional Networks. Within each ensemble, each GCN model consists of several graph convolution lay-

---

**Table 1**

| Interaction Type | Graph Edges | Particle Association |
|-----------------|-------------|----------------------|
| $\alpha$-$\beta$ | 1 (marked with an $S$) | $G_{\text{Tube}}(24,13,0)$ |
| $\alpha$-$\beta$ | 2 (no $S$) | $G_{\text{Tube}}(24,13,1)$ |

---

**Figure 4**

The graph used in our simulation is thus $G_{\text{int}} = G_{\text{Tube}}(48,13,3)$. We pick the medium scale model $G_{\text{inter}}$ to be $G_{\text{Tube}}(24,13,1)$, as this is the result of combining each $\alpha$-$\beta$ pair of tubulin monomer units in the fine scale, into one tubulin dimer unit in the medium scale. We pick the coarsest graph $G_{\text{coarse}}$ by searching over possible offset tube graphs. Namely, we vary $k \in \{3, 4, \ldots, 12\}$ and $p \in \{0, 1, 2, 3\}$, and compute the optimal $P^*$ and its associated distance $D(G_{\text{Tube}}(24,k,p), G_{\text{int}}|P = P^*)$. Figure 4 shows the distance between $G_{\text{int}}$ and various other tube graphs as parameters $p$ and $k$ are varied. The nearest $G_{\text{Tube}}(24,k,p)$ to $G_{\text{int}}$ is that with $p = 0$ and $k = 3$. Note that Figure 4 has two columns for each value of $k$: these represent the coarse edges along the seam having weight (relative to the other edges) 1 (marked with an $S$) or having weight 2 (no $S$). This is motivated by the fact that our initial condensing of each dimer pair condensed pairs of seam edges into single edges.
Table 1. Description of energetic interactions in microtubule simulation, according to the labels in Figure 3.

| ASSOCIATION INTERACTIONS | Description | Examples | Resting Length | Strength Param. |
|--------------------------|-------------|----------|----------------|-----------------|
| Lateral association inside lattice | (1,3),(2,4) | 5.15639nm | LATASSOC |
| Lateral association across seam | (5,8),(6,9) | 5.15639nm | LATASSOC |
| Longitudinal association | (1,2),(3,4) | 5.0nm | LONGLASSOC |

| ANGLE INTERACTIONS | Description | Examples | Resting Angle | Strength Param. |
|-------------------|-------------|----------|---------------|-----------------|
| Pitch angle inside lattice | (1,3,5),(2,4,6) | 153.023° | LATANGLE |
| Longitudinal angle | (5,6,7),(8,9,10) | 180° | LONGLANGLE |
| Lattice cell acute angle | (3,4,6),(3,5,6),(5,8,9),(6,9,10) | 77.0694° | QUADANGLES |
| Lattice cell obtuse angle | (4,3,5),(4,6,5),(6,5,8),(6,9,8) | 102.931° | QUADANGLES |

Figure 4. Directed Graph Diffusion Distance (GDD) between off-set tube graphs and $G_{int}$. Table cells colored by value. We see from this comparison that the two graphs which are closest to $G_{int}$ are $G_{Tube(24,3,0)}$ and $G_{Tube(24,3,0)}$ with an edge weight of 2 for connections along the seam, motivating our choice of $G_{Tube(24,3,0)}$ (unweighted) as the coarsest graph in our hierarchy.

We also compare our model to the work of Abu-El-Haija et al. (2018), who introduce the N-GCN model: an ensemble GCN in which each ensemble member uses a different power $Z^{r}$ of the structure matrix (to aggregate information from neighborhoods of radius $r$). We include a N-GCN with radii (1,2,4) and a N-GCN with radii (1,2,4,8,16).

