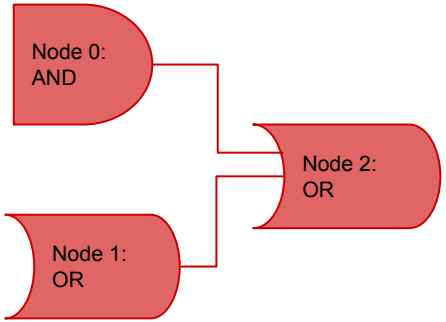
Behavioral Verilog	Structural RTL	Tree	Edge List	Matrix	CNN
C = A + B D = X * Y	OR(...) AND(...) XOR(...)	AND Inputs: ... Outputs: ... OR Inputs: ... Outputs: ...	A — AND — B C — OR — D	[1 2 3 4 5 . . . .]	1 Conv Layer 1 Max Pool Layer 1 FC Layer L2 Loss

Behavioral Verilog	Structural RTL	Tree	Graph
C = A + B D = X * Y	OR(...) AND(...) XOR(...)	AND Inputs: ... Outputs: ... OR Inputs: ... Outputs: ...	AND1 - AND2 AND2 - OR7 ...

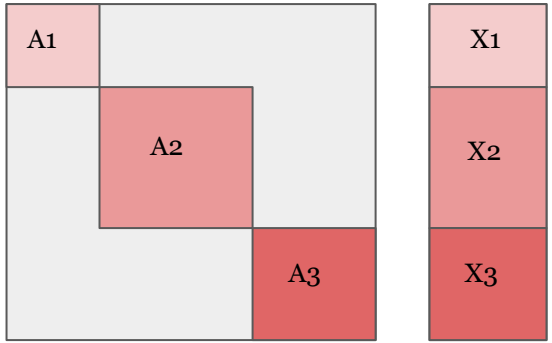


$$\begin{array}{c} \\ \text{Node 0} \\ \text{Node 1} \\ \text{Node 2} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Node 0} & \text{Node 1} & \text{Node 2} \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

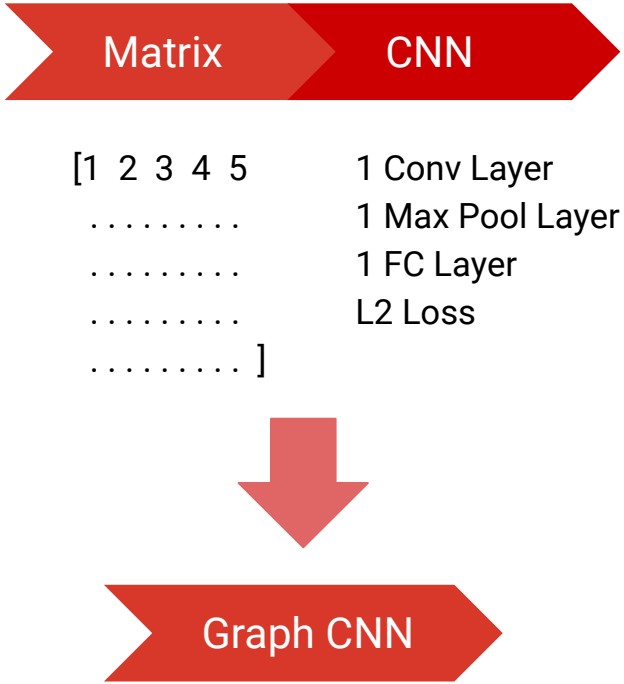
Adjacency Matrix

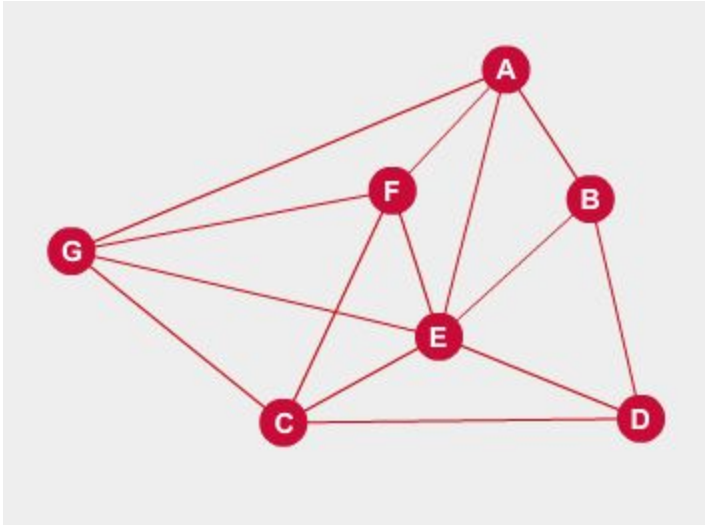
$$\begin{array}{c} \\ \text{Node 0} \\ \text{Node 1} \\ \text{Node 2} \end{array} \begin{array}{c} \\ \begin{pmatrix} AND \\ OR \\ OR \end{pmatrix} \end{array}$$

