# INVESTIGATING SPARSITY IN RECURRENT NEURAL NETWORKS

DARJI, HARSHIL JAGADISHBHAI

SUPERVISORS:

1. PROF. DR. MICHAEL GRANITZER
2. PROF. DR. HARALD KOSCH

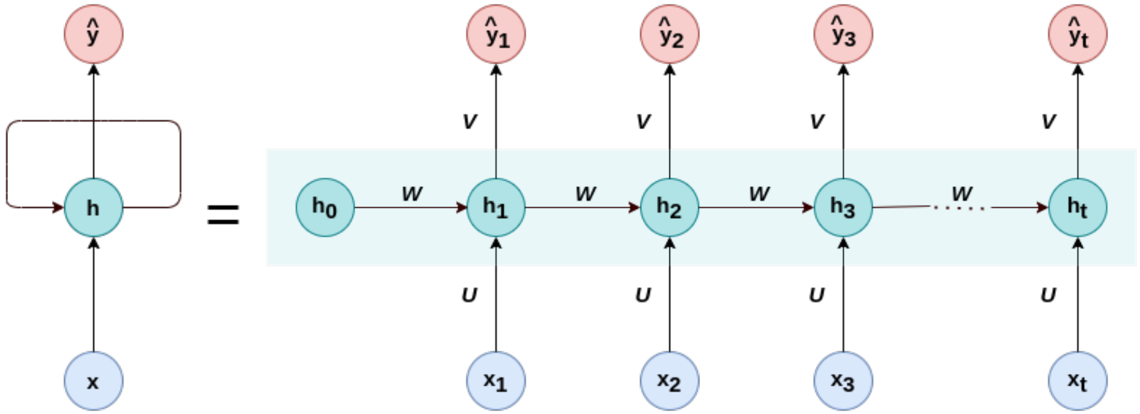
ADVISOR: JULIAN STIER

# AGENDA


1. Introduction
2. Motivation
3. Related work
4. Research goals
5. Dataset
6. Experiments
7. Results
8. Conclusion

# 1. INTRODUCTION

- Recurrent Neural Networks are standard models that have shown exceptional performance in many NLP tasks that make use of **sequential information**.

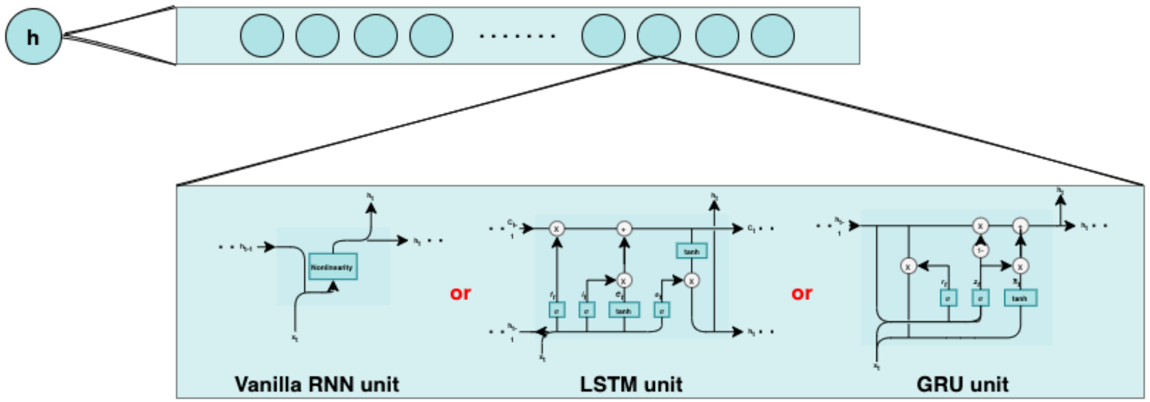



Unrolled RNN

- A standard RNN is parameterized with three weight matrices (*U, V, W*) and two bias vectors ($b_h$, $b_o$).
- Deep extensions of such basic RNNs can be constructed by **stacking** multiple recurrent hidden states on top of each other.

# 1. INTRODUCTION (cont.)

- Different variations of RNNs are created based on various internal architectures of hidden states as shown below:




Internal architecture of a hidden state

- A vanilla RNN differs based on the nonlinearity function used.
- Tanh and ReLU are two of the most used nonlinearity functions.

# 2. MOTIVATION

- Deep Neural Networks are more likely to have increased performance but also **increased fast memory** requirements.
- One way to decrease these memory requirements is to introduce sparsity into a network's connection[1].
- Sparsity in traditional neural networks is being studied widely from the past few years but is not explored much in case of recurrent neural networks.
- Sparse structures have shown a training potential in traditional neural networks[2] which if applied to RNNs, can also make training them **less difficult** while retaining their performance.

