CLIQUE POOLING FOR GRAPH CLASSIFICATION

Enxhell Luzhnica[*], **Ben Day**[*] **& Pietro Lió**
Department of Computer Science & Technology
University of Cambridge
Cambridge, United Kingdom.
ben.day@cl.cam.ac.uk

ABSTRACT

We propose a novel graph pooling operation using cliques as the unit pool. As this approach is purely topological, rather than featural, it is more readily interpretable, a better analogue to image coarsening than filtering or pruning techniques, and entirely nonparametric. The operation is implemented within graph convolution network (GCN) and GraphSAGE architectures and tested against standard graph classification benchmarks. In addition, we explore the backwards compatibility of the pooling to regular graphs, demonstrating competitive performance when replacing two-by-two pooling in standard convolutional neural networks (CNNs) with our mechanism.

1 INTRODUCTION & RELATED WORK

The ongoing deep learning renaissance has proved remarkably fruitful and long lived, with state-of-the-art performance in tasks spanning the breadth of machine learning. The dominance of CNNs in the domain of image classification is of particular note, with superhuman performance becoming almost pedestrian. Images can be thought of as highly regular, Euclidean graphs, where pixels are nodes connected to the eight neighbouring pixels by edges. Graphs of different structures are used to represent problems from the biological, social and physical (Kipf et al., 2018; Gilmer et al., 2017) sciences as well as more abstract problems such as knowledge representation. Generalising the advances made for CNNs for use on irregular graphs has thus become an important direction for the application of deep learning, under the umbrella term geometric deep learning (Bronstein et al., 2017).

The key operations in CNNs are the convolution and pooling. Convolutions extract features, with adaptations for the exploitation of locality and translational invariance. Pooling literally reduces the spatial dimensionality, aiding in the expansion of the receptive field and building consensus and saliency through coarsening. The convolution has been well adapted to non-Euclidean graphs (Kipf & Welling, 2016; Defferrard et al., 2016; Veličković et al., 2017; Gilmer et al., 2017) with many variations as a result of a strong interest from the research community. Pooling has not received the same treatment, unsurprisingly given its CNN and not PNN, and prior to recent developments the standard method was to pool globally[1] and then feed this into a multilayer perceptron (MLP) (Duvenaud et al., 2015). The current state-of-the-art methods implement gradual coarsening in hierarchies of representations (Gao & Ji, 2018; Cangea et al., 2018; Ying et al., 2018) but require hyperparameterisation, in the form of a preset allowed number of clusters or number of nodes to prune, and are not purely topologically derived. This means they are poor analogues of pooling in CNNs. If the aim of the community is to produce robust, transferable algorithms able to be used with many graphs then the inapplicability of these methods to regular graphs should be viewed as a serious deficiency.

In this work we introduce an operation that is purely topological, static, nonparametric and, has a natural correspondence in regular graphs and when substituted into GNN or traditional CNN for image classification, achieves performance competitive with the state-of-the-art parametric alternatives and

[*]Equal contribution

[1]Take all node features and form a single set of features through some aggregation, typically element-wise max or averaging.

improves on the performance of other nonparametric approaches. Being nonparametric renders the approach far more interpretable as there are no learned parameters to pick apart – we can state quite clearly what our approach does, it puts nodes into groups where every member is connected to every other member. In addition to these properties the operation is biased towards producing a dendritic pooling hierarchy which, in combination with being static and precomputable, permits concurrent processing of the graph without loss of accuracy.

