Discrete Optimizations using Graph Deep Learning

Radu Balan

Department of Mathematics, CSCAMM and NWC
University of Maryland, College Park, MD

joint work with Naveed Haghani

March 11, 2019
Workshop on Machine Learning and Applications at UMD

Acknowledgments

"This material is based upon work partially supported by the National Science Foundation under grant no. DMS-1413249, ARO under grant W911NF-16-1-0008, and LTS under grant H9823013D00560049. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation."

Combinatorial Problems
Approach

In this talk we consider the class of combinatorial problems,

$$\begin{array}{ll} \textit{maximize} & J(\Pi; \textit{Input}) \\ \text{subject to:} & \\ \Pi \in S_n & \end{array}$$

where *Input* stands for a given set input data, and S_n denotes the symmetric group of permutation matrices.

We analyze two specific objective functions:

1. Linear Assignment, $J(\Pi; C) = trace(\Pi C^T)$
2. Quadratic Assignment, $J(\Pi; A, B) = trace(A \Pi B \Pi^T)$

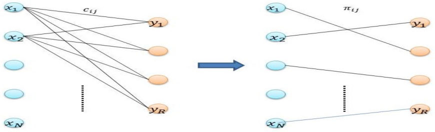
Idea: Use a two-step procedure:

1. Perform a latent representation of the Input Data using a Graph Convolutive Network;
2. Apply a direct algorithm (e.g., a greedy-type algorithm) or solve a convex optimization problem to obtain an estimate of the optimal Π.

The Linear Assignment Problem

Consider a $N \times R$ cost/reward matrix $C = (C_{i,j})_{1 \leq i \leq N, 1 \leq j \leq R}$ of non-negative entries associated to edge connections between two sets of nodes, $\{x_1, \cdots, x_N\}$ and $\{y_1, \cdots, y_R\}$ with $N \geq R$. The problem is to find the minimum cost/maximum reward matching/assignment, namely:

$$\begin{array}{cc} minimize/maximize & \sum_{i=1}^{N} \sum_{j=1}^{R} \pi_{i,j} C_{i,j} = trace(\Pi \tilde{C}^T) \\ \text{subject to:} & \\ \pi_{i,j} \in \{0,1\} \ , \ \forall i,j & \\ \sum_{i=1}^{N} \pi_{i,j} = 1 \ , \ \forall 1 \leq j \leq R & \\ \sum_{j=1}^{R} \pi_{i,j} \leq 1 \ , \ \forall 1 \leq i \leq N & \end{array}$$



Quadratic Assignment Problem

Consider two symmetric (and positive semidefinite) matrices $A, B \in \mathbb{R}^{n \times n}$. The *quadratic assignment problem* asks for the solution of

$$\begin{array}{ll} \textit{maximize} & \textit{trace}(A\Pi B\Pi^T) \\ \text{subject to:} & \\ \Pi \in S_n & \end{array}$$

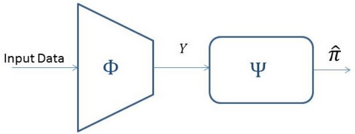
In turns this is equivalent to the minimization problem:

$$\begin{array}{ll} \textit{minimize} & \|\Pi A - B\Pi\|_F^2 \\ \text{subject to:} & \\ \Pi \in S_n & \end{array}$$

In the case A, B are graph Laplacian, an efficient solution to this optimization problem would solve the millenium problem of whether two graphs are isomorphic.

Novel Approach: Optimization in a Latent Representation Domain

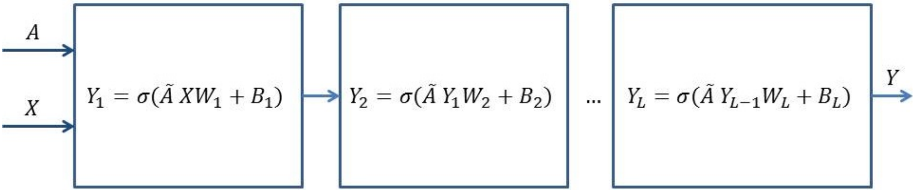
Idea: Perform a two-step procedure: (1) perform a nonlinear representation of the input data; (2) perform optimization in the representation space.



The nonlinear representation map $\Phi : \mathrm{Input\ Data} \mapsto Y$ is implemented using a GCN.
The Optimization map $\Psi : Y \mapsto \hat{\pi}$ can be implemented using a specific nonlinear map (e.g., greedy algorithm, or turning into stochastic matrix) or by solving a convex optimization problem.

Graph Convolutive Networks (GCN)

Kipf and Welling introduced a network structure that performs local processing according to a modified adjacency matrix:



Here $\tilde{A} = I + A$, where A is an input adjacency matrix, or graph weight matrix. The L-layer GCN has parameters $(W_1, B_1, W_2, B_2, \cdots, W_L, B_L)$. As activation map σ we choose the ReLU (Rectified Linear Unit).

Linear Assignment Problems using GCN

The GCN design: Consider the GCN with $N+R$ nodes, adjacency/weight matrix $\mathbf{A} = \begin{bmatrix} 0 & C \\ C^T & 0 \end{bmatrix}$ and data matrix $X = \begin{bmatrix} \nu(C(i,:)) \\ \nu(C^T(j,:)) \end{bmatrix}$.

Linear Assignment Problems using GCN

The GCN design: Consider the GCN with $N+R$ nodes, adjacency/weight matrix $\mathbf{A} = \begin{bmatrix} 0 & C \\ C^T & 0 \end{bmatrix}$ and data matrix $X = \begin{bmatrix} \nu(C(i,:)) \\ \nu(C^T(j,:)) \end{bmatrix}$.