[1] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.1109/CVPR.2016.90

[2] Simon Alford et al. Pruned and Structurally Sparse Neural Networks. Tech. rep. MIT, 2018. arXiv: 1810.00299v1

# 3. RELATED WORK

- *Exploring Sparsity in Recurrent Neural Networks*[1] (November 2016) :
  - Pruned the linear layers that feed into the recurrent layers, the forward and backward recurrent layers and fully connected layer before the CTC layer.
- *Pruned and Structurally Sparse Neural Networks*[2] (September 2018) :
  - Tested pruning based sparse topologies by pruning a pre-trained dense network and by using RadiX-Nets.
- *Exploring Randomly Wired Neural Networks for Image Recognition*[3] (April 2019) :
  - Explored randomly wired neural networks driven by random graph models from graph theory.
- *Structural Analysis of Sparse Neural Networks*[4] (September 2019) :
  - Predicted the performance of convolutional neural networks using its structural properties.

[1] Sharan Narang et al. "Exploring Sparsity in Recurrent Neural Networks". In: ICLR 2017 Conference. arXiv: 1704.05119v2
[2] Simon Alford et al. Pruned and Structurally Sparse Neural Networks. Tech. rep. MIT, 2018. arXiv: 1810.00299v1
[3] Saining Xie et al. Exploring Randomly Wired Neural Networks for Image Recognition. Tech. rep. Facebook AI Research (FAIR), 2019. arXiv: 1904.01569v2
[4] Julian Stier and Michael Granitzer. "Structural Analysis of Sparse Neural Networks". In: 23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems. DOI: 10.1016/j.procs.2019.09.165

# 4. RESEARCH GOALS

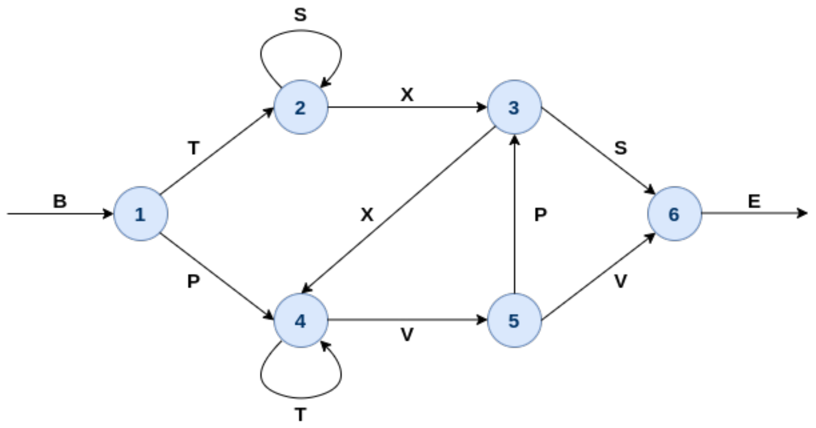
- The primary goal is to investigate, both pruned and structural sparsity in RNNs by answering the following research questions:
    1. What is the **effect of weights pruning** on a recurrent network's accuracy?
    2. What **percentage of weights pruning** is permissible without triggering a significant reduction in the performance?
    3. After pruning a certain percent of weights, if we see a significant reduction in the accuracy, how many re-training epochs can **regain accuracy**?
    4. How does a randomly structured recurrent network's **performance correlate with the graph properties** of its internal structure?
    5. Is it possible to **predict** a randomly structured recurrent network's performance using the graph properties of its base random graph?
- I will answer these questions by conducting several experiments on different datasets.

# 5. DATASETS

- Consists of a total of 25000 grammar sequences.
- 12500 are true Reber sequences, and 12500 are false Reber sequences.
- Train-Test split:

| | Training set | Test set |
|---|---|---|
| True (Valid) | 9339 | 3161 |
| False (Invalid) | 9411 | 3089 |