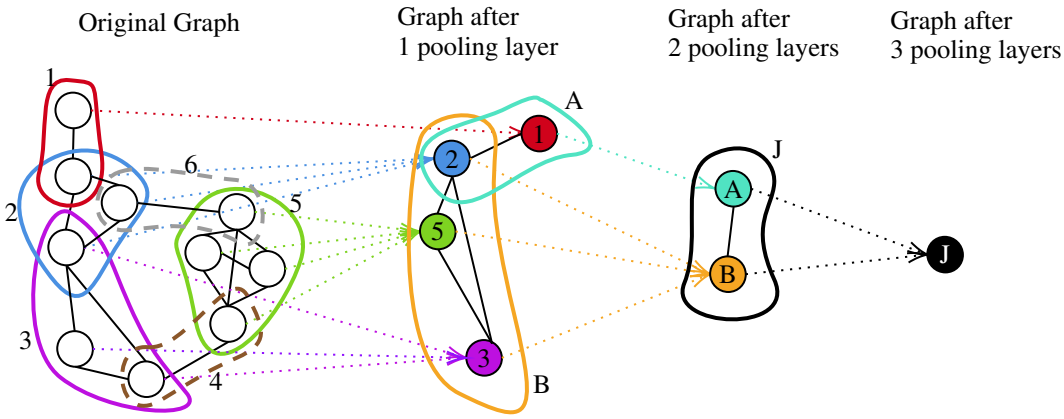

Figure 1: Clique Pooling for an irregular graph. The colored borders represent the maximal cliques (also labeled with numbers or letters next to them), dotted arrows indicate the cliques to which the nodes are assigned. Notice that although some of the nodes belong to more than one clique they do not necessarily contribute to the respective node in the coarsened graph. This is the case for the node belonging to the red (1) and blue (2) maximal clique. Since the blue clique is bigger (in terms of nodes), the node is assigned to the blue (2) cluster only. In the case of the node intersecting the blue (2) and purple (3) maximal cliques, the node is assigned to both cliques since the cliques have the same size. Hence, it contributes to the features of both of the respective nodes in the coarsened graph. The grey maximal clique (6) is not represented in the new coarsened graph since the nodes in that clique have already been assigned to larger cliques: the green (5) and blue (2) cliques. The nodes is the coarsened graph are connected if any two nodes in the respective cliques are connected.

2 Proposed Method

2.1 Preliminaries

We use the standard representation of graphs in graph classification tasks. The graph G with N nodes is represented as a pair (A, X) where the adjacency matrix $A \in \mathbb{R}^{N \times N}$ and the node feature matrix, with F features per node, $X \in \mathbb{R}^{N \times F}$. Additionally, our method assumes undirected graphs.

Graph Convolution In principle, our method does not require a convolution operator. However, a CNN-like architecture requires one. The graph convolution needs to be inductive. In our experiments we use GraphSAGE and GCN as particular implementations of the message-passing scheme.

Readout Function We use readout functions to combine pools of nodes to form the features in the coarsened graph and to generate representations of entire layers to be fed into the MLP classifier. For pool aggregation we take the mean over nodes in the pool. For whole-layer representation we use the concatenation of the mean and maximum over all nodes in the layer. The whole input to the MLP is the concatenation of each layer representation. CNNs typically take the maximum within pools until the final layer when a whole-field pool reduces the 3-tensor, $[H, W, F]$, to a vector, $[1, 1, F] \rightarrow [F]$.

2.2 Proposed Operation

Our approach is to coarsen the graph by aggregating maximal cliques. We attempt to limit the dispersal of nodes by assigning each to a single clique-pool, ranked by size, where possible. Only in the case of equally large options is the node assigned to multiple pools.

To accomplish this, the maximal cliques are found using the Bron-Kerbosch algorithm (Bron & Kerbosch, 1973) with modifications shown to improve performance on large real-world graphs (Eppstein et al., 2010; Eppstein & Strash, 2011). Nodes are assigned to pools greedily starting with the largest. When a node has been assigned it is removed from the remaining smaller pools and does not count towards that pool's size for the assignment of other nodes. Edges are inherited from the clique members such that if a node in one clique shares an edge with a node in another, the cliques will share an edge in the coarsened graph. The resulting graph shares some features with the clique-graph[2] though with fewer nodes and more edges in all but the simplest cases. The assigned cliques are then pooled using whichever pooling function is desired, average- and max-pooling in the experiments presented here.

Lemma 1 (Convergence)**.** *Given a connected and finite graph, the clique-pooling operator converges to a single node after finitely many steps.*

The proof for the above can be found in Appendix C. This guarantees that in any graph we can have a CNN-like architecture. As the approach is based only on the topology of the original graph it is both nonparametric and static. This means that the pools can be precomputed and opens the door to greater parallelization for dealing with very large graphs. Having found the entire pooling structure it is trivial to generate the dependency diagram, allowing graph partitions to be loaded and operated on separately, which will be key to dealing with very-large graphs, as discussed in Zhang et al. 2005 for the case of systems biology.

