# Deep Statistical Solvers

Course Project — DS303: Machine Learning

**Team AlphaSolvers**

Sai Aneesh Suryadevara - 190100125

Bhargav Rongali - 190100102

Indian Institute of Technology, Bombay

April 22, 2022

# Overview

1. Problem Statement
2. Before Midterm review
3. Midterm Review
4. Work Done after Midterm Review
5. Experiments & Results
6. Conclusions
7. Future Directions
8. Team Contributions

# Problem Statement

- **Optimization Problem:** The elementary question is to solve the following optimization problem for a given Interaction Graph $G$

$$U^*(G) = \arg\min_{U \in \mathcal{U}} L(U, G)$$

- **Learning Goal:** Because the above optimization problem is not easy to solve, we want to learn a single solver that best approximates the mapping $G \to U^*(G)$ for all $G$.

$$\hat{U}(G) = Solver(G) \quad \text{where, } \hat{U} \text{ is an approximation is } U^*$$

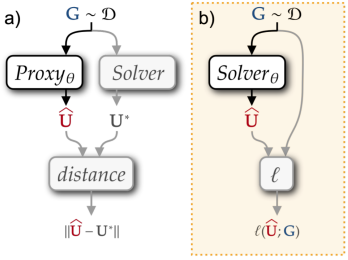



Figure: Proxy approach (a) vs. DSS (b)

# Graph Neural Network implementation of a DSS

- **Node Embedding**: For each node, $H_i$ is an embedding of actual state $U_i$, which is used for training and prediction

- **Message passing**: This step performs $\overline{k}$ updates on the latent state variable $\mathbf{H}$ using an **UPDATE** function $M_\theta^k$ and **AGGREGATE** function $\mathbf{\Psi}$

$$\phi^k_{\rightarrow,i} = \sum_{j \in \mathcal{N}^\star(i;\mathbf{G})} \Phi^k_{\rightarrow,\theta}(H_i^{k-1}, A_{ij}, H_j^{k-1}) \qquad \textit{outgoing edges}$$

$$\phi^k_{\leftarrow,i} = \sum_{j \in \mathcal{N}^\star(i;\mathbf{G})} \Phi^k_{\leftarrow,\theta}(H_i^{k-1}, A_{ji}, H_j^{k-1}) \qquad \textit{ingoing edges}$$

$$\phi^k_{\circlearrowleft,i} = \Phi^k_{\circlearrowleft,\theta}(H_i^{k-1}, A_{ii}) \qquad \textit{self loop}$$

Latent states $H_i^k$ are then computed using trainable mapping $\Psi_\theta^k$, in a ResNet-like fashion:

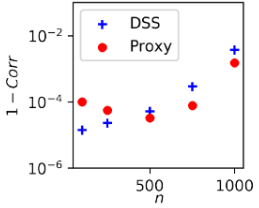
$$\mathbf{H}^k = \mathbf{M}_\theta^k(\mathbf{H}^{k-1}, \mathbf{G}) := (H_i^k)_{i\in[n]}, \text{ with } H_i^k = H_i^{k-1} + \Psi_\theta^k(H_i^{k-1}, B_i, \phi^k_{\rightarrow,i}, \phi^k_{\leftarrow,i}, \phi^k_{\circlearrowleft,i})$$

# Linear Solver Experiment

- We had seen how tempetrature distribution for a 2D domain can be obtined by solving Poisson equation using the DSS
- Following were its results

| Method | **DSS** | Proxy | LU |
|---|---|---|---|
| Correlation w/ LU | **>99.99%** | **>99.99%** | - |
| NRMSE w/ LU | 1.6e-3 | **1.1e-3** | - |
| Time per instance (ms) *Inference time divided by batch size | **1.8*** | **1.8*** | 2.4 |
| Loss $10^{th}$ percentile | **3.9e-4** | 7.0e-3 | 4.5e-27 |
| Loss $50^{th}$ percentile | **1.2e-3** | 1.6e-2 | 6.1e-26 |
| Loss $90^{th}$ percentile | **4.1e-3** | 4.0e-2 | 6.3e-25 |

- Next we had analyzed how well trained model is able to generalize to a distribution that is different from the training distribution



# Comments on Midterm Presentation

- Motivation could have been more elaborate and clear
- For the past work only the name of the approach or method is mentioned, could have covered a brief theme of these approaches
- Skipped a part on universal approximation property
- More emphasis regarding the performance metric (correlation between two solvers) could have been discussed as it is not a common performance metric.
- For future work: Include a code walkthrough of replication of work.

