Optimizations using Deep Learning

Radu Balan, Naveed Haghani

Department of Mathematics, Applied Mathematics Applied Statistics and Scientific Computing
University of Maryland, College Park, MD

Joint work with **Maneesh Singh** (Verisk)

October 20, 2018
AMS Regional Meeting, University of Michigan, Ann Arbor, Michigan

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."

Table of Contents:

Optimization Problems

Consider a general optimization problem:

$$\begin{aligned} &\textit{minimize} && J(x;y) \\ &\textit{subject to}: && \\ &x \in D && \\ &g(x;y) \leq 0 && \\ &h(x;y) = 0 && \end{aligned}$$

where $y \in E \subset \mathbb{R}^d$ denotes a set of parameters (or input) into the optimization problem, D denotes the allowable domain for the unknown variable x, $g, h : D \times E \to \mathbb{R}$ (or $\bar{R}$) define the constraints (D can be define implicit by an indicator function constraint), and $J(x;y)$ is the objective function.

Purpose of this talk: How can deep learning solve optimization problems?

Optimization Problems
General Approaches

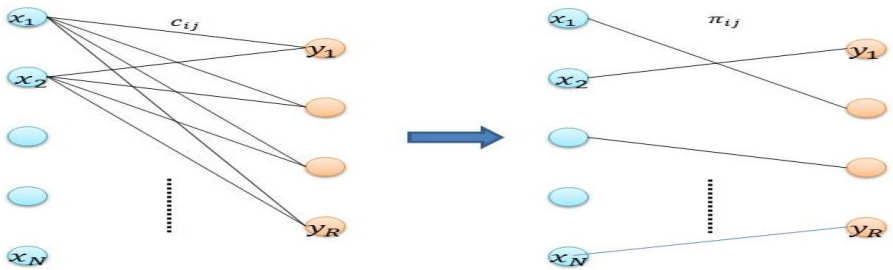
We plan to discuss three approaches to an optimization problem:

1. Deep Neural Network (DNN) as a Universal Approximator (UA)
2. Neural Network as an auxiliary function of an iterative algorithm
3. Deep Neural Network as a representation tool

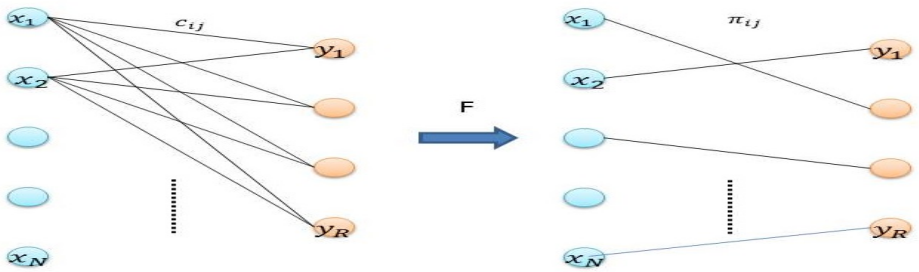
The Specific Optimization Problem

Consider a $N \times R$ cost 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 matching/assignment, namely:

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



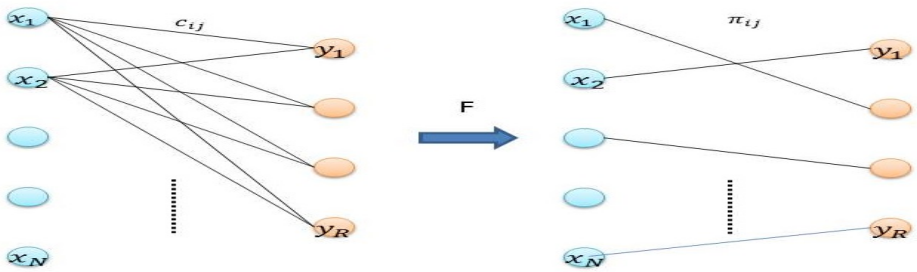
First Approach: DNN as a Universal Approximator



The optimal solution of this (or any optimization problem) is a nonlinear map $\pi = F(C)$.

Idea: Generate optimal pairs $\{(C, \pi)\}$ and then train a Neural Network to reproduce these values.

First Approach: DNN as a Universal Approximator



The optimal solution of this (or any optimization problem) is a nonlinear map $\pi = F(C)$.

Idea: Generate optimal pairs $\{(C, \pi)\}$ and then train a Neural Network to reproduce these values.

Issues to be discussed:

1. How to generate optimal pairs?
2. Network architecture: how many layers? how many nodes in each layer?
3. How to enforce feasibility?
4. Primal problem – what about the dual problem?

Second Approach: Emulate an Iterative Descent Algorithm

For the optimization problem $min_x\ J(x)$ use an iterative algorithm:

$$x^{(t+1)} = x^{(t)} - \alpha_t p^{(t)}$$

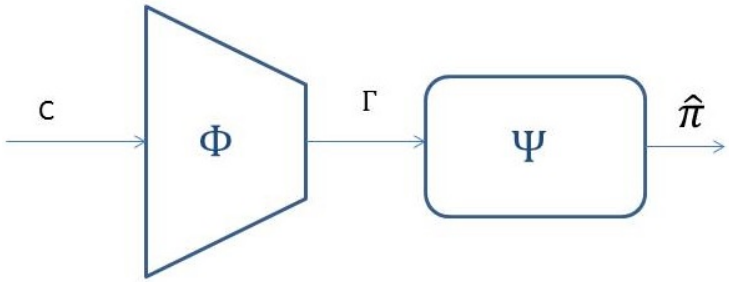
where $x^{(t)}$ is the current estimate, α_t is the step size at step t, and $p^{(t)}$ is the marching direction. Possible choices:

- Gradient Descent: $p_g^{(t)} = \nabla J(x^{(t)})$
- Newton method: $p_n^{(t)} = Hess^{(-1)} J(x^{(t)}) \nabla J(x^{(t)})$
- Mixed methods: $p^{(t)} = a_t p_g + b_t p_n + c_t p^{(t-1)}$

Idea: Implement a descent strategy, but adapt step size and learning rates using Neural Networks – Recurrent Neural Network.
See also: Google DeepMind group [Andrychowicz et.al. '16];

Third Approach: Optimization in a Representation Space

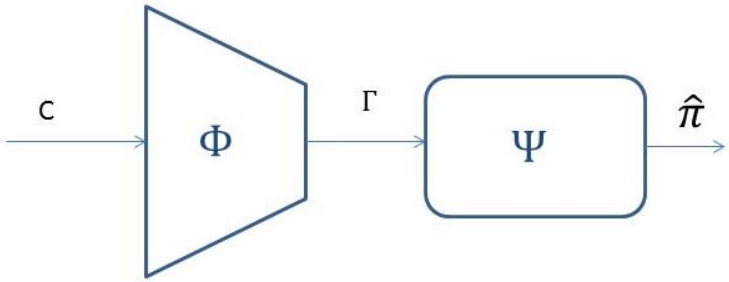
Idea: Perform a two-step procedure: (1) perform a nonlinear representation of the input data; (2) perform optimization in the representation space.



The nonlinear representation map $\Phi : C \mapsto \Gamma$ can be learned using a Variational Auto-Encoder method.
The Optimization map $\Psi : \Gamma \mapsto \hat{\pi}$ can be implemented using a neural-network such as in the first approach.

Third Approach: Optimization in a Representation Space

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 $\Phi : C \mapsto \Gamma$ can be learned using a Variational Auto-Encoder method.
The Optimization map $\Psi : \Gamma \mapsto \hat{\pi}$ can be implemented using a neural-network such as in the first approach.
Why it makes sense? feasible solutions are graph representable – Use Graph Convolutive Networks (GCN – KipfWeiling) as a launching pad.
See also: [Nowak et.al.'18]

First Approach: DNN as a UA

Exact Solutions to Our Problem

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

Luckily, this problem admits an exact convex relaxation: the associated Linear Program produces the same optimal solution:

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

First Approach: DNN as a UA

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 $\hat{\pi}$:

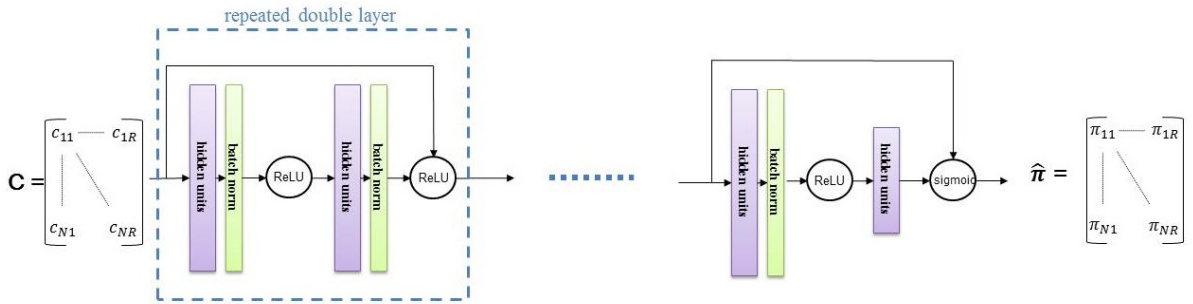
Issues to discuss:

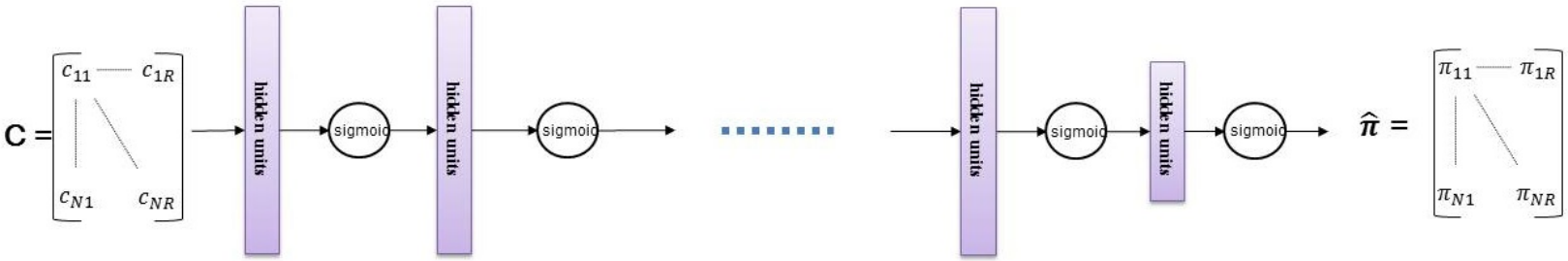
1. DNN architecture: how many layers; how many neurons per layer?
2. P, the feasibility enforcer

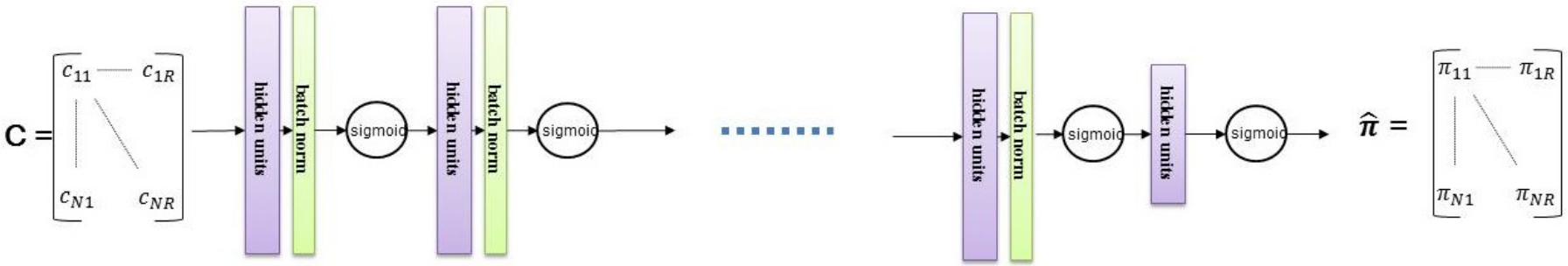
First Approach: DNN as a UA

DNNs

We study three architectures:





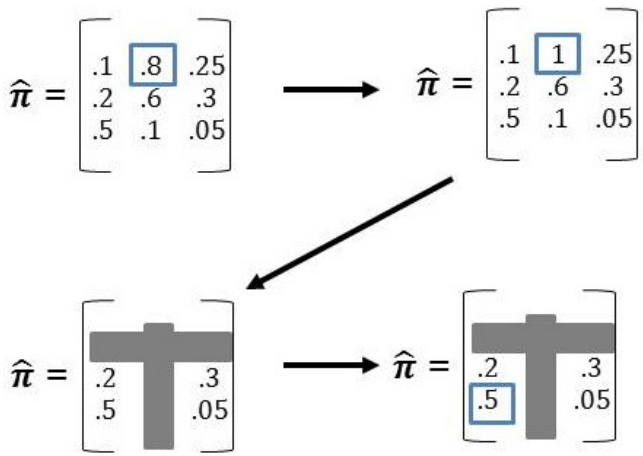


First Approach: DNN as a UA

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 $\hat{\pi}$.



