Adaptive Filters and Aggregator Fusion for Efficient Graph Convolutions

Shyam A. Tailor [1] Felix L. Opolka [1] Pietro Liò [1] Nicholas D. Lane [1,2]

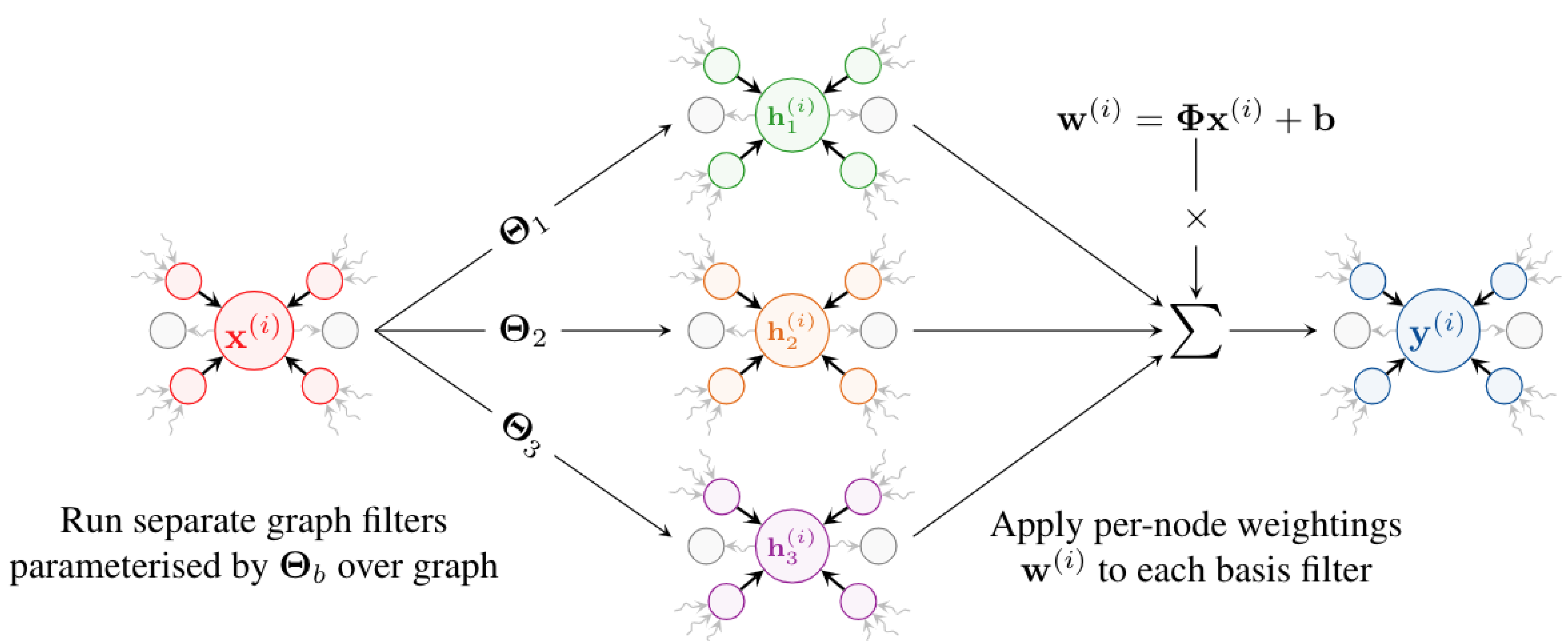
[1]Department of Computer Science and Technology, University of Cambridge [2]Samsung AI Center, Cambridge, UK

Our Contributions

Our work can be viewed in the same way as MobileNet for CNNs. We think carefully about efficiency *without* sacrificing expressivity.

1. We achieve SOTA accuracy with only $\mathcal{O}(V)$ memory consumption by using ***adaptive filters***.
2. We propose ***aggregator fusion***, a technique that leverages observation about hardware implementation to enable large boosts in accuracy for small increases in latency. This technique can be widely adopted by the community.
3. We carefully consider the state of accelerator design, and ***design our architecture so that it can be hardware accelerated***.

Adaptive Filtering for GNNs



Our approach employs B separate basis filters, each of which is parameterised like GCN. These are then combined using ***nodewise-adaptive*** weightings. This can be intuitively interpreted as ***giving each node its own weight matrix***, but also has interesting spectral interpretations. This can be generalised to multiple heads, and to generalised aggregators:

$$\mathbf{y}^{(i)} = \Big\Vert_{h=1}^{H} \sum_{\oplus\in\mathcal{A}} \sum_{b=1}^{B} w^{(i)}_{h,\oplus,b} \bigoplus_{j\in\mathcal{N}(i)\cup\{i\}} \mathbf{\Theta}_b \mathbf{x}^{(j)}$$

Results

Table 1:EGC consistently obtains the best performance against normalised baselines. Any results marked with $\ast$ ran out of memory on the popular Nvidia 1080Ti or 2080Ti GPUs.

Architecture	ZINC (MAE ↓)	CIFAR (Acc. ↑)	MolHIV (ROC-AUC ↑)	Arxiv (Acc. ↑)	Code-V2 (F1 ↑)
GCN	0.459 ± 0.006	55.71 ± 0.38	76.14 ± 1.29	71.92 ± 0.21	0.1480 ± 0.0018
GAT	0.475 ± 0.007	64.22 ± 0.46	77.17 ± 1.37	$\ast\, 71.81 \pm 0.23$	0.1513 ± 0.0011
GIN	0.387 ± 0.015	55.26 ± 1.53	76.02 ± 1.35	67.33 ± 1.47	0.1481 ± 0.0027
MPNN-Sum	0.381 ± 0.005	65.39 ± 0.47	75.19 ± 3.57	$\ast\, 66.11 \pm 0.56$	0.1470 ± 0.0017
MPNN-Max	0.468 ± 0.002	69.70 ± 0.55	77.07 ± 1.37	$\ast\, 71.02 \pm 0.21$	0.1552 ± 0.0022
PNA	0.320 ± 0.032	70.21 ± 0.15	$\mathbf{79.05 \pm 1.32}$	$\ast\, 71.21 \pm 0.30$	$\ast\, 0.1570 \pm 0.0032$
EGC-S	0.364 ± 0.020	66.63 ± 0.26	77.21 ± 1.10	$\mathbf{72.19 \pm 0.16}$	0.1528 ± 0.0025
EGC-M	$\mathbf{0.281 \pm 0.008}$	$\mathbf{71.04 \pm 0.45}$	78.18 ± 1.53	71.96 ± 0.23	$\mathbf{0.1595 \pm 0.0019}$

EGC-S is a single aggregator variant. We beat GAT on every dataset despite the reduction in memory consumption. EGC-M, which uses multiple aggregators, is a clear match for PNA.

Our results are surprising given the efficiency of our approach, and raises further questions about which aspects of GNN architecture design are most important.

Aggregator Fusion

Sparse operations are ***memory-bound***: unstructured sparsity results in difficult to optimise memory access patterns. If our processor spends so much time sitting idle just waiting for data to arrive from memory, why not do some additional computation during the wait? This is the key to aggregator fusion: ***apply all aggregators at once, rather than performing multiple fetches***. We find this results in an average latency increase of ***19%*** at inference time; the naive approach results in an increase of ***305%***. In principle this approach can be readily integrated into upstream libraries, and applied to architectures such as PNA as well.

Learning More About This Work

We release code and pretrained models on GitHub. A blog post describing our research in more depth can be found with the QR code. Please feel welcome to contact Shyam Tailor (sat62@cam.ac.uk) with queries and thoughts.