Permutation Invariance and Combinatorial Optimizations with Graph Deep Learning

Radu Balan

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

March 18, 2019
American University, Washington DC

Acknowledgments

"This material is based upon work partially supported by the National Science Foundation under grant no. DMS-1413249 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."

Collaborators:
Naveed Haghani (UMD)
Maneesh Singh (Verisk)
Debdeep Bhattacharya (GWU)

Table of Contents:

1. Permutation Invariant Representations

2. Optimizations using Deep Learning

3. DNN as UA

4. Numerical Results

Permutation Invariant induced Representations

Consider the equivalence relation $\sim$ on $\mathbb{R}^{n \times d}$ indiced by the group of permutation S_n: for any $X, X' \in \mathbb{R}^{n \times d}$,

$$X \sim X' \quad \Leftrightarrow \quad X' = PX \ , \ \text{for some } P \in S_n$$

Let $\mathbb{M} = \mathbb{R}^{n \times d} / \sim$ be the quotient space endowed with the natural distance induced by Frobenius norm $\| \cdot \|_F$

$$d(\hat{X}_1, \hat{X}_2) = \min_{P \in S_n} \|X_1 - PX_2\|_F \quad , \quad \hat{X}_1, \hat{X}_2 \in \mathbb{M}.$$

Permutation Invariant induced Representations

Consider the equivalence relation $\sim$ on $\mathbb{R}^{n\times d}$ indiced by the group of permutation S_n: for any $X, X' \in \mathbb{R}^{n\times d}$,

$$X \sim X' \quad \Leftrightarrow \quad X' = PX \ , \ \text{for some } P \in S_n$$

Let $\mathbb{M} = \mathbb{R}^{n\times d}/\sim$ be the quotient space endowed with the natural distance induced by Frobenius norm $\|\cdot\|_F$

$$d(\hat{X}_1, \hat{X}_2) = \min_{P\in S_n} \|X_1 - PX_2\|_F \quad , \quad \hat{X}_1, \hat{X}_2 \in \mathbb{M}.$$

The Problem: Construct a Lipschitz embedding $\hat{\alpha} : \mathbb{M} \to \mathbb{R}^m$, i.e., an integer $m = m(n, d)$, a map $\alpha : \mathbb{R}^{n\times d} \to \mathbb{R}^m$ and a constant $L = L(\alpha) > 0$ so that for any $X, X' \in \mathbb{R}^{n\times d}$,

1. If $X \sim X'$ then $\alpha(X) = \alpha(X')$
2. If $\alpha(X) = \alpha(X')$ then $X \sim X'$
3. $\|\alpha(X) - \alpha(X')\|_2 \leq L\, d(\hat{X}, \hat{X}')$

Motivation (1)

Graph Learning Problems

Consider data graphs such as: social networks, transportation networks, citation networks, chemical networks, protein networks, biological networks, etc. Each such network is modeled as a (weighted) graph $(\mathcal{V}, \mathcal{E}, A)$ of n nodes, and a set of feature vectors $\{x_1^T, \cdots, x_n^T\} \subset \mathbb{R}^d$ that form the matrix $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}$.

Two important problems involving a map $f : (A, X) \to f(A, X)$:

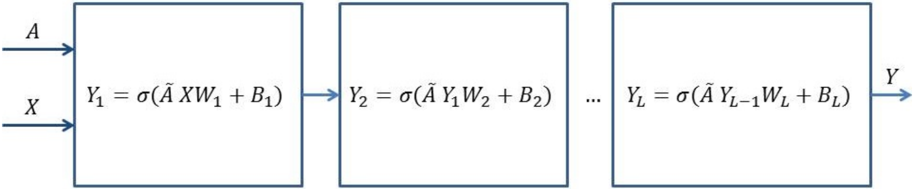
1. classification: $f(A, X) \in \{1, 2, \cdots, c\}$
2. regression/prediction: $f(A, X) \in \mathbb{R}$.

In each case we expect the task to be invariant to vertices permutation: $f(PAP^T, PX) = f(A, X)$, for every $P \in S_n$.

Motivation (2)

Graph Convolutive Networks (GCN)

Kipf and Welling ('16) introduced a network structure that performs local processing according to a modified adjacency matrix:

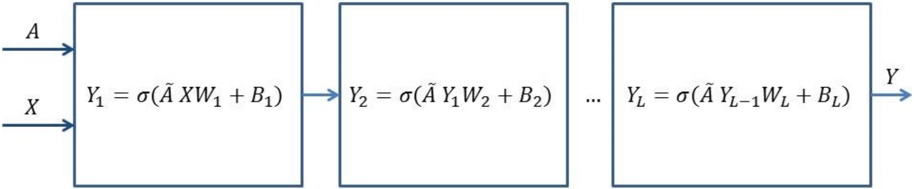


$\tilde{A} = I + A$, where A is the adjacency matrix, or the graph weight matrix; σ is the activation map. L-layer GCN has parameters $(W_1, B_1, \cdots, W_L, B_L)$.

Motivation (2)

Graph Convolutive Networks (GCN)

Kipf and Welling ('16) introduced a network structure that performs local processing according to a modified adjacency matrix:



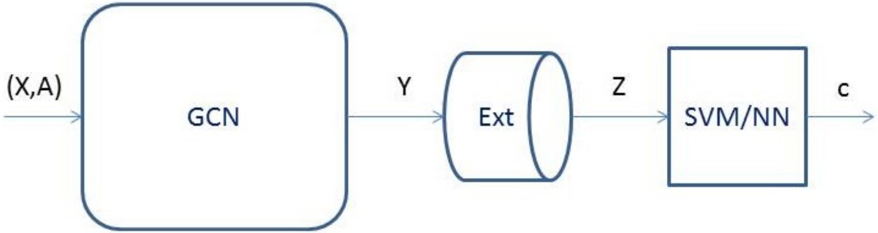
$\tilde{A} = I + A$, where A is the adjacency matrix, or the graph weight matrix; σ is the activation map. L-layer GCN has parameters $(W_1, B_1, \cdots, W_L, B_L)$.