First Approach: DNN as a UA

Baseline solution: The Greedy Algorithm

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

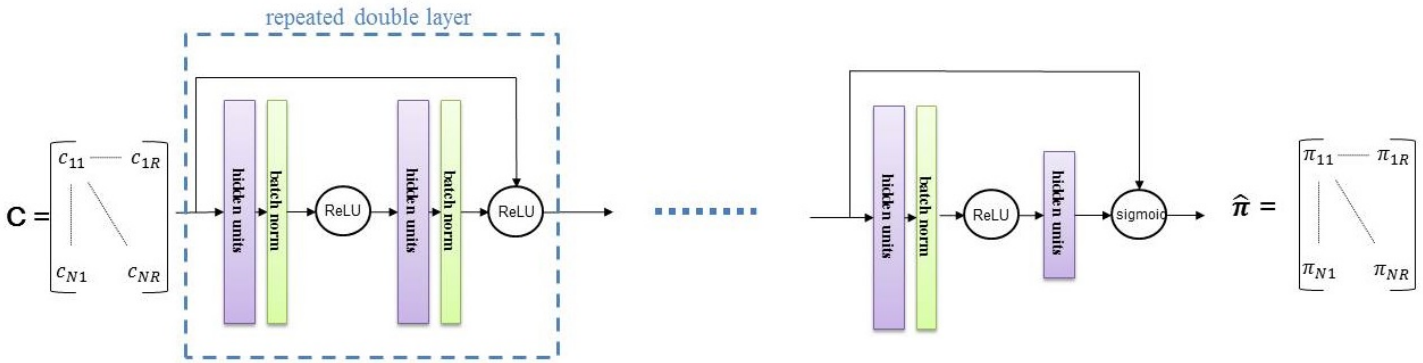
1. Initialize $E = C$ and $\hat{\pi} = 0_{N \times R}$
2. Repeat R times:
   - Find $(i, j) = argmin_{(a,b)} E_{a,b}$;
   - Set $\hat{\pi}_{i,j} = 1$, $\hat{\pi}_{i,l} = 0 \ \forall l \neq j$, $\hat{\pi}_{l,j} = 0 \ \forall l \neq i$;
   - Set $E_{i,:} = \infty$, $E_{:,j} = \infty$.

Proposition

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

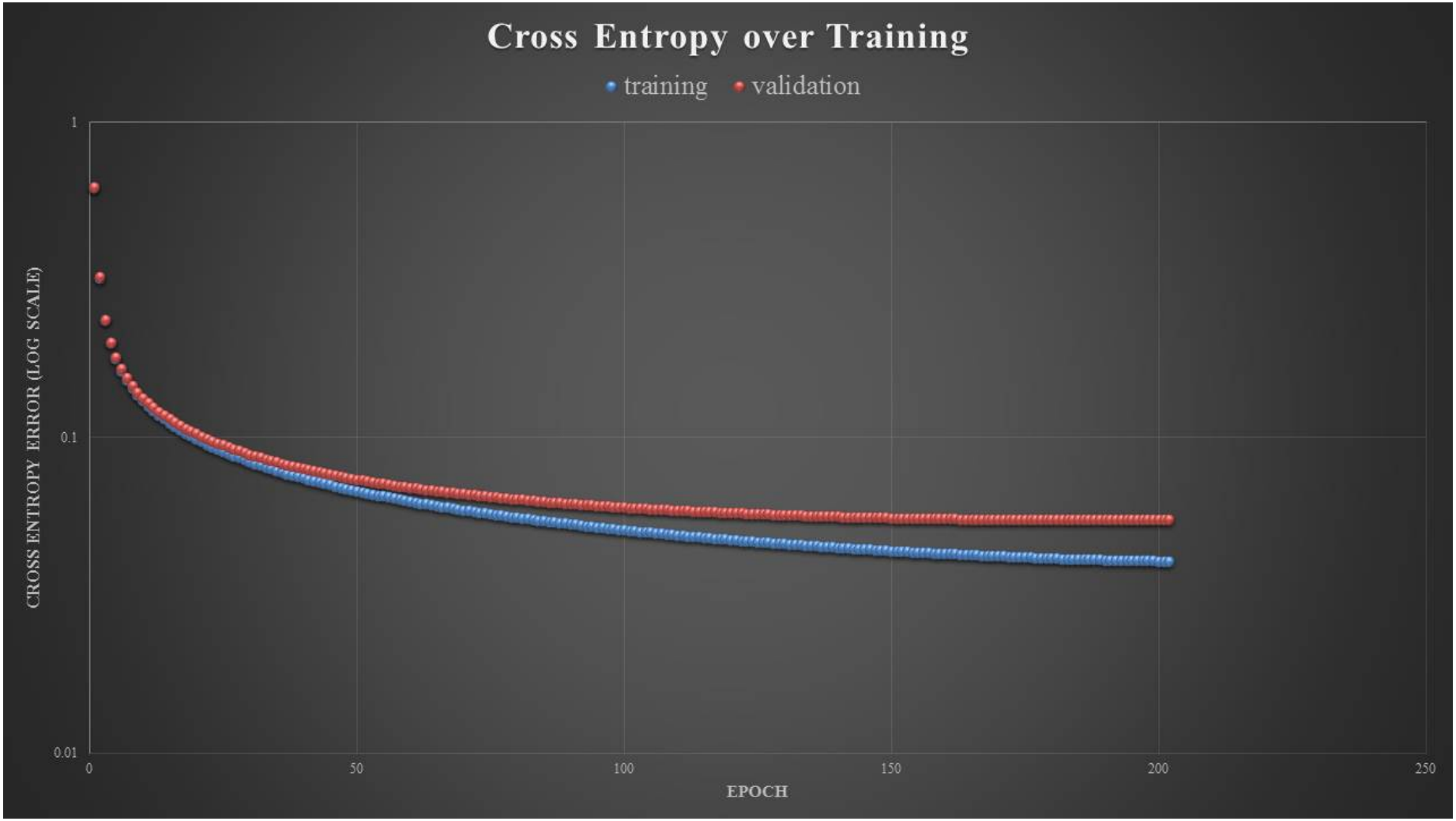
Exp.1 : $N = 5$, $R = 4$ with ReLU activation

First architecture:

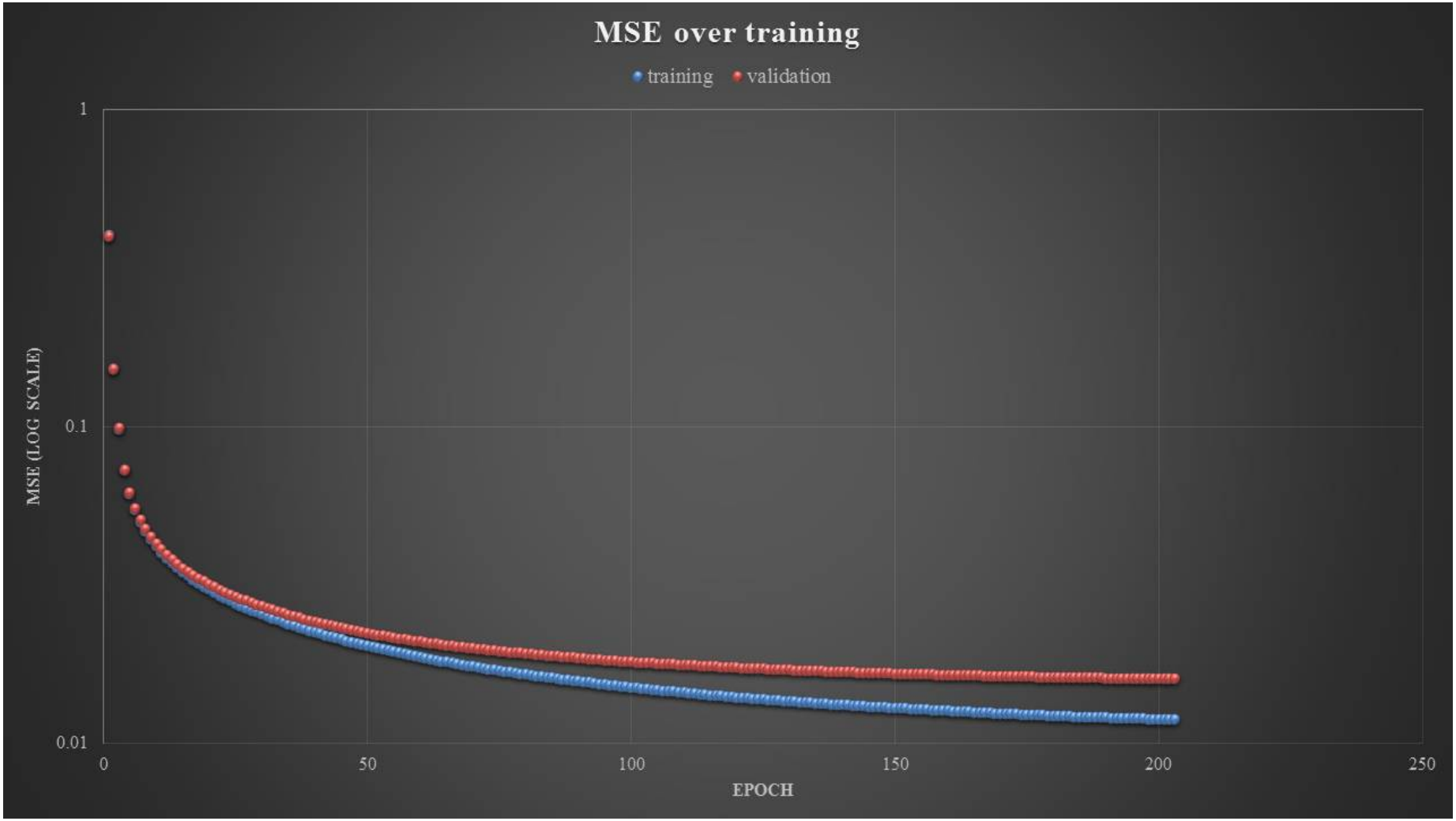


- Number of internal layers: 9
- Number of hidden units per layer: 250
- Batch size: 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 1 million random instances $U(0, 1)$ i.i.d.
- Validation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