# Addressing the Comments

- **Universal approximation property** now explained in detail
- The performance metrics used **Pearson Correlation Coefficient** and NRMSE have been discussed in detail and also explained during the code demonstration
- Code has now been discussed properly and a code walkthorugh has been included
- We learnt about **GRNN** and made an attempt to incorporate it in the code

# Universal Approximation Property

Let us say that all the interaction graphs, $G$ are sampled from a given distribution $\mathcal{D}$. Let $supp(\mathcal{D})$ (support of $\mathcal{D}$) follow four hypotheses:

- **Permutation Invariance:** For any $G \in supp(\mathcal{D})$ and $\sigma \in \Sigma_n$, $\sigma \star G \in supp(\mathcal{D})$
- **Compactness:** $supp(\mathcal{D})$ is a compact subset
- **Connectivity:** For any $G \in supp(\mathcal{D})$, $\tilde{G}$ has only one connected component.
- **Seperability of external inputs:** There exist $\delta > 0$ such that for any $G = (n, A, B) \in supp(\mathcal{D})$ and any $i \neq j \in \{1, 2, ..., n\}, ||B_i - B_j|| \geq \delta$

and let $l$ be a continuous and permutation-invariant loss function such that for any $G \in supp(\mathcal{D})$, our problem has a unique minimizer $U^*(G)$, continuous w.r.t $G$. Then $\forall\, \epsilon > 0\ \exists\, Solver_\theta$, such that

$$||Solver_\theta(G) - U^*(G)|| \leq \epsilon$$

# Normalization - Change of Variables (For Poisson Equation)

- In the Poisson case study, the nodes at the boundary are constrained (i.e. $A_{ii} = 1$ and $A_{ij} = 0$ if $i \neq j$), and the interior nodes are not
- Also, the coefficients of matrix A at these interior nodes satisfy a conservation equality ($A_{ii} = -\sum_{j \in [n] \setminus i} A_{ij}$)
- Due to this, the distributions of their respective $B_i$ are very different

To overcome this issue we use change of variables as follows:

$$B_i^{'} = \begin{cases} [B_i, 0, 0], & \text{if node } i \text{ is not constrained} \\ [0, 1, B_i], & \text{otherwise} \end{cases}$$

$$A_{ij}^{'} = \begin{cases} A_{ij}, & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases}$$

# Normalization and Evaluation Metric

So the now the loss function $l(U, G) = \sum_{i=1}^{n}(-B_i + \sum_{j=1}^{n} A_{ij} U_j)^2$ takes the form,

$$l^{'}(U, G^{'}) = \sum_{i=1}^{n} \left( (1 - B_i^{'2})(-B_i^{'1}) + B_i^{'2}(U_i - B_i^{'3}) + \sum_{j=1}^{n} A_{ij}^{'}(U_j - U_i) \right)^2$$

where $B_i^{'p}$ denotes the $p^{th}$ component of vector $B_i$

**Pearson Correlation Metric:** The correlation metric used by the authors is known as the Pearson correlation. It is defined as follows,

$$r_{12} = \frac{\sum_{i=1}^{n}(U_i^1 - \bar{U}^1)(U_i^2 - \bar{U}^2)}{\sqrt{\sum_{i=1}^{n}(U_i^1 - \bar{U}^1)^2}\sqrt{\sum_{i=1}^{n}(U_i^2 - \bar{U}^2)^2}}$$

# AC power flow experiments

The second SSP example is the AC power flow prediction. Here, We know

- The amount of power being produced & consumed throughout the grid - encoded into B
- The way power lines are interconnected and their physical properties - encoded into A
- Our goal is to compute the voltage defined at each electrical node - encoded in state U

Let's consider a power grid with n nodes. We define the complex voltage at node i

$$V_i = |V_i| e^{j\theta_i}$$

The real part of the power consumed is denoted by $P_{d,i}$ and the imaginary part by $Q_{d,i}$.

Current State-of-the-art AC power flow computation uses the **Newton-Raphson** method

# Loss Function : Kirchhoff's laws

The system of equations that govern the power grid are the following:

$$\forall i \in [n] \setminus \{i_s\}, \quad P_{g,i} - P_{d,i} = \sum_{j \in [n]} |V_i||V_j|(\mathrm{Re}(Y_{ij})\cos(\theta_i - \theta_j) + \mathrm{Im}(Y_{ij})\sin(\theta_i - \theta_j))$$

$$\forall i \in I_{PQ}, \qquad -Q_{d,i} = \sum_{j \in [n]} |V_i||V_j|(\mathrm{Re}(Y_{ij})\sin(\theta_i - \theta_j) - \mathrm{Im}(Y_{ij})\cos(\theta_i - \theta_j))$$

$$\forall i \in I_{PV}, \qquad |V_i| = V_{g,i}$$

This set of complex equations can be converted into a SSP using A, B and U and loss function-

$$\ell(\mathbf{U},\mathbf{G}) = \sum_{i\in[n]}(1-B_i^5)\left(-B_i^1 + U_i^1 \sum_{j\in[n]} A_{ij}^1 U_j^1 \cos(U_i^2 - U_j^2 - A_{ij}^2)\right)^2$$
$$+ \sum_{i\in[n]} B_i^3\left(-B_i^2 + U_i^1 \sum_{j\in[n]} A_{ij}^1 U_j^1 \sin(U_i^2 - U_j^2 - A_{ij}^2)\right)^2 + \lambda \sum_{i\in[n]}(1-B_i^3)\left(U_i^1 - B_i^4\right)^2$$

# Details of Experiments

- Experiments are conducted on two standard benchmarks (n = 14) and (n = 118).
- For case 14 (resp. case 118), the dataset is split into 16064/2008/2008 (resp. 18432/2304/2304).

