1 Operator learning for Parametric PDEs

Operators are mappings between function spaces. For example, given a Hilbert space $\mathcal{H} : A \to B$, mappings $\hat{O} : \mathcal{H} \to \mathcal{H}$ are operators on $\mathcal{H}$. The goal of **operator learning** is to approximate operators $\hat{O}$ with neural networks $\hat{O}_\theta$. We want to do this as doing calculation with operators using traditional methods (e.g. FEM) are slower compared to forward pass with NN, and they require complex mathematical derivations.

One common approach is to discretize operators: (1) select N *arbitrary* points $G = \{x_i\}_{i=1}^N$ from the domain A; (2) input to the operator becomes the point set $\{(x, f(x)),\ x \in G\}$ in instead of the infinite dimensional object f (3) the approximated operator $\hat{O}_\theta$ maps a function $f \in \mathcal{H}$ to $g \in \mathcal{H}$ with this discretization: $\forall x_i \in G, \quad \hat{O}_\theta(x_i, f(x_i)) = \tilde{g}(x_i) \approx g(x_i) = (\hat{O}f)(x_i)$.

Learned neural-network based operator can be used to solve parametric PDEs. Parametric PDEs are of the form $\mathcal{L}_a u = f$, where $\mathcal{L}_a$ is some operator, a is the parameter, f is a known function, and u is the solution to the equation. We can define a *implicit operators* for a given *parametric PDEs*: $\hat{O} : a \mapsto u$ Learning an approximate $\hat{O}_\theta$ would allow us to get the solution of the given PDEs for any parameter a. In this paper, the form of parametric PDEs considered is the **2D Darcy Flow**:

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \quad x \in (0,1)^2 \\ u(x) &= 0, \qquad x \in \partial(0,1)^2 \end{aligned}$$

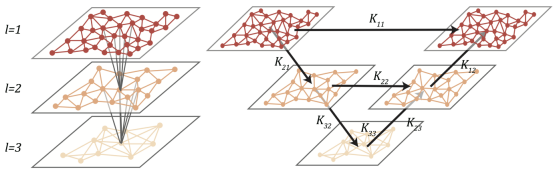

Figure 1: V-cycle
Left: the multi-level graph. **Right**: one V-cycle iteration for the multipole graph kernel network.

Figure 1: Multi-level mesh and V-cycle

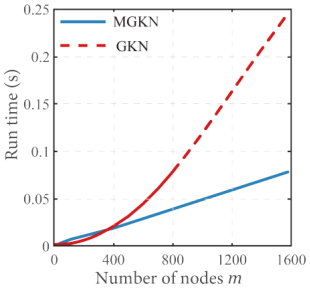

Figure 2: MGKN has linear scaling

The above PDE can be written as $\mathcal{L}_a u = f$. Suppose we have learnt $\hat{O}_\theta$, then we can obtain solutions for this equation given arbitrary $a(\cdot)$: $\hat{O}_\theta(x, a(x)) \approx (\hat{O}_a u)(x) = f(x)$. The dataset for learning the operator $\hat{O}_a$ consist of 100 samples of $a(\cdot)$ sampled from the distribution $\psi_\# \mathcal{N}(0, (-\Delta + 9I)^{-2})$, where for each sampled $a(\cdot)$ the solution u is obtained by using a finite difference solver on 2d grid of size 241×241.

2 Multipole Graph Kernel Network (MGKN)

Since $\mathcal{L}_a$ is uniformly elliptic, using well-known result from boundary element method literature, the solution has the **Green's representation**: given the Green's function $G_a(x, y)$ of $\mathcal{L}_a$, the solution u has the following representation

$$u_\Omega(x) = \underbrace{\int_{\partial\Omega} G_a(x,y) \frac{\partial u(y)}{\partial \mathbf{n}(y)} \cancel{dS(y)}}_{:=[\mathcal{V}_a u_\mathbf{n}](x)} - \underbrace{\int_{\partial\Omega} u(y) \frac{\partial G_a(x,y)}{\partial \mathbf{n}(y)} \cancel{dS(y)}}_{:=[\mathcal{K} u_\Gamma](x)} + \underbrace{\int_\Omega G_a(x,y) f(y)\, dV(y)}_{[\mathcal{N} f](x)} = \int_\Omega G_a(x,y) f(y)\, dV(y).$$

where G_a is a Newtonian potential and Γ_a is some operator. Note that the single ($\mathcal{V}_a$) ane double layer potential operator ($\mathcal{K}_a$) vanished due to the boundary condition $u(x) = 0, \forall x \in \partial(0,1)^2$. Since the operator are discretized and the discretization of domain can be seen as a graph, it make sense to model the Newtonian potential $[\mathcal{N} f](x)$ using **graph convolution**:

$$u(x) = \int_{(0,1)^2} \kappa_\theta(x, a(x), \nabla a(x)) h_{\text{emb}}(a(y)) dy, \quad h_{\text{emb}}(a(y)) \in \mathbb{R}^d,\ \kappa_\theta(x, a(x), \nabla a(x)) \in \mathbb{R}^{d \times d}. \tag{1}$$

where κ_θ is an **edge-conditioned graph convolution network**. However, (a) if we naively use the fully connect graph constructed from $\{x_i\}_{i=1}^N$ the model would not scale well (Figure 2); (b) if we only connect nodes with nearby neighbors, the non-locality of $\kappa_\theta(x, y)$ would not be modelled well. The paper solve the scaling issue by using a multi-resolution graph model called MGKN, which is inspired by the fast multipole method (FMM). *MGKN has local connectivity so it scales linearly (Figure 2), and it also model non-locality well since it operates on multiple resolutions.* The key is to use *Nyström approximation*: $K_{nn} \approx K_{nm} K_{mm} K_{mn}, m \ll n$, where we reduce the kernel size from n^2 to $m^2 + 2nm$. For example, if $m = \frac{n}{4}$, then the low-rank kernel has size $\frac{9}{16} n^2$. This factorization is realized by the v-cycle algorithm (Figure 1), which basically apply K_{mn}, K_{mm}, K_{mn} sequentially. Also, the factorization can be applied recursively on K_{mm}, which further increase the efficiency.

Let's walk through an example with two resolution levels, where the level 1 has 100 nodes and the neighborhood radius is 0.25, and level 2 has 25 nodes and the neighborhood radius is 0.5. We first sample two sets of non-overlapping nodes of size 100 and 25 from the 241×241 grid , and connect the edge within each resolution level. We further create edges between nodes of level 1 and 2 with radius $0.25 * 1.41$. Then we apply GCN (equation 1) on the multi-resolution graph, following the v-cycle.