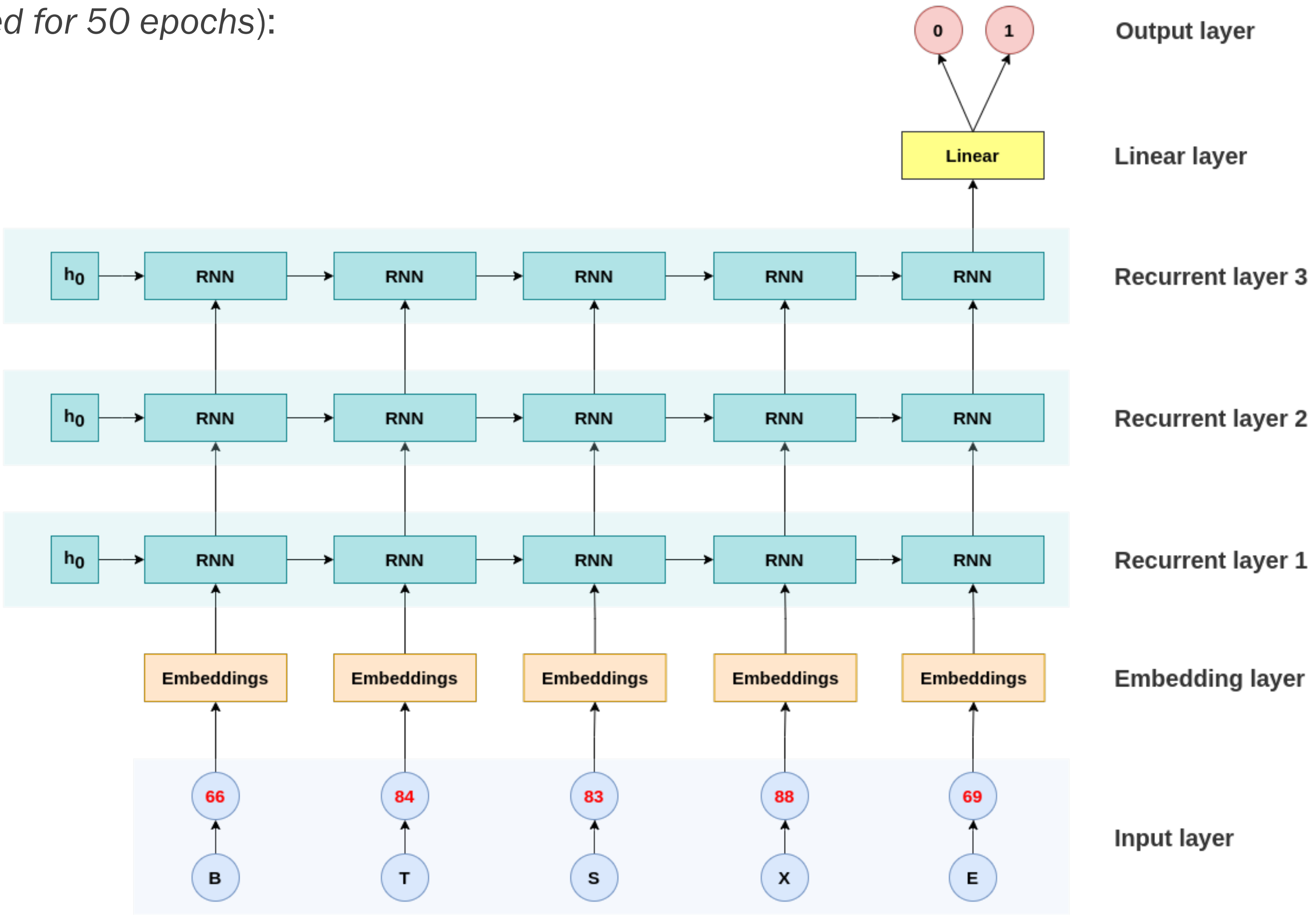
- Minimum string length: 11 (*216 strings*)
- Maximum string length: 53 (*1 string*)



Reber grammar flowchart

# 6. EXPERIMENTS

- Base model (*trained for 50 epochs*):



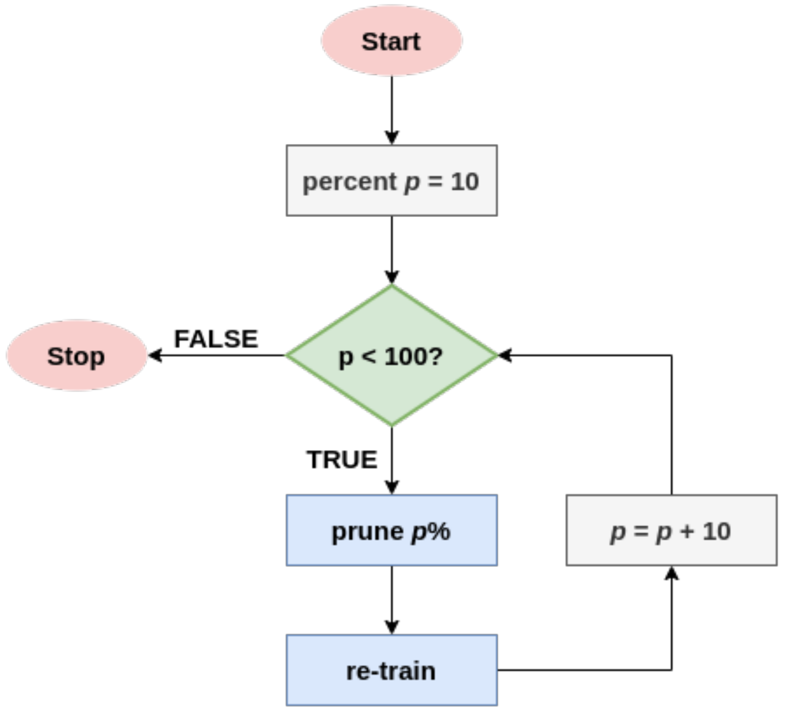

Base model

# 6. EXPERIMENTS (cont.)

- Pruning:
  1. Pruning both, input-to-hidden and hidden-to-hidden weights simultaneously,
  2. Pruning only input-to-hidden weights,
  3. Pruning only hidden-to-hidden weights
- Randomly structured RNN
- Performance prediction of Randomly Structured RNNs