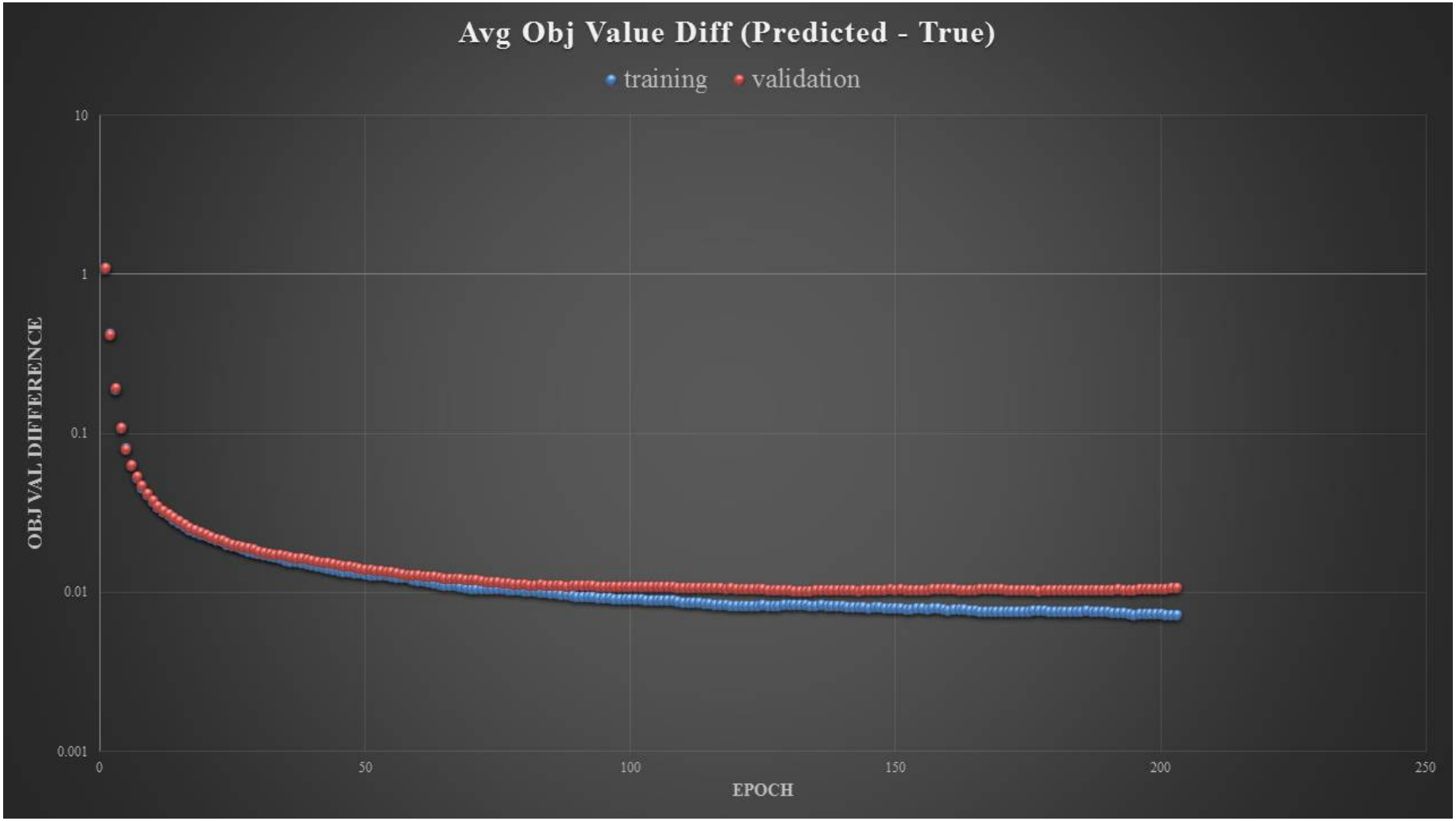
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



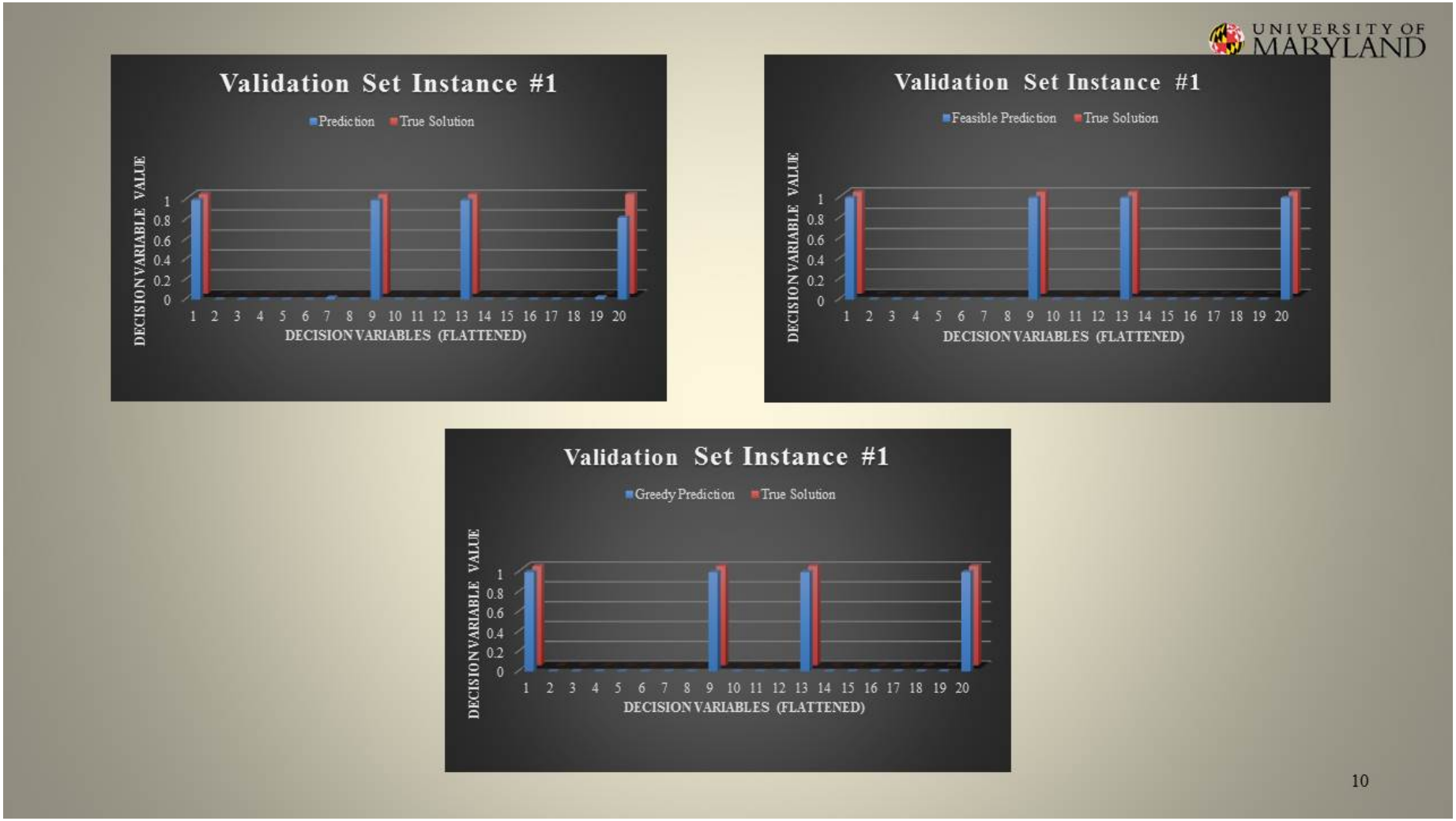
Exp.1 : $N = 5$, $R = 4$ with ReLU activation



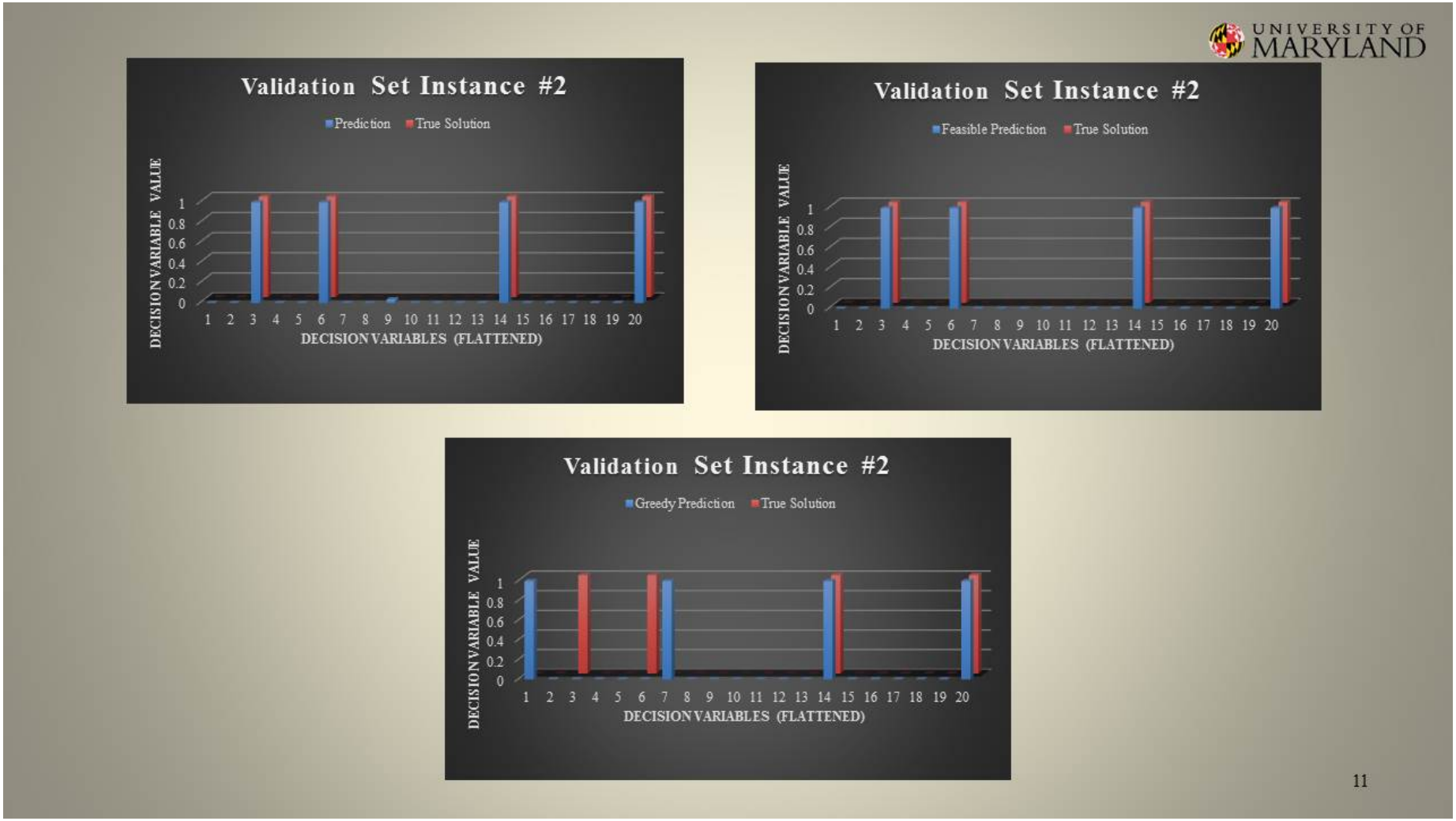
Exp.1 : $N = 5,\ R = 4$ with ReLU activation



