Neural Sculpting:
Uncovering hierarchically modular task structure in neural networks through pruning and network analysis

Shreyas Malakarjun Patil, Loizos Michael, Constantine Dovrolis

Coauthors and affiliations

Shreyas Malakarjun Patil [1]
sm_patil@gatech.edu

Loizos Michael [3,4]
loizos@ouc.ac.cy

Constantine Dovrolis [2,1]
constantine@gatech.edu

1 Georgia Tech

2 The Cyprus Institute
RESEARCH·TECHNOLOGY·INNOVATION

3 Open University of Cyprus

4 CYENS Centre of Excellence

Most tasks/functions in nature are hierarchically modular

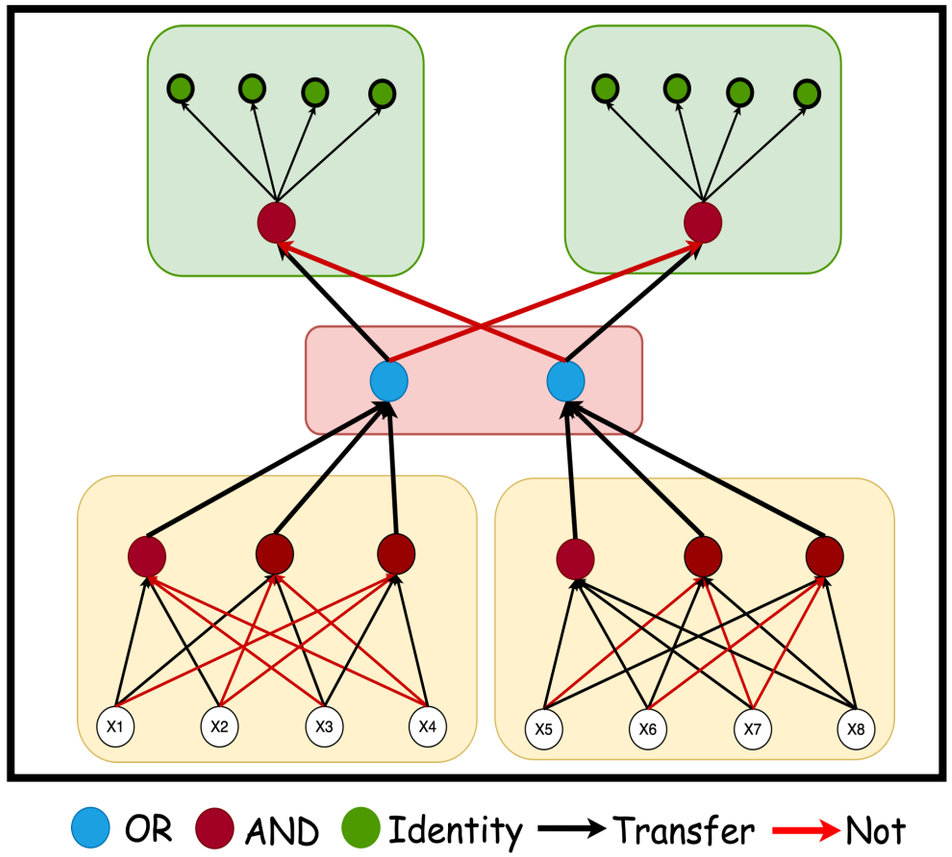
- *Modularity:* Functions can be decomposed into sub-functions

Example : Learning to classify visual, auditory or haptic inputs

- *Hierarchy:* Simpler sub-functions are used as inputs in functions of higher complexity

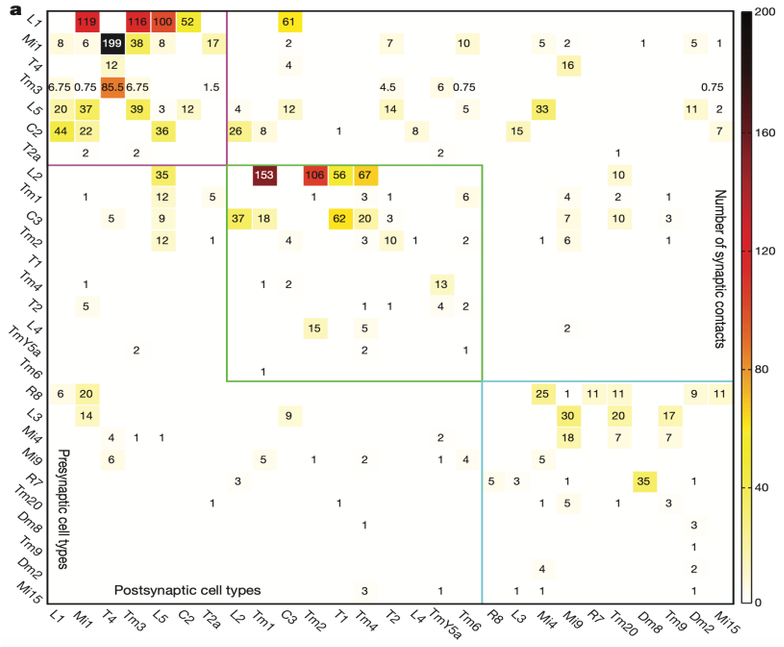
Example : Learning to read sentences by first learning letters -> words -> context and so on

A hierarchically modular Boolean function

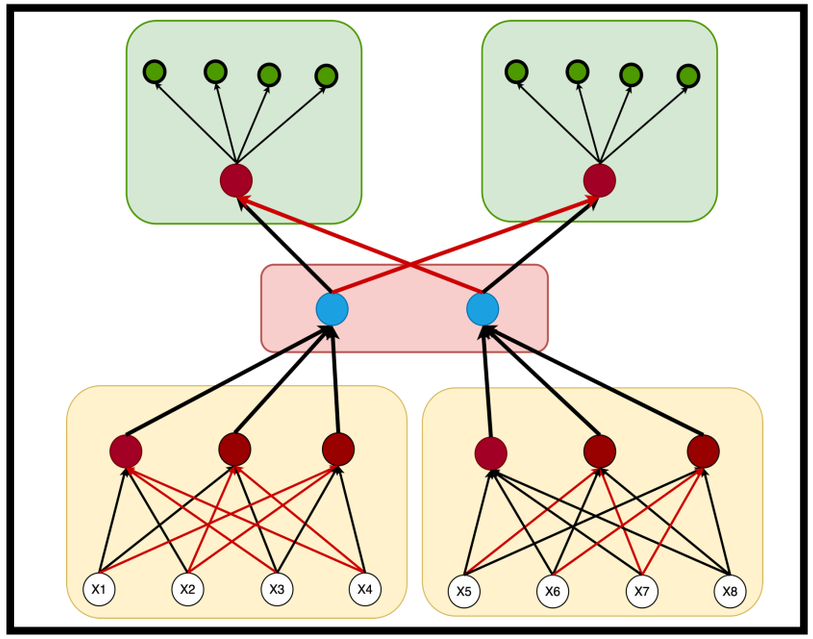


In nature, functional organization shapes structural organization

Hierarchically Modularity
in optic medulla connectome



Hierarchically Modularity
in natural tasks



- Consists of densely connected clusters of neurons with sparse inter-cluster connectivity.
- The repeated module of the optic medulla is a motion detection circuit.

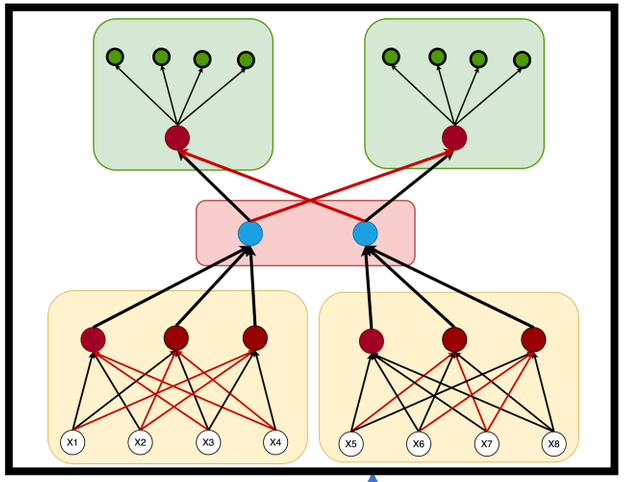
More efficient learning & inference

Fast adaptability

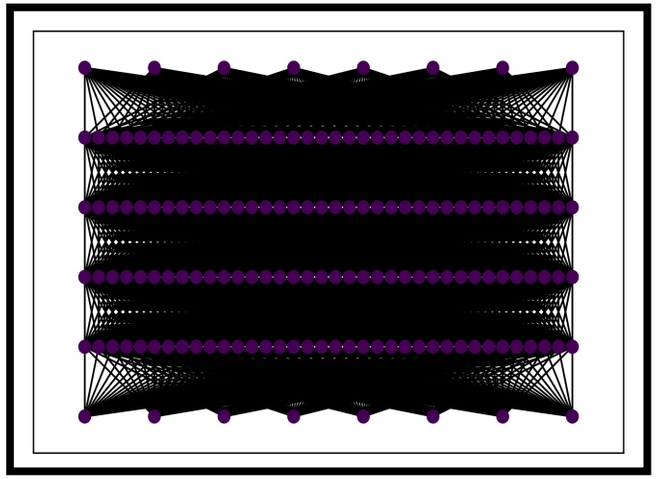
Takemura et. al : **A visual motion detection circuit suggested by *Drosophila* connectomics**, *Nature*, *500*(7461), 175-181.