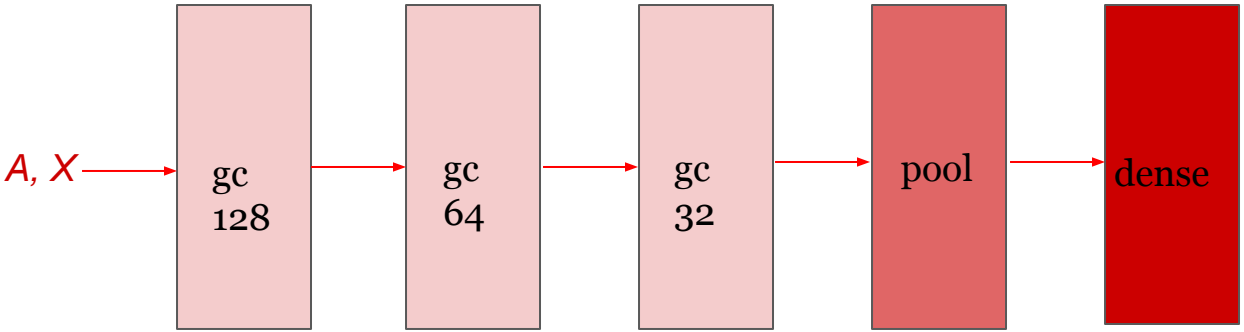
Node Feature Matrix

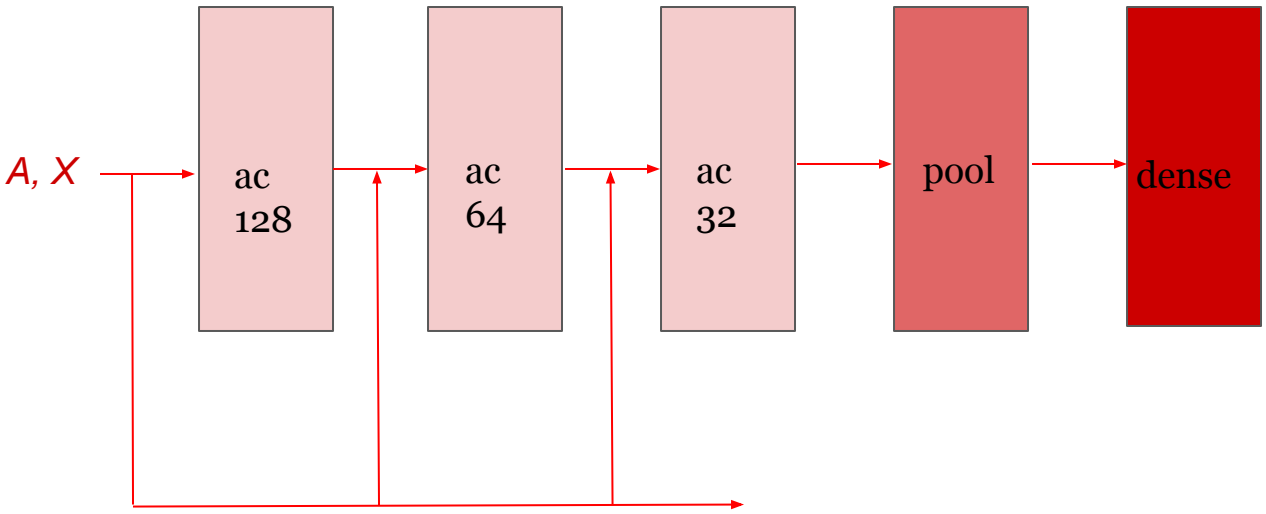

A1
A2
A3

X1
X2
X3


Matrix
CNN
[1 2 3 4 5
. . . . . .
. . . . . .
. . . . . .
. . . . . .]
1 Conv Layer
1 Max Pool Layer
1 FC Layer
L2 Loss
Graph CNN


A
F
B
G
E
C
D


A, X
gc
128
gc
64
gc
32
pool
dense


A, X
ac
128
ac
64
ac
32
pool
dense

Predicting Power and Area for Chip Design

Kalhan Koul
kkoul@stanford.edu

Kartik Prabhu
kprabhu7@stanford.edu

Introduction

Motivation

- Chip designers write code in a hardware description language (ex. Verilog), which is consumed by computer-aided design tools to realize the actual transistors on a chip, as well as their layout
- These CAD tools have a long runtime, owing to their computationally expensive algorithms
- As a result, it can take several hours for a designer to get feedback on the power consumption and area of their design, making it difficult to effectively iterate on designs
- With a deep learning approach, it is possible to predict the area and power of a design in a fraction of the time that CAD tools take