2.3 Application to Images as Regular Graphs

Images can be thought of as highly regular graphs. Indeed, this is the primary motivation for most work in the field of graph neural networks (Kipf & Welling, 2016; Defferrard et al., 2016) and often image recognition techniques are straightforwardly adapted to graphs with great success (Veličković et al., 2017; Cangea et al., 2018). Our method goes against the grain in this sense as, whilst it is based on concepts native to irregular graphs, we are able to apply it to images without issue.

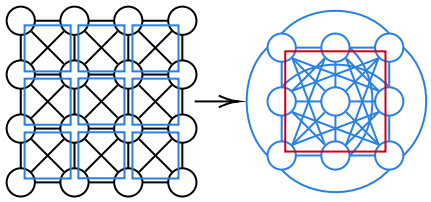

Figure 2: Clique pooling for a regular graph. The left graph represents a 4 pixel by 4 pixel image with nodes and edges in black and blue squares showing the maximal cliques. The right graph is the result of pooling where the maximal cliques have become nodes. Note that the neighbourhoods of each node have grown to include their neighbours' neighbours, resulting in a maximal clique of nine nodes shown by the red square.

To do this we must first define the graph structure for images. We argue that the use of 3-by-3 convolutions implies the graph structure as shown in figure 2, that is, pixels are connected to their eight immediate neighbours. As the figure shows, the first pool will be 2-by-2 with a stride of 1. The second pool will then be 3-by-3 with a stride of 1. We can analyse how this progresses by considering the 1-dimensional case, without loss of generality, as the 2D case is no more complicated than the same thing happening in two directions at the once[3].

Lemma 2 (Length Reduction)**.** *Applying clique-pool n-times on a 1-dimensional grid (a chain) reduces the length of the grid by $r_n = 2^n - 1$.* [4]

In a typical CNN, the architecture will be such that the pools reduce the input to a single pixel in the spatial dimension by the final layer, using (2×2) pooling to do so. The length can be expressed in terms of the number of pools, n, as $L = 2^n$ and therefore, conveniently,

$$L - r_n = L - 2^n + 1 = 1$$

so the same number of clique-pooling operations will reduce the input to a single spatial dimension. As such the clique pool operation can be substituted into existing CNN architectures directly, replacing pools without any additional changes.

[2] The graph formed by maximal cliques where edges occur where cliques intersect.

[3] And the same can be said for an N-dimensional grid.

[4] See Appendix C for proof and illustration.

3 EXPERIMENTAL SETUP & RESULTS

We test the approach extensively on irregular graph classification problems and also demonstrate how the pooling works being substituted into a standard CNN architecture in the VGG-form.

	DATASETS			
MODEL	ENZYMES	DD	COLLAB	PROTEINS
GRAPHLET	41.03	74.85	64.66	72.91
SHORTEST-PATH	42.32	78.86	59.10	76.43
1-WL	53.43	74.02	78.61	73.76
WL-QA	60.13	79.04	80.74	75.26
PATCHYSAN	-	76.27	72.60	75.00
GRAPHSAGE	54.25	75.42	68.25	70.48
ECC	53.50	74.10	67.79	72.65
SET2SET	60.15	78.12	71.75	74.29
SORTPOOL	57.12	79.37	73.76	75.54
DIFFPOOL-DET	58.33	75.47	**82.13**	75.62
DIFFPOOL-NOLP	62.67	79.98	75.63	77.42
DIFFPOOL	64.23	**81.15**	75.50	**78.10**
SPARSE HGC CANGEA ET AL. (2018)	**64.17**	78.59	74.54	75.46
CLIQUEPOOL (OURS)	60.71	77.33	74.50	72.59

Table 1: Classification accuracy percentages.

We benchmark the performance on irregular graphs on standard datasets. For *Enzymes, DD, Proteins* we use the same architecture: two blocks of Graph Convolution Network + Pool followed by a convolutional layer. We use the mean as the readout function. For *Enzymes* we use GraphSAGE ('mean' variant) for the convolutions, and GCN for others. The outputs of each layer are normalised by the L^2-norm. For *Collab* we use the GCN and one layer of pooling, followed by another convolution layer. To provide a comparison of the number of pooling parameters, in the case of *DD*, DiffPool requires $27,776$ parameters, Graph U-Net requires 192 whilst our method is nonparametric (0 parameters). We use 128 hidden units for *Enzymes*, and 64 for the other datasets. We use the Adam optimizer (learning rate of 0.0001, weight decay of 0.001). We train for 1000 epochs for *Enzymes*, 100 for *DD*, 500 for *Proteins*, and 200 for *Collab*.