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

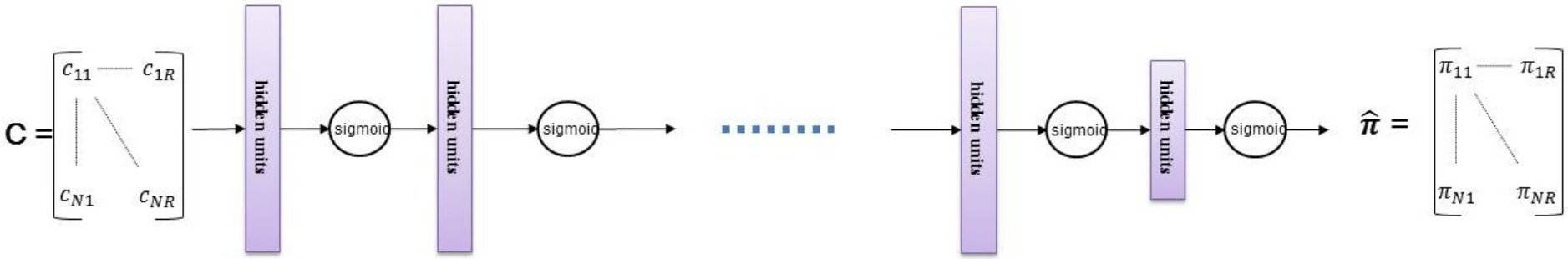


Exp.1 : $N = 5,\ R = 4$ with ReLU activation



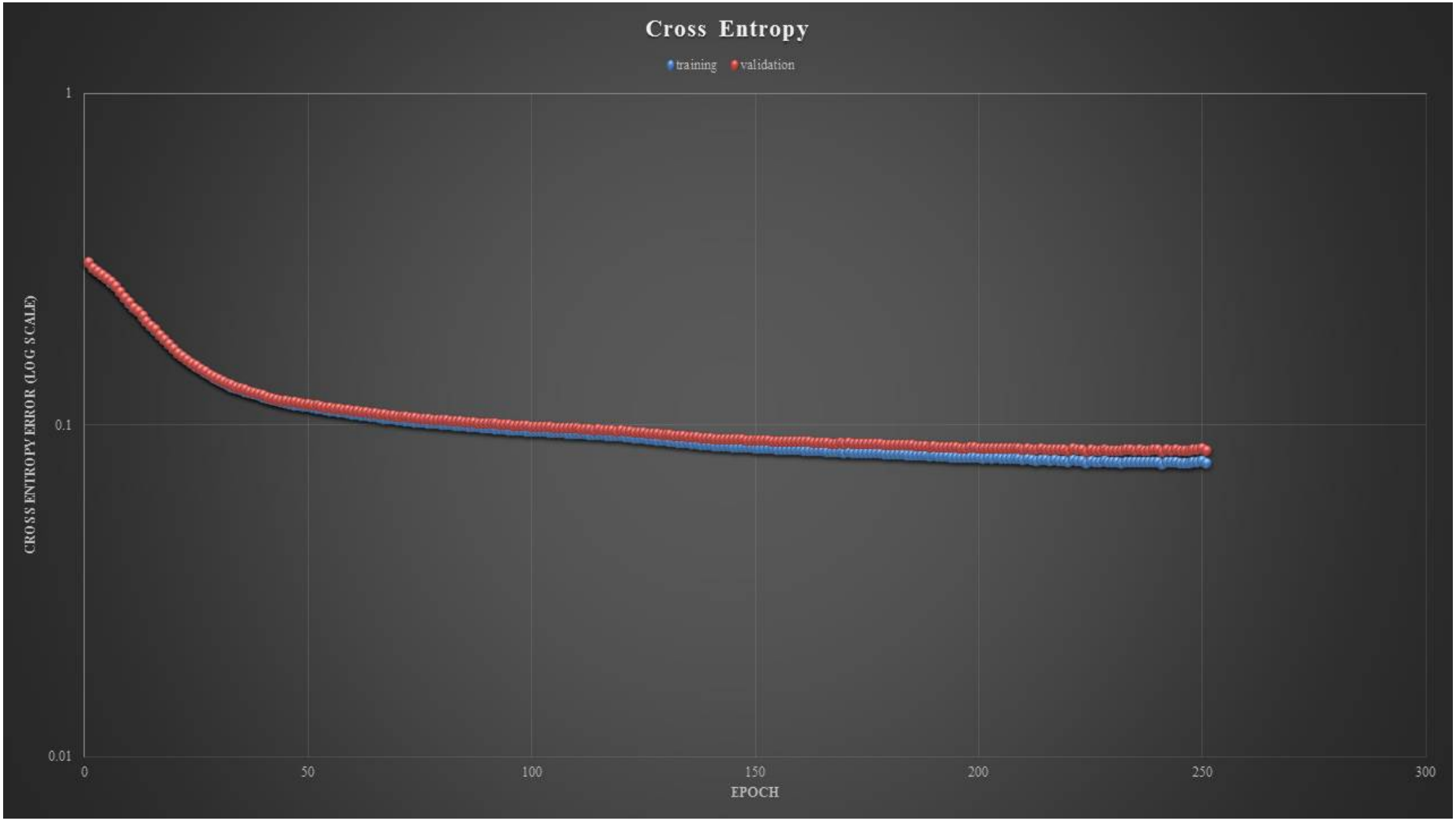
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation

Second architecture:

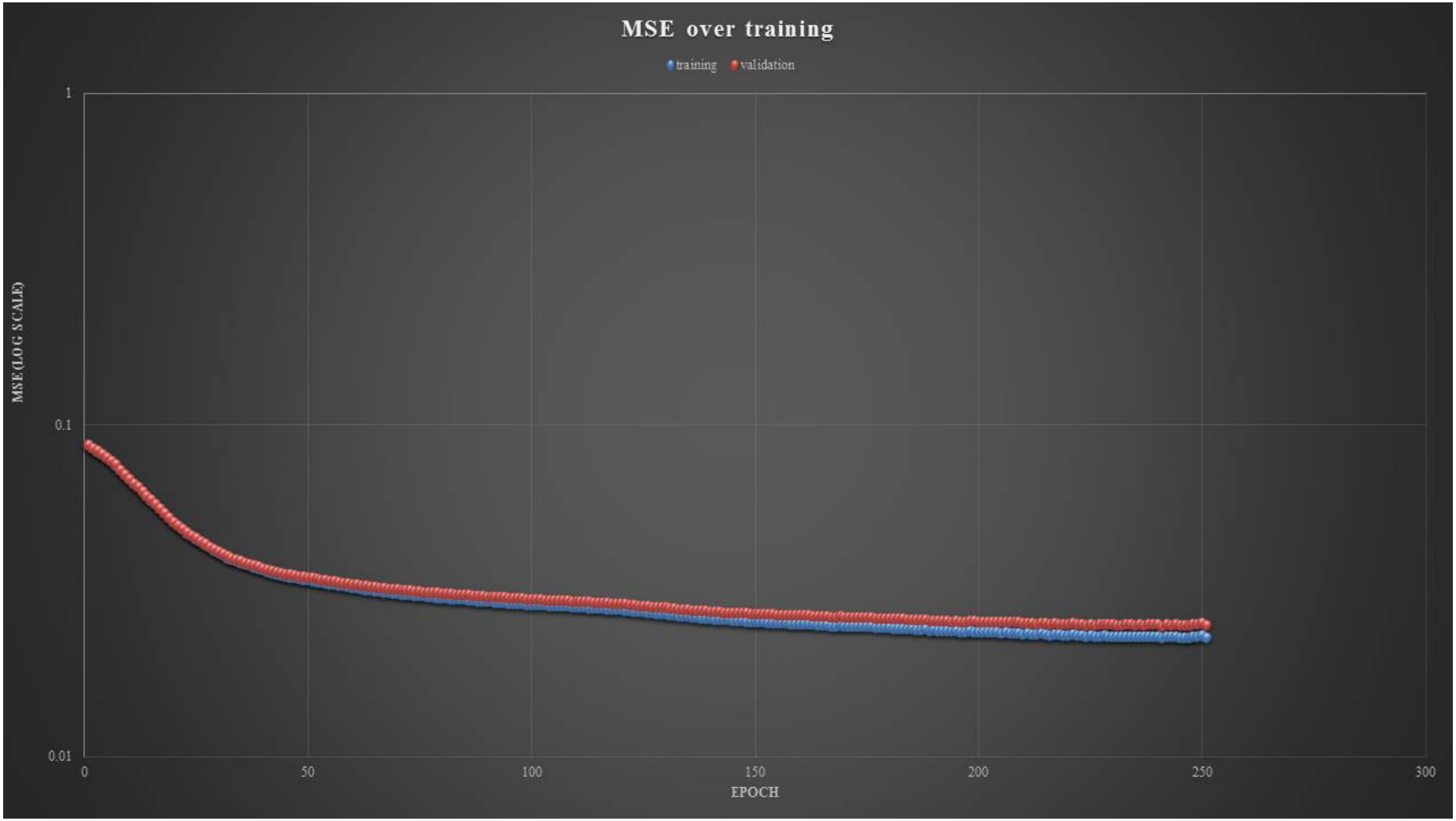


- Number of internal layers: 10
- Number of hidden units per layer: 250
- No Batch; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 1 million random instances $U(0, 1)$ i.i.d.
- Validation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

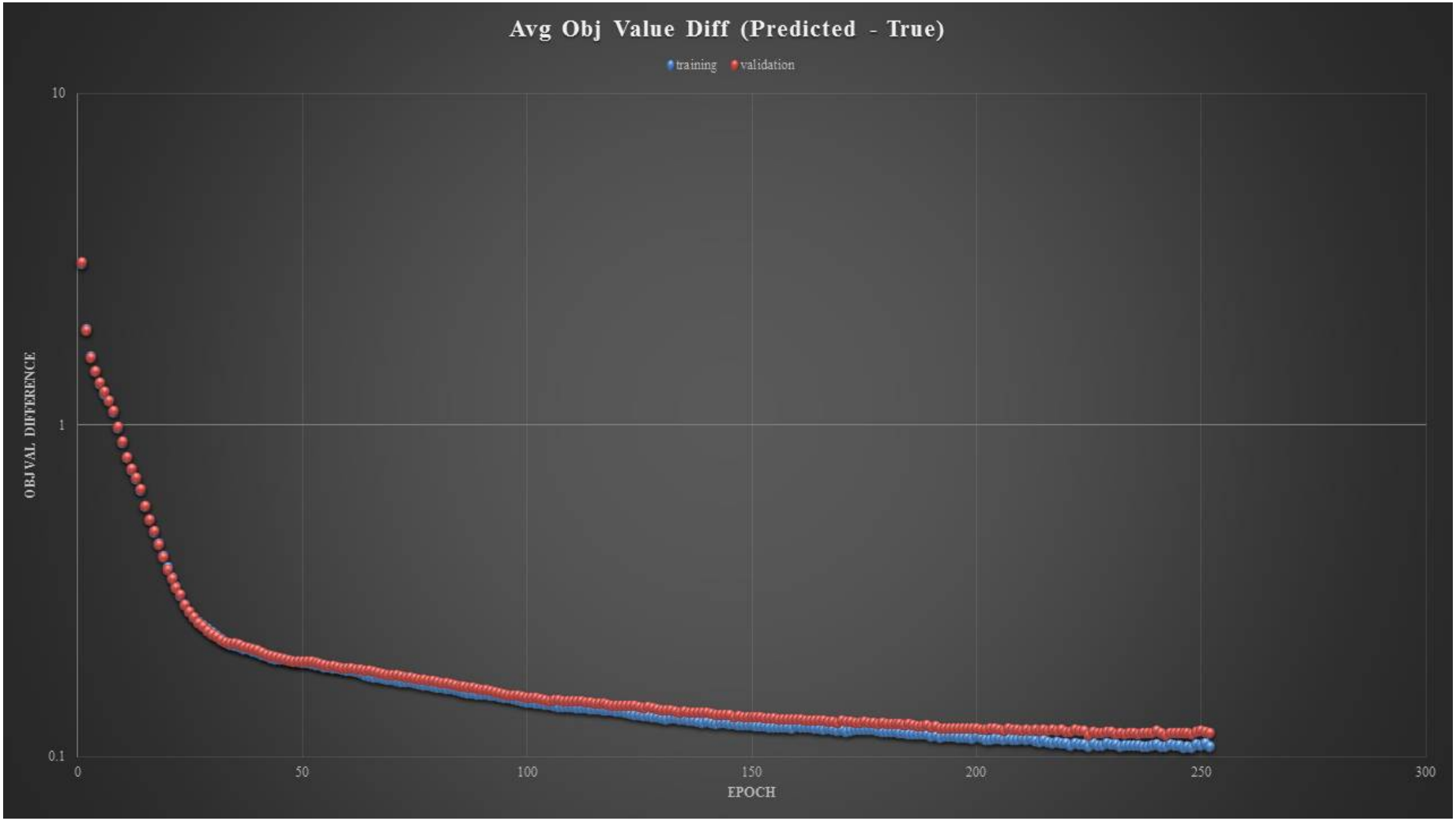
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