All models were trained with the same train/validation split, using ADAM with default hyperparameters, in TensorFlow (Abadi et al., 2016). Random seeds for Python, TensorFlow, Numpy, and Scipy were all initialized to the same value for each training run, to ensure that the train/validation split is the same across all experiments, and the batches of drawn data are the same. See supplementary material for version numbers of all software packages used. Training batch size was set to 8, all GCN layers have ReLU activation, and all dense layers have sigmoidal activation with the exception of the output layer of each network (which is linear). All modes were trained for 1000 epochs of 20 batches each. The time per batch of each model is listed in Table 4. Since hardware implementations may differ, we estimate the computational cost in FLOPs of each operation.
Table 2. Filter specifications for ensemble models in comparison experiment.

| Structure Matrix | GCN Filters | Dense Filters |
|------------------|-------------|---------------|
| \( L_{\text{mat}} \) | 64,64,64 | 256, 32, 8, 1 |
| \( L_{\text{inter}} \) | 64,64,64 | 256, 32, 8, 1 |
| \( L_{\text{intra}} \) | 32,32,32 | 256, 32, 8, 1 |
| \( L_{\text{mat}} \) | 16,16,16 | 256, 32, 8, 1 |

2-level GPCN

| Structure Matrix | GCN Filters | Dense Filters |
|------------------|-------------|---------------|
| \( L_{\text{mat}} \) | 64,64,64 | 256, 32, 8, 1 |
| \( L_{\text{inter}} \) | 64,64,64 | 256, 32, 8, 1 |
| \( L_{\text{intra}} \) | 32,32,32 | 256, 32, 8, 1 |
| \( L_{\text{mat}} \) | 16,16,16 | 256, 32, 8, 1 |

3-level GPCN

| Structure Matrix | GCN Filters | Dense Filters |
|------------------|-------------|---------------|
| \( L_{\text{mat}} \) | 64,64,64 | 256, 32, 8, 1 |
| \( L_{\text{inter}} \) | 32,32,32 | 256, 32, 8, 1 |
| \( L_{\text{intra}} \) | 16,16,16 | 256, 32, 8, 1 |

N-GCN (radii 1,2,4)

| Structure Matrix | GCN Filters | Dense Filters |
|------------------|-------------|---------------|
| \( L_{\text{mat}} \) | 64,64,64 | 256, 32, 8, 1 |

N-GCN (radii 1,2,4,8,16)

| Structure Matrix | GCN Filters | Dense Filters |
|------------------|-------------|---------------|
| \( L_{\text{mat}} \) | 64,64,64 | 256, 32, 8, 1 |

Figure 6. Comparison of Normalized MSE on held-out validation data as a function of FLOPs expended for a variety of ensemble Graph Convolutional Network Models. We see that especially in early stages of training, our model formulation learns faster than an ensemble of 2, 3 or 5 GCNs with the same number of filters. The error plotted is the model’s minimum error thus far (on the validation data). See supplementary material for a non-minimized version, and versions of same plots with wall-clock time substituted for estimated FLOP count.

3.4. Comparison: All-at-Once or Coarse-to-Fine Training

In this Section we compare the computational cost of training the entire GPCN at once, versus training the different ‘resolutions’ (meaning the different GCNs in the hierarchy) of the network according to a more complicated training schedule. This approach is motivated by recent work in coarse-to-fine training of both flat and convolutional neural networks (Scott & Mjolsness, 2019; Zhao et al., 2019; Haber et al., 2018; Dou & Wu, 2015; Ke et al., 2017), as well as the extensive literature on Algebraic MultiGrid (AMG) methods (Vaněk et al., 1996).

AMG solvers for differential equations on a mesh (which arises as the discretization of some volume to be simulated) proceed by performing numerical “smoothing steps” at multiple resolutions of discretization. The intuition behind this approach is that modes of error should be smooth at a spatial...
Table 3. Mean error and uncertainty of several GCN ensemble models across ten random trials. For each trial, the random seed was set to the same value for each model. Reported values are the minimum error on the validation set during training (not the error at the final epoch). Normalized Mean Squared Error (NMSE) values are unitless.