Key observation: When $C = uv^T$, that is, when the cost matrix is rank one then:

1. Objective Function: $J(\Pi; C) = u^T \Pi v = \langle \Pi v, u \rangle$
2. GCN output when no bias ($B_j = 0$): $\Gamma = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix}$ satisfies $\Gamma_1 \Gamma_2^T = \alpha C$.

Consequence: the "greedy" algorithm produces the optimal solution.

Linear Assignment Problems using GCN

The GCN design: Consider the GCN with $N+R$ nodes, adjacency/weight matrix $\mathbf{A} = \begin{bmatrix} 0 & C \\ C^T & 0 \end{bmatrix}$ and data matrix $X = \begin{bmatrix} \nu(C(i,:)) \\ \nu(C^T(j,:)) \end{bmatrix}$.

Key observation: When $C = uv^T$, that is, when the cost matrix is rank one then:

1. Objective Function: $J(\Pi; C) = u^T \Pi v = \langle \Pi v, u \rangle$
2. GCN output when no bias ($B_j = 0$): $\Gamma = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix}$ satisfies $\Gamma_1 \Gamma_2^T = \alpha C$.

Consequence: the "greedy" algorithm produces the optimal solution.

Network Objective: Once trained, the GCN produces a latent representation $Z = \Gamma_1 \Gamma_2^T$ close to the input cost matrix C so that the greedy algorithm applied on Z produces the optimal solution.

Quadratic Assignment Problem using GCN

Preliminary result

The GCN Design: Consider the GCN with n nodes, adjacency/weight matrix $\mathbf{A} = \begin{bmatrix} 0 & AB \\ BA & 0 \end{bmatrix}$ and data matrix $X = \begin{bmatrix} A \\ B \end{bmatrix}$.

Quadratic Assignment Problem using GCN

Preliminary result

The GCN Design: Consider the GCN with n nodes, adjacency/weight matrix $\mathbf{A} = \begin{bmatrix} 0 & AB \\ BA & 0 \end{bmatrix}$ and data matrix $X = \begin{bmatrix} A \\ B \end{bmatrix}$.

Key observation: When $A = uu^T$ and $B = vv^T$, that is, when the matrices are rank one then:

1. Objective function: $J(\Pi; A, B) = (u^T \Pi v)^2 = (\langle \Pi v, u \rangle)^2$
2. GCN output when no bias (($B_j = 0$): $\Gamma = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix}$ satisfies $\Gamma_1 \Gamma_2^T \sim uv^T$.

Consequence: the "greedy" algorithm or the solution to the linear assignment problem associated to uv^T produces the optimal solution.

Quadratic Assignment Problem using GCN
Preliminary result

The GCN Design: Consider the GCN with n nodes, adjacency/weight matrix $\mathbf{A} = \begin{bmatrix} 0 & AB \\ BA & 0 \end{bmatrix}$ and data matrix $X = \begin{bmatrix} A \\ B \end{bmatrix}$.

Key observation: When $A = uu^T$ and $B = vv^T$, that is, when the matrices are rank one then:

1. Objective function: $J(\Pi; A, B) = (u^T \Pi v)^2 = (\langle \Pi v, u \rangle)^2$
2. GCN output when no bias (($B_j = 0$): $\Gamma = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix}$ satisfies $\Gamma_1 \Gamma_2^T \sim uv^T$.

Consequence: the "greedy" algorithm or the solution to the linear assignment problem associated to uv^T produces the optimal solution.

Network Objective: Once trained, the GCN produces a latent representation $Z = \Gamma_1 \Gamma_2^T$ so that the linear assignment problem associated to Z produces the same optimal permutation.

Deep Neural Networks as Universal Approximators

$$
\begin{array}{cc}
\textit{minimize/maximize} & \sum_{i=1}^{N}\sum_{j=1}^{R}\pi_{i,j}C_{i,j} \\
\text{subject to:} & \\
\pi_{i,j}\in\{0,1\}\ ,\ \forall i,j & \\
\sum_{i=1}^{N}\pi_{i,j}=1\ ,\ \forall 1\leq j\leq R & \\
\sum_{j=1}^{R}\pi_{i,j}\leq 1\ ,\ \forall 1\leq i\leq N &
\end{array}
$$

Luckily, the convex relaxation (Linear Program) produces the same optimal solution:

$$
\begin{array}{cc}
\textit{minimize} & \sum_{i=1}^{N}\sum_{j=1}^{R}\pi_{i,j}C_{i,j} \\
\text{subject to:} & \\
0\leq\pi_{i,j}\leq 1\ ,\ \forall i,j & \\
\sum_{i=1}^{N}\pi_{i,j}=1\ ,\ \forall 1\leq j\leq R & \\
\sum_{j=1}^{R}\pi_{i,j}\leq 1\ ,\ \forall 1\leq i\leq N &
\end{array}
$$