# 6.1. PRUNING

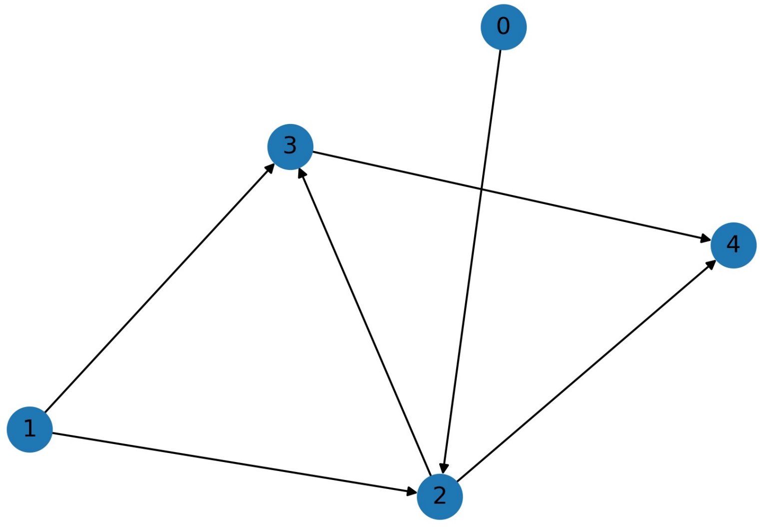
- Pruning is performed on trained base model.
- After pruning *p* percentage, pruned performance is stored.
- Afterwards, this pruned model is retrained to identify the number of epochs required to regain the accuracy.
- This pruning experiement is performed for,
  - RNN_Tanh,
  - RNN_ReLU,
  - LSTM,
  - GRU
- Results of this experiment is presented in section 7.1.



Pruning flow chart

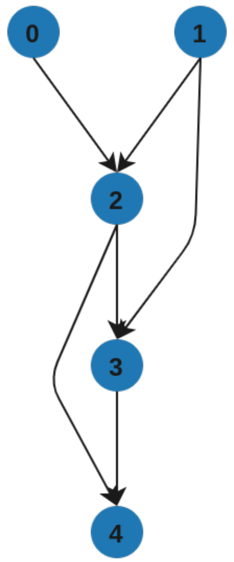
# 6.2. RANDOMLY STRUCTURED RNN

- Start with a Random Graph and make it directed (*if not!*).




Directed random graph

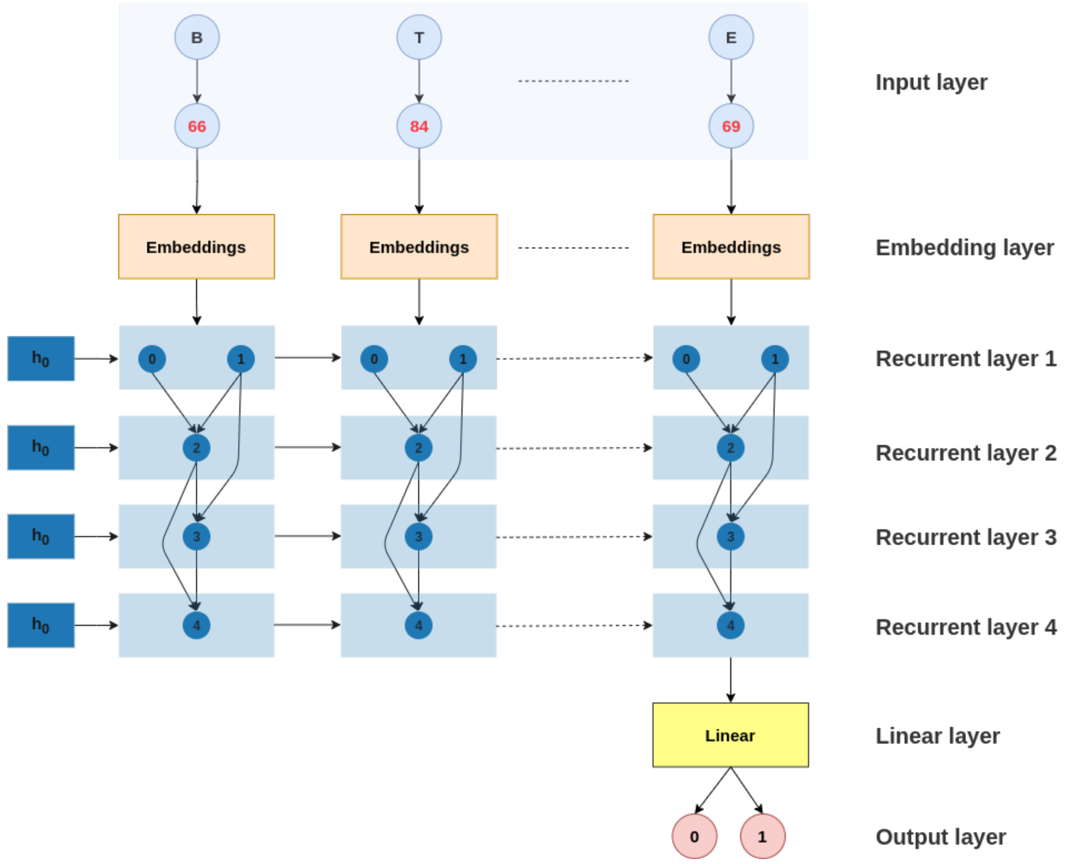
- Using the layer indexing algorithm, compute layer indexing of each node in the random graph.