| Model Name                | Mean NMSE ± Std. Dev $\times 10^{-3}$ | Min NMSE $\times 10^{-3}$ |
|---------------------------|----------------------------------------|---------------------------|
| Single GCN                | 1.50 ± 0.09                            | 1.37796                   |
| Ensemble - 2 GCNs         | 1.38 ± 0.09                            | 1.16949                   |
| Ensemble - 3 GCNs         | 1.44 ± 0.16                            | 1.24315                   |
| 2-level GPCN              | 1.40 ± 0.14                            | 1.18357                   |
| 2-level A-GPCN            | 0.23 ± 0.05                            | 0.14109                   |
| 3-level GPCN              | 1.95 ± 0.20                            | 1.69807                   |
| 3-level A-GPCN (0.181 ± 0.029) | 0.13726                   |
| N-GCN radii (1,2,4)       | 1.41 ± 0.11                            | 1.31325                   |
| N-GCN radii (1,2,4,8,16)  | 1.41 ± 0.08                            | 1.30904                   |
| DiffPool                  | 4.6 ± 1.2                              | 3.14470                   |

Table 4. Mean wall-clock time to perform feed-forward and back-propagation for one batch of data, for various GCN ensemble models. Times were collected on a single Intel(R) Xeon(R) CPU core and an NVIDIA TITAN X GPU.

| Model Name                | Mean time per batch (s) |
|---------------------------|-------------------------|
| Single GCN                | 0.0312                  |
| Ensemble - 2 GCNs         | 0.0471                  |
| Ensemble - 3 GCNs         | 0.0588                  |
| 2-level GPCN              | 0.0524                  |
| 2-level A-GPCN            | 0.0339                  |
| 3-level GPCN              | 0.0324                  |
| 3-level A-GPCN            | 0.0371                  |
| N-GCN radii (1,2,4)       | 0.0862                  |
| N-GCN radii (1,2,4,8,16)  | 0.138                   |
| DiffPool                  | 0.0580                  |

scale which is equivalent to their wavelength, i.e. the solver shouldn’t spend many cycles resolving long-wavelength errors at the finest scale, since they can be resolved more efficiently at the coarse scale. Given a solver and a hierarchy of discretizations, the AMG literature defines several types of training procedures or “cycle” types (F-cycle, V-cycle, W-cycle). These cycles can be understood as being specified by a recursion parameter $\gamma$, which controls how many times the smoothing or training algorithm visits all of coarser levels of the hierarchy in between smoothing steps at a given scale. For example, when $\gamma = 1$ the algorithm proceeds from fine to coarse and back again, performing one smoothing step at each resolution - a ‘V’ cycle.

We investigate the efficiency of training 3-level GPCN and A-GPCN (as described in Section 3.3), using multigrid-like training schedules with $\gamma \in \{0, 1, 2, 3\}$, as well as “coarse-to-fine” training: training the coarse model to convergence, then training the coarse and intermediate models together (until convergence), then finally training all three models at once. For coarse-to-fine training convergence was defined to have occurred once 10 epochs had passed without improvement of the validation error.

Our experiments (see Figure 7) show that these training schedules do result in a slight increase in efficiency of the GPCN model, especially during the early phase of training. However, we also find that these schedules are seemingly not compatible with the adaptive GPCN, as demonstrated by the fact that there are long periods of training with no improvement in validation loss. Notably, the coarse-to-fine schedule is an exception: the coarse-to-fine training of the A-GPCN outperformed even the GPCN with no multigrid training.

Figure 7. Effect of varying training schedule for training a GPCN model. Notably, coarse-to-fine training outperforms all multigrid schedules and even the original training procedure.
We see that our GPCN model achieves comparable validation loss with less computational work, and our A-GPCN model additionally achieves lower absolute validation loss.

In addition to GCN layers as described in Section 2, Ying et al. define a pooling operation at layer $l$:

$$S^{(l)} = \sigma \left( \text{GCN}_{\text{pool}} \left( Z^{(l)}, X^{(l)}, \left\{ \theta_1^{(i)} \right\}_{i=1}^m \right) \right)$$

where $\text{GCN}_{\text{pool}}$ is an auxiliary GCN with its own set of parameters $\left\{ \theta_1^{(i)} \right\}_{i=1}^m$, and $\sigma$ is the softmax function. The output of $\text{GCN}_{\text{pool}}$ is a $n \times n_{\text{coarse}}$ matrix, each row of which is softmaxed to produce an affinity matrix $S$ whose rows each sum to 1, representing each fine-scale node being connected to one unit’s worth of coarse-scale nodes. The coarsened structural and data matrices for the next layer are then calculated as:

$$X^{(l+1)} = S^{(l)^T} X^{(l)}$$
$$Z^{(l+1)} = S^{(l)^T} Z^{(l)} S^{(l)}$$

Clearly, the additional GCN layers required to produce $S^{(l)}$ incur additional computational cost. We compare our 3-level GPCN (adaptive and not) models from the experiment in Section 3.3 to a model which has the same structure, but in which each $P$ matrix is replaced by the appropriately-sized output of a DIFFPOOL module, and furthermore the coarsened structure matrices are produced as in Equation 2.