Regular graph pooling is tested against the CIFAR-10 benchmark using the standard architecture proposed by Jetley et al. 2018, an adaptation of the VGG ImageNet entry Simonyan & Zisserman (2015). The architecture consists of alternating stacks of convolutional layers, zero-padded to maintain spatial dimensions, and (2×2) max-pools with a stride of 2 to reduce the spatial dimensions by half, with a final multilayer perceptron (MLP) classifier[5]. CIFAR images are $(32, 32, 3)$ so five stride-2 pools are needed to reduce to $(1, 1, F)$. Following the derivation in section 2.3, these are replaced by $(2 \times 2), (3 \times 3), (5 \times 5), (9 \times 9)$ & (17×17) pools all with stride of 1.

Our experiments show that our non-parametric approach is competitive with the parametric approaches, as detailed in table 3. The method outperforms the GraphSAGE baseline and most of the kernel-based and GNN approaches. Moreover, because we do not introduce any additional parameters our method is fast to train and does not suffer the instabilities associated with DIFFPOOL. It also outperforms the DIFFPOOL with deterministic clustering on two datasets. The image investigation found a small, but significant ($p = 0.02$), reduction of in mean accuracy over the 2-BY-2 pool baseline in a 10-fold cross-validation comparison, presented in table 3.

MODEL	MEAN	STD. DEV.
2-BY-2 POOL	**92.4**	0.3
CLIQUE POOL	92.0	0.5

TABLE 2: CIFAR-10 RESULTS

[5]Full specification is given in appendix A

REFERENCES

Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 9 1973. ISSN 00010782. doi: 10.1145/362342.362367. URL http://portal.acm.org/citation.cfm?doid=362342.362367.

Michael M. Bronstein, Yann LeCun, Arthur Szlam, Pierre Vandergheynst, and Joan Bruna. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 11 2017. ISSN 1053-5888. doi: 10.1109/msp.2017.2693418. URL http://arxiv.org/abs/1611.08097http://dx.doi.org/10.1109/MSP.2017.2693418.

Ctlina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards Sparse Hierarchical Graph Classifiers. 11 2018. URL http://arxiv.org/abs/1811.01287.

Alessio Conte, Roberto De Virgilio, Antonio Maccioni, Maurizio Patrignani, and Riccardo Torlone. Finding All Maximal Cliques in Very Large Social Networks. doi: 10.5441/002/edbt.2016.18. URL https://openproceedings.org/2016/conf/edbt/paper-54.pdf.

Michal Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. 6 2016. URL http://arxiv.org/abs/1606.09375.

David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Aln Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. 9 2015. URL http://arxiv.org/abs/1509.09292.

David Eppstein and Darren Strash. Listing All Maximal Cliques in Large Sparse Real-World Graphs. 3 2011. URL http://arxiv.org/abs/1103.0318.

David Eppstein, Maarten Löffler, and Darren Strash. Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time. 6 2010. URL http://arxiv.org/abs/1006.5440.

Hongyang Gao and Shuiwang Ji. Graph u-net. 2018.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. 4 2017. URL http://arxiv.org/abs/1704.01212.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2 2015. URL http://arxiv.org/abs/1502.03167.

Saumya Jetley, Nicholas A. Lord, Namhoon Lee, and Philip H. S. Torr. Learn To Pay Attention. 4 2018. URL https://arxiv.org/abs/1804.02391.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural Relational Inference for Interacting Systems. 2 2018. URL http://arxiv.org/abs/1802.04687.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. 9 2016. URL http://arxiv.org/abs/1609.02907.

J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 3 1965. ISSN 0021-2172. doi: 10.1007/BF02760024. URL http://link.springer.com/10.1007/BF02760024.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Technical report, 2015. URL http://www.robots.ox.ac.uk/.

Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. 10 2017. URL http://arxiv.org/abs/1710.10903.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. 2018. ISSN 18160948. doi: arXiv:1806.08804v3. URL https://arxiv.org/pdf/1806.08804.pdfhttp://arxiv.org/abs/1806.08804.