Approach

- Representing code as an input to a neural network is nontrivial and will greatly influence the accuracy of the neural network
- Some work exists that demonstrate approaches to representing code as a vector, but these are only effective for programming languages, not hardware description languages
- Instead, our approach converts Verilog code into a graph of logic gates, which can be represented as an adjacency matrix and node feature matrix to the neural network

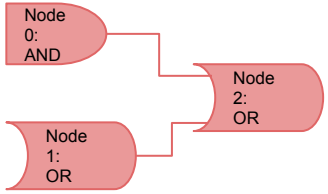


Figure 1: Example Circuit

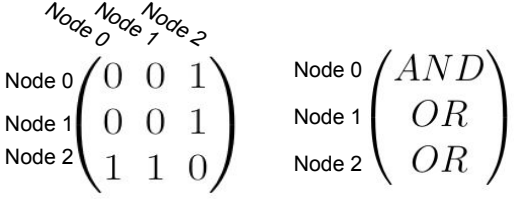


Figure 2: Adjacency and Node Feature Matrix for Example Circuit

Dataset

- Synopsys DesignWare library, consisting of basic building blocks of digital circuits
- Generated the adjacency matrix and node feature matrix for 4569 designs and their corresponding power consumption and area values

Model

Model Inputs

- Circuits have a variable number of gates, so each graph has a variable number of nodes
- Utilize a disjoint union of each graph per batch

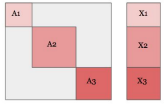


Figure 3: Disjoint Union of each graph

Baseline Model

- Consists of GraphConv (gc) layers
- X is node features matrix and $\tilde{A}$ is the normalized Laplacian matrix
- W, b are trainable parameters
- Activation Function: ReLU

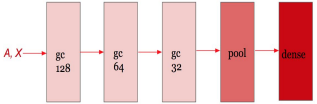


Figure 4: Baseline Model Diagram

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$\textit{Laplacian Matrix} = \textit{Degree Matrix} - \textit{Adjacency Matrix}$$

$$Z = \sigma(\tilde{A}XW + b)$$

(1): Traditional GCN Layer

ARMA Model

- Consists of ARMAConv (ac) layers
- X is node features matrix and $\tilde{A}$ is the normalized Laplacian matrix
- W, V are trainable parameters
- Activation Function: ReLU
- Utilizes skip connections

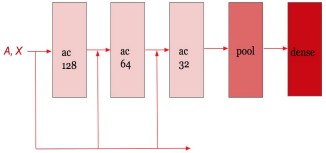


Figure 5: ARMAModel Diagram

$$\bar{X}_k^{(t+1)} = \sigma(\tilde{A}\bar{X}^{(t)}W^{(t)} + XV^{(t)})$$

(2): Skip Layer

$$Z = 1/K \sum_{k=1}^{K} \bar{X}_k^{(T)}$$

(3): ARMA Layer

Results and Discussion

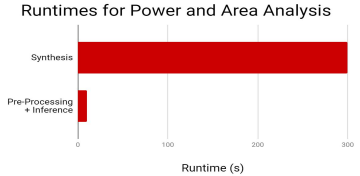
Metric

- Percentage of power and area estimates within 20%

Model	Training data (4113)	Test Data (456)
GraphConv	Power: 68% Area: 65%	Power: 58% Area: 61%
ARMA	Power: 75% Area: 69%	Power: 63% Area: 61%

Runtime

- Synthesis runtime is design size dependent, but inference is constant
- Runtime on 64-bit carry look ahead adder



Discussion

- The model does relatively well in the context of the problem
- Provides an almost instantaneous method of getting a rough estimate of power and area

Future Work

- Add constraints (clock speed, etc.) as inputs to the neural network
- Add process technology as inputs
- Train larger, varied networks, with data from real designs

References

[1] Filippo Maria Bianchi, Daniele Grattarola, Cesare Alippi, and Lorenzo Livi. Graph neural networks with convolutional arma filters, 2019.
[2] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864.ACM, 2016.
[4] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.5
[5] Hao Peng, Lili Mou, Ge Li, Yuxuan Liu, Lu Zhang, and Zhi Jin. Building program vector representations for deep learning. Lecture Notes in Computer Science, page 547–553, 2015.
[6] Clifford Wolf and Johann Glaser. Yosys-a free verilog synthesis suite.