Training One-Hidden-Layer Sparse Neural Networks

Jayant Agrawal
Indian Institute of Technology, Kanpur
agjayant@cse.iitk.ac.in

Purushottam Kar
Indian Institute of Technology, Kanpur
purushot@cse.iitk.ac.in

Abstract

In the recent past, deep neural networks have shown remarkable results for complex tasks in various domains such as computer vision and natural language processing. However, their reach is limited because they normally have a large number of redundant parameters along with other factors like computational complexity and huge data requirements. Sparse Neural Networks(SNN), with sparse connections between layers, are much more efficient, require fewer data points to train and can even match the performance of their dense counterparts. In this project, we consider regression problems with one-hidden layer networks and use Tensor Methods for weight initialization and Iterative Hard Thresholding Algorithm(IHT) for training. We analyze the performance of these algorithms for recovery of weights in one hidden layer sparse networks.

1 Introduction

Performance of deep learning models comes with various caveats. The number of parameters to be learned in these models is quite large(of the order of millions in some state-of-the-art CNNs). Consequently, the amount of training data required to train these models effectively is also large, which is not always available. Additionally, one forward pass in these models requires high computation power and huge memory space, the luxary of which is not available in most devices. If deep learning has to be of any benefit in the real world for tackling real world problems, these models should be trainable with a limited number of training samples, and should not just be made for high-end immobile hardwares.

Various studies in the recent past have shown that most of these models have a lot of redundant parameters. These redundant parameters do not contribute much to the output(their weight is close to zero). Also, there are many parameters which might learn the same ”feature”. These observations have lead researchers to come up with Sparse Networks.

1.1 Sparse Networks

Unlike dense networks, where every node in a layer is connected to every other node in the next layer, sparse networks have sparse connections. These models have significantly less number of parameters than the regular dense models saving on the amount of time, computation and memory.

[1] Submitted as UGP Report for CS498A

They have also shown similar accuracy results when compared with their dense counterparts.

Training these models is however not as simple as the dense models. Apart from learning the parameters, one has to choose an optimal sparsity structure. For more effective results, the sparsity structure can be learned from data itself. In this work, we attempt to do the latter.

We consider training one-hidden layer networks with sparsity for the problem of regression. The detailed problem formulation is described in Section 2. Recently, *Zhong et. al* have proposed an efficient algorithm for using Tensor Methods [4] for initializing weights in one-hidden layer networks. This method can help initialize the weights near the local convexity region. We describe this in detail in Section 3.

For training the network, we use the Iterative Hard Thresholding(Section 4), as used by *Prateek et. al* [1]. This training process helps us to enforce sparsity while training as opposed to many recent works which do the same after training. We give an outline to the complete algorithm in Section **??** before proceeding to the experiments(Section 5).

2 Problem Formulation

As described above, we consider one-hidden layer networks for the regression problem. We have a set(S) of n training samples where each example is d-dimensional($x_i \in \mathbb{R}^d$) and the output, $y \in \mathbb{R}$,

$$S = \{(x_1, y_1), (x_2, y_2)..(x_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$$

D is the underlying distribution over $\mathbb{R}^d \times \mathbb{R}$ with parameters

$$\{w_1^*, w_2^*, .., w_k^*\} \subset \mathbb{R}^d$$
$$\{v_1^*, v_2^*, .., v_k^*\} \subset \mathbb{R}$$

where $w_i^* \in \mathbb{R}^d$ represent the ground truth weights of connections from the input layer to the i^{th} hidden node and $v_i^* \in \mathbb{R}$ represent the ground truth weight of the connection from the i^{th} node to the output. k is the number of hidden nodes. Each sample $(x, y) \in S$ is sampled(i.i.d) from D as

$$D : x \sim \mathcal{N}(0, I_d), y = \sum_{i=1}^{k} v_i^* * \phi(w_i^{*T} x)$$

where ϕ is the activation function. The activation function is chosen assuming the following:

- It should be continuous
- If it is non-smooth then the first derivative should be the left derivative

Apart from these, the activation function has to satisfy certain important properties described in [4]. We omit them here, since most of the activation functions (ReLU, squared ReLU etc) satisfy those.