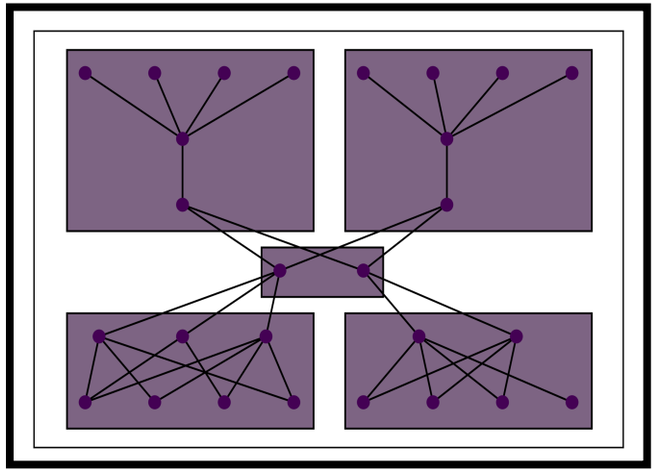
Can this also happen in artificial neural networks?

Given a hierarchically modular target function,



if a deep NN is trained to learn that function





how can we uncover the underlying functional hierarchy from the resulting network?

Two key properties of the sub-functions we aim to detect

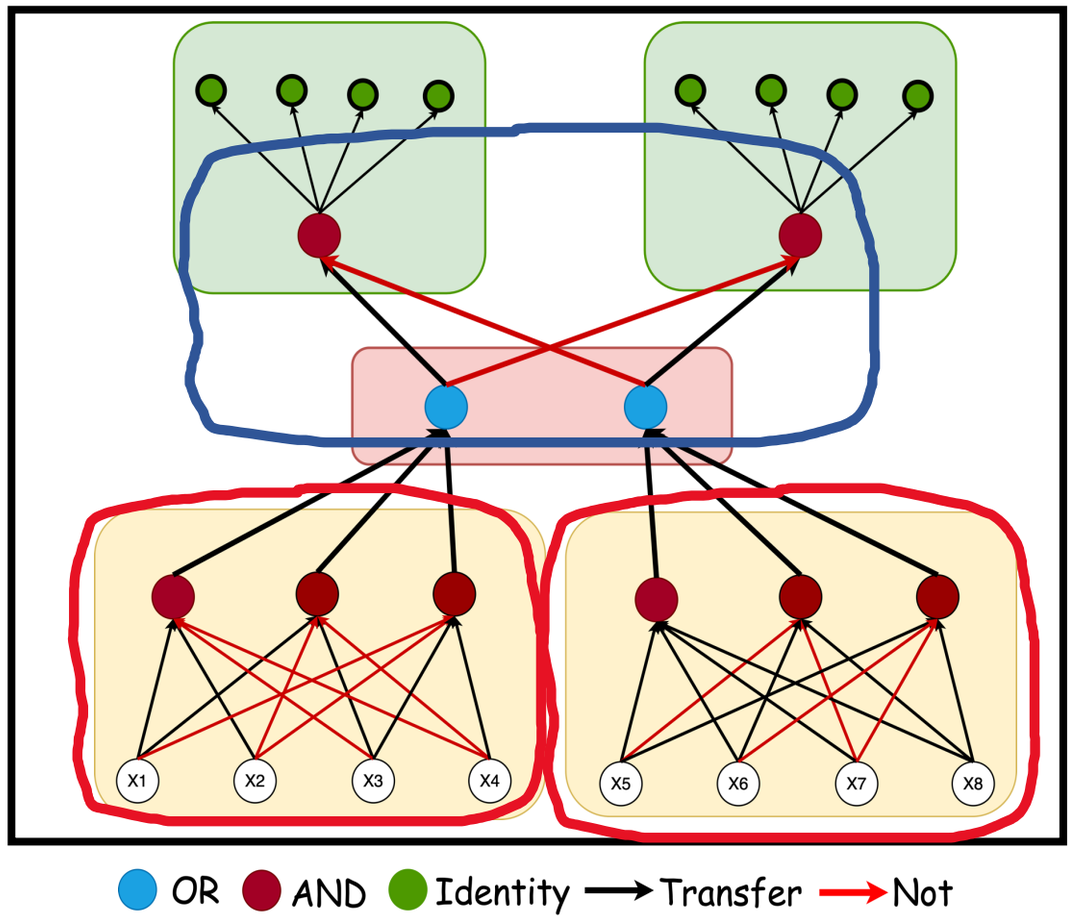
Reused

They are reused as inputs higher in the hierarchy

Input-separable

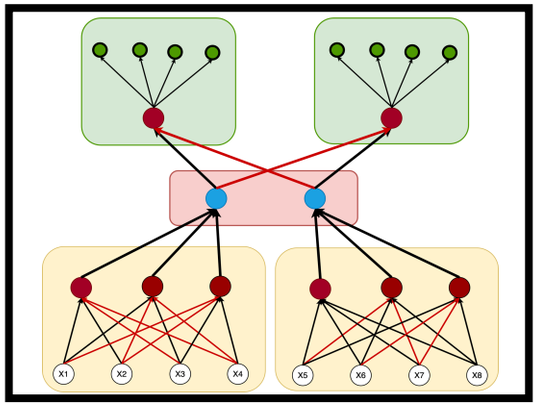
They have a distinct set of inputs at a given hierarchical level

Hierarchically Modular Function

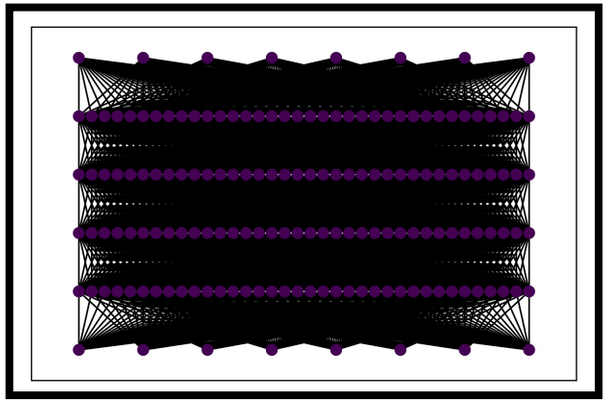


Neural Sculpting:
A method to uncover hierarchical modularity of the task

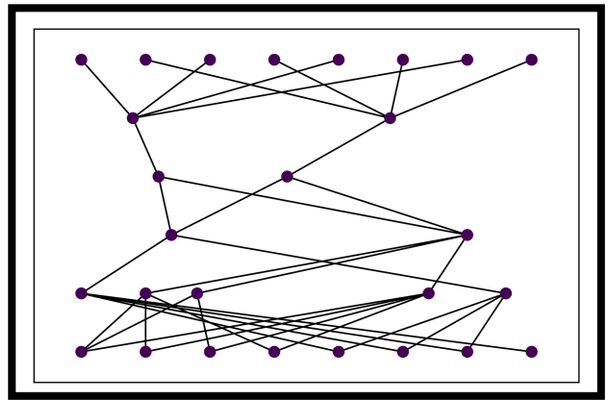
Suppose task is described by hierarchically modular function



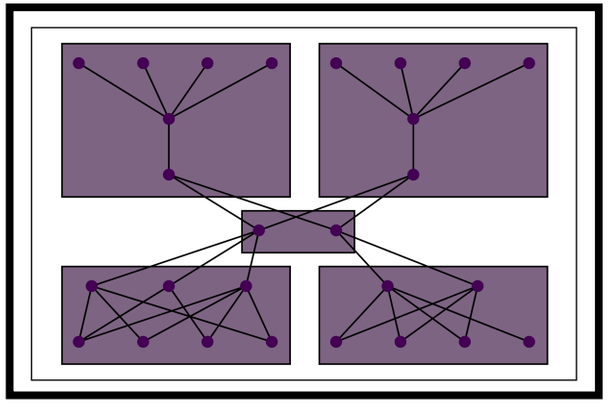
Start from a dense NN



Iteratively, sparsify network as much as possible while learning task



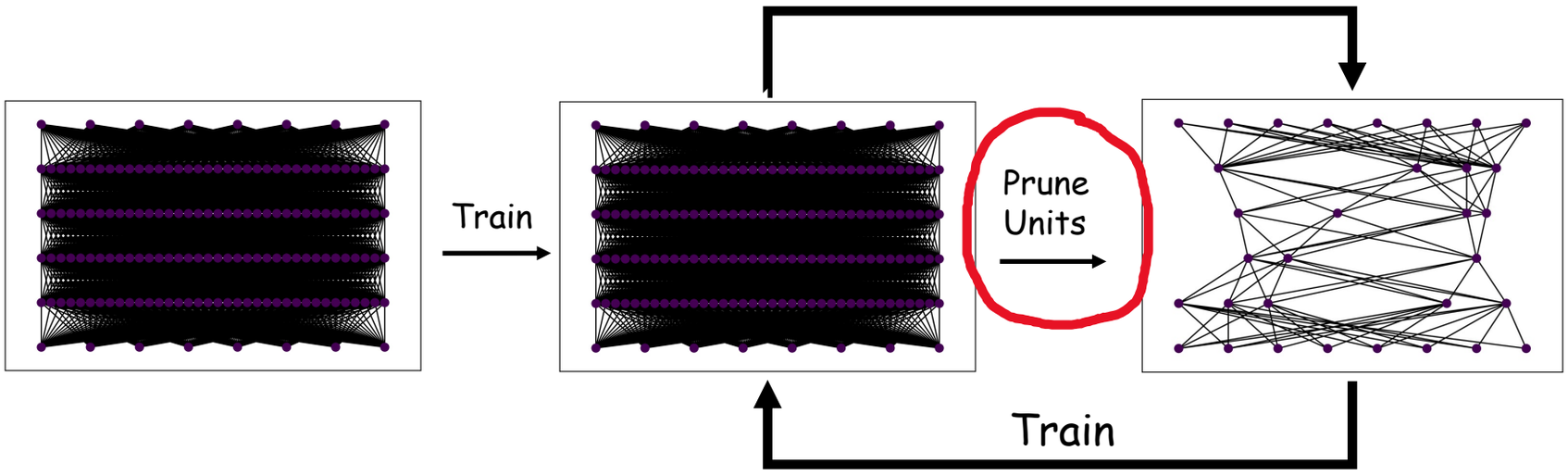
Cluster units into modules



Neural network sparsification : Hidden unit pruning → edge pruning

1) Train the NN
2) Score units
3) Remove lowest scoring units
4) Re-train
5) Repeat steps 2, 3 and 4 iteratively,
    - As long as NN accuracy remains >= threshold