Deep Neural Networks as Universal Approximators

Architectures

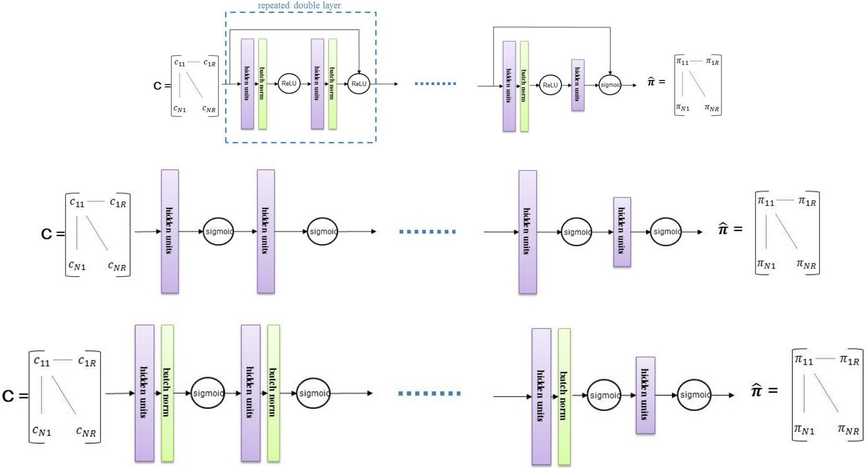
The overall system must output feasible solutions $\hat{\pi}$. Our architecture compose two components: (1) a deep neural network (DNN) that outputs a (generally) unfeasible estimate $\bar{\pi}$; (2) an enforcer (P) of the feasibility conditions that outputs the estimate $\hat{\pi}$:

Issues:

1. DNN architecture: how many layers; how many neurons per layer?
2. P, the feasibility enforcer

Deep Neural Networks as Universal Approximators
DNNs

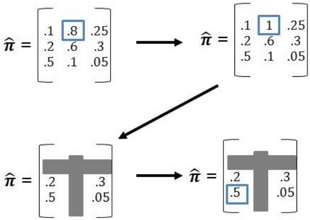
We studied three architectures:



Deep Neural Networks as Universal Approximators
Feasibility Enforcer P

An "optimal" feasibility condition enforcer would minimize some "distance" to the feasibility set. However this may be a very computationally expensive component. An intermediate solution is to alternate between different feasibility conditions (equalities and inequalities) until convergence.
Instead we opt for a simpler and "greedier" approach:

Repeat R times:
1. Find (i,j) the largest entry in $\bar{\pi}$
2. Set $\hat{\pi}_{i,j} = 1$; set to 0 other entries in row i and column j;
3. Remove row i and column j from both $\bar{\pi}$ and $\hat{\pi}$.



Deep Neural Networks as Universal Approximators

Baseline solution: The Greedy Algorithm

The "greedy" enforcer can be modified into a "greedy" optimization algorithm:

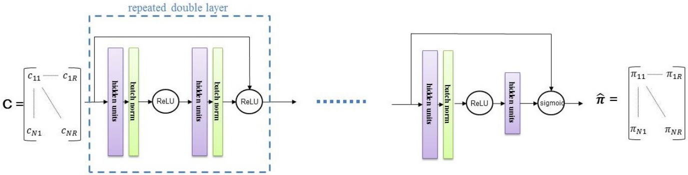
1. Initialize $E = C$ and $\hat{\pi} = 0_{N \times R}$
2. Repeat R times:
   - Find $(i, j) = argmin_{(a,b)} E_{a,b}$;
   - Set $\hat{\pi}_{i,j} = 1$, $\hat{\pi}_{i,l} = 0 \ \forall l \neq j$, $\hat{\pi}_{l,j} = 0 \ \forall l \neq i$;
   - Set $E_{i,:} = \infty$, $E_{:,j} = \infty$.

Proposition

The greedy algorithm produces the optimal solution if there is a positive number $\lambda > 0$ and two nonnegative vectors u, v such that $C = \lambda 1 \cdot 1^T - u \cdot v^T$.

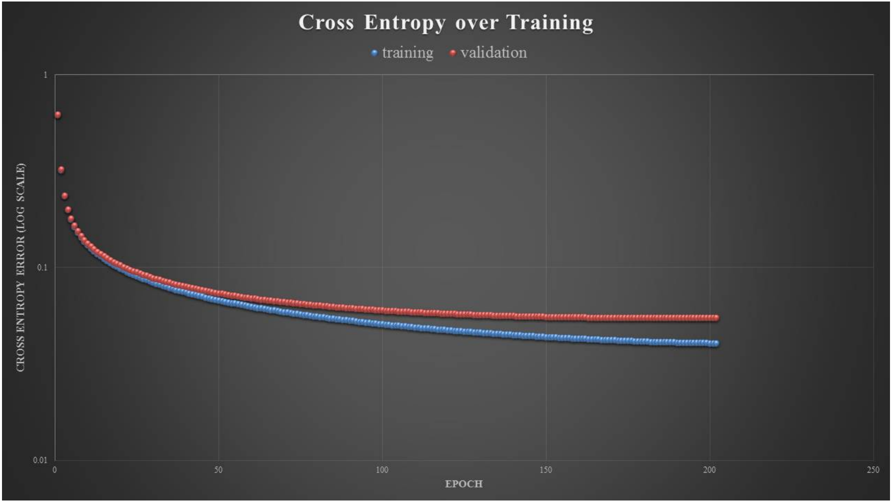
Exp.1 : $N = 5$, $R = 4$ with ReLU activation

First architecture:

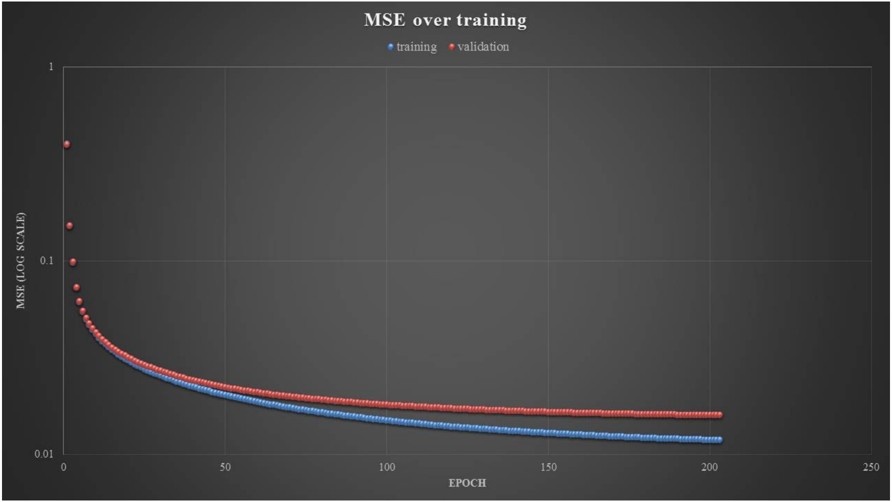


- Number of internal layers: 9
- Number of hidden units per layer: 250
- Batch size: 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 1 million random instances $U(0,1)$ i.i.d.
- Validation set: 20,000 random instances.

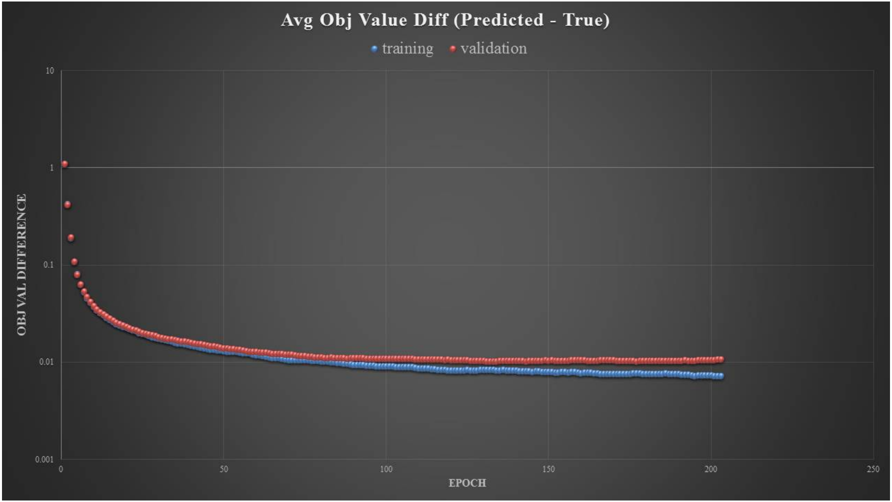
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



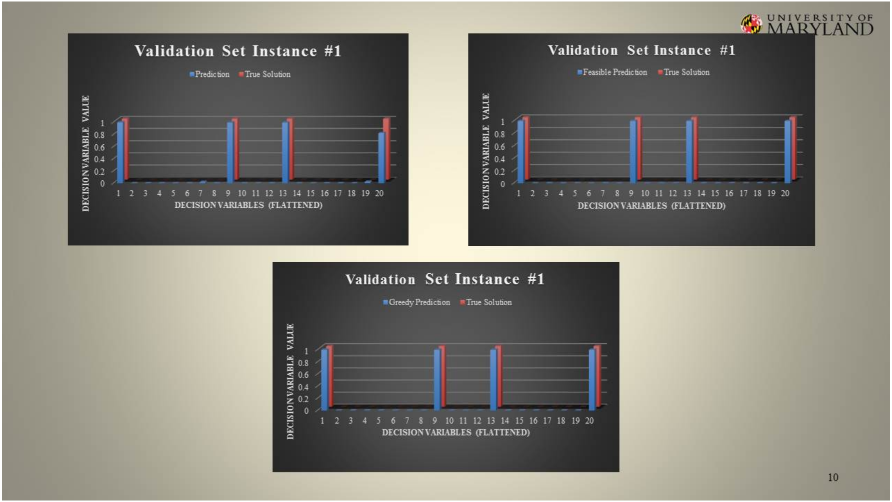
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



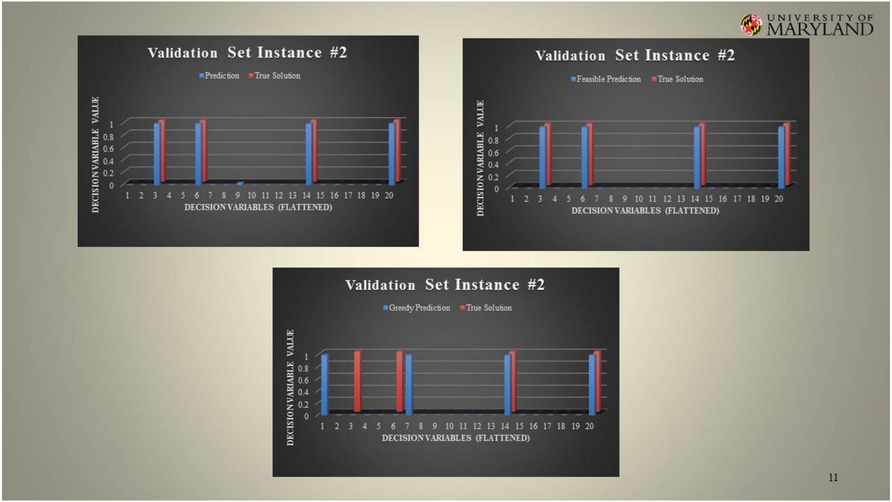
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