Layered Random Graph

# 6.2. RANDOMLY STRUCTURED RNN (cont.)

- Randomly Structured RNN model is then generated by introducing recurrent connections between consecutive layered RG:



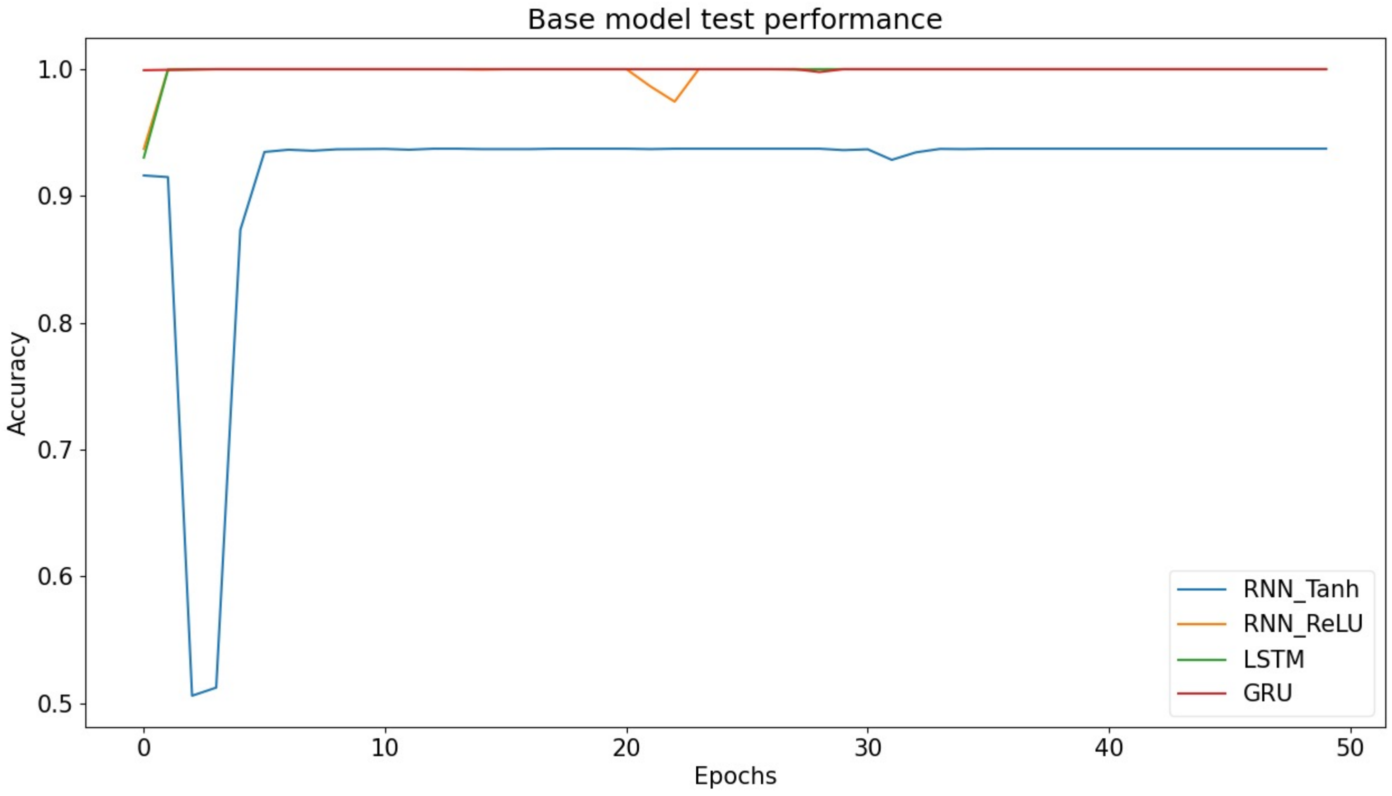

Randomly Structured RNN model

# 6.3. PERFORMANCE PREDICTION RS RNN

- During training and evaluation of Randomly Structured RNN, graph properties of the base Random Graphs are stored along with its corresponding performance.