Note the *covariance property*: for any $P \in O(n)$ (including S_n), $(A, X) \mapsto (PAP^T, PX)$ and $B_i \mapsto PB_i$ then $Y \mapsto PY$.

Motivation (3)

Deep Learning with GCN

The two learning tasks (classification or regression) can be solved by the following scheme:



where *Ext* is a permutation invariant feature extractor, and SVM/NN is a single-layer or a deep neural network (Support Vector Machine or a Fully Connected Neural Network).

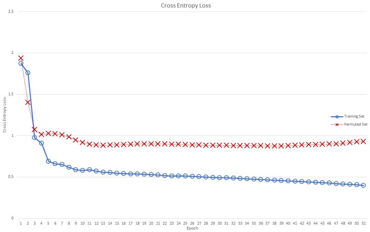
The purpose of this (part of the) talk is to analyze the *Ext* component

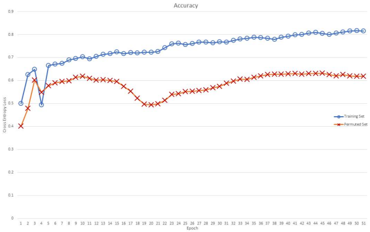
Motivation (4)

Enzyme Classification Example

Protein Dataset where task is classification into *enzyme* vs. *non-enzyme*.
Dataset: 450 enzymes and 450 non-enzymes.
Architecture (ReLU activation):

- GCN with $L = 3$ layers and $d = 25$ feature vectors in each layer;
- No Permutation Invariant Component: $Ext = Identity$
- Fully connected NN with dense 3-layers and 120 internal units.





The Measure Theoretic Embedding

First approach: Consider the map

$$\mu : \mathbb{M} \rightarrow \mathcal{P}(\mathbb{R}^d) \quad , \quad \mu(X)(x) = \frac{1}{n}\sum_{k=1}^{n} \delta(x - x_k)$$

where $\mathcal{P}(\mathbb{R}^d)$ denotes the convex set of probability measures over $\mathbb{R}^d$, and δ denotes the Dirac measure.
Clearly $\mu(X') = \mu(X)$ iff $X' = PX$ for some $P \in S_n$.

Main drawback: $\mathcal{P}(\mathbb{R}^d)$ is infinite dimensional!

Finite Dimensional Embeddings
Architectures

Two classes of extractors:

1. Pooling Map – based on Max pooling
2. Readout Map – based on Sum pooling

Finite Dimensional Embeddings

Architectures

Two classes of extractors:

1. Pooling Map – based on Max pooling
2. Readout Map – based on Sum pooling

Intuition in the case $d = 1$:

Max pooling:

$$\lambda : \mathbb{R}^n \to \mathbb{R}^n \quad , \quad \lambda(x) = (x_{\pi(k)})_{k=1}^n \ , \ x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots \geq x_{\pi(n)}$$

Finite Dimensional Embeddings
Architectures

Two classes of extractors:

1. Pooling Map – based on Max pooling
2. Readout Map – based on Sum pooling

Intuition in the case $d = 1$:

Max pooling:

$$\lambda : \mathbb{R}^n \to \mathbb{R}^n \quad , \quad \lambda(x) = (x_{\pi(k)})_{k=1}^n \; , \; x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots \geq x_{\pi(n)}$$

Sum pooling:

$$\sigma : \mathbb{R}^n \to \mathbb{R}^n \quad , \quad \sigma(x) = (y_k)_{k=1}^n \; , \; y_k = \sum_{j=1}^n \nu(a_k, x_j)$$

where kernel $\nu : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, e.g. $\nu(a,t) = e^{-(a-t)^2}$, or $\nu(a = k, t) = t^k$.

Pooling Mapping Approach

Fix a matrix $R \in \mathbb{R}^{d\times D}$. Consider the map:

$$\Lambda : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times D} \equiv \mathbb{R}^{nD} \quad , \quad \Lambda(X) = \lambda(XR)$$

where λ acts columnwise (reorders monotonically decreasing each column). Since $\Lambda(\Pi X) = \Lambda(X)$, then $\Lambda : \widehat{\mathbb{R}^{n\times d}} \to \mathbb{R}^{n\times D}$.

Theorem

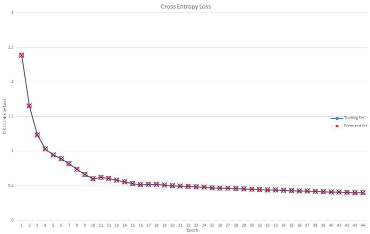
For any matrix $R \in \mathbb{R}^{n,d+1}$ so that any $n \times n$ submatrix is invertible, there is a subset $Z \subset \widehat{\mathbb{R}^{n\times d}}$ of zero measure so that $\Lambda : \widehat{\mathbb{R}^{n\times d}} \setminus Z \to \mathbb{R}^{n\times d+1}$ is faithful (i.e., injective).

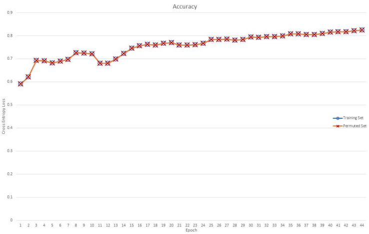
No known tight bound yet as to the minimum $D = D(n, d)$ so that there is a matrix R so that Λ is faithful (injective).
However, due to local linearity, if Λ is faithful (injective), then it is stable.

Enzyme Classification Example
Extraction with Hadamard Matrix

Protein Dataset where task is classification into *enzyme* vs. *non-enzyme*.
Dataset: 450 enzymes and 450 non-enzymes.
Architecture (ReLU activation):

- GCN with $L = 3$ layers and $d = 25$ feature vectors in each layer;
- $Ext = \Lambda$, $Z = \lambda(YR)$ with $R = [I \; Hadamard]$. $D = 50$, $m = 50$.
- Fully connected NN with dense 3-layers and 120 internal units.





Readout Mapping Approach

Kernel Sampling

Consider:

$$\Phi : \mathbb{R}^{n\times d} \to \mathbb{R}^m \quad , \quad (\Phi(X))_j = \sum_{k=1}^{n} \nu(a_j, x_k) \text{ or } (\Phi(X))_j = \prod_{k=1}^{n} \nu(a_j, x_k)$$

where $\nu : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel, and $x_1, \cdots, x_n$ denote the rows of matrix X.