Yun Zhang, Faisal N. Abu-Khzam, Nicole E. Baldwin, Elissa J. Chesler, Michael A. Langston, and Nagiza F. Samatova. Genome-scale computational approaches to memory-intensive applications in systems biology. In *Proceedings of the ACM/IEEE 2005 Supercomputing Conference, SC'05*, 2005. ISBN 1595930612. doi: 10.1109/SC.2005.29.

A NETWORK ARCHITECTURES

In the interests of clarity and reproducibility we present here the network architectures used in our investigations. We believe the following to be a complete description though the authors would gladly welcome any correspondence requesting further specification, clarification or suggestions relating to improving these descriptions.

For compactness, we've used abbreviations. CONV.* is a block consisting of a convolutional layer with a 3×3 kernel, batch-normalization (Ioffe & Szegedy, 2015) with numerical stabilisation ($\epsilon = 1 \times 10^{-}5$) and momentum of 0.1 for the affine transform parameters (γ, β), with a RELU activation. '$\times$ N' refers to how many times this block is repeated, which could also be inferred by from the numbering in the LAYER column. FLATTEN returns the layer input with dimensions of size 1 removed, sometimes called *squeezing*. LINEAR layers are fully-connected and followed by a RELU activation except in the final layer where a softmax is used. For pooling layers the size is given as $(\Delta x \times \Delta y), stride$.

LAYER	TYPE	SIZE/FEATURES	SHAPE
INPUT	-	-	(32,32,3)
1,2	CONV.* $\times$ 2	64	(32,32,64)
-	MAXPOOL	(2×2), 2	(16,16,64)
3,4	CONV.* $\times$ 2	128	(16,16,128)
-	MAXPOOL	(2×2), 2	(8,8,128)
5,6,7	CONV.* $\times$ 3	256	(8,8,256)
-	MAXPOOL	(2×2), 2	(4,4,256)
8,9,10	CONV.* $\times$ 3	512	(4,4,512)
-	MAXPOOL	(2×2), 2	(2,2,512)
11,12,13	CONV.* $\times$ 3	512	(2,2,512)
-	MAXPOOL	(2×2), 2	(1,1,512)
-	FLATTEN	-	(512)
14	LINEAR	512	(512)
-	DROPOUT	$p = 0.3$	(512)
15	LINEAR	10	(10)

TABLE 3: BASELINE VGG-LIKE ARCHITECTURE.

LAYER	TYPE	SIZE/FEATURES	SHAPE
INPUT	-	-	(32,32,3)
1,2	CONV.* $\times$ 2	64	(32,32,64)
-	MAXPOOL	(2×2), 1	(31,31,64)
3,4	CONV.* $\times$ 2	128	(31,31,128)
-	MAXPOOL	(3×3), 1	(29,29,128)
5,6,7	CONV.* $\times$ 3	256	(29,29,256)
-	MAXPOOL	(5×5), 1	(25,25,256)
8,9,10	CONV.* $\times$ 3	512	(25,25,512)
-	MAXPOOL	(9×9), 1	(17,17,512)
11,12,13	CONV.* $\times$ 3	512	(17,17,512)
-	MAXPOOL	(17×17), 1	(1,1,512)
-	FLATTEN	-	(512)
14	LINEAR	512	(512)
-	DROPOUT	$p = 0.3$	(512)
15	LINEAR	10	(10)

TABLE 4: CLIQUE-POOLED VGG-LIKE ARCHITECTURE.

B MAXIMAL CLIQUES

The enumeration of maximal cliques has been a core component in gene expression networks analysis, cis regulatory motif finding, and the study of quantitative trait loci for high-throughput molecular phenotypes Zhang et al. (2005). As such, significant effort has gone to developing efficient methods

of enumerating all the maximal cliques in a graph. The upper bound of the number of cliques is $3^{n/3}$. The algorithm for finding maximal cliques presented by Bron & Kerbosch 1973, was adapted by Tomita et al. 2006 to find the maximal cliques in an iterative way without having to store previous cliques or many candidates in memory. Zhang et al. (2005) showed a way of enumerating the cliques, sorted by size, in a parallelized way. While their primary focus is parallelizing this method in shared-memory machines, they show that it is possible to do the same in distributed machines. They also demonstrate that it is possible to load-balance the sub-tasks efficiently, showing a near ideal relative speedup (defined as the ratio between $2p$ processors and p processors run times). Finally, several modifications have been made showing improved performance on large real-world graphs (including social graphs) Eppstein et al. (2010); Eppstein & Strash (2011); Conte et al.