Exp.1 : $N = 5$, $R = 4$ with ReLU activation

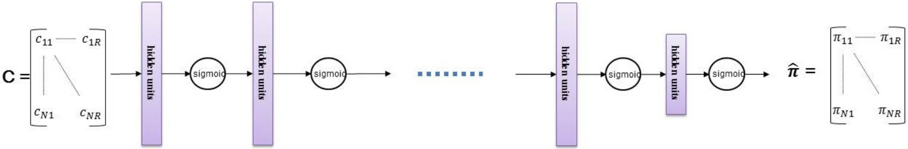


Exp.1 : $N = 5$, $R = 4$ with ReLU activation



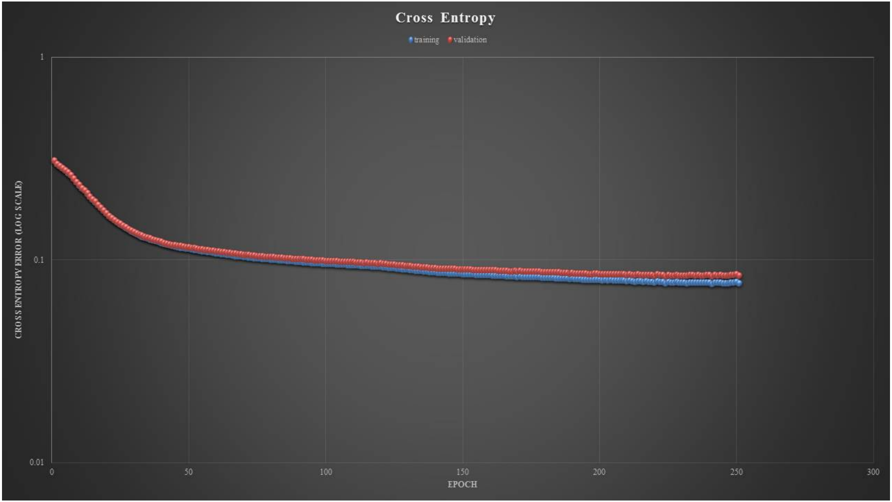
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation

Second architecture:

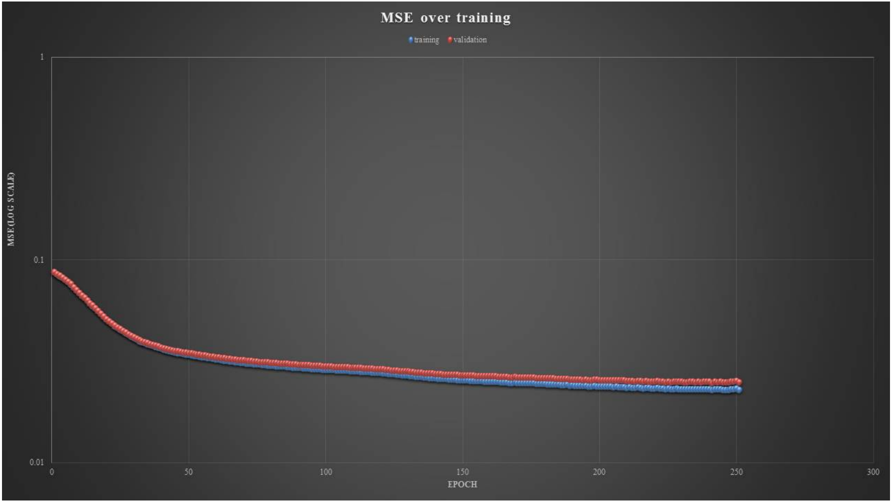


- Number of internal layers: 10
- Number of hidden units per layer: 250
- No Batch; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1-\pi_{i,j})(-log(1-\hat{\pi}_{i,j}))$
- Training data set: 1 million random instances $U(0,1)$ i.i.d.
- Validation set: 20,000 random instances.

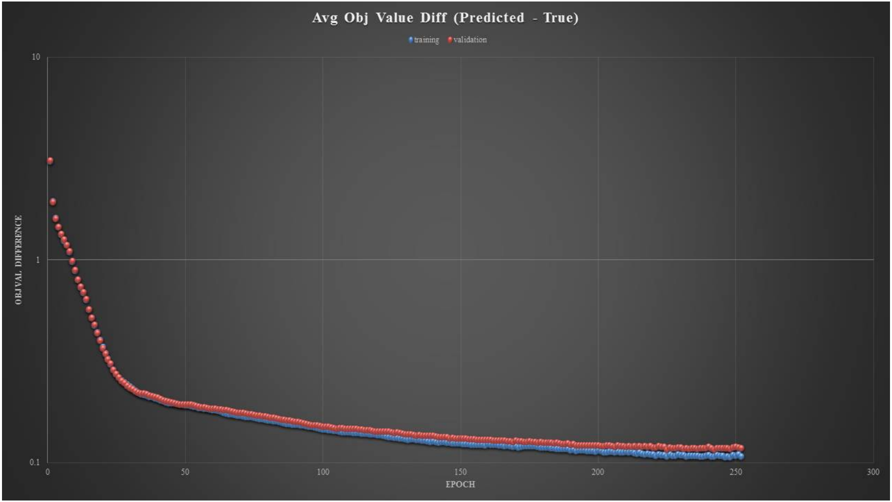
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