Loss-sensitivity w.r.t unit activation

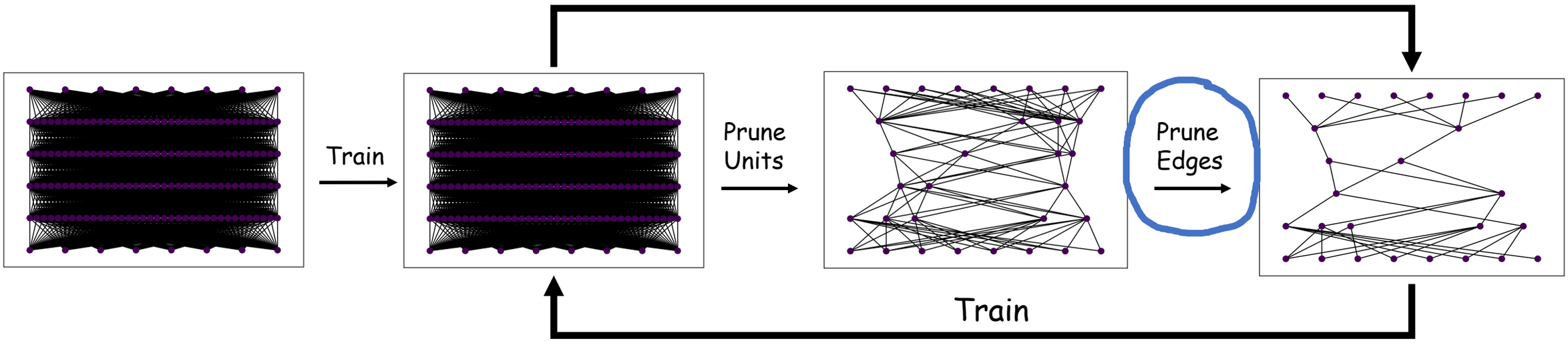


Neural network sparsification : Hidden unit pruning → edge pruning

1) Train the NN
2) Score ~~units~~ edges
3) Remove lowest scoring ~~units~~ edges
4) Re-train
5) Repeat steps 2, 3 and 4 iteratively,
   - As long as NN accuracy remains >= threshold

Loss-sensitivity w.r.t unit activation

Edge-weight magnitude



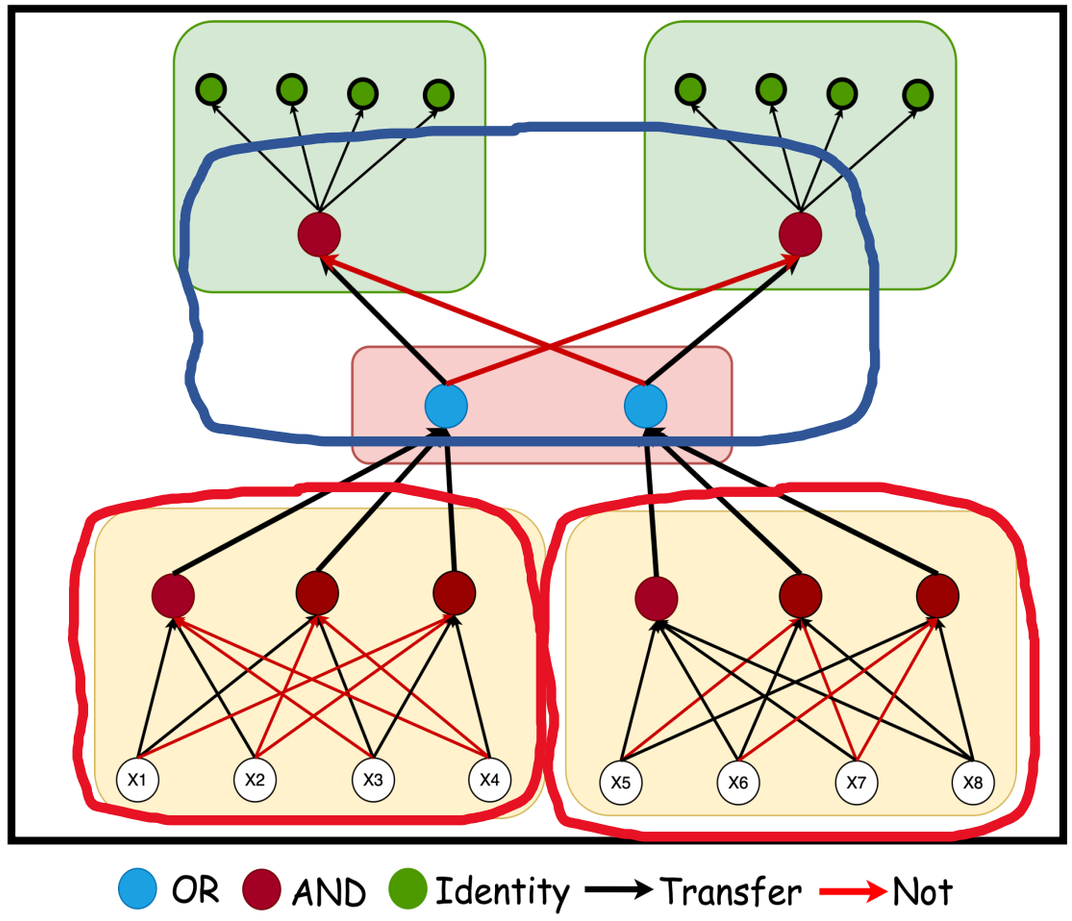
Two key structural properties of the sub-functions are uncovered through pruning

Unit Pruning → *Reused*

Edge Pruning → *Input separability*

Sparse Connectivity

Hierarchically Modular Function



Module detection through layer-wise unit clustering and cluster merging across layers

Unit feature vector:
units that can be reached within L hops

Agglomerative clustering

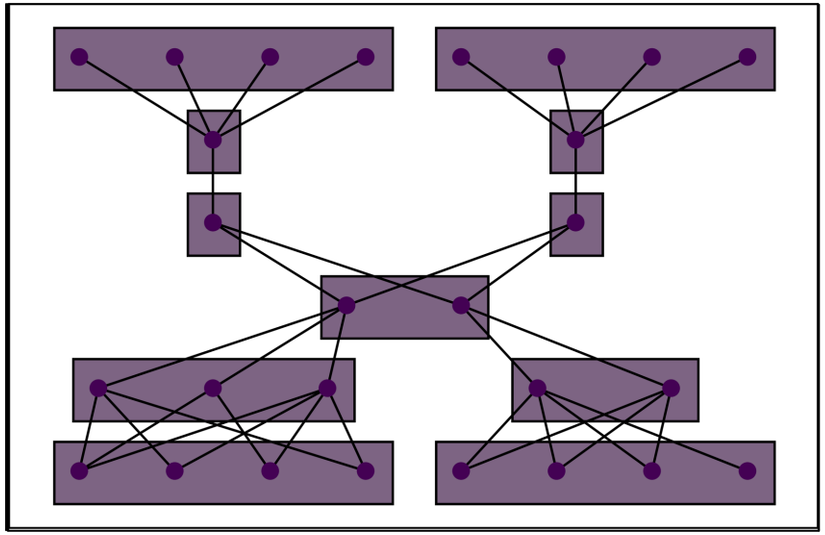
Number of clusters

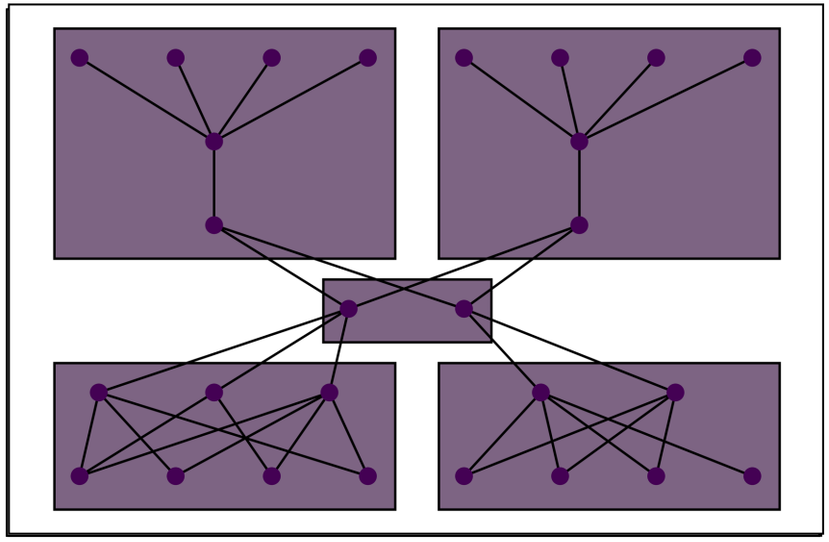
Modularity metric

Starting from input layer, merge clusters bottom-up

Merge clusters i,j, if both:

- Edges (i -> j) / Edges (i->) > 0.9
- Edges (i -> j) / Edges (->j) > 0.9



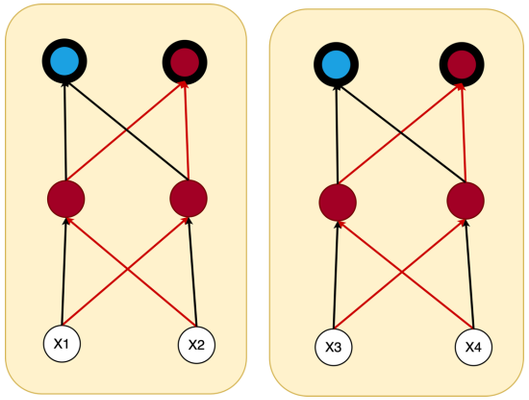


Merge clusters

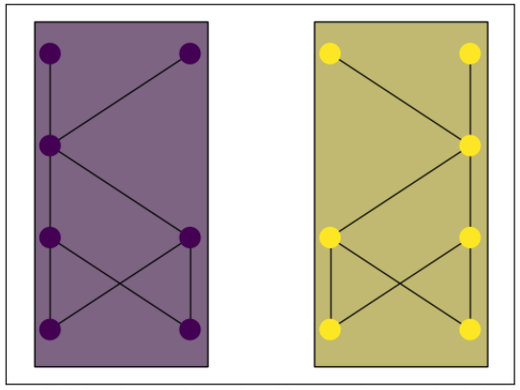
Results for some simple functions

Input-separable subfunctions

Success rate: 36/36



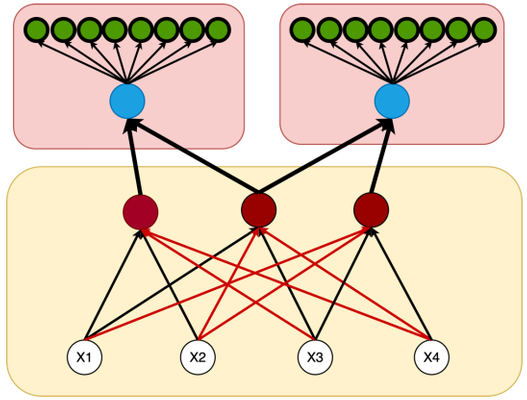
Modularity[2]: Arch[4, 24, 24, 4]: Density = 2.08%



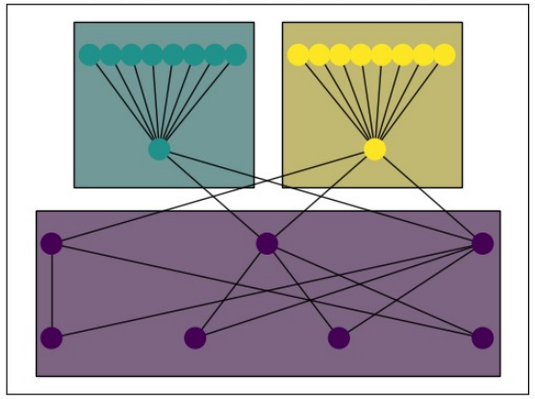
lr = 0.05, batch size =8, epochs =10, seed =0, arch =[4, 4, 2, 4]
P_u =65.0, P_e =1.5

Reused subfunction

Success rate: 35/36



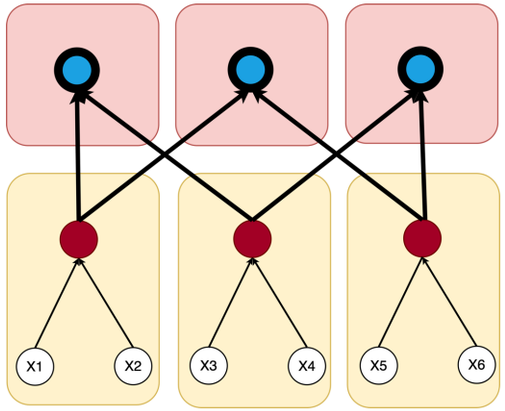
Modularity[1, 2]: Arch[4, 24, 24, 16]: Density = 2.75%



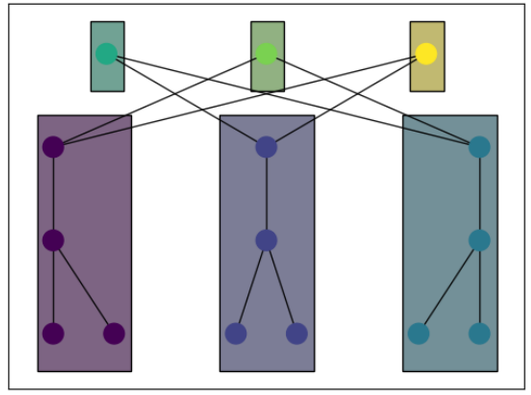
lr = 0.1, batch size =8, epochs =40, seed =0, arch =[4, 3, 2, 16]
P_u =70.0, P_e =2.5

Input-separable and reused sub-functions

Success rate: 36/36



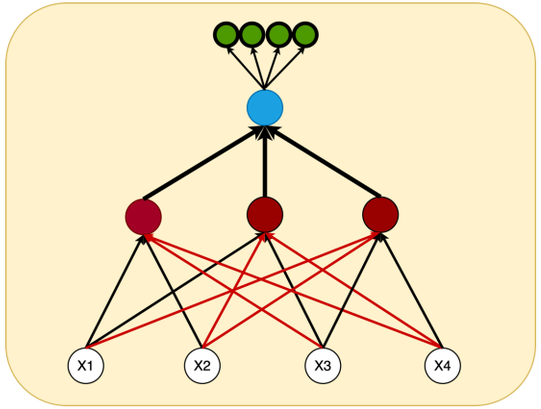
Modularity[3, 3]: Arch[6, 24, 24, 3]: Density = 1.89%



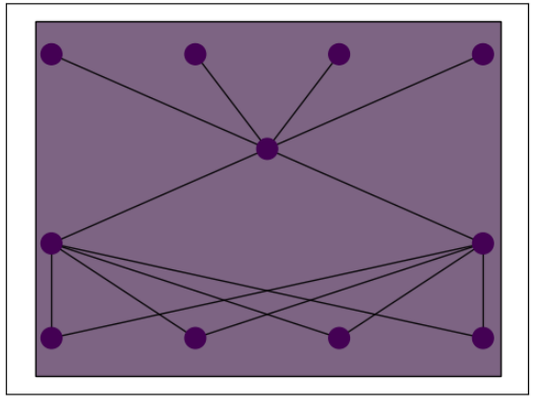
lr = 0.05, batch size =32, epochs =10, seed =0, arch =[6, 3, 3, 3]
P_u =60.0, P_e =0.5

Single subfunction

Success rate: 36/36

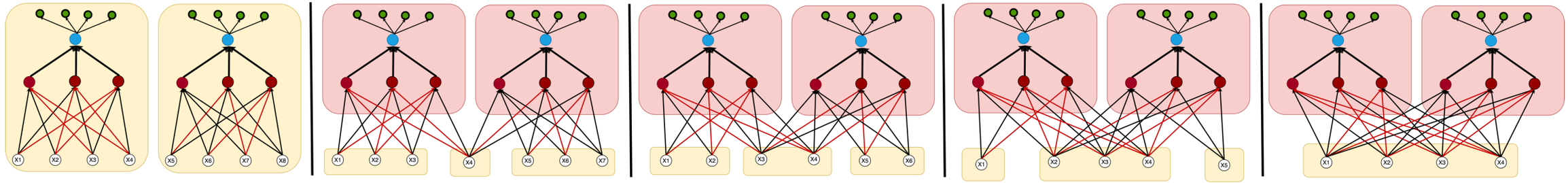


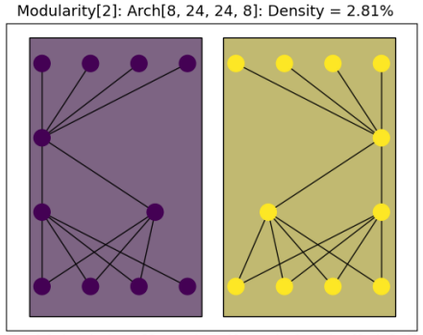
Modularity[1]: Arch[4, 24, 24, 4]: Density = 1.82%