Known solutions: If $m = \infty$, then there exists a Φ that is globally faithful (injective) and stable on compacts.

Interesting mathematical connexion: On compacts, some kernels ν define Repreducing Kernel Hilberts Spaces (RKHSs) and yield a decomposition

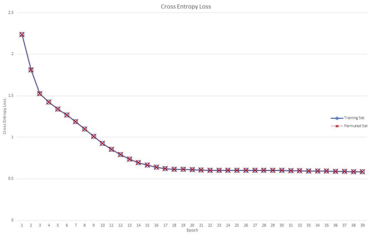
$$(\Phi(X))_j = \sum_{p\geq 1} \sigma_p f_p(a_j) g_p(X)$$

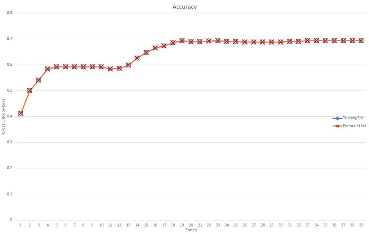
Enzyme Classification Example

Feature Extraction with Exponential Kernel Sampling

Protein Dataset where task is classification into *enzyme* vs. *non-enzyme*.
Dataset: 450 enzymes and 450 non-enzymes.
Architecture (ReLU activation):

- GCN with $L = 3$ layers and $d = 25$ feature vectors in each layer;
- $Ext : Z_j = \sum_{k=1}^{n} exp(-\pi \|y_k - z_j\|)$ with $m = 120$ and z_j random.
- Fully connected NN with dense 3-layers and 120 internal units.





Readout Mapping Approach

Polynomial Expansion - Quadratics

Another interpretation of the moments for $d = 1$: coefficients of linear expansion

$$P(X) = \frac{1}{n}\sum_{k=1}^{n}(X - x_k)^n = X^n + \sum_{k=1}^{n} a_k X^{n-k}$$

Readout Mapping Approach

Polynomial Expansion - Quadratics

Another interpretation of the moments for $d=1$: coefficients of linear expansion

$$P(X)=\frac{1}{n}\sum_{k=1}^{n}(X-x_k)^n = X^n+\sum_{k=1}^{n} a_k X^{n-k}$$

For $d>1$, consider the quadratic d-variate polynomial:

$$\begin{aligned} P(Z_1,\cdots,Z_d) &= \prod_{k=1}^{n}\left((Z_1-x_k(1))^2+\cdots+(Z_d-x_k(d))^2\right) \\ &= \sum_{p_1,\ldots,p_d=0}^{2n} a_{p_1,\ldots,p_d} Z_1^{p_1}\cdots Z_d^{p_d} \end{aligned}$$

Encoding complexity:

$$m = O\left(\begin{array}{c} 2n+d \\ d \end{array}\right) \sim (2n)^d.$$

Algebraic Embedding

Encoding using Complex Roots

Idea: Consider the case $d = 2$. Then each $x_1, \cdots, x_n \in \mathbb{R}^2$ can be replaced by n complex numbers $z_1, \cdots, z_n \in \mathbb{C}$, $z_k = x_k(1) + ix_k(2)$. Then consider the complex polynomial:

$$Q(z) = \prod_{k=1}^{n}(z - z_k) = z^n + \sum_{k=1}^{n} \sigma_k z^{n-k}$$

which requires n complex numbers, or $2n$ real numbers.

Algebraic Embedding

Encoding using Complex Roots

Idea: Consider the case $d = 2$. Then each $x_1, \cdots, x_n \in \mathbb{R}^2$ can be replaced by n complex numbers $z_1, \cdots, z_n \in \mathbb{C}$, $z_k = x_k(1) + ix_k(2)$. Then consider the complex polynomial:

$$Q(z) = \prod_{k=1}^{n}(z - z_k) = z^n + \sum_{k=1}^{n} \sigma_k z^{n-k}$$

which requires n complex numbers, or $2n$ real numbers.
For $d > 3$ encode each combination of two columns of $X \in \mathbb{R}^{n \times d}$: Total of $d(d-1)/2$ combinations, each using $2n$ real numbers.

Encoding complexity: $m = nd(d-1)$

Combinatorial Optimization Problems
Approach

Consider the class of combinatorial problems,

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

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, $J(\Pi; A, B) = trace(A \Pi B \Pi^T)$

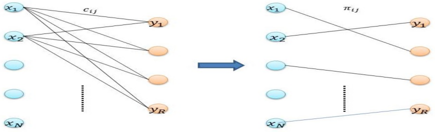
Idea: Use a two-step procedure:

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:

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



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

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

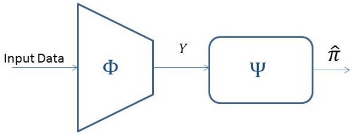
In turns this is equivalent to the minimization problem:

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

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

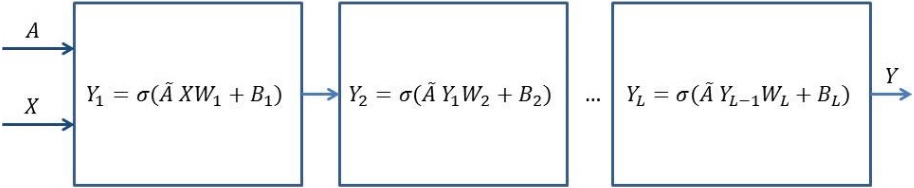
Idea: Perform a two-step procedure: (1) perform a nonlinear representation of the input data; (2) perform optimization in the representation space.