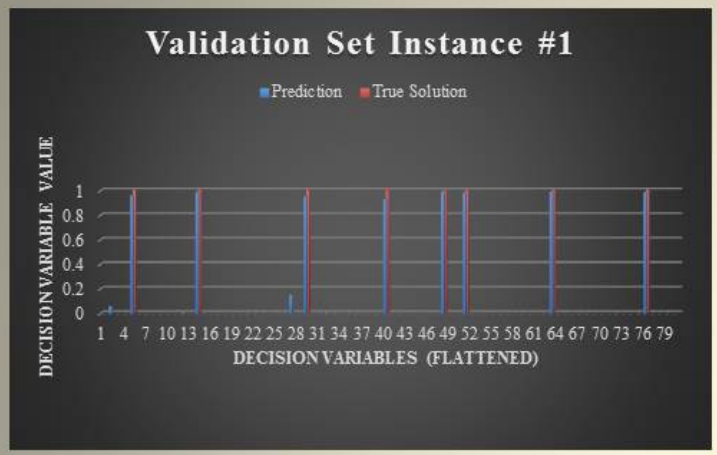
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation

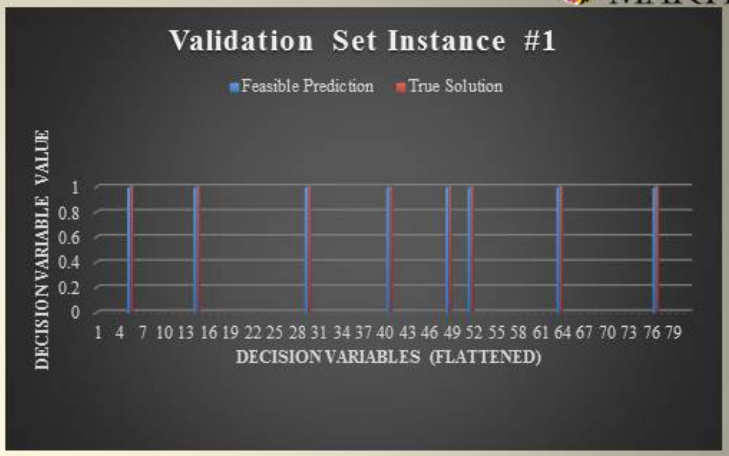


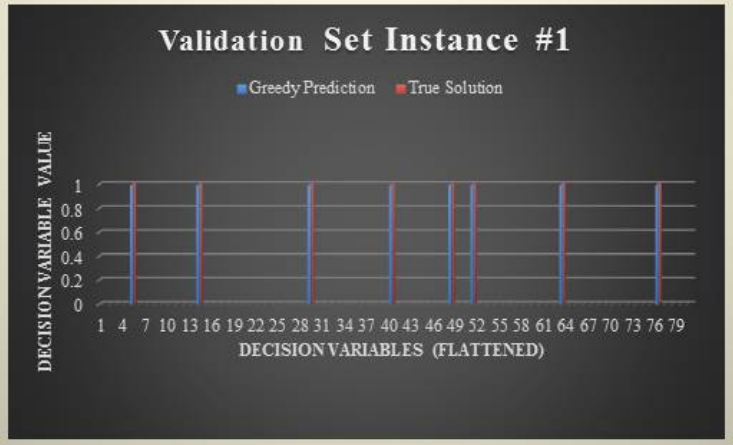
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



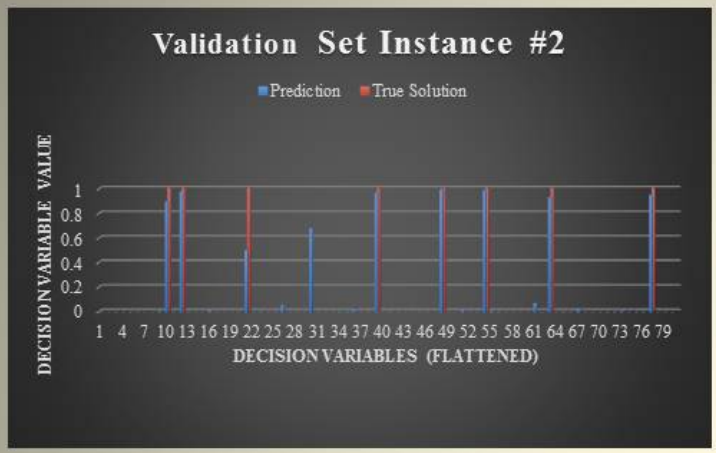
Exp.2 : $N = 10$, $R = 8$ with sigmoid activation

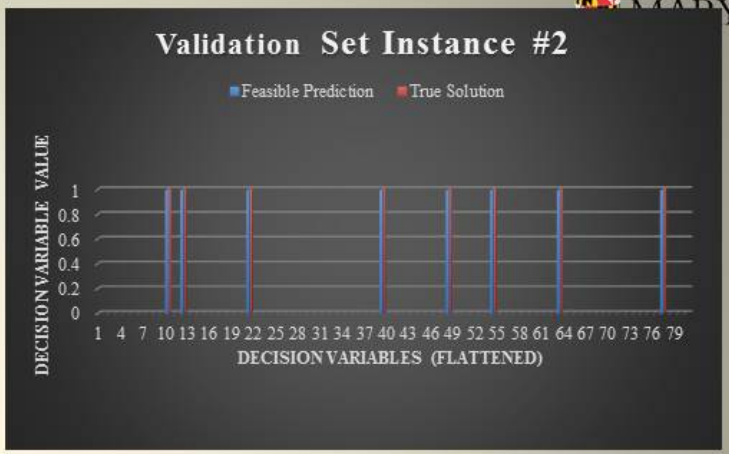


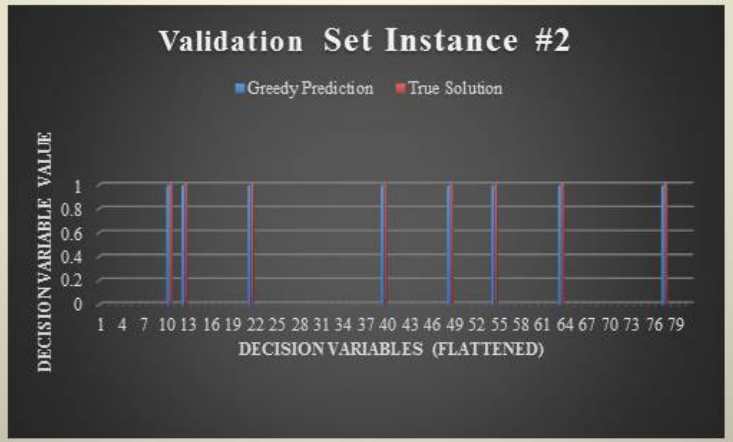




Exp.2 : $N = 10$, $R = 8$ with sigmoid activation



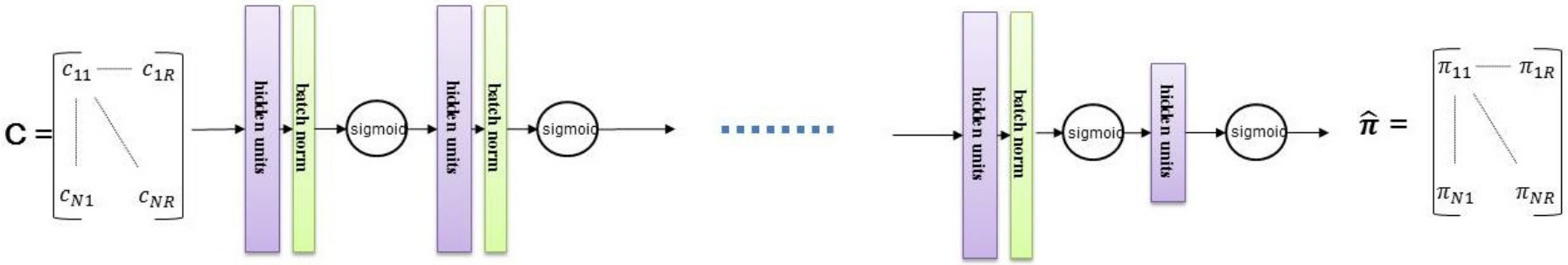




20

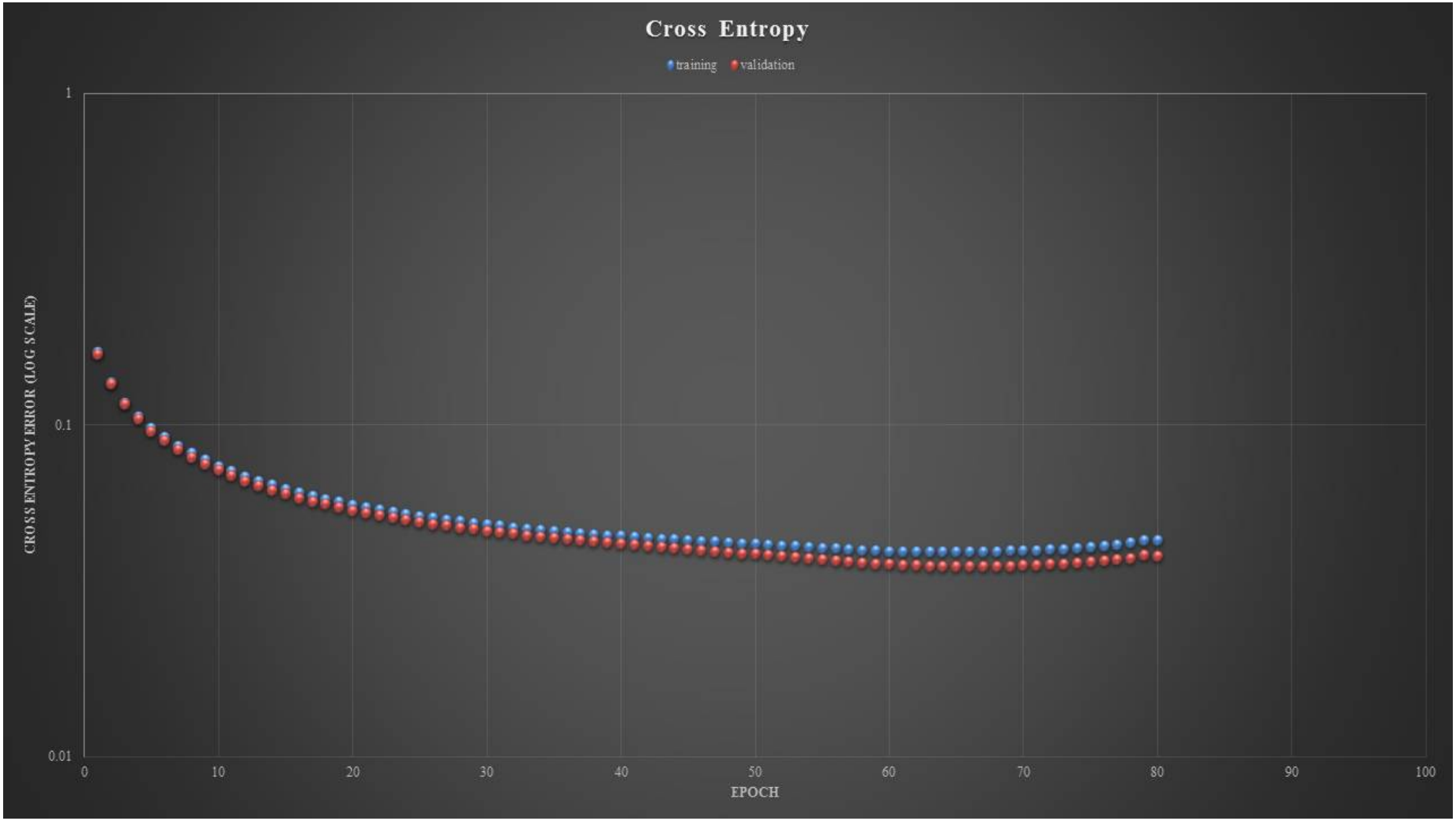
Exp.3 : $N = 5,\ R = 4$ with sigmoid activation