- Three regressor algorithms, namely Bayesian Ridge, Random Forest, and AdaBoost, are then trained on this data, with graph properties as features and performance as the target.
- An $R^2$-value is then reported to understand how these data fit each regressor model.

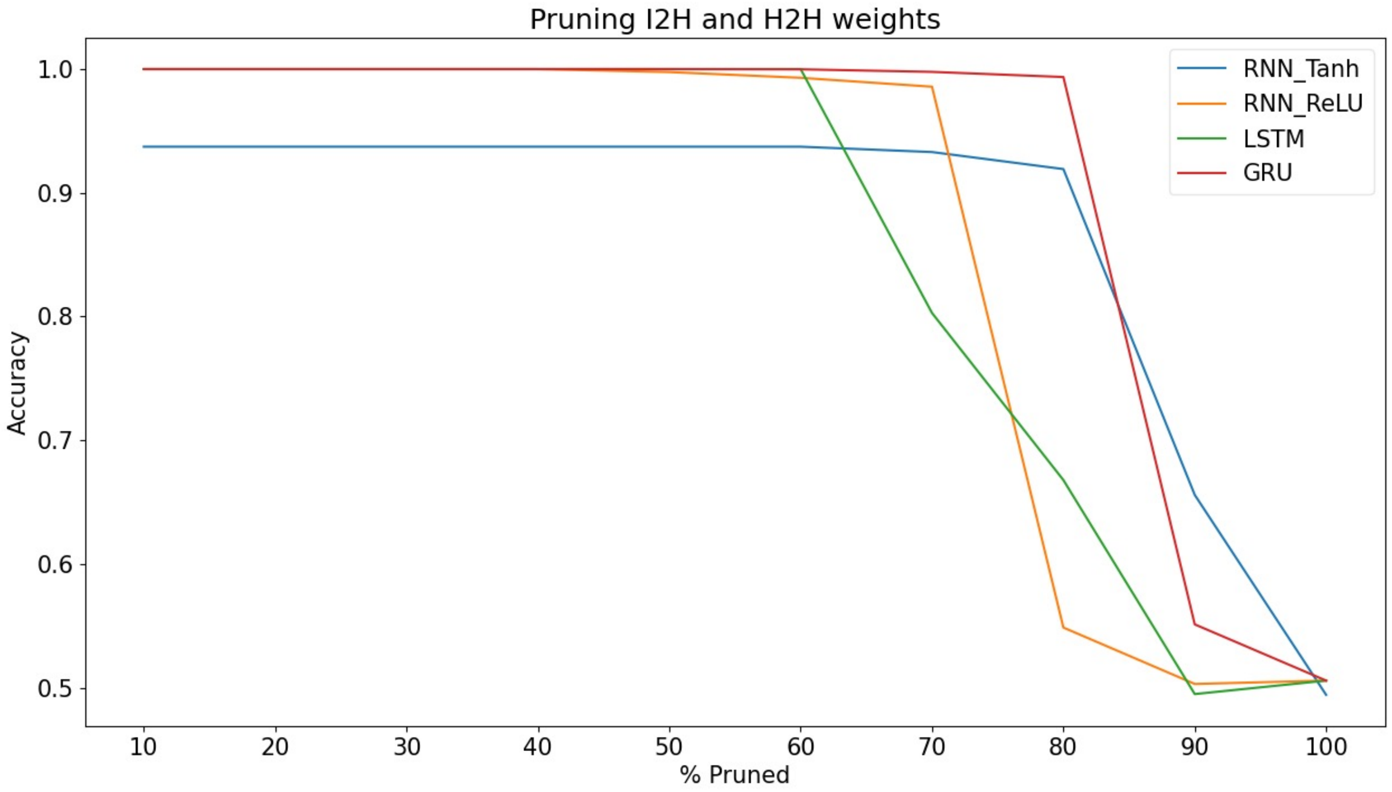
# 7. RESULTS

- Base model performance:



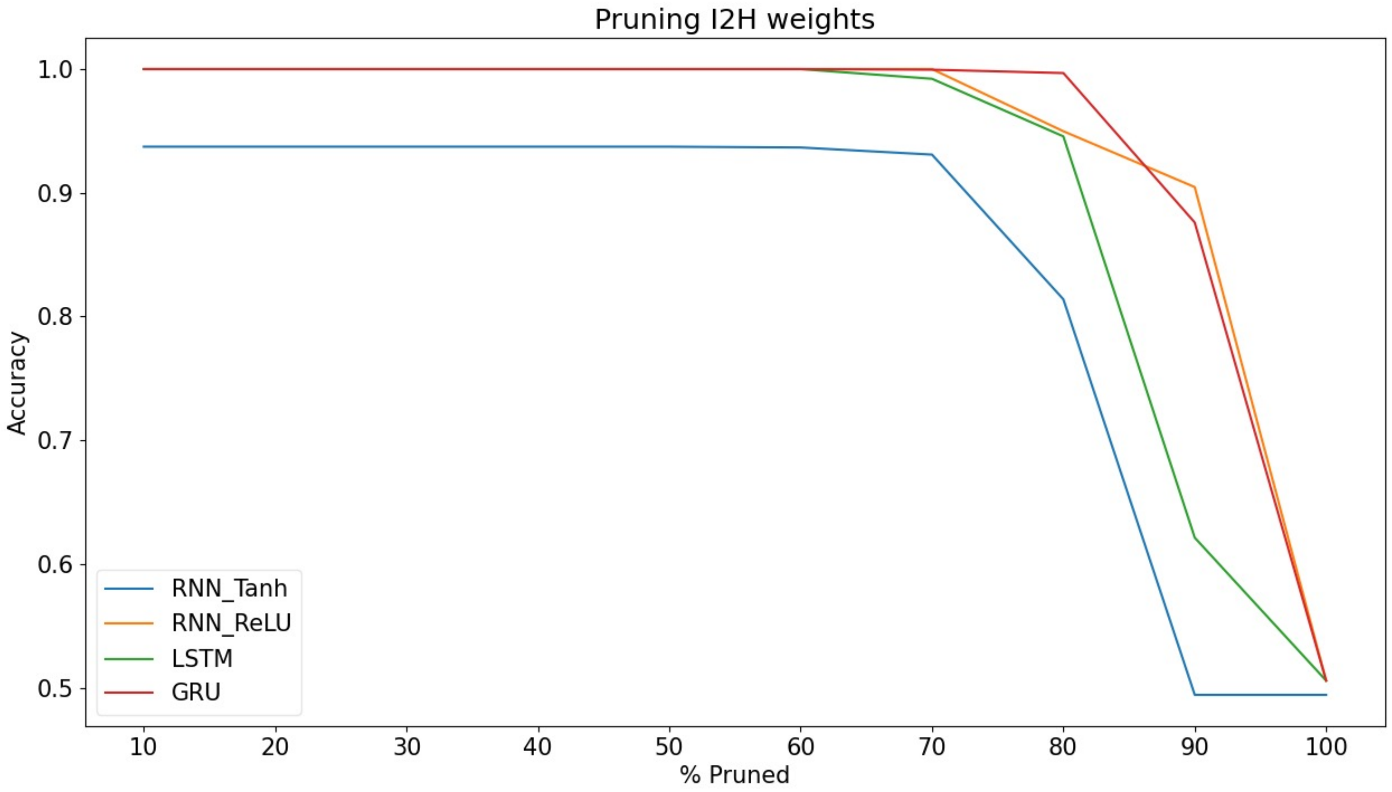
# 7.1. PRUNING RESULTS

- Pruning performance (*both I2H and H2H*):