The nonlinear representation map $\Phi : \text{Input Data} \mapsto Y$ 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 $\hat{\pi}$:

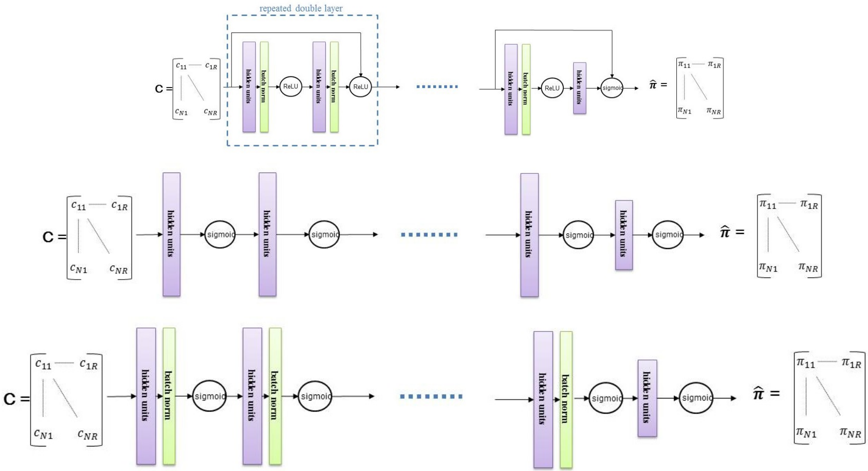
Issues:

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

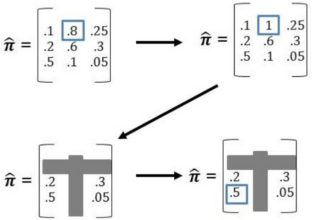
Feasibility Enforcer P

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:

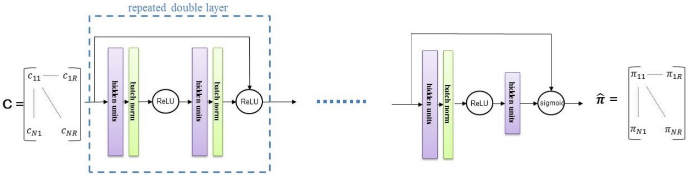
1. Initialize $E = C$ and $\hat{\pi} = 0_{N \times R}$
2. Repeat R times:
   - Find $(i, j) = argmin_{(a,b)} E_{a,b}$;
   - Set $\hat{\pi}_{i,j} = 1$, $\hat{\pi}_{i,l} = 0 \ \forall l \neq j$, $\hat{\pi}_{l,j} = 0 \ \forall l \neq i$;
   - Set $E_{i,:} = \infty$, $E_{:,j} = \infty$.

Proposition

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

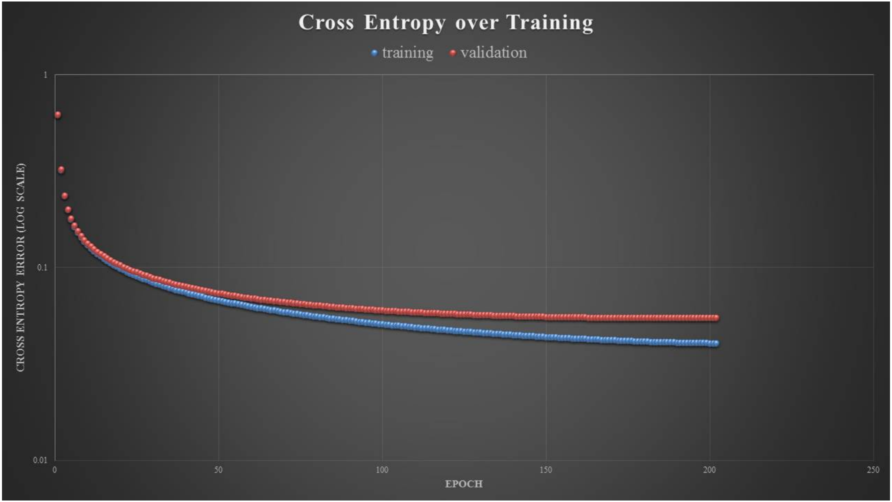
Exp.1 : $N = 5$, $R = 4$ with ReLU activation

First architecture:

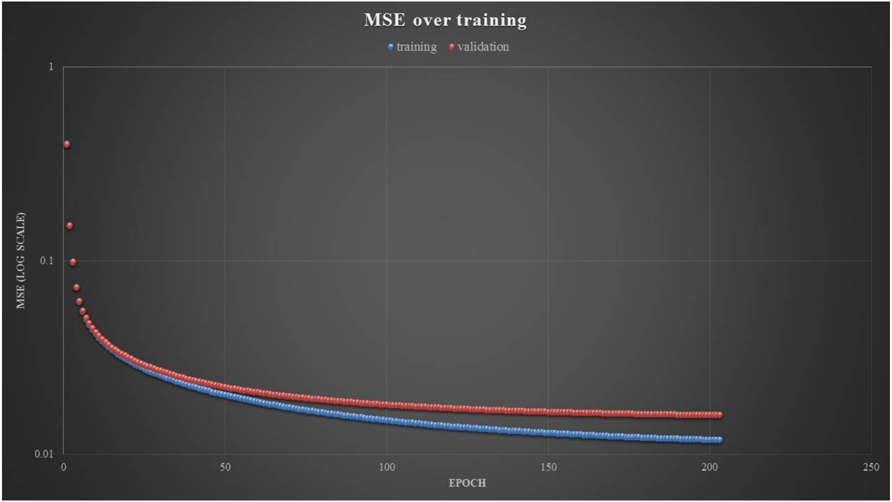


- Number of internal layers: 9
- Number of hidden units per layer: 250
- Batch size: 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 1 million random instances $U(0,1)$ i.i.d.
- Validation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

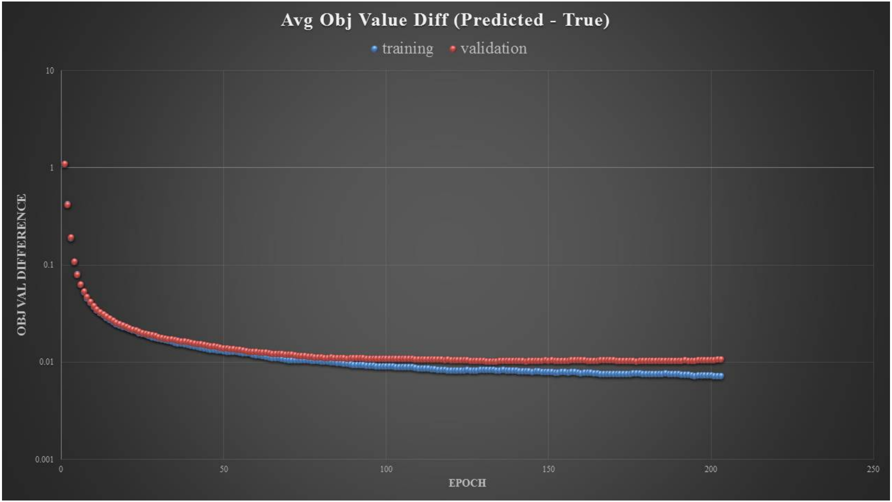
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