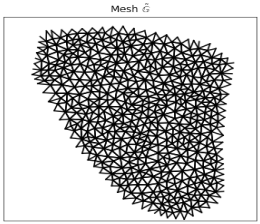
**Neural Network Details**

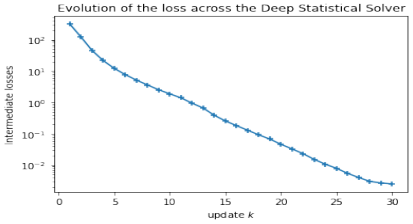
- Each NN block : single hidden layer of dimension d = 10 & leaky-ReLU non linearity
- Xavier Initialization, Adam Optimizer and Gradient Clipping were used
- For case n = 14, $\overline{k}$ was set to 10 ; we have $\alpha = 10^{-2}$, $lr = 3 * 10^{-3}$ and $\gamma = 0.9$
- For case n = 118, $\overline{k}$ was set to 30 ; we have $\alpha = 3 * 10^{-4}$, $lr = 3 * 10^{-3}$ and $\gamma = 0.9$)
- The number of weights is 1,722 for each of the $\overline{k}$ (M, D) blocks

# Results

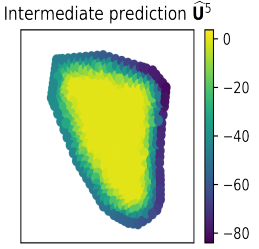
| Dataset | | IEEE 14 nodes | | | IEEE 118 nodes | | |
|---|---|---|---|---|---|---|---|
| Method | | **DSS** | Proxy | NR | **DSS** | Proxy | NR |
| Corr. w/ NR | $\|V_i\|$ | 99.93% | $>$**99.99%** | - | 99.79% | $>$**99.99%** | - |
| | $\theta_i$ | 99.86% | $>$**99.99%** | - | 81.31% | $>$**99.99%** | - |
| | $P_{ij}$ | $>$**99.99%** | $>$**99.99%** | - | $>$**99.99%** | $>$**99.99%** | - |
| | $Q_{ij}$ | $>$**99.99%** | $>$**99.99%** | - | $>$**99.99%** | $>$**99.99%** | - |
| NRMSE w/ NR | $\|V_i\|$ | 2.0e-3 | **4.9e-4** | - | 1.4e-3 | **1.2e-3** | - |
| | $\theta_i$ | 7.1e-3 | **1.7e-3** | - | 5.7e-2 | **4.5e-3** | - |
| | $P_{ij}$ | 6.2e-4 | **2.6e-4** | - | 1.0e-3 | **3.9e-4** | - |
| | $Q_{ij}$ | 4.2e-4 | **2.0e-4** | - | 1.1e-4 | **1.7e-4** | - |
| Time per instance (ms) * Inference time divided by batch size | | **1e-2*** | **1e-2*** | 2e1 | **2e-1*** | 2e-1* | 2e1 |
| Loss $10^{th}$ percentile | | **4.2e-6** | 2.3e-5 | 1.4e-12 | **1.3e-6** | 6.2e-6 | 2.9e-14 |
| Loss $50^{th}$ percentile | | **1.0e-5** | 4.0e-5 | 2.1e-12 | **1.7e-6** | 8.3e-6 | 4.2e-14 |
| Loss $90^{th}$ percentile | | **4.4e-5** | 1.2e-4 | 3.3e-12 | **2.5e-6** | 1.3e-5 | 6.4e-14 |

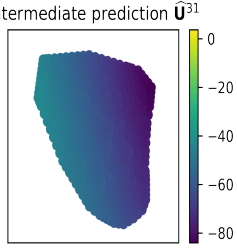
# Replication of results -Linear Systems