3 Tensor Methods for Initialization

Tensor Methods can recover parameters such that they fall into the local convexity region(near a local minima). *Zhong et. al* reduce the computational complexity from cubic dependency[2] to linear dependency[4]. We use the algorithm given by *Zhong et. al* and evaluate their performance in the sparse setting. We begin by describing some moments:

$$M_1 = \mathbb{E}_{(x,y)\sim D}[y * x]$$
$$M_2 = \mathbb{E}_{(x,y)\sim D}[y * (x \otimes x - I)]$$
$$M_3 = \mathbb{E}_{(x,y)\sim D}[y(x^{\otimes 3} - x\tilde{\otimes} I)]$$
$$M_4 = \mathbb{E}_{(x,y)\sim D}[y(x^{\otimes 4} - (x \otimes x)\tilde{\otimes} I + I\tilde{\otimes} I)]$$

[0] $v\tilde{\otimes} I = \sum_{j=1}^{d} [v \otimes e_j \otimes e_j + e_j \otimes v \otimes e_j + e_j \otimes e_j \otimes v]$

Algorithm 1 Initialization via Tensor Method

procedure INITIALIZATION(S)
  $S_2, S_3, S_4 \leftarrow$ PARTITION$(S, 3)$
  $\widehat{P}_2 \leftarrow \mathbb{E}_{S_2}[P_2]$
  $V \leftarrow$ POWERMETHOD$(\widehat{P}_2, k)$
  $\widehat{R}_3 \leftarrow \mathbb{E}_{S_3}[P_3(V, V, V)]$
  $\{\widehat{u}_i\}_{i\in[k]} \leftarrow \mathrm{KCL}(\widehat{R}_3)$
  $\{w_i^{(0)}, v_i^{(0)}\}_{i\in[k]} \leftarrow$ RECMAGSIGN$(V, \{\widehat{u}_i\}_{i\in[k]}, S_4)$
  Return $\{w_i^{(0)}, v_i^{(0)}\}_{i\in[k]}$
end procedure

Figure 1: Tensor Methods for Initialisation, Figure Credits: [4]

The idea is to use a third order moment P_3 that has a tensor decomposition formed by $\{w_1^*, w_2^*, .., w_k^*\}$[1]

$$P_3 = M_{j_3}(I, I, I, \alpha, ..., \alpha)$$

where $j_3 = \min\{j \geq 3 | M_j \neq 0\}$ and $\alpha \in \mathbb{R}^d$ is a random vector. To obtain an approximation of the parameters, P_3 can be decomposed using the non-orthogonal decomposition method [3]. But, the complexity for tensor decomposition on a $d \times d \times d$ tensor is $\Omega(d^3)$, which is quite expensive.

Zhong et. al in [4] propose an algorithm to approximate the parameters using tensor decomposition on $R_3 \in \mathbb{R}^{k\times k\times k}$. which can be obtained as

$$R_3 = P_3(V, V, V)$$

where V is the subspace(V) spanned by $\{w_1^*, w_2^*, .., w_k^*\}$. The subspace can be approximated from the 2^{nd} order moment P_2 using the Power Method.

$$P_2 = M_{j_2}(I, I, \alpha, ..\alpha)$$

where $j_2 = \min\{j \geq 2 | M_j \neq 0\}$. Tensor decomposition of R_3 is computationally much more efficient as compared to that of P_3. The overall algorithm is described in Figure 1.

4 Iterative Hard Thresholding

The idea is to "forcefully" make the network sparse while training. The algorithm proceeds as the regular gradient descent algorithm with the following addition: after regular intervals during training, some connections are selected and thrown.

Connections with "small" weights are generally redundant. Their contribution towards the output is minimal and removing them does not affect the accuracy. Selection of the edges which have to be removed can be done in two ways:

- *Thresholf Based-* All edges with value of weights less than a threshold
- *Top p- p* edges with least weights

5 Experiments

We begin by describing the setup in Section 5.1 and evaluation metric in Section 5.2. The results are shown in Section 5.3.