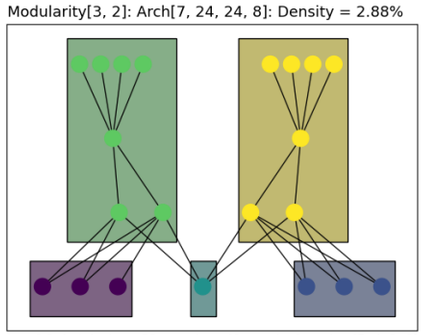
lr = 0.05, batch size =8, epochs =30, seed =0, arch =[4, 2, 1, 4]
P_u =15.0, P_e =2.5

Sub-functions with higher separability are uncovered more accurately

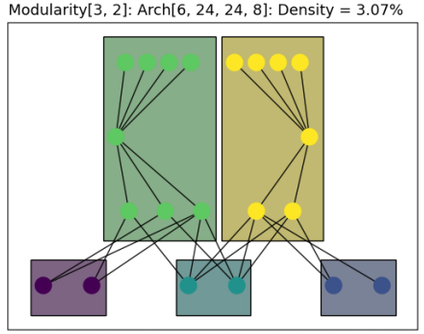



Modularity[2]: Arch[8, 24, 24, 8]: Density = 2.81%

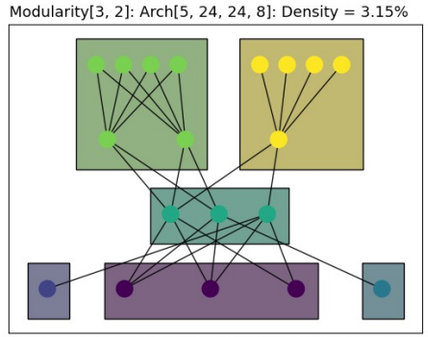
lr = 0.1, batch size =32, epochs =40, seed =0, arch =[8, 4, 2, 8]
P_u =70.0, P_e =2.5


Modularity[3, 2]: Arch[7, 24, 24, 8]: Density = 2.88%

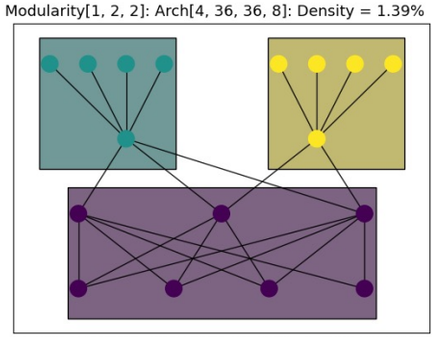
lr = 0.1, batch size =32, epochs =40, seed =0, arch =[7, 4, 2, 8]
P_u =55.0, P_e =1.0


Modularity[3, 2]: Arch[6, 24, 24, 8]: Density = 3.07%

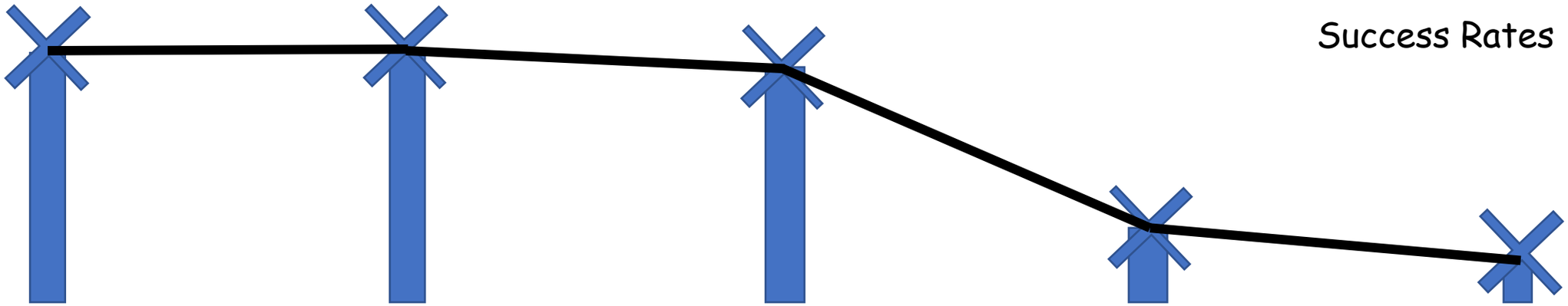
lr = 0.1, batch size =32, epochs =40, seed =0, arch =[6, 5, 2, 8]
P_u =65.0, P_e =2.0


Modularity[3, 2]: Arch[5, 24, 24, 8]: Density = 3.15%

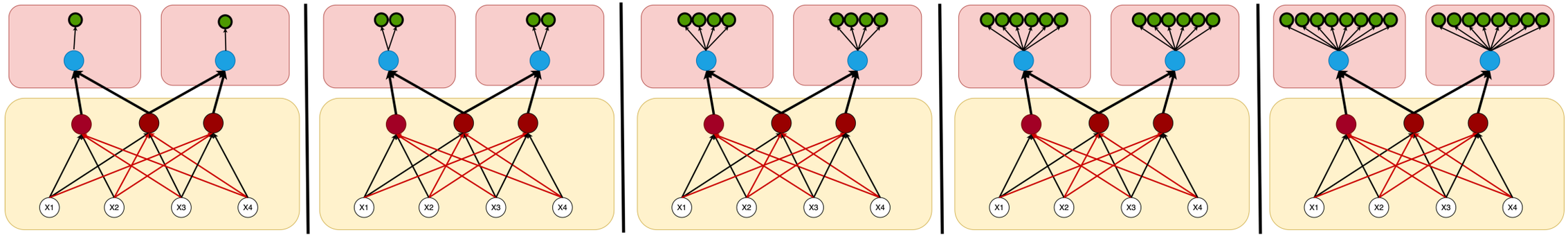
lr = 0.05, batch size =16, epochs =20, seed =0, arch =[5, 3, 3, 8]
P_u =25.0, P_e =2.0

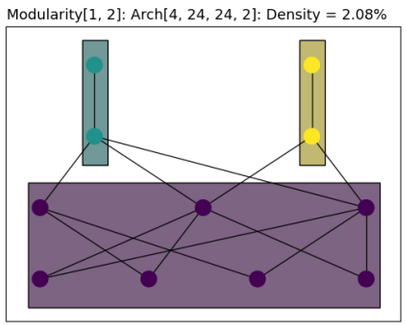

Modularity[1, 2, 2]: Arch[4, 36, 36, 8]: Density = 1.39%

lr = 0.1, batch size =16, epochs =20, seed =1, arch =[4, 3, 2, 8]
P_u =60.0, P_e =2.5

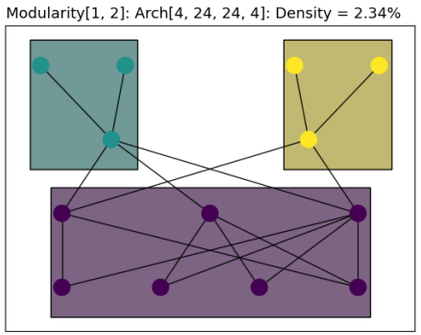


Sub-functions with higher reuse are detected more accurately

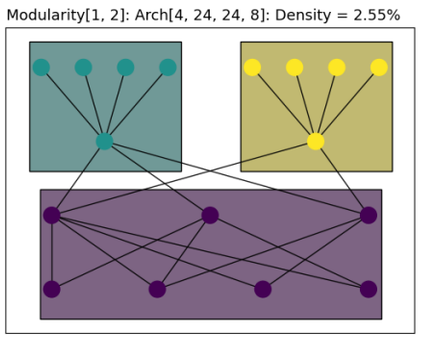




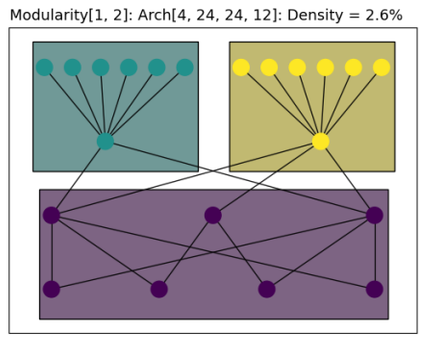
lr = 0.1, batch size =8, epochs =30, seed =5, arch =[4, 3, 2, 2]
P_u =45.0, P_e =1.0



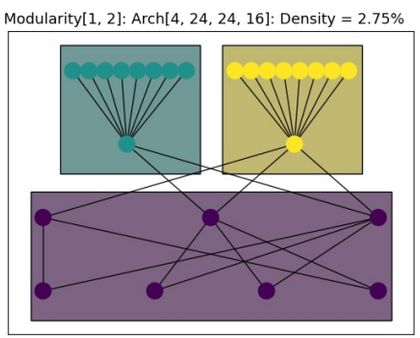
lr = 0.1, batch size =8, epochs =40, seed =0, arch =[4, 3, 2, 4]
P_u =55.0, P_e =2.5



lr = 0.1, batch size =16, epochs =30, seed =0, arch =[4, 3, 2, 8]
P_u =40.0, P_e =2.5