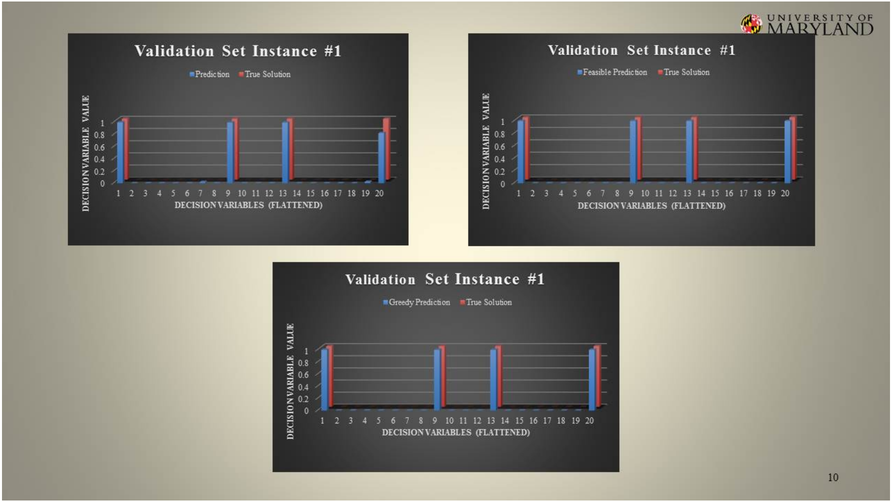
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



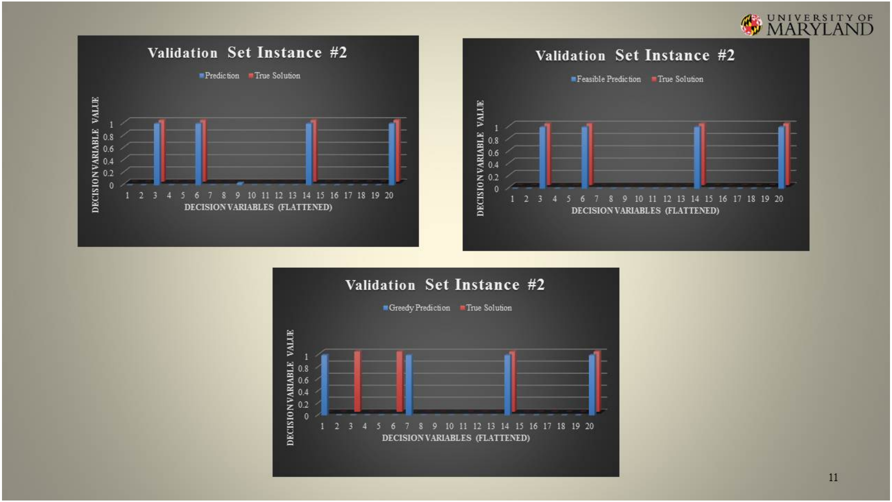
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



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

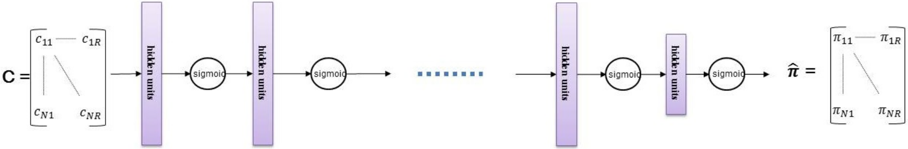


Exp.1 : $N = 5$, $R = 4$ with ReLU activation



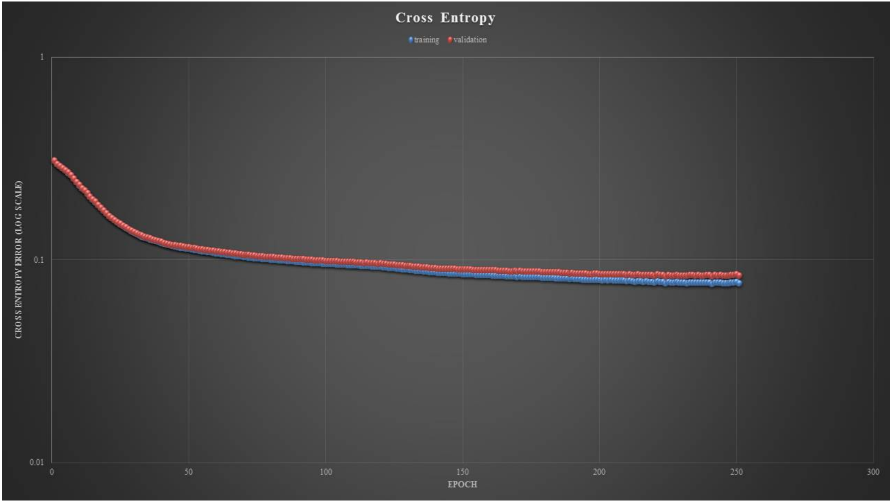
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation

Second architecture:

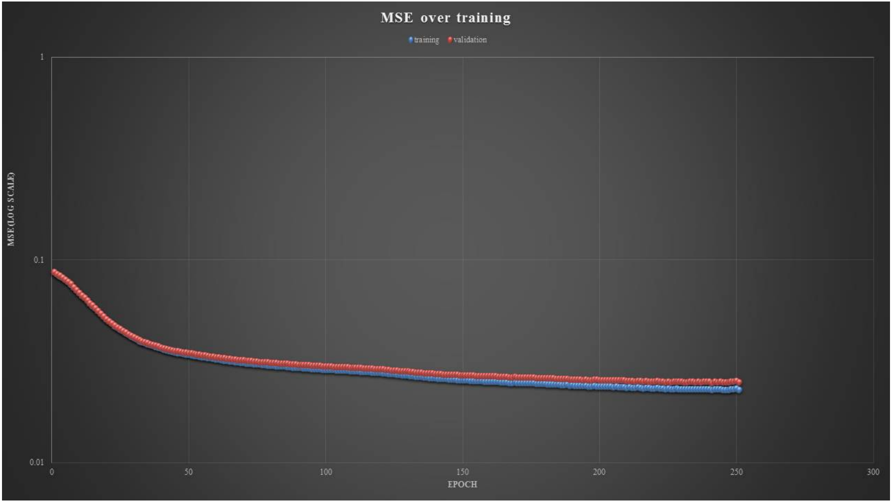


- Number of internal layers: 10
- Number of hidden units per layer: 250
- No Batch; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 1 million random instances $U(0, 1)$ i.i.d.
- Validation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

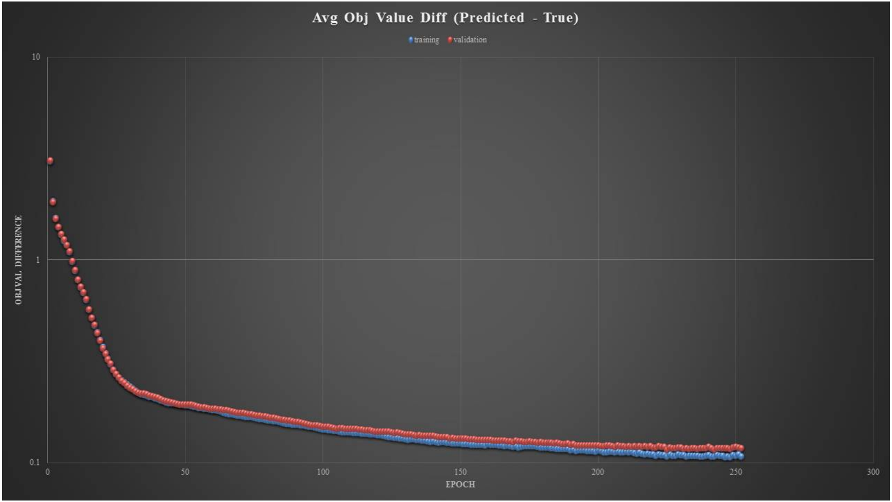
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



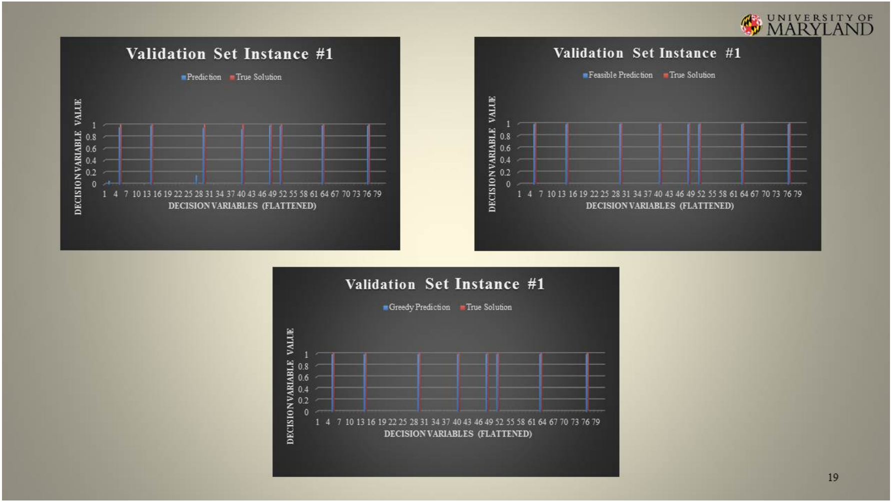
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



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



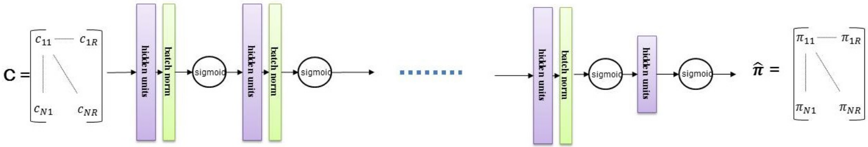
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



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

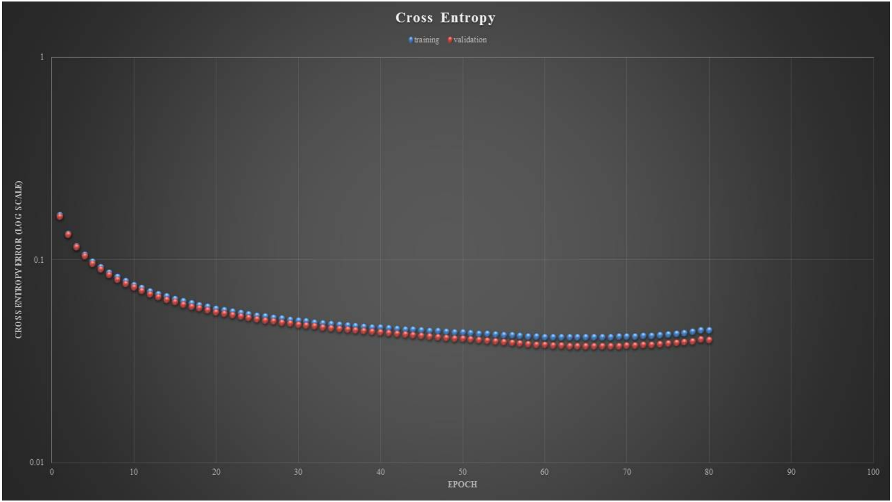
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation

Second architecture:

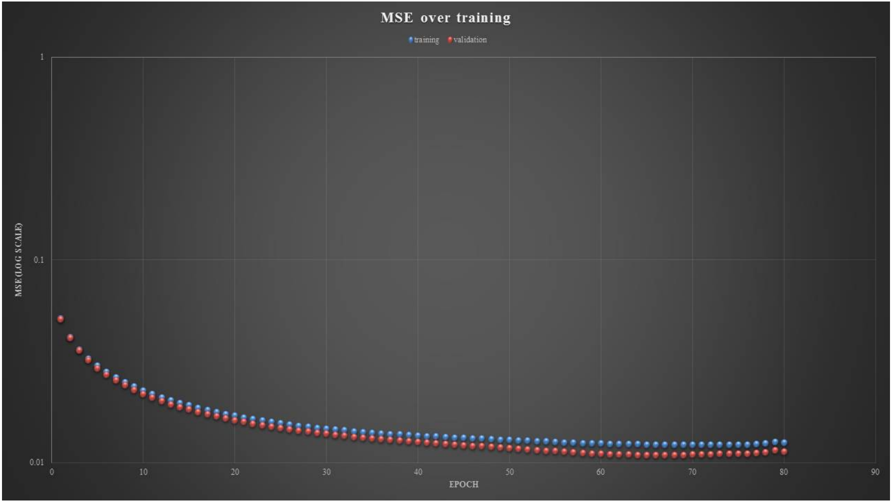


- Number of internal layers: 10
- Number of hidden units per layer: 250
- Batch size 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 500,000 random instances $U(0,1)$ i.i.d.
- Validation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

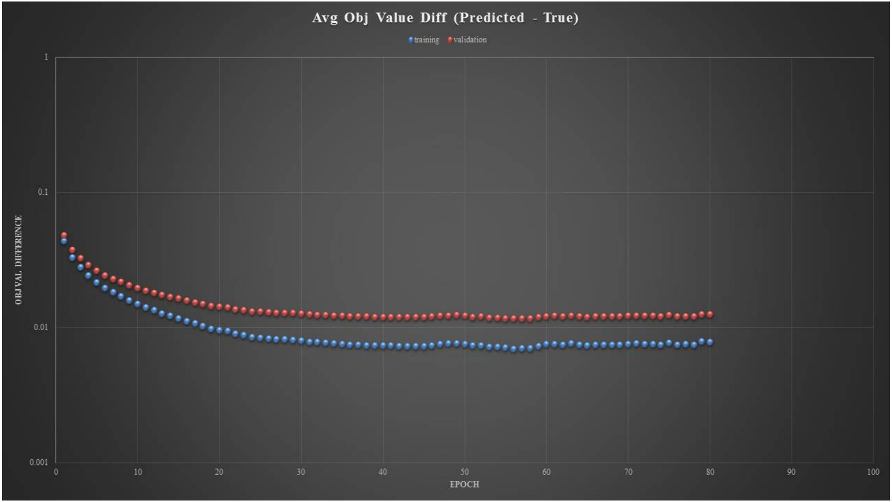
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



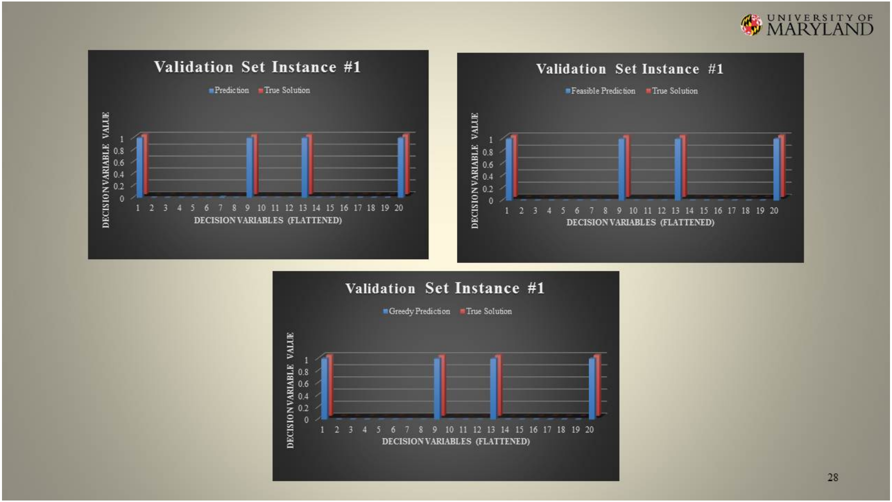
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



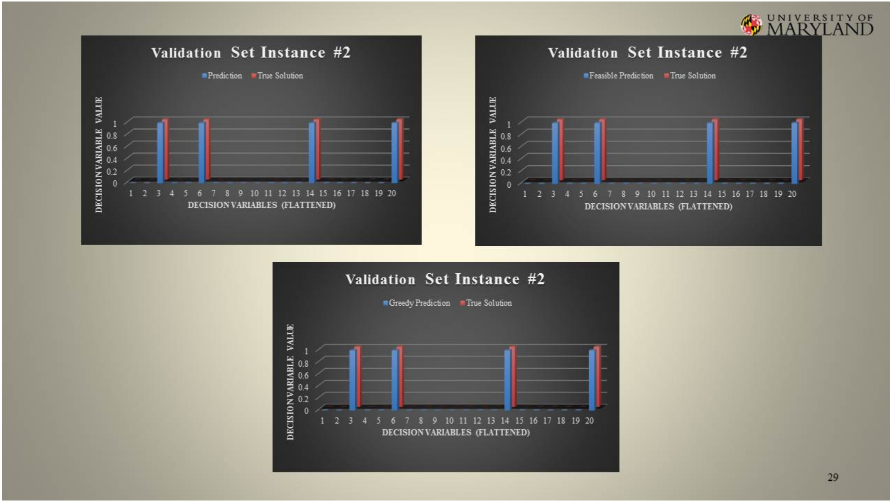
Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