C PROOFS

C.1 CONVERGENCE

We recall,

Lemma 1 (Convergence)**.** *Given a connected and finite graph, the clique-pooling operator converges to a single node after finitely many steps.*

Proof. Consider the shortest-path between two nodes, $d(u, v)$, and the corresponding distance in the new graph, $d(u', v')$ where u' and v' are the most distant cliques containing u and v, respectively. For each assigned maximal clique that the path between u and v traverses, $d(u', v')$ is reduced from $d(u, v)$ by 1. If the path does not traverse any pooled cliques then the distance remains constant as the path will be unchanged. As there is always a largest clique, the distance between some nodes is always reduced in the newly formed graph. Therefore the sum of the distances between all pairs of original nodes *must* decrease in each pooling. As the graph is finite and connected the sum of the distances between nodes must also be finite and so will reduce to 0, a single node, within a finite number of pooling operations. □

However, it is possible to construct graphs that will initially grow in the number of nodes through pooling, the most straightforward example being a bipartite graph. In this case every pair of nodes sharing an edge forms a maximal clique, all of size 2 and thus all equally large. The pooled-graph then has as many nodes as the original graph had edges. In the worst possible case, the number of nodes in the pooled graph will balloon to the maximal clique limit, $\mathcal{O}(3^{n/3})$ (Moon & Moser, 1965), although this will be fully-connected and collapse immediately.

C.2 RECEPTIVE FIELD

We recall,

Lemma 2 (Length Reduction)**.** *Applying clique-pool n-times on a 1-dimensional grid (a chain) reduces the length of the grid by $r_n = 2^n - 1$.*

Proof. Figure 3 illustrates how the pools grow over successive iterations. The trend appears to be $2^n - 1$ and we can show that this is indeed the case. Consider the size of the cliques at a particular iteration, c_i, and the distance along the chain each node is connected, d_i. By inspection, each node is connected to every neighbour up to d_i and so this group forms a clique of size

$$c_i = d_i + 1$$

with the additional 1 accounting for the node itself. Tracing the inheritance of connections into the next layer gives the connected distance of the pooled nodes as

$$\begin{aligned} d_{i+1} &= \tfrac{1}{2}(c_i - 1) + d_i + \tfrac{1}{2}(c_i - 1) \\ &= c_i - 1 + d_i = 2d_i. \end{aligned}$$

Where we first go up one side of the clique, along the connected distance and then back down the side of another clique. So the distance does double each time with the cliques, and pools, being one more. The pooling routine as applied to images then, is to use pools of increasing size, with no

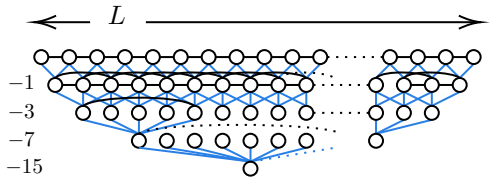

Figure 3: In this diagram we consider the pooling operation applied to a 1D chain. The 2D case, that of images, only differs in that the process occurs in both directions in parallel – the pools remain the same size. Nodes and edges are shown in black, blue indicates how the nodes are pooled into the cliques with the next chain being the result of the pooling operation. In this way we can illustrate four iterations of the pooling operation. In the first the cliques are of size two and the chain reduces in length by one – half from each end. In the second the cliques are of size three, due to the greater reach of the inherited edge connections, and the chain has reduced by two more, one from each end, for a total of three. This process continues with the reach becoming greater, the size of the pools increasing and the chain reducing more rapidly, as indicated on the left.

padding, and a unit stride. Specifically, given the initial degree of 1 implied by the convolutions, $(2 \times 2), (3 \times 3), (5 \times 5), (9 \times 9) \ldots (2^{n-1} + 1 \times 2^{n-1} + 1)$

In addition to the simplicity of the operation, if we observe the reduction in the length a second benefit is made apparent. On the left we reduce by the left half of the pool, on the right by the right half, but not by the middle on either. So the length is reduced by $c_i - 1 = d_i$. As d_i doubles each time, starting from 1, the reduction after $n-$pools is

$$r_i = \sum_{i=1}^{n} 2^{n-1} = 2^n - 1.$$

□