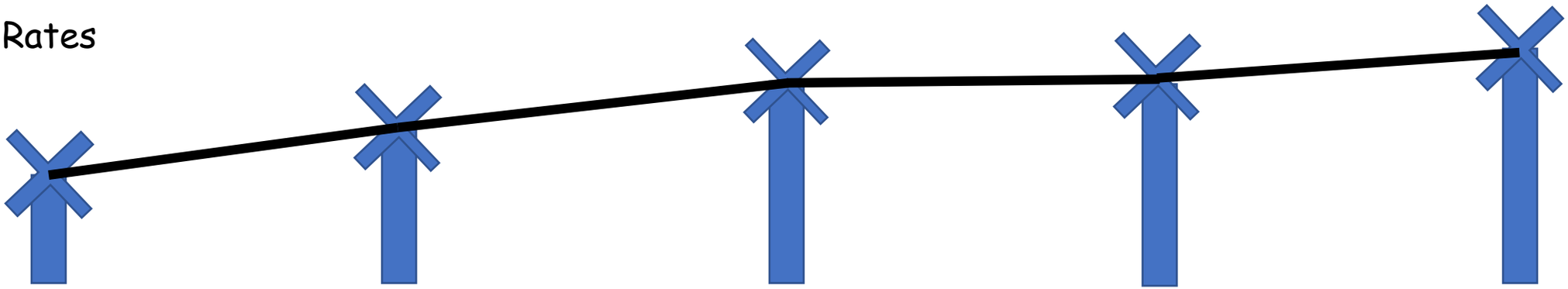


lr = 0.1, batch size =8, epochs =40, seed =0, arch =[4, 3, 2, 12]
P_u =65.0, P_e =2.0

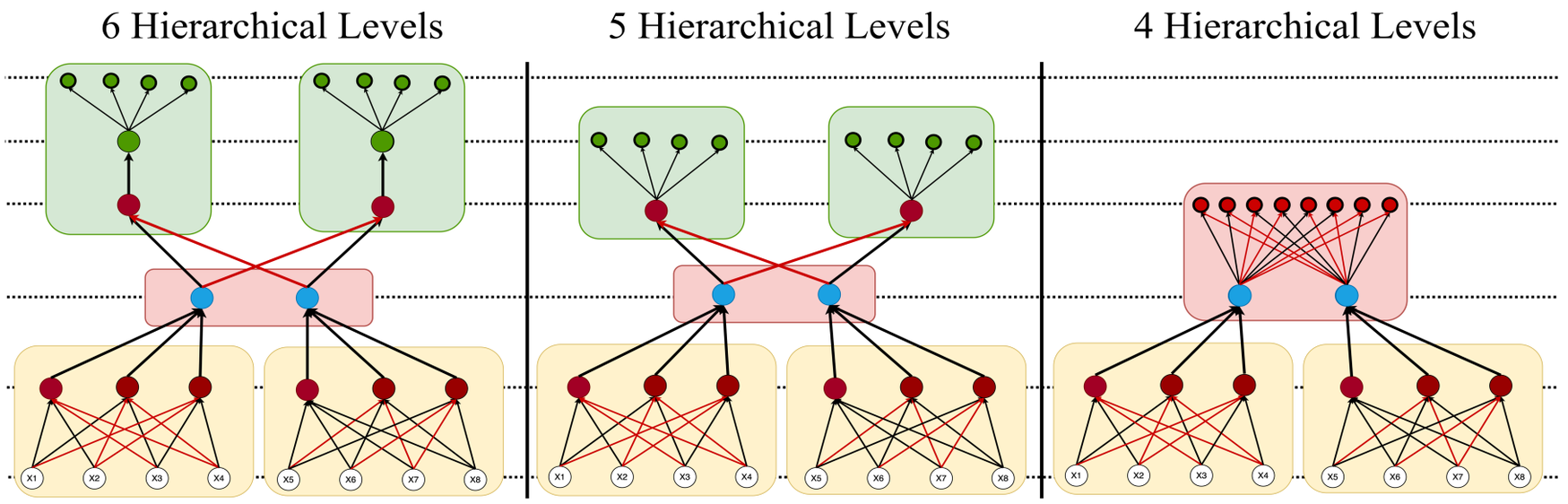


lr = 0.1, batch size =8, epochs =40, seed =0, arch =[4, 3, 2, 16]
P_u =70.0, P_e =2.5

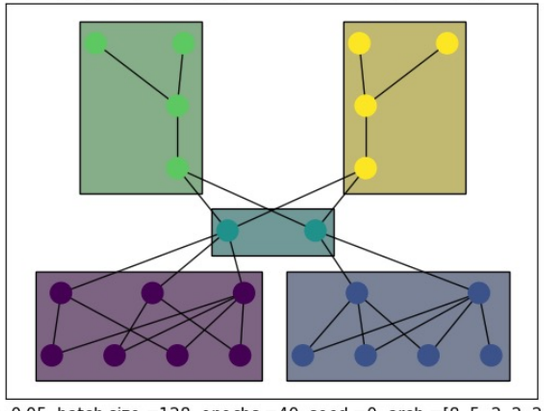
Success Rates



Hierarchy detection depends on NN depth

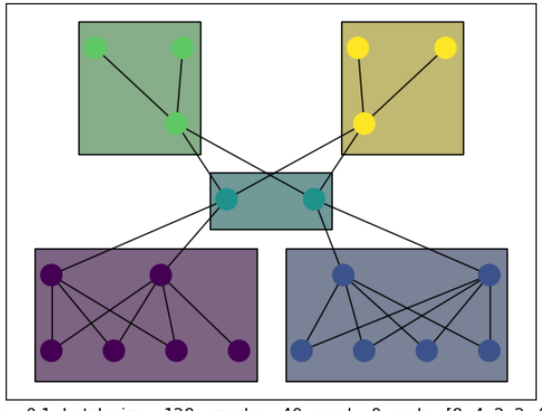


Modularity[2, 1, 2]: Arch[8, 48, 48, 48, 48, 4]: Density = 0.4%



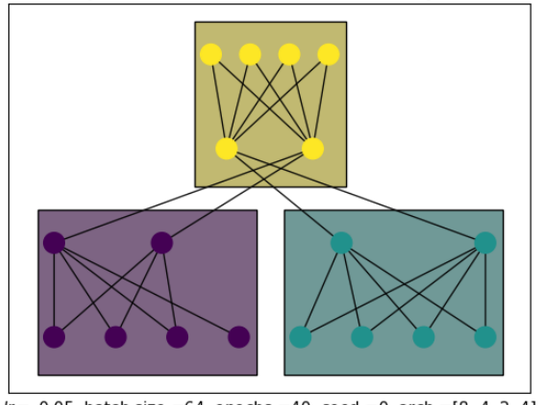
lr = 0.05, batch size =128, epochs =40, seed =0, arch =[8, 5, 2, 2, 2, 4]
P_u =10.0, P_e =1.0

Modularity[2, 1, 2]: Arch[8, 48, 48, 48, 4]: Density = 0.52%



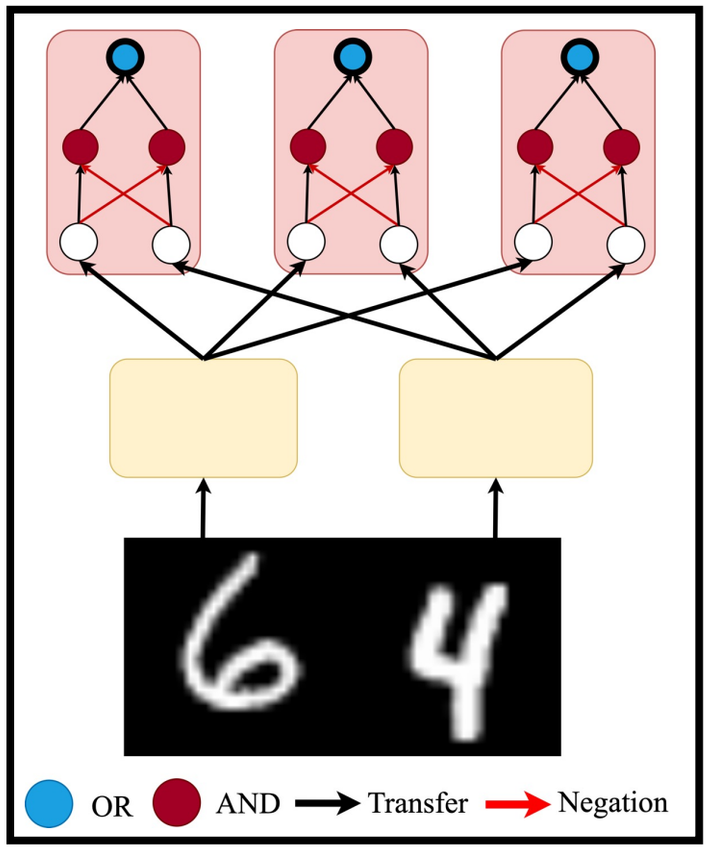
lr = 0.1, batch size =128, epochs =40, seed =0, arch =[8, 4, 2, 2, 4]
P_u =35.0, P_e =1.0