We see that our GPCN model achieves comparable validation loss with less computational work, and our A-GPCN model additionally achieves lower absolute validation loss.

4. Future Work

4.1. Differentiable Models of Molecular Dynamics

This work demonstrates the use of feed-forward neural networks to approximate the energetic potentials of a mechanochemical model of an organic molecule. Per-timestep, GCN models may not be as fast as highly-parallelized, optimized MD codes. However, neural networks are highly flexible function approximators: the GCN training approach outlined in this paper could also be used to train a GCN which predicts the energy levels per particle at the end of a simulation (once equilibrium is reached), given the boundary conditions and initial conditions of each particle. In the case of our MT experiments, approximately $3 \times 10^5$ steps were required to reach equilibrium. The computational work to generate a suitably large and diverse training set would then be amortized by the GCN’s ability to generalize to initial conditions, boundary conditions, and hyperparameters outside of this data set. Furthermore, this GCN reduced model would be fully differentiable, making it possible to perform gradient descent with respect to any of these inputs.

4.2. Tensor Factorization

Recent work has re-examined GCNs in the context of the extensive literature on tensor decompositions. LanczosNet (Liao et al., 2019), uses QR decomposition of the structure matrix to aggregate information from large neighborhoods of the graph. The “Tensor Graph Convolutional Network” of Zhang et al. (2018), is a different decomposition method, based on graph factorization; a product of GCNs operating on each factor graph can be as accurate as a single GCN acting on the product graph. Since recent work (Scott & Mjolsness, 2019) has shown that the GDD of a graph product is bounded by the distances between the factor graphs, it seems reasonable to combine both ideas into a model which uses a separate GPCN for each factor. One major benefit of this approach would be that a transfer-learning style approach can be used. For example, we could train a product of two GCN models on a short section of microtubule; and then re-use the weights in a model that predicts energetic potentials for a longer microtubule. This would allow us to extend our approach to MT models whose lengths are biologically relevant, e.g. $10^3$ tubulin monomers.

5. Conclusion

We introduce a new type of graph ensemble model which explicitly learns to approximate behavior at multiple levels of coarsening. Our model outperforms several other types of GCN, including both other ensemble models and a model which coarsens the original graph using DiffPool. We also explore the effect of various training schedules, discovering that A-GPCNs can be effectively trained using a coarse-to-fine training schedule. We present the first use of GCNs to approximate energetic potentials in a model of a microtubule.
Acknowledgements

This work was funded by the Human Frontiers Science Program [grant HFSP - RGP0023/2018]; and the National Institute of Health [grant R01 HD073179], as well as the hospitality of the Center for NonLinear Studies (CNLS) at Los Alamos National Laboratory.

References

Abadi, M. et al. Tensorflow: A System for Large-Scale Machine Learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pp. 265–283, 2016.

Abu-El-Haija, S., Kapoor, A., Perozzi, B., and Lee, J. N-GCN: Multi-Scale Graph Convolution for Semi-supervised Node Classification, 2018.

Author, A. and Author, A. Novel Diffusion-Derived Distance Measures for Graphs. Suppressed for Anonymity, 2019.

Bacciu, D., Errica, F., Micheli, A., and Podda, M. A Gentle Introduction to Deep Learning for Graphs. arXiv preprint arXiv:1912.12693, 2019.

Chakrabortty, B., Blilou, I., Scheres, B., and Mulder, B. M. A Computational Framework for Cortical Microtubule Dynamics in Realistically Shaped Plant Cells. PLoS Computational Biology, 14(2):e1005959, 2018.