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

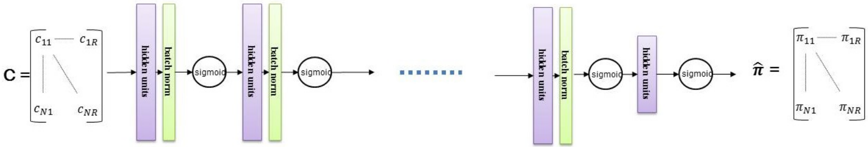


Exp.3 : $N = 5$, $R = 4$ with sigmoid activation



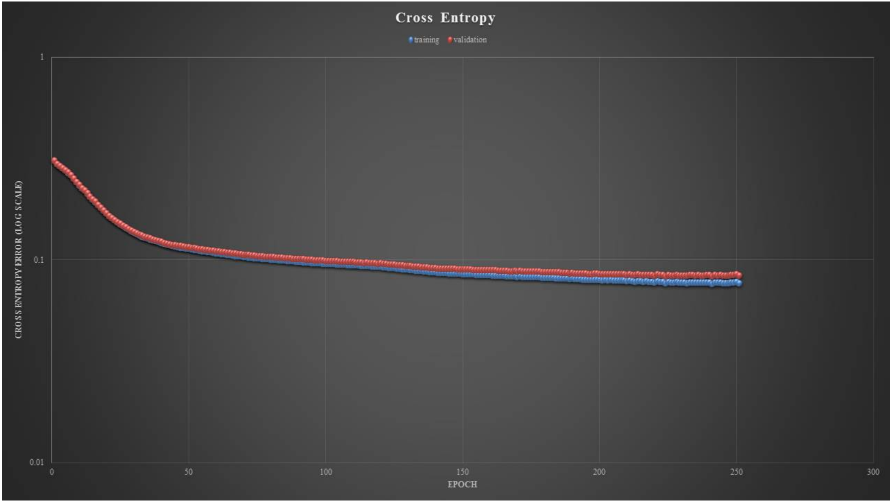
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation

Second architecture:

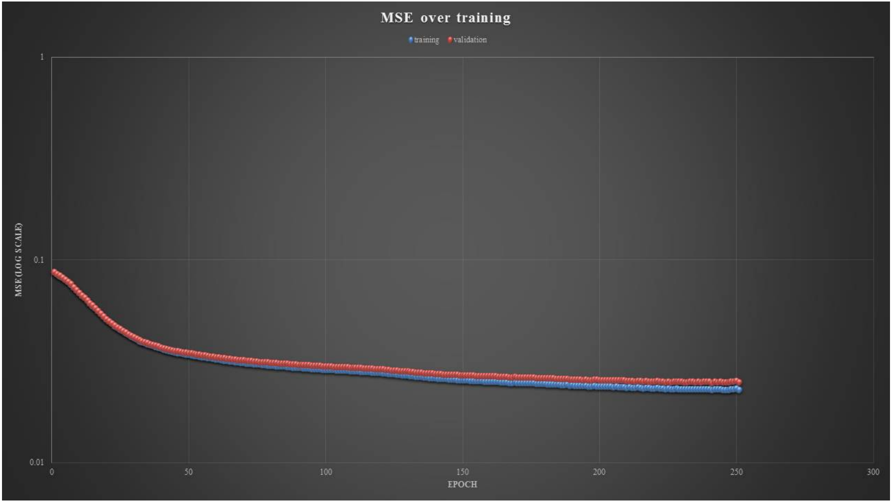


- Number of internal layers: 10
- Number of hidden units per layer: 300
- Batch size 200; ADAM optimizer
- Loss function: cross-entropy: $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 500,000 random instances $U(0,1)$ i.i.d.
- Validation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

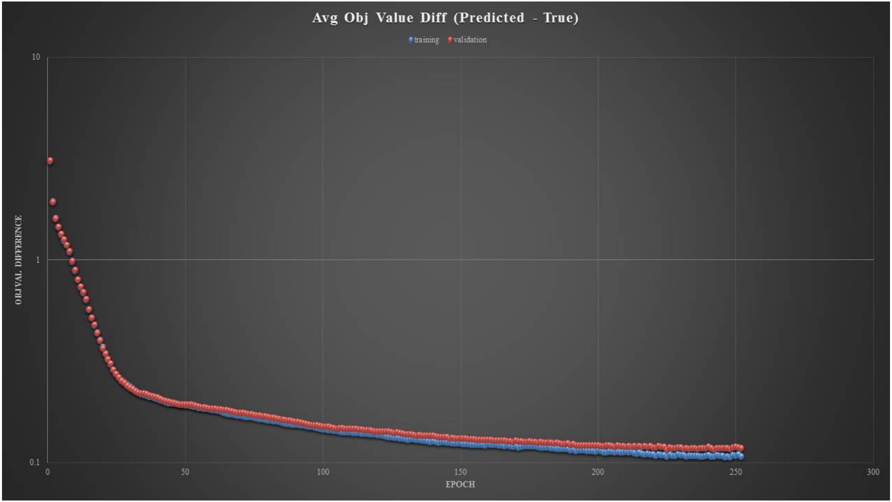
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



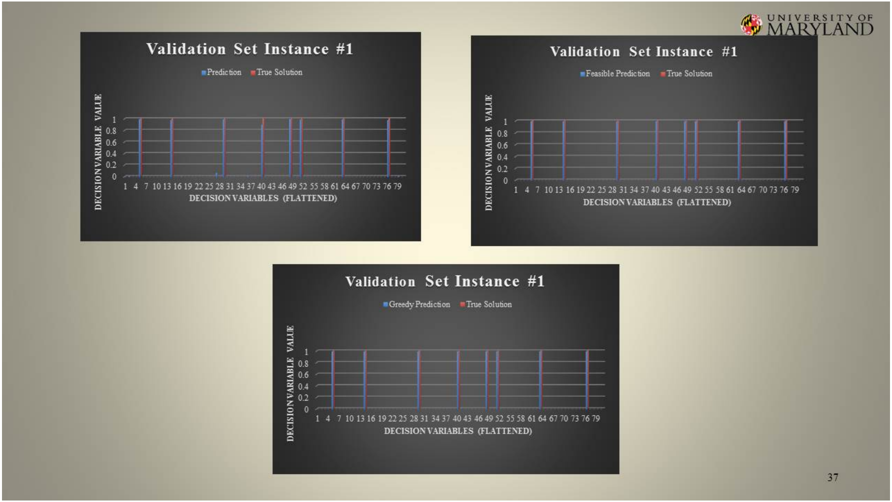
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



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, *Variational 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]