Modularity[2, 1, 2]: Arch[8, 48, 48, 4]: Density = 0.94%

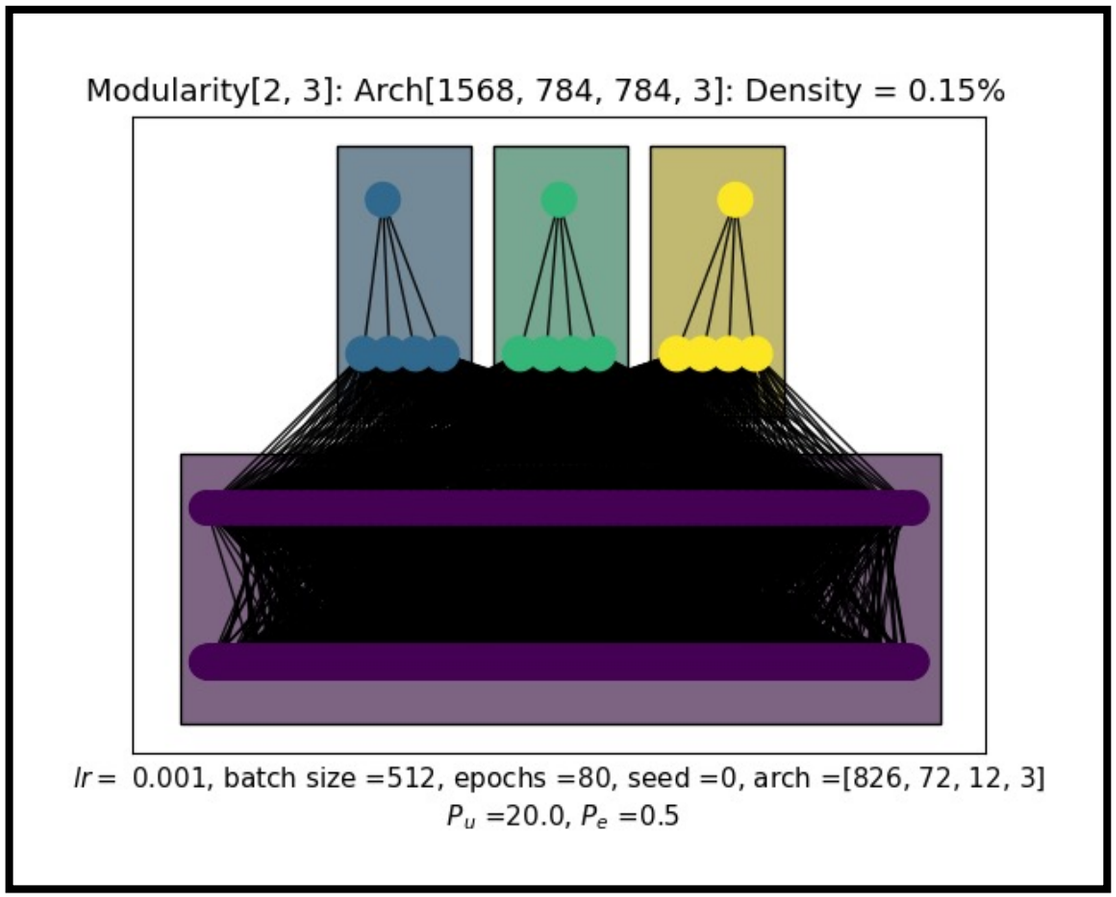


lr = 0.05, batch size =64, epochs =40, seed =0, arch =[8, 4, 2, 4]
P_u =70.0, P_e =2.0

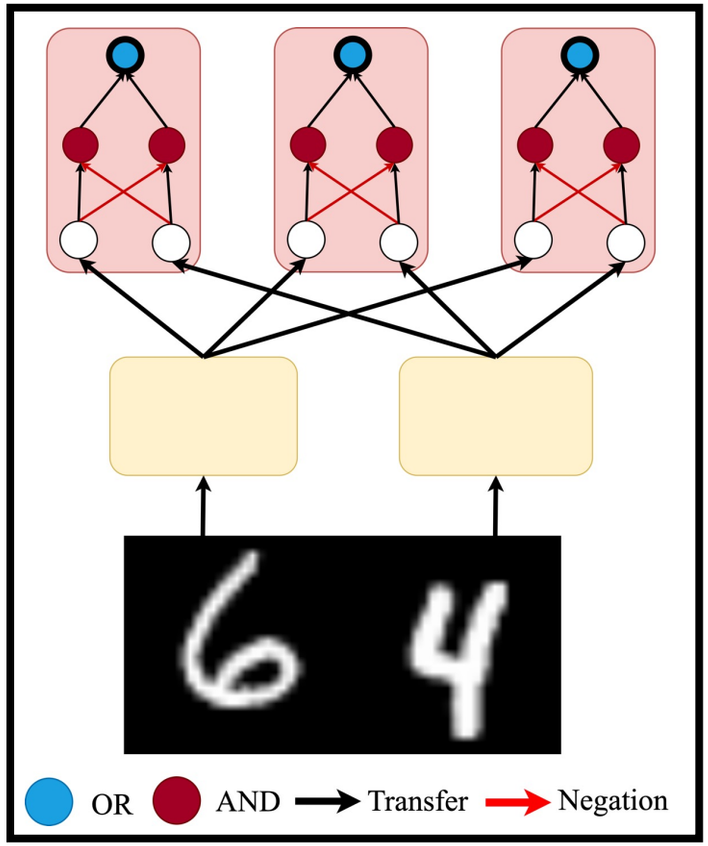
Hierarchical and modular task with MNIST digits



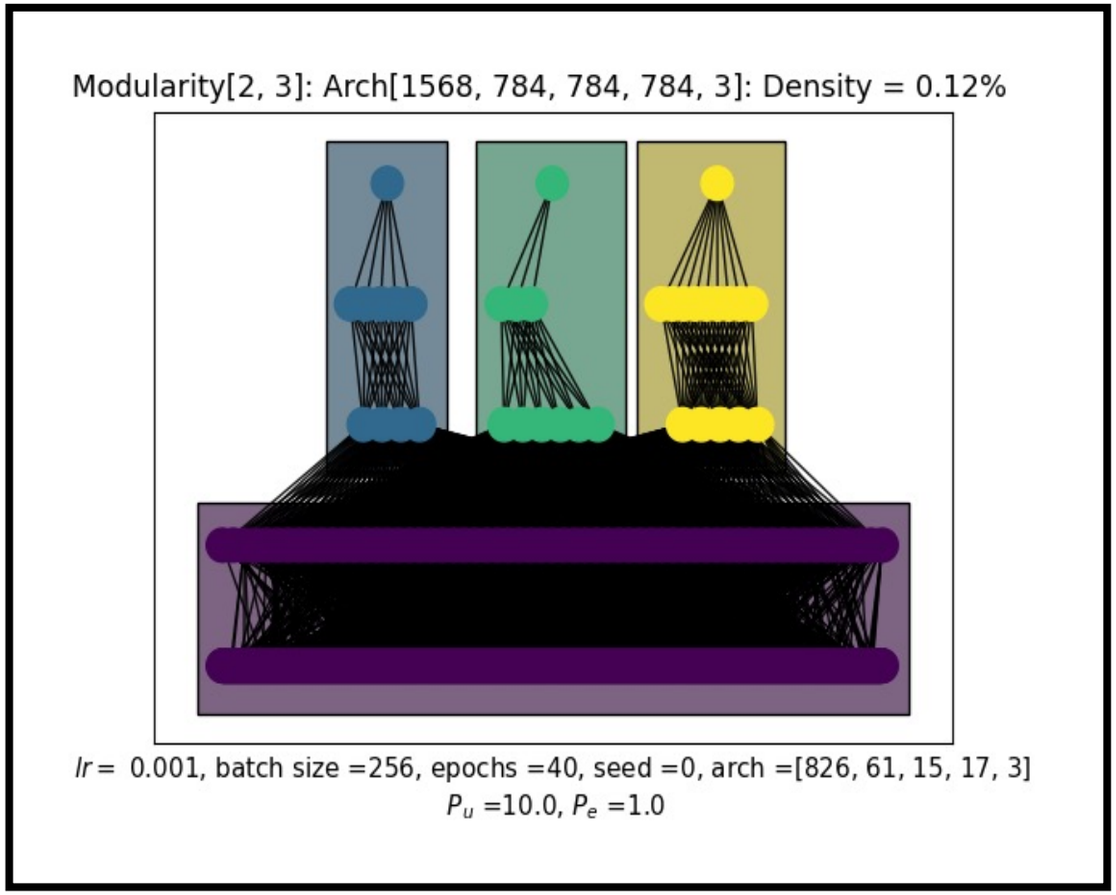
NN with 2 hidden layers



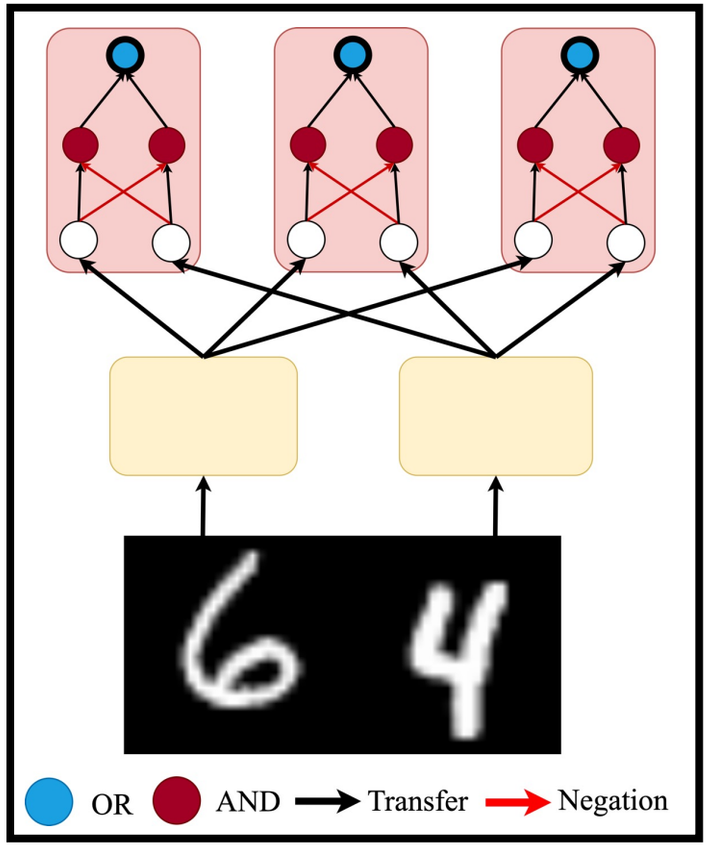
Hierarchical and modular task with MNIST digits