Second architecture:

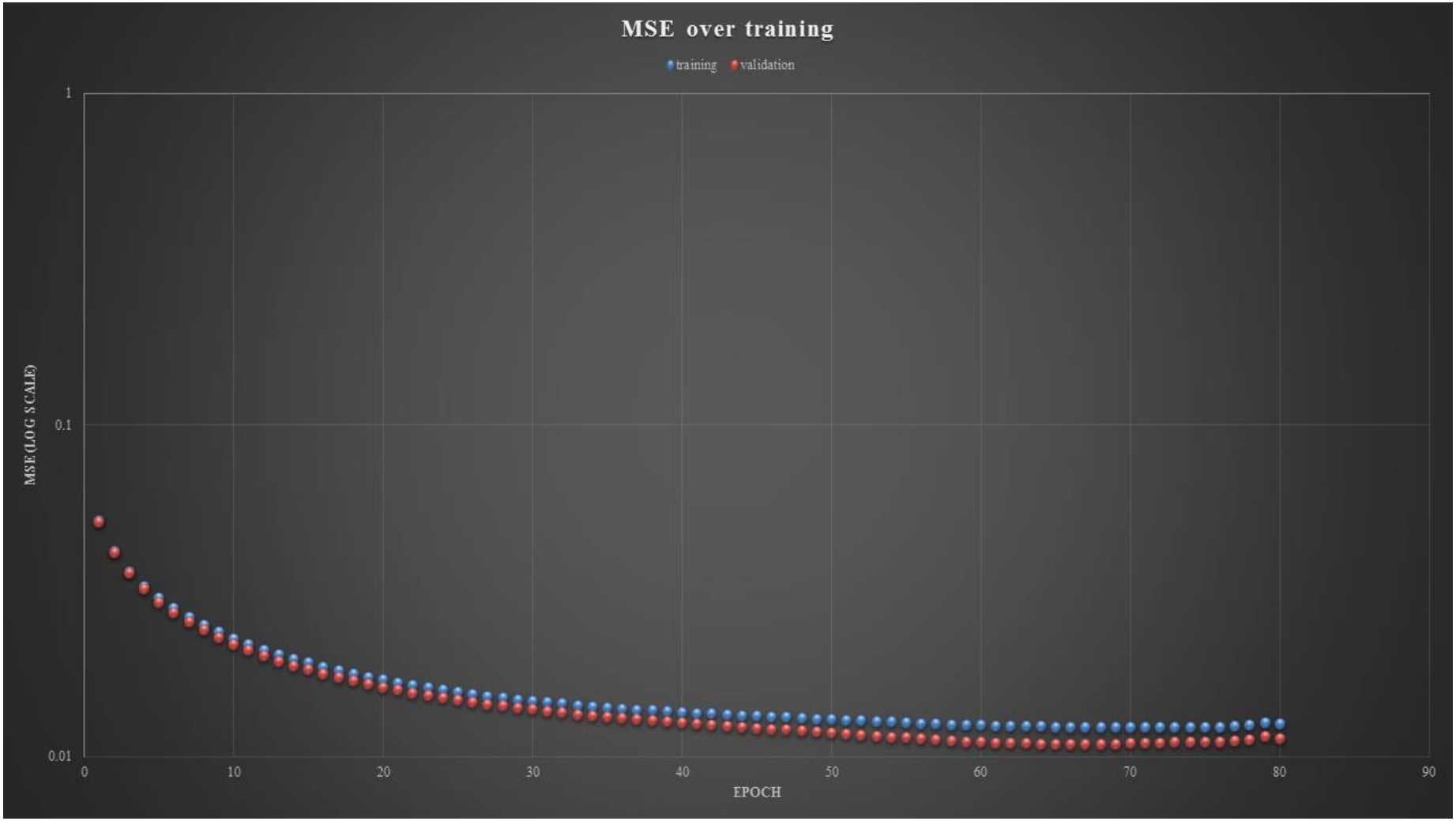


- Number of internal layers: 10
- Number of hidden units per layer: 250
- Batch size 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 500,000 random instances $U(0, 1)$ i.i.d.
- Validation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

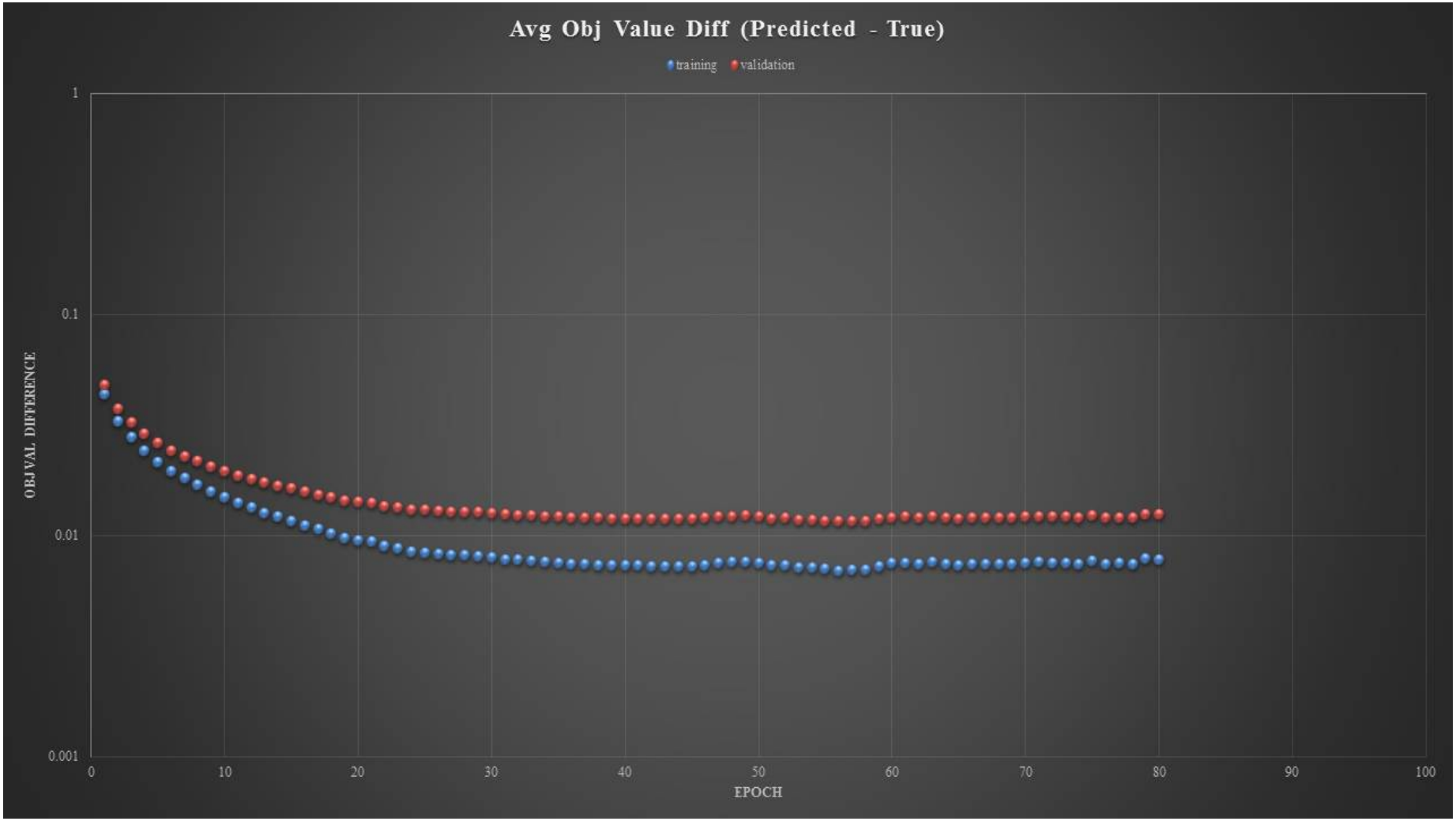
Exp.3 : $N = 5,\ R = 4$ with sigmoid activation



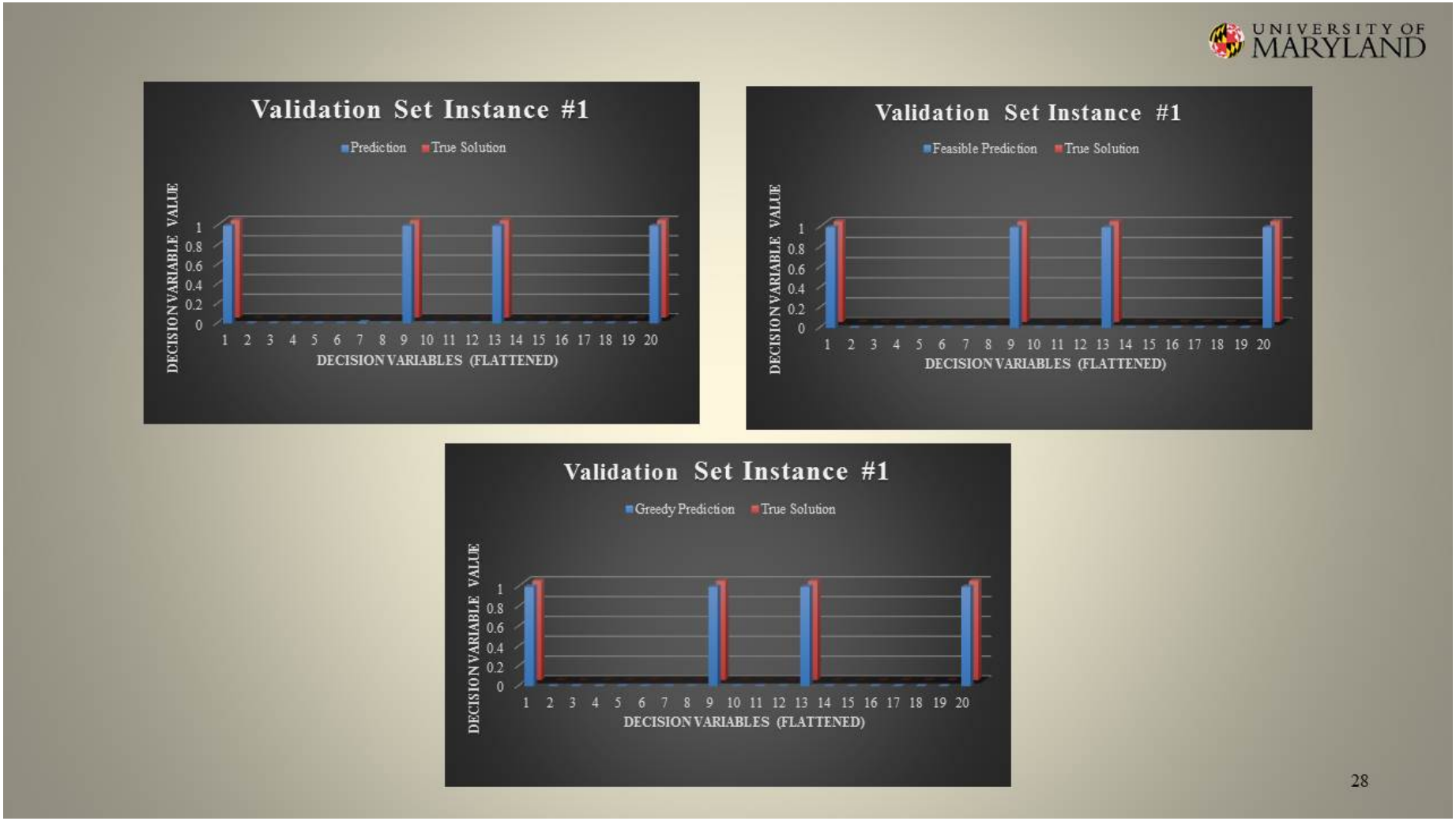
Exp.3 : $N = 5,\ R = 4$ with sigmoid activation