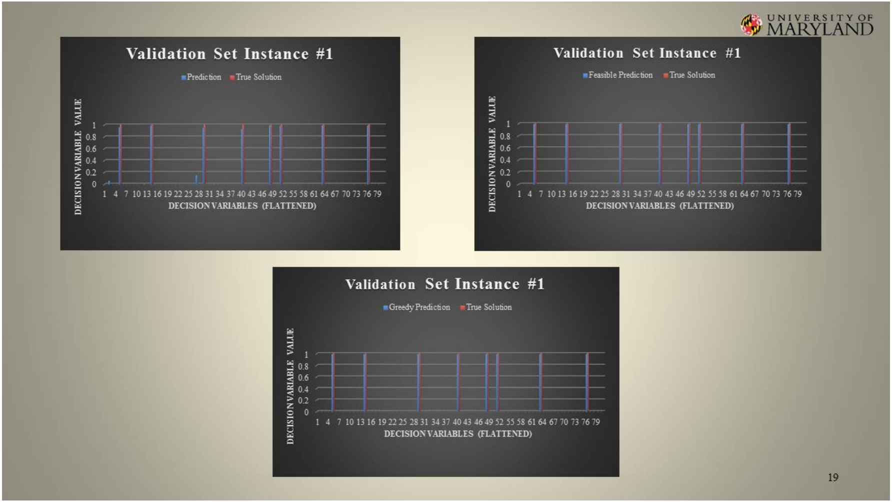
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



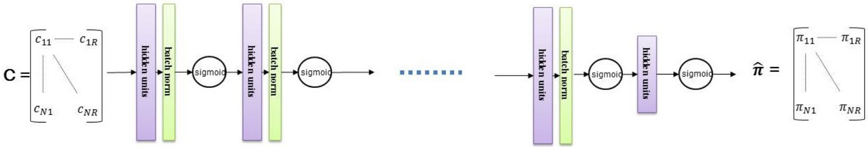
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



Exp.2 : $N = 10$, $R = 8$ with sigmoid activation

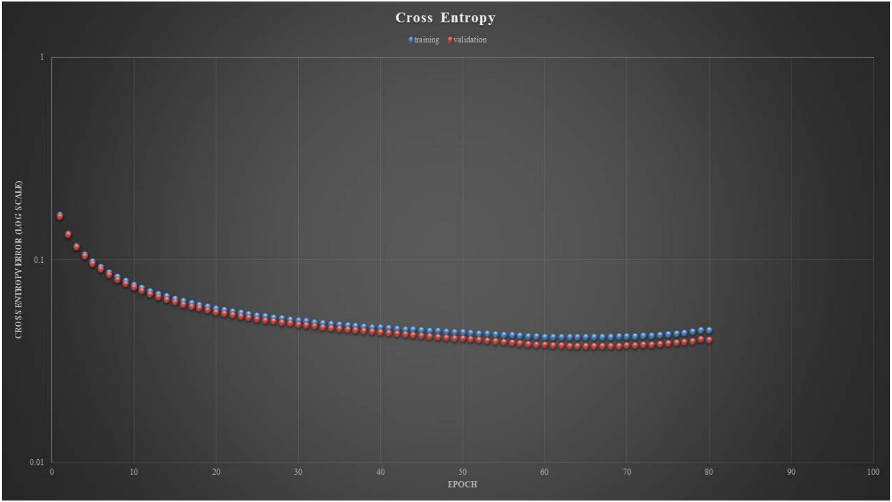
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation

Second architecture:

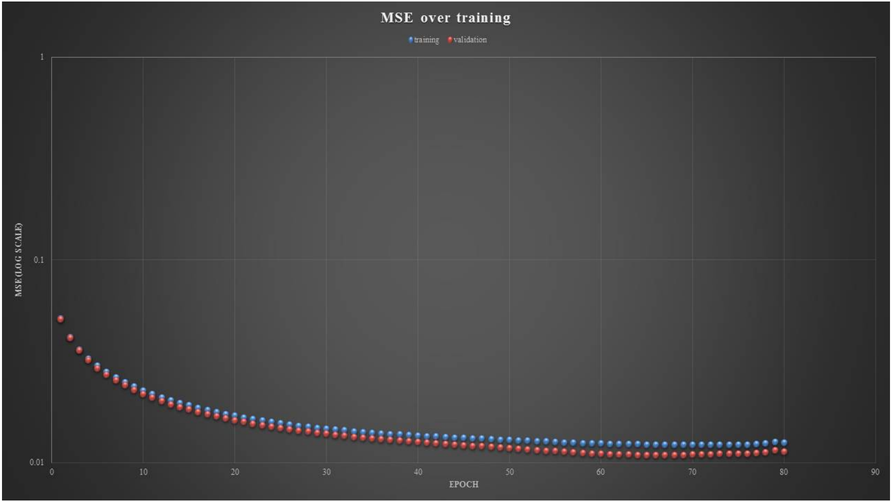


- Number of internal layers: 10
- Number of hidden units per layer: 250
- Batch size 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 500,000 random instances $U(0, 1)$ i.i.d.
- Validation set: 20,000 random instances.