Dou, H. and Wu, X. Coarse-to-Fine Trained Multi-Scale Convolutional Neural Networks for Image Classification. In 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–7. IEEE, 2015.

Gardner, M. K., Zanic, M., and Howard, J. Microtubule Catastrophe and Rescue. Current Opinion in Cell Biology, 25(1):14–22, 2013.

Haber, E., Ruthotto, L., Holtham, E., and Jun, S.-H. Learning Across Scales - Multiscale Methods for Convolution Neural Networks. In Thirty-Second {AAAI} Conference on Artificial Intelligence, 2018.

Jewett, A. I., Zhuang, Z., and Shea, J.-E. Moltemplate: a Coarse-Grained Model Assembly Tool. Biophysical Journal, 104(2):169a, 2013.

Ke, T.-W., Maire, M., and Yu, S. X. Multigrid Neural Architectures. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6665–6673, 2017.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. arXiv preprint arXiv:1609.02907, 2016.

Kis, A., Kasas, S., Babić, B., Kulik, A., Benoit, W., Briggs, G., Schönenberger, C., Catsicas, S., and Forro, L. Nanomechanics of Microtubules. Physical Review Letters, 89(24):248101, 2002.

Liao, R., Zhao, Z., Urtasun, R., and Zemel, R. S. LanczosNet: Multi-Scale Deep Graph Convolutional Networks. arXiv preprint arXiv:1901.01484, 2019.

Pampaloni, F. and Florin, E.-L. Microtubule Architecture: Inspiration for Novel Carbon Nanotube-based Biomimetic Materials. Trends in Biotechnology, 26(6):302–310, 2008.

Plimpton, S. Fast Parallel Algorithms For Short-Range Molecular Dynamics. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1993.

Schneider, T. and Stoll, E. Molecular-Dynamics Study of a Three-Dimensional One-Component Model for Distortive Phase Transitions. Physical Review B, 17(3):1302, 1978.

Scott, C. and Mjolsness, E. Multilevel Artificial Neural Network Training for Spatially Correlated Learning. SIAM Journal on Scientific Computing, 41(5):S297–S320, 2019.

Shaw, S. L., Kamyar, R., and Ehrhardt, D. W. Sustained Microtubule Treadmilling in Arabidopsis Cortical Arrays. Science, 300(5626):1715–1718, 2003.

Stukowski, A. Visualization and Analysis of Atomistic Simulation Data with OVITO - the Open Visualization Tool. Modelling Simulation in Materials Science and Engineering, 18(1), JAN 2010. doi: {10.1088/0965-0393/18/1/015012}.

Tange, O. GNU Parallel - The Command-Line Power Tool. :login: The USENIX Magazine, 36(1):42–47, Feb 2011. doi: http://dx.doi.org/10.5281/zenodo.16303. URL http://www.gnu.org/s/parallel.

Tindemans, S. H., Deinum, E. E., Lindeboom, J. J., and Mulder, B. Efficient Event-Driven Simulations Shed New Light on Microtubule Organization in the Plant Cortical Array. Frontiers in Physics, 2:19, 2014.

Townsend, J., Koep, N., and Weichwald, S. Pymanopt: A python Toolbox for Optimization on Manifolds using Automatic Differentiation. The Journal of Machine Learning Research, 17(1):4755–4759, 2016.

VanBuren, V., Cassimeris, L., and Odde, D. J. Mechanochemical Model of Microtubule Structure and Self-Assembly Kinetics. Biophysical Journal, 89(5):2911–2926, 2005.
Vaněk, P., Mandel, J., and Brezina, M. Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. *Computing*, 56(3):179–196, 1996.

Verlet, L. Computer “Experiments” on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98, 1967.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

Zhang, T., Zheng, W., Cui, Z., and Li, Y. Tensor Graph Convolutional Neural Network. *arXiv preprint arXiv:1803.10071*, 2018.

Zhao, J., Dai, L., Zhang, M., Yu, F., Li, M., Li, H., Wang, W., and Zhang, L. PGU-net+: Progressive Growing of U-net+ for Automated Cervical Nuclei Segmentation. *Lecture Notes in Computer Science*, pp. 5158, Dec 2019.