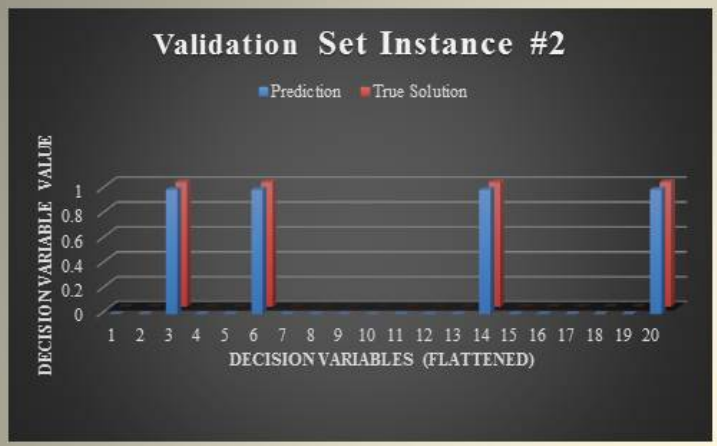
Exp.3 : $N = 5,\ R = 4$ with sigmoid activation

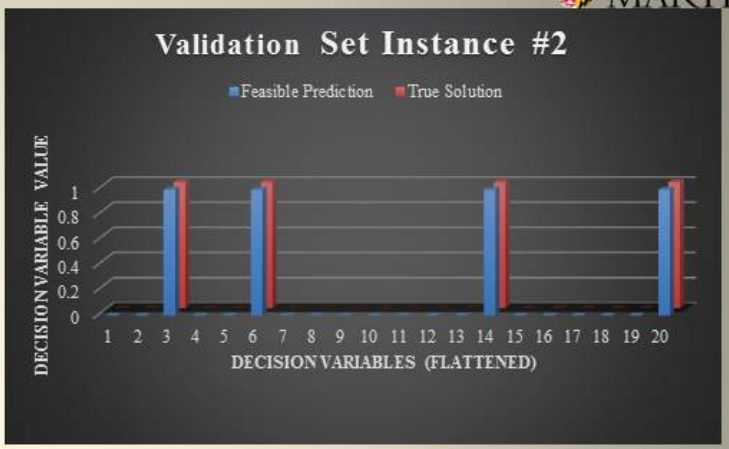


Exp.3 : $N = 5,\ R = 4$ with sigmoid activation



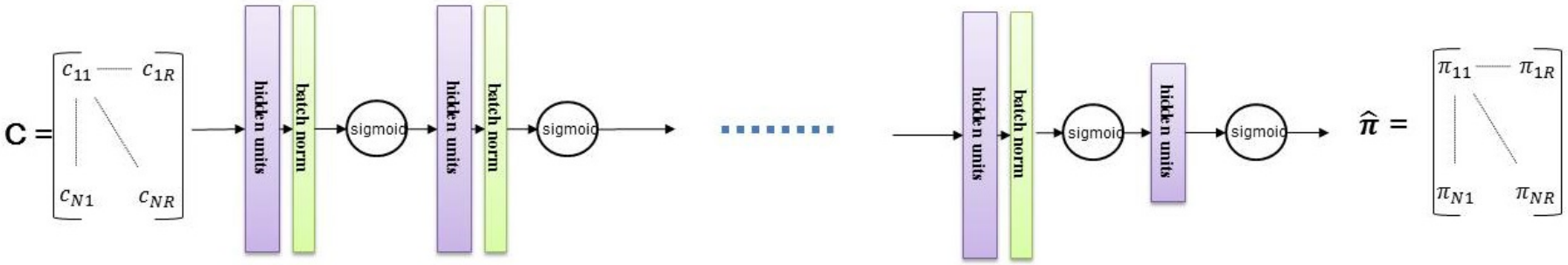
Exp.3 : $N = 5,\ R = 4$ with sigmoid activation





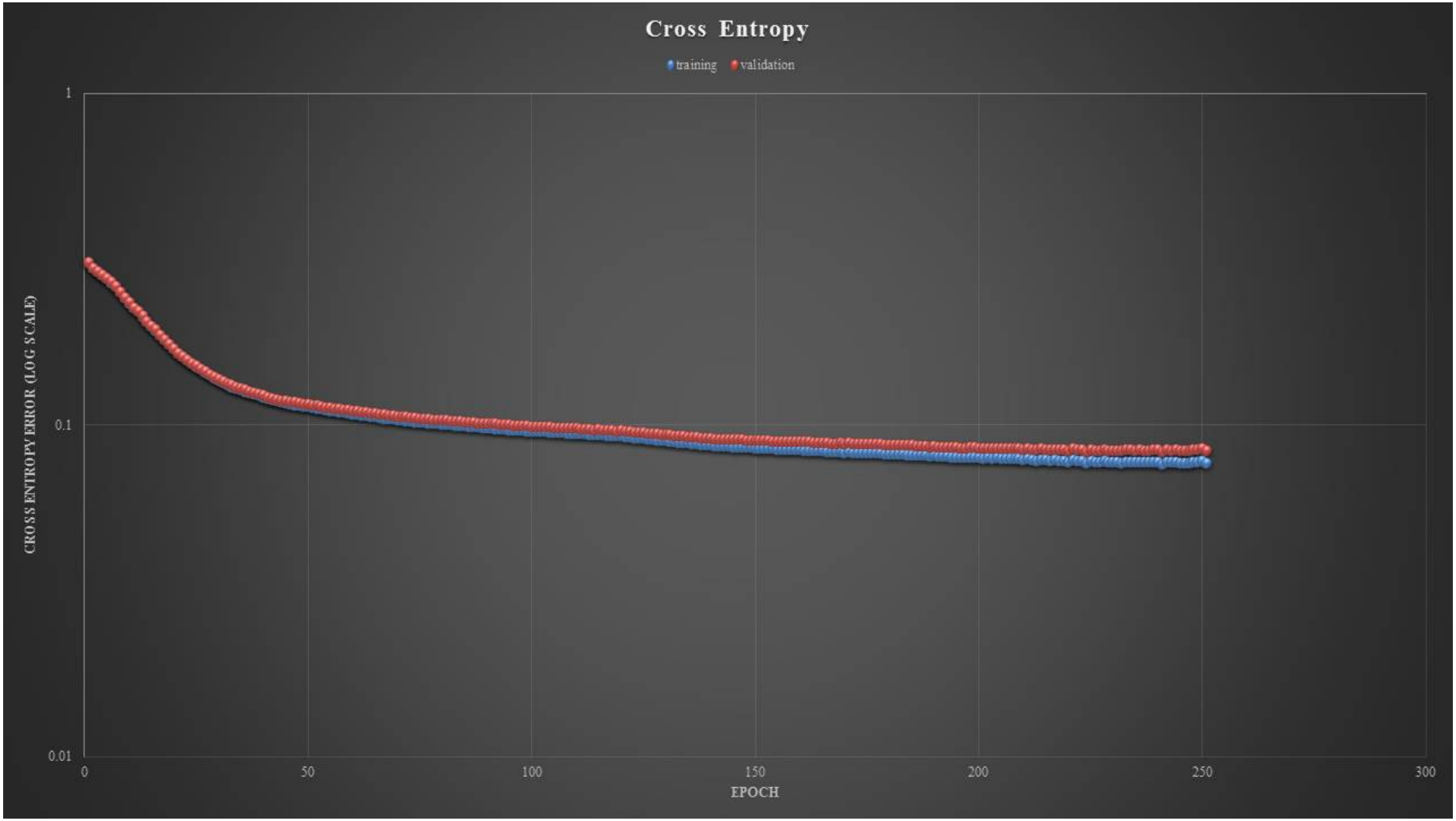
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation

Second architecture:

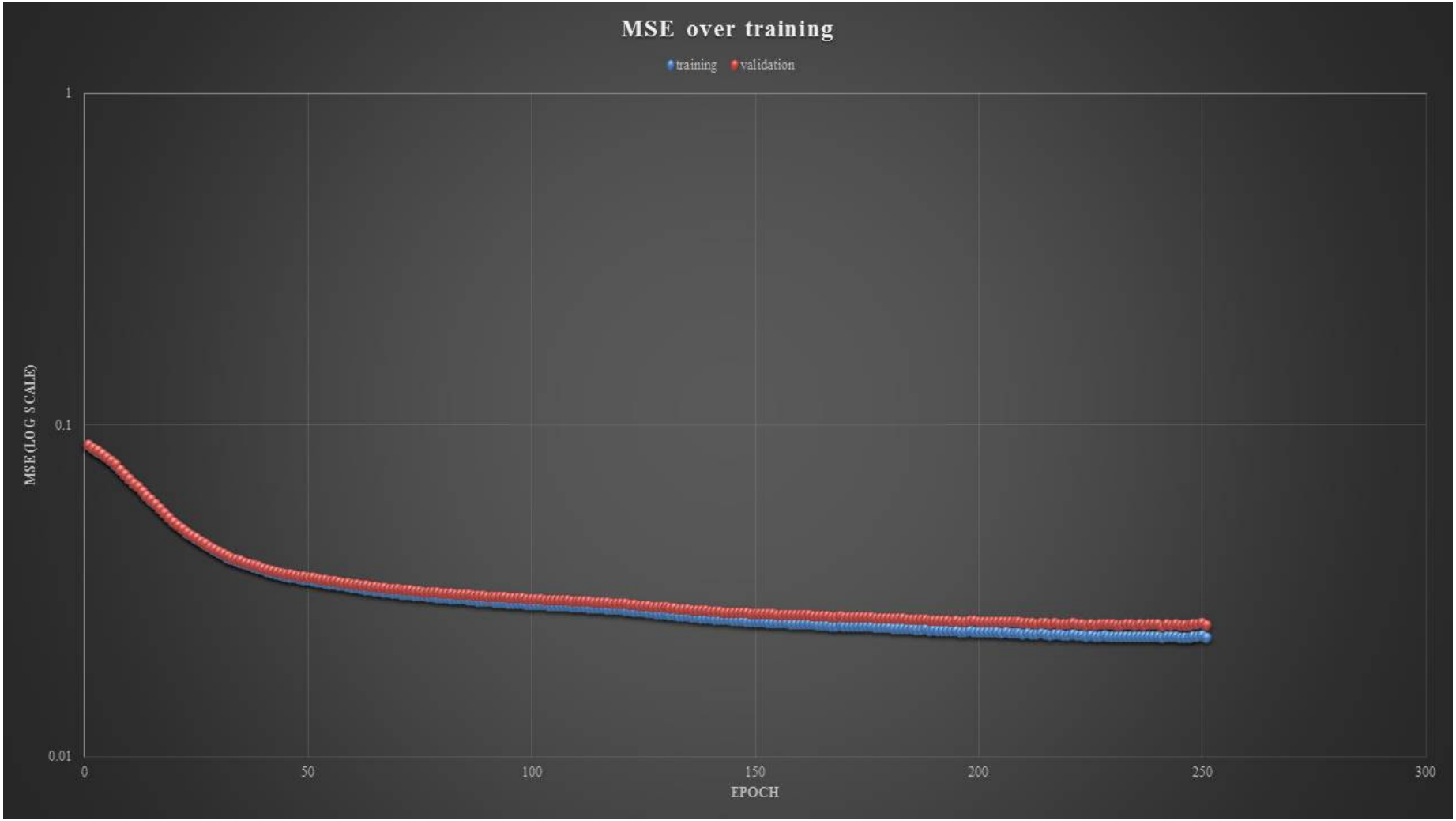


- Number of internal layers: 10
- Number of hidden units per layer: 300
- Batch size 200; ADAM optimizer
- Loss function: cross-entropy:
  $\sum_{i,j} \pi_{i,j}(-log(\hat{\pi}_{i,j})) + (1 - \pi_{i,j})(-log(1 - \hat{\pi}_{i,j}))$
- Training data set: 500,000 random instances $U(0, 1)$ i.i.d.
- Validation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