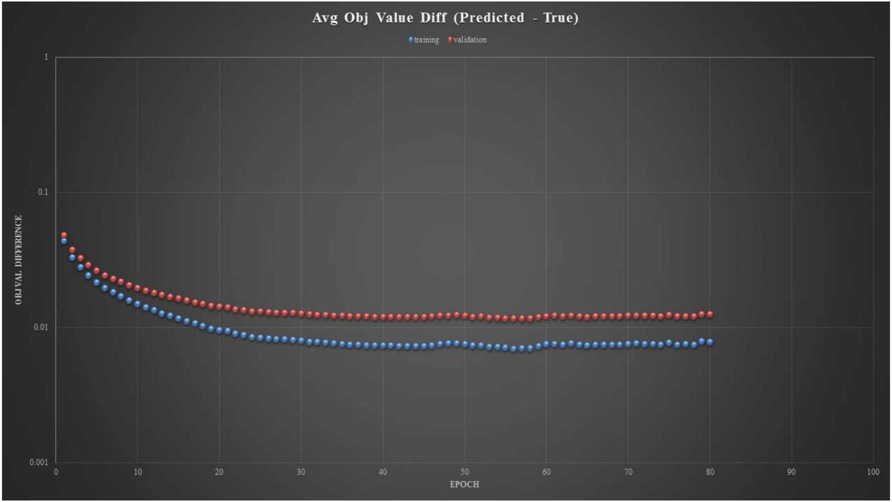
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



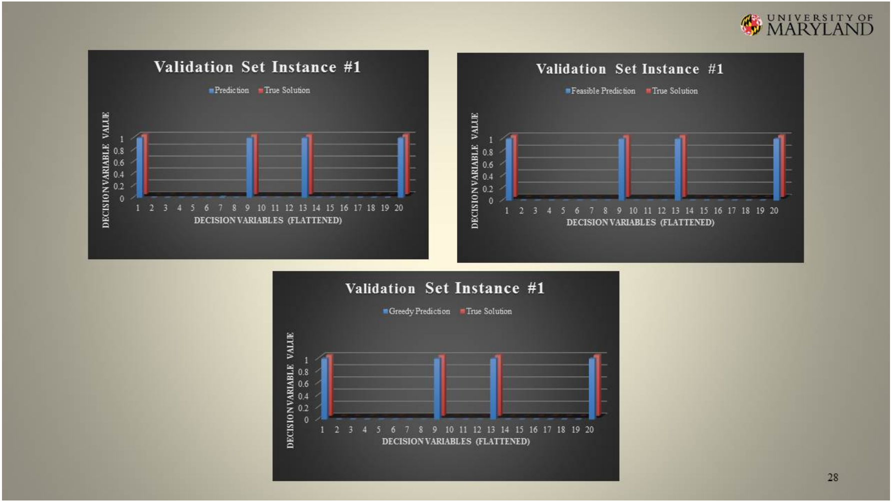
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



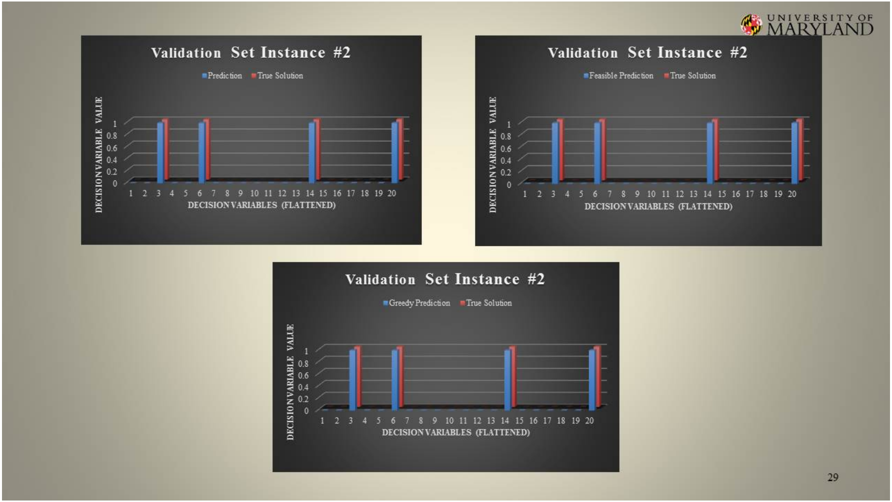
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



Exp.3 : $N = 5$, $R = 4$ with sigmoid activation

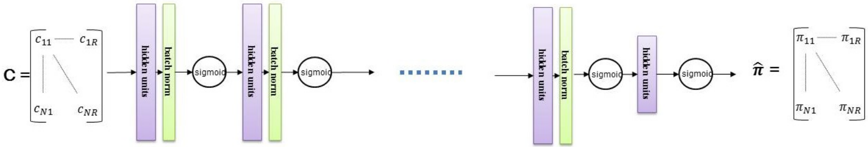


Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



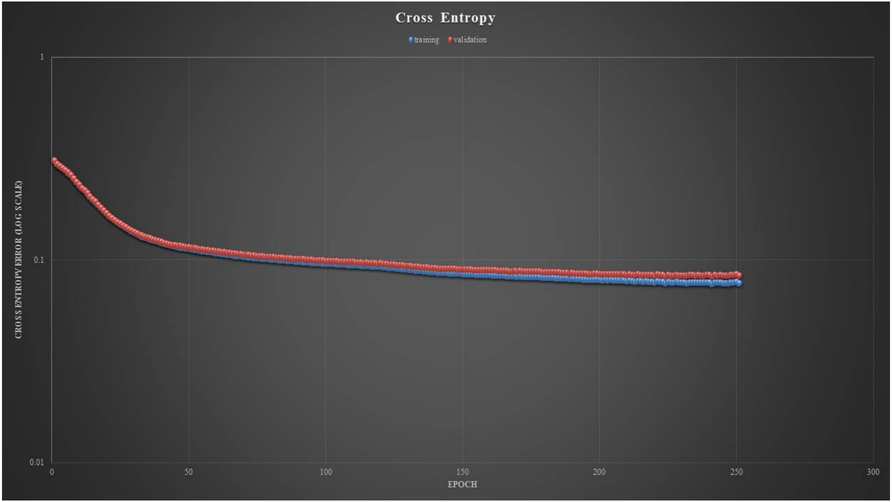
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation

Second architecture:

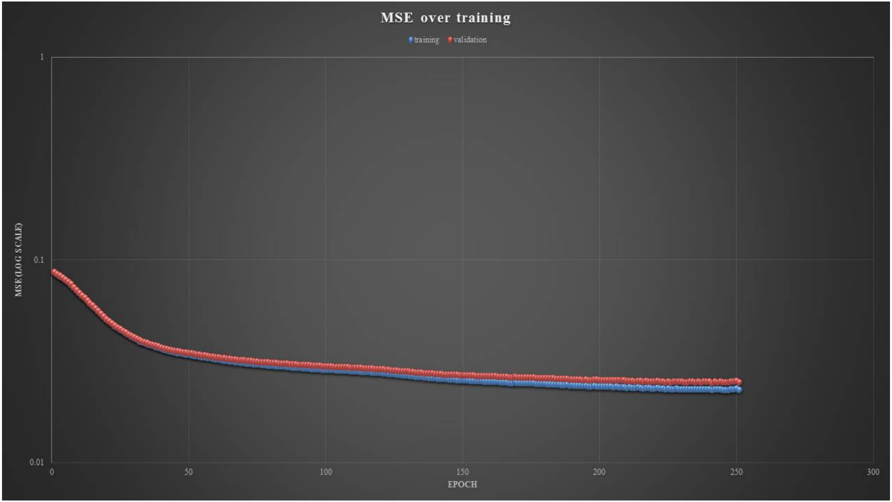


- Number of internal layers: 10
- Number of hidden units per layer: 300
- Batch size 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 500,000 random instances $U(0,1)$ i.i.d.
- Validation set: 20,000 random instances.

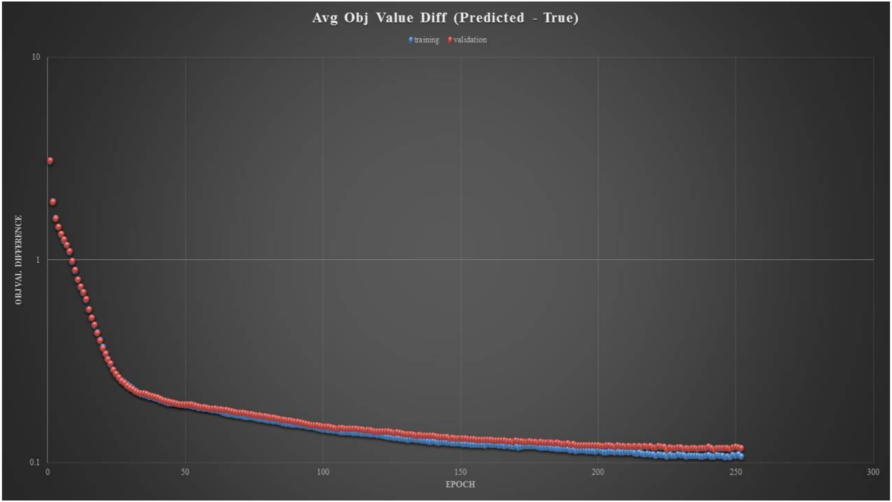
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



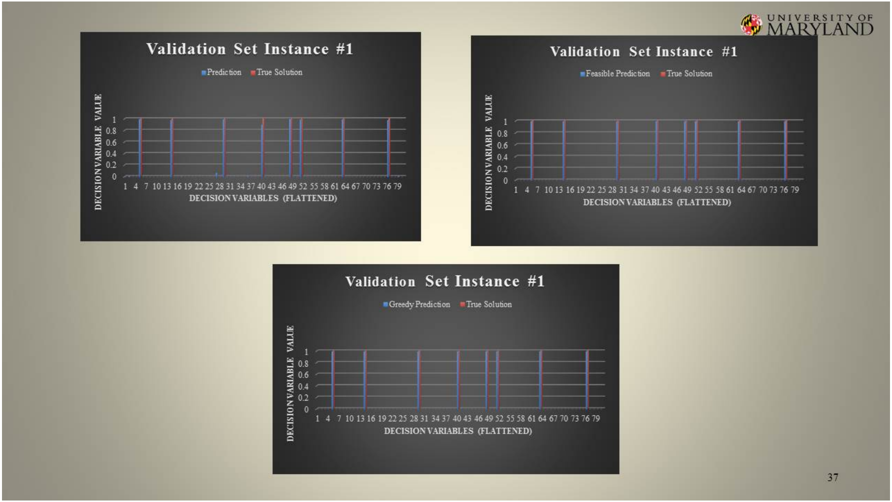
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



Exp.4 : $N = 10$, $R = 8$ with sigmoid activation

Bibliography

1. M. Andrychowicz, M. Denil, S.G. Colmenarejo, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N.de Freitas, *Learning to learn by gradient descent by gradient descent*, arXiv:1606.04474v2 [cs.NE]
2. T.N. Kipf, M. Welling, *Variatioal Graph Auto-Encoder*, arXiv:1611.07308 [stat.ML]
3. A. Nowak, S. Villar, A. Bandeira, J. Bruna, *Revised Note on Learning Quadratic Assignment with Graph Neural Network*, arXiv: 1706.07450 [stat.ML]