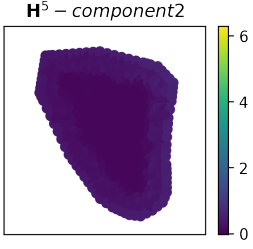


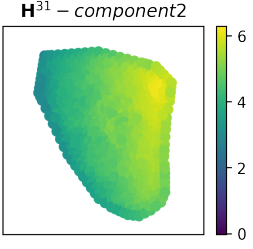




Intermediate predictions of State U and latent state H evolution

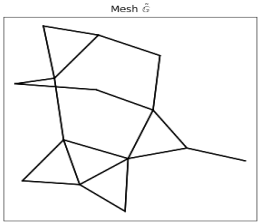


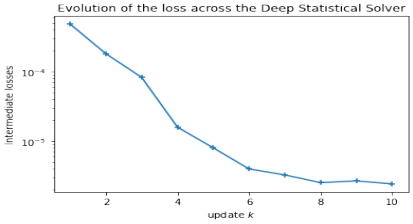



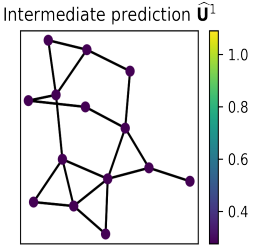



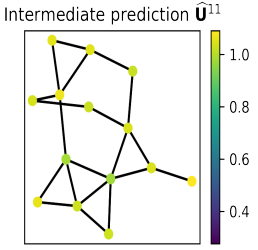
# Replication of results - AC Power Flow Prediction

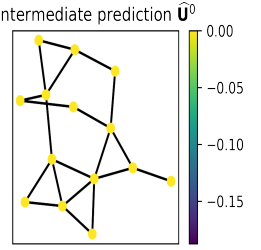


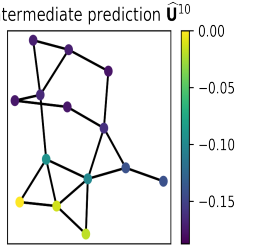




Intermediate predictions of Voltage and Phase Angle



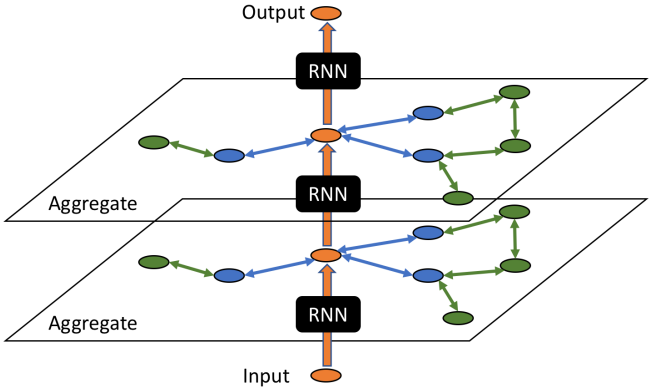



# Conclusions

- We have performed an in-depth analysis of the Deep Statistical Solvers and in doing so we also learnt about Graph Neural Networks(GNN) which are widely used for physics based simulations.
- We covered the Universal Approximation Property of DSS. This property gives a theoretical backup to the network hence justifying its use in various domains.
- The effectiveness of DSSs was also experimentally demonstrated on two problems. We saw that the accuracy on power flow computations matches that of state-of-the-art approaches while speeding up calculations by 2 to 3 orders of magnitude.
- Another application of DSS is that it can be used as an initialization heuristic for classical optimization algorithms.

# Extensions to the Work

- **Modification - Using Recurrent Graph Neural Networks (GRNN)** :
  In the paper authors have used GNN model of a limited depth. Instead, we can use **GNN with recurrent graph layers**. This will help in building deeper GNNs, without increasing the complexity of the training phase, while improving on the predictive performances.



# Team Contributions

1. **Aneesh:**
   - Replicated the code for both Linear Systems and AC power flow and verified the results
   - Ideated on the inclusion of Recurrent modules to the existing GNN architecture
   - Learnt about GNNs and GRNNs using Reference [4],[6]
   - Read the entire Deep Statistical Solver paper(Reference [1], [2])
   - Made slides corresponding to sections 3, 4, 5 and 7

2. **Bhargav:**
   - Worked on the comments given in mid-term review and explained Universal approximation property, Pearson correlation metric and change of variables
   - Verified the code replication for both experiments
   - Learnt about GNNs and GRNNs using Reference [4],[6]
   - Read the entire Deep Statistical Solver paper(Reference [1], [2])
   - Made slides corresponding to sections 1, 2, 4 and 6

# References


[1] Deep Statistical Solvers, 2020 by *Balthazar Donon, Wenzhuo Liu, Antoine Marot, Zhengying Liu, Isabelle Guyon, Marc Schoenaue* [Link]

[2] Author's presentation video. [Link]

[3] GitHub repository of Deep Statistical solvers [Link]

[4] YouTube Playlist on GNNs [Link]

[5] Penn Engineering, Graph Neural Networks [Link]

[6] Residual or Gate? Towards Deeper Graph Neural Networks for Inductive Graph Representation Learning, 2019, *Binxuan Huang, Kathleen M. Carley* [Link]