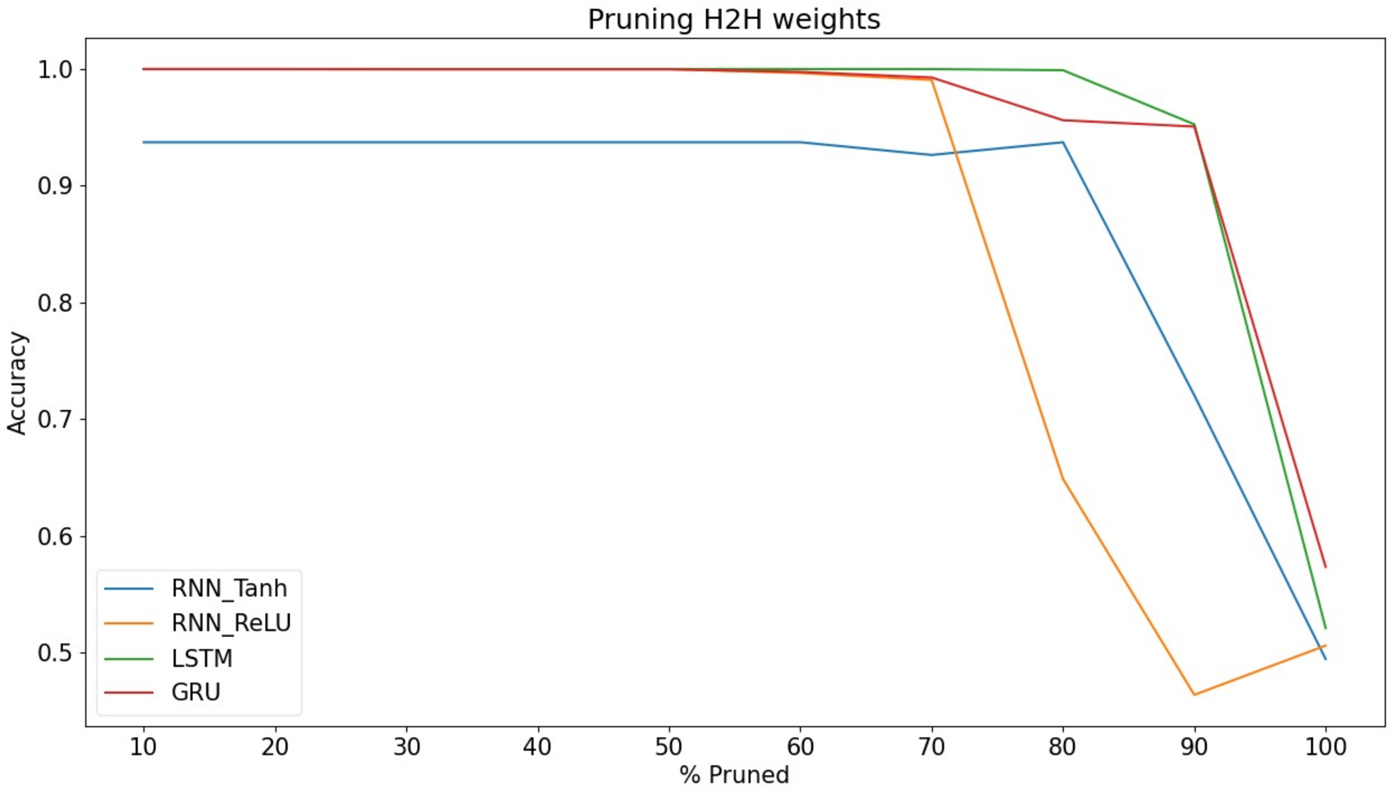
# 7.1. PRUNING RESULTS (cont.)

- Pruning I2H weights:



# 7.1. PRUNING RESULTS (cont.)

- Pruning H2H weights:



# 7.1. PRUNING RESULTS (cont.)

- In most cases, each pruned model recovers only after one epoch,
  - except in the case of RNN_Tanh, after pruning 90% I2H weights, it requires two epochs to recover.
- In almost all the cases, pruned models never recover after 100% pruning,
  - except in the case of LSTM and GRU, both models still recovers in just one epoch, even with 100% pruning of H2H weights.

# 7.2. RANDOMLY STRUCTURED RNN PERFORMANCE

| Property | Correlation with *test_acc* | | | |
|---|---|---|---|---|
| | RNN_Tanh | RNN_ReLU | LSTM | GRU |
| nodes | **0.40** | **0.44** | **0.44** | **0.49** |
| edges | 0.38 | **0.43** | **0.42** | **0.49** |
| source_nodes | 0.35 | **0.47** | **0.57** | **0.74** |
| degree_var | -0.28 | -0.26 | -0.39 | **-0.58** |
| closeness_var | **-0.46** | -0.39 | **-0.51** | **-0.67** |
| nodes_betweenness_var | **-0.49** | **-0.41** | **-0.56** | **-0.52** |
| edge_betweenness_var | -0.34 | -0.30 | **-0.44** | -0.26 |

Pearson correlation between test accuracy and different graph and recurrent network properties

# 7.3. PERFORMANCE PREDICTION

| RNN variant | Bayesian Ridge | Random Forest | AdaBoost |
|---|---|---|---|
| RNN_Tanh | **0.47919** | 0.43163 | 0.35698 |
| RNN_ReLU | 0.36075 | **0.61504** | 0.53469 |
| LSTM | 0.37206 | 0.57933 | **0.59514** |
| GRU | 0.67224 | **0.87635** | 0.78313 |

$R^2$-value from each regressor, for each RNN variant

# 8. CONCLUSION

- Two different methods to induce sparsity:
    1. Pruning,
    2. Randomly structured RNN
- Pruning results conclude that the weight complexity of various RNN variants can safely be reduced by more than 60%.
- In most cases of pruning, it only takes one epoch for a model to recover.
- Random structure experiments identified two essential graph properties that are mutual in all four RNN variant.
- Performance prediction experiment shows data from RNN_ReLU and GRU fits well with Random Forest, while the data from LSTM fits good with AdaBoost.

# THANK YOU

QUESTIONS?