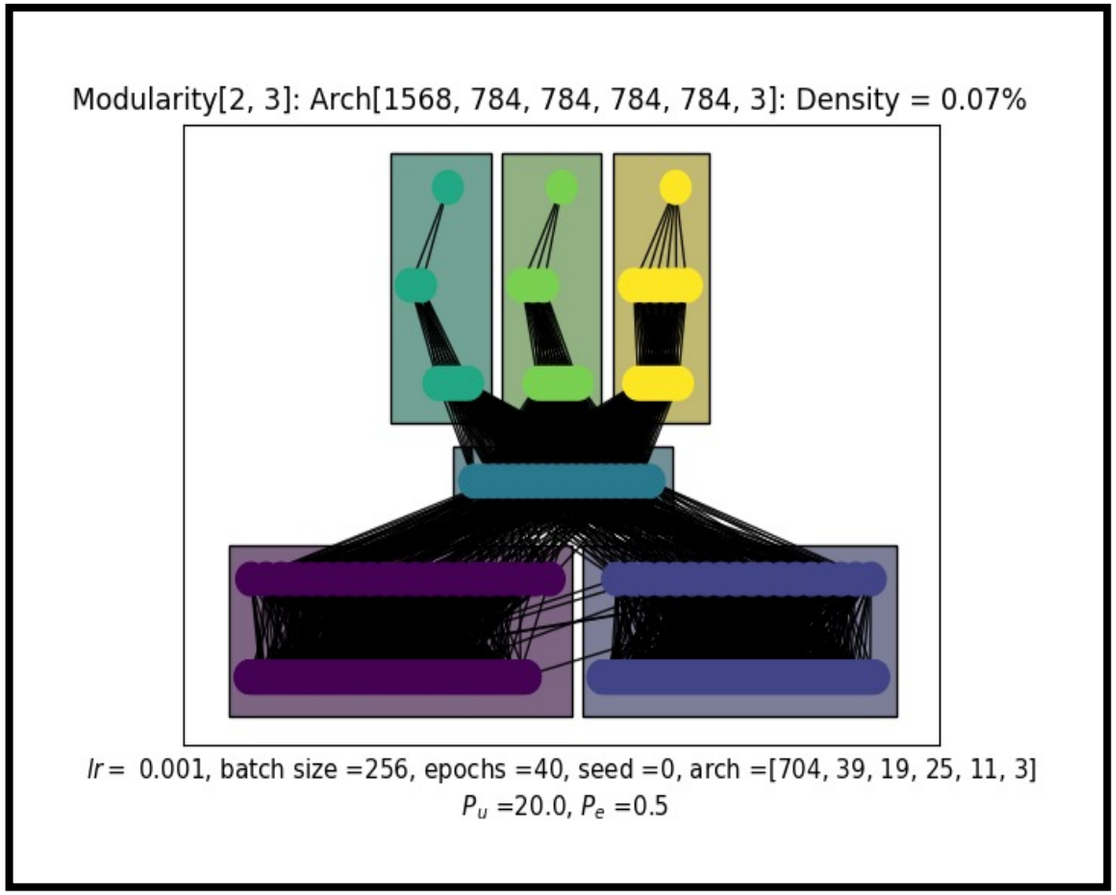
NN with 3 hidden layers



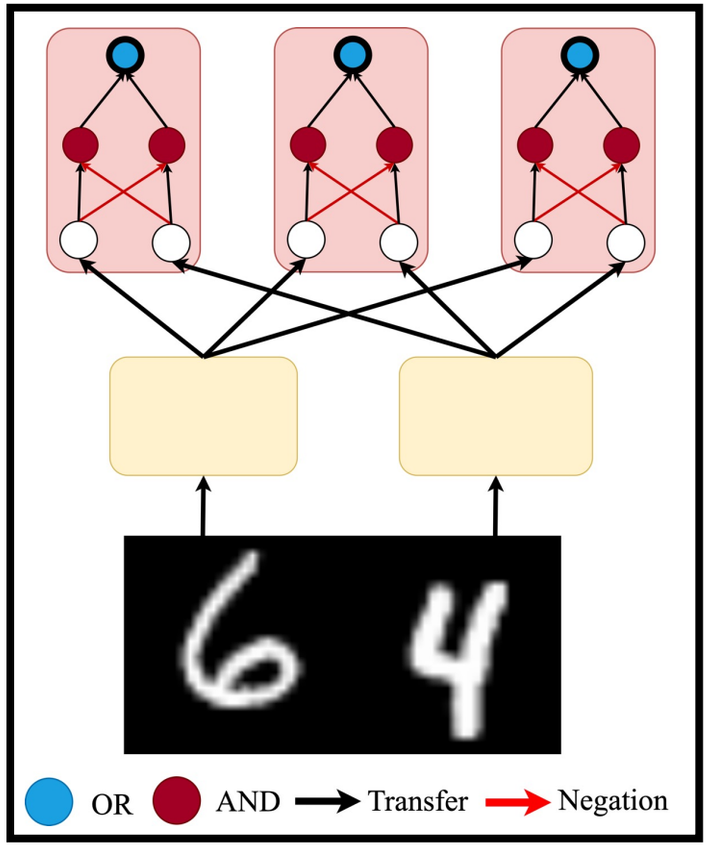
Hierarchical and modular task with MNIST digits



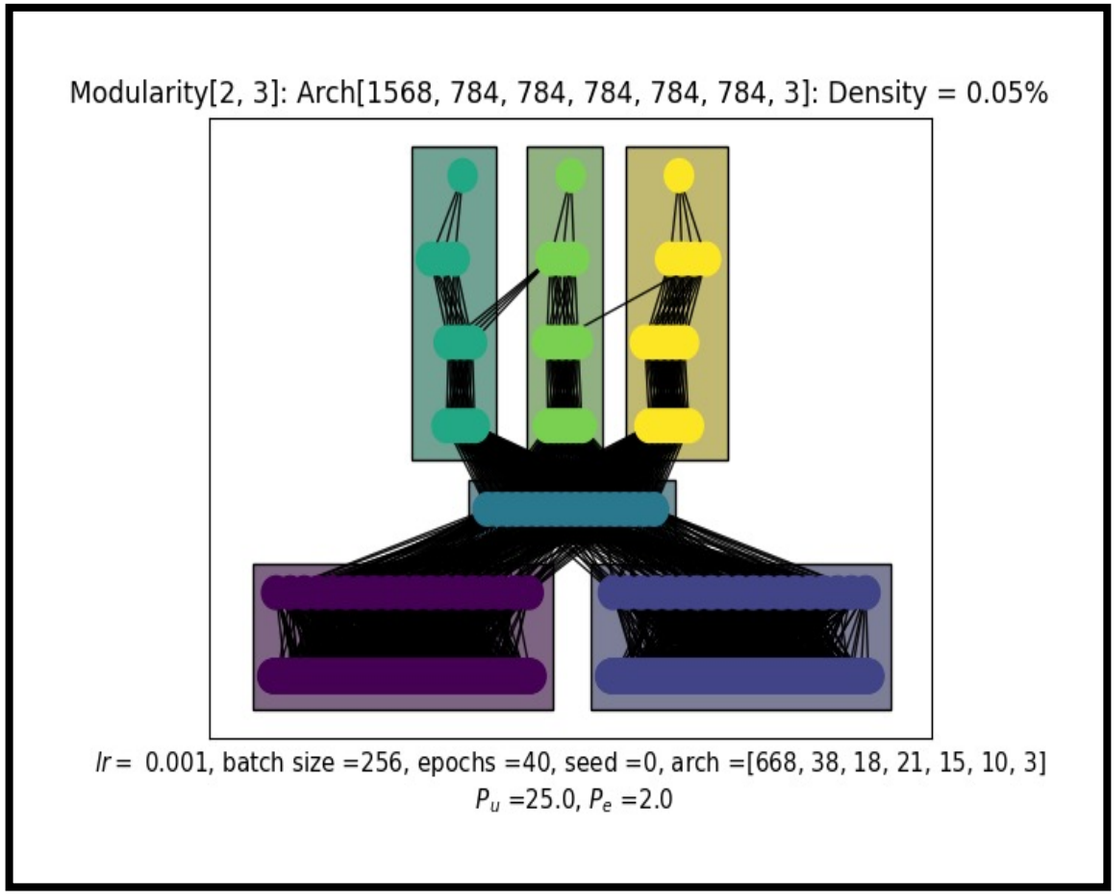
NN with 4 hidden layers



Hierarchical and modular task with MNIST digits



NN with 5 hidden layers



Future work

- Analyze the benefits of hierarchically modular NNs
- Improve the computational efficiency of the method
- Apply Neural Sculpting to larger models and datasets