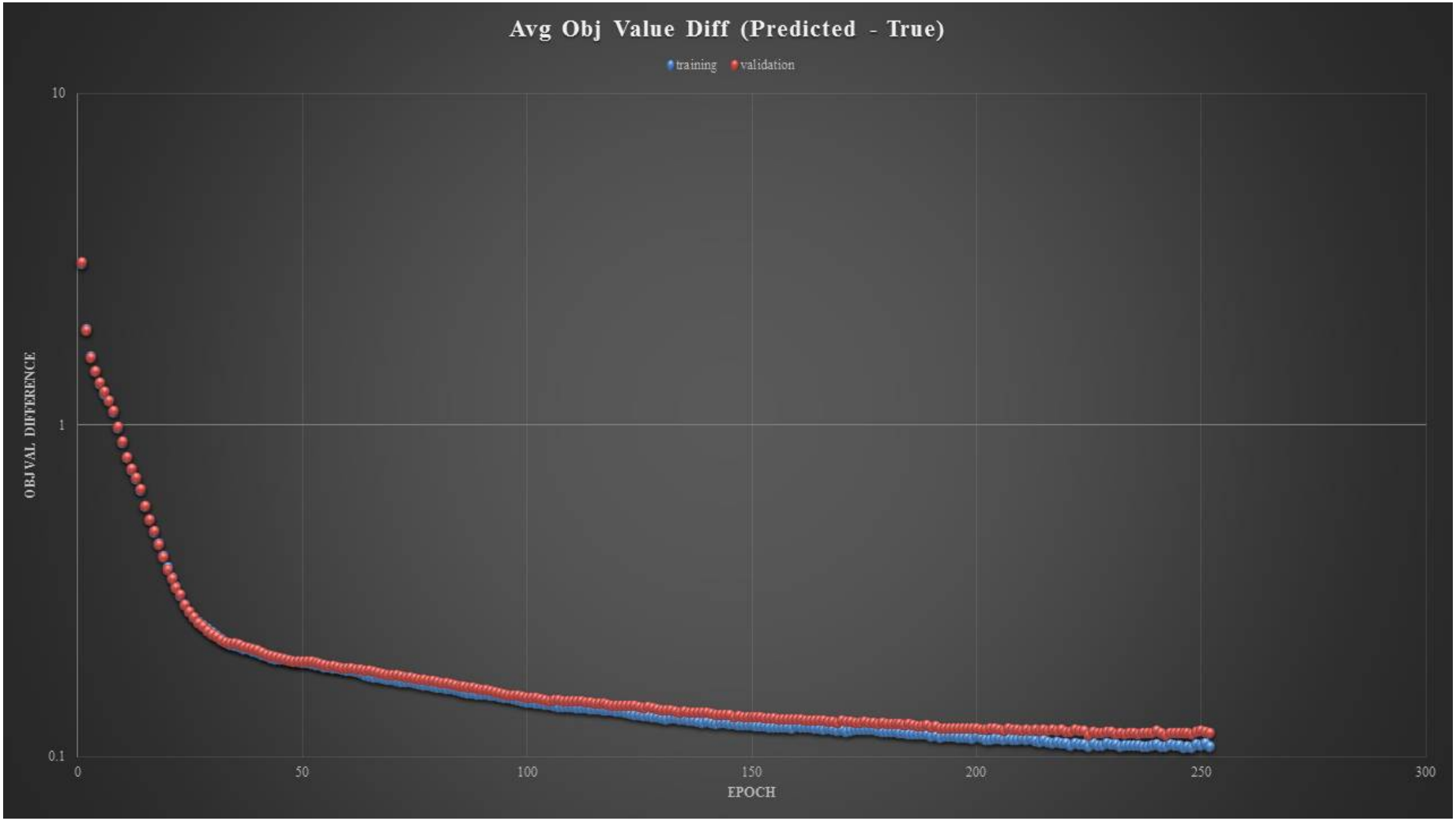
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



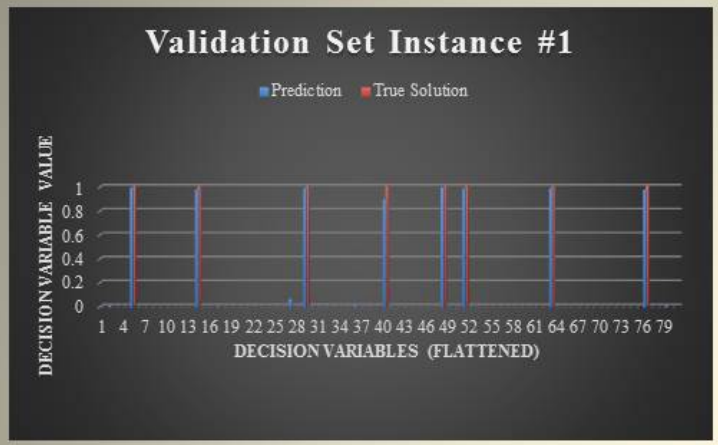
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation

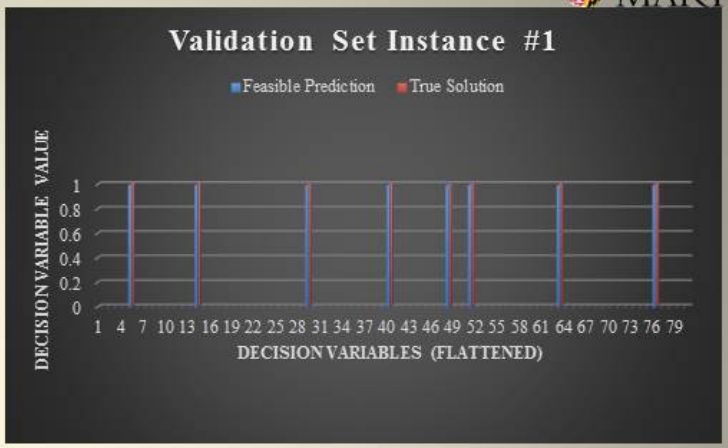


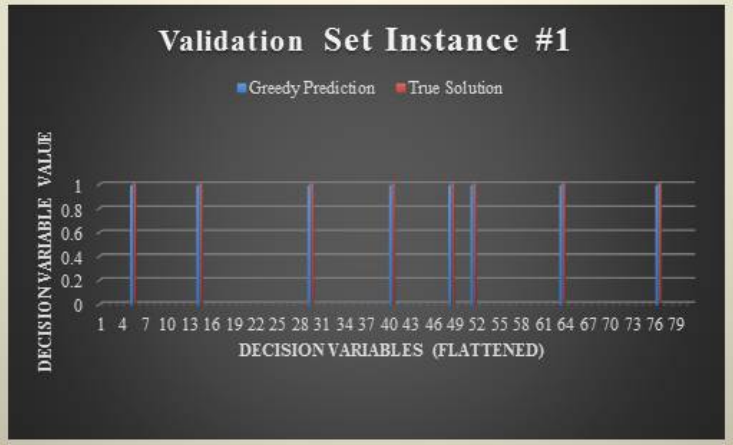
Exp.4 : $N = 10$, $R = 8$ with sigmoid activation



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

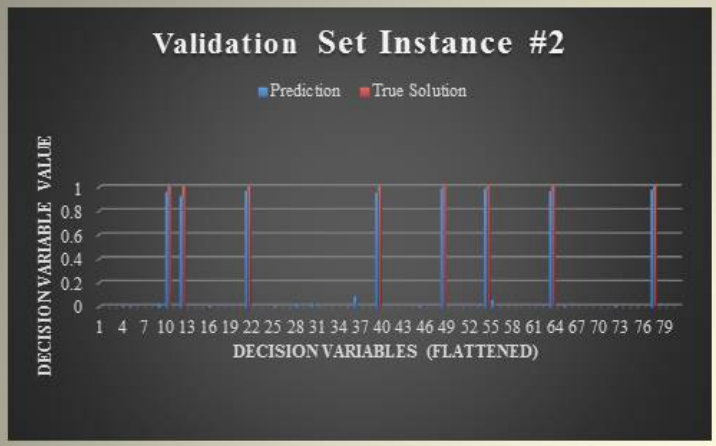


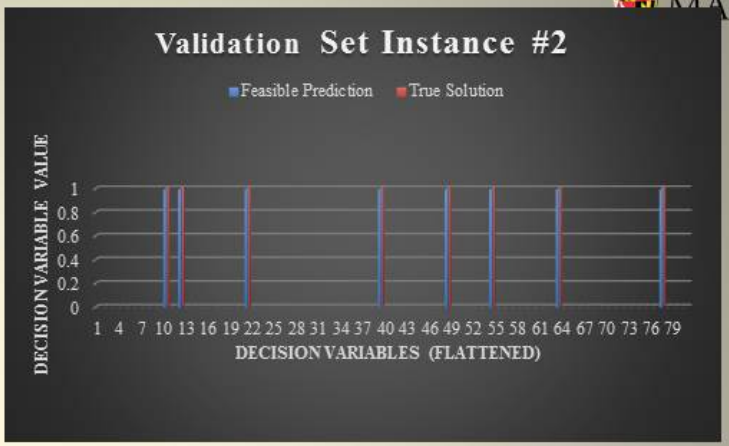


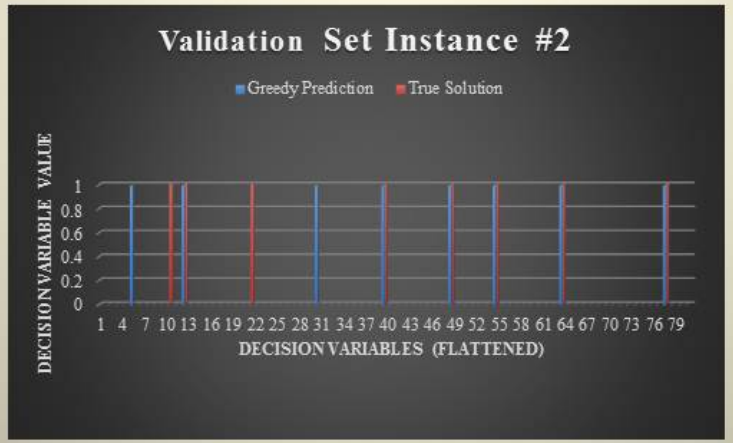


37

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







38

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. Weiling, *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]