[1]For detailed proof : [4]

[1]For details on POWERMETHOD and RECMAGSIGN : [4]

Algorithm 1 Iterative Hard Thresholding

procedure TRAINIHT(W_0, s, T, η)
  $t \leftarrow 0, \textit{epoch} \leftarrow 1$
  while $\textit{epoch} \leq T$ **do**
    $i, j \leftarrow SelectRedundant(W_t, s)$ ▷ returns indices of edges to be removed
    $W_t^{i,j} \leftarrow 0$
    $W_{t+1} \leftarrow W_t - \eta \bigtriangledown_W f(W_t)$ ▷ Optimization Function f
    $t \leftarrow t + 1, \textit{epoch} \leftarrow \textit{epoch} + 1$

5.1 Setup

For setup, we employ the same strategy and use the same values of hyperparameters as used in [4]. Data points, $\{x_i, y_i\}_{i=1}^N$ are generated as described above in Section 2. W^* is set as $U\Sigma V^T$, where $U \in \mathbb{R}^{d\times k}$ and $V \in \mathbb{R}^{k\times k}$ are orthogonal matrices generated from QR decomposition of Gaussian matrices. Σ is a diagonal matrix whose diagonal elements are $1, 1+\frac{\kappa-1}{k-1}, 1+\frac{2(\kappa-1)}{k-1}, .., \kappa$. The value of κ is kept at 2. v_i^* is initialized to be randomly picked from $\{1, -1\}$. The activation function, ϕ, chosen is squared ReLU.

For the *Threshold* based sparsity scheme, we have two hyperparameters, $thresh\_gt$ and $thresh\_train$ which represent the ground truth sparsity threshold and the IHT training threshold respectively. For *Top-p* based sparsity scheme, we have p_gt and p_train. p_gt represents the fraction of edges present in ground truth weights. Finally, we use $noise\_sd$ to represent the standard deviation of the noise added to y_i before training.

We vary the eight parameters $(n, d, k, noise\_sd, thresh\_gt, thresh\_train, p\_gt.p\_train)$ against some base values and evaluate recovery error, sparsity structure and testing error(Section 5.2). The base values for various parameters are : $n = 6000, d = 20, k = 5, noise\_sd = 0, thresh\_gt = 0.15, thresh\_train = 0.15, p\_gt = 0.75.p\_train = 0.75$.

Learning rate is set at 0.001. The values reported are average over five trials.

5.2 Evaluation Metrics

Apart from the testing error, recovery error is also used as the evaluation metric. Recovery error is defined as follows:

$$min_{\pi:v_\pi=v^*}(||W_\pi^{rec} - W^*||/||W^*||)$$

where π is a permutation($\pi : [k] \rightarrow [k]$) and W_π^{rec} is the recovered W. The sparsity structure is also important and the four metrics given in Figure 2 define the recovery of the sparsity structure.

Criteria	Ground Truth	Recovered Weights
True Negative	Absent	Absent
True Positive	Present	Present
False Alarm	Absent	Present
Mis-Detection	Present	Absent

Figure 2: Sparsity Structure

5.3 Results

Figure 3 shows the effectiveness of the Tensor Initialization. The training errors for the initial epochs are clearly lower for tensor initialization as opposed to random initialization.

Figure 4 shows the plot of recovery error vs the training threshold for the threshold based IHT. The recovery error is the lowest at $thresh\_train = 0.075$. It is better to train the model using dense

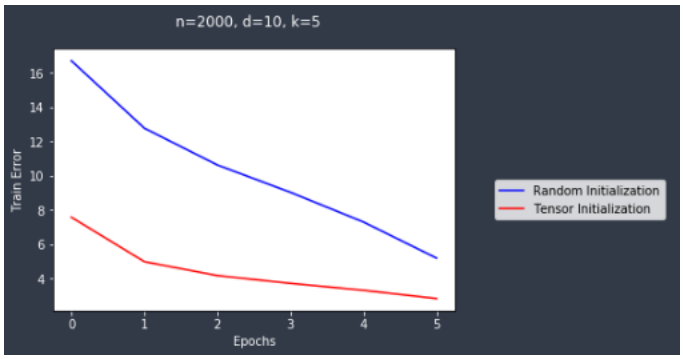


Figure 3: Tensor Methods vs Random Initialization

training for some epochs(pre-training) before using IHT to help the model adapt first. Figure 4 also shows that pre-training performs slightly better than IHT Threshold based training. Figure 6 shows the plot of recovery error vs p_train. Figures 5 and 7 shows the plot of the percentage of true negatives vs the IHT params for both the threshold based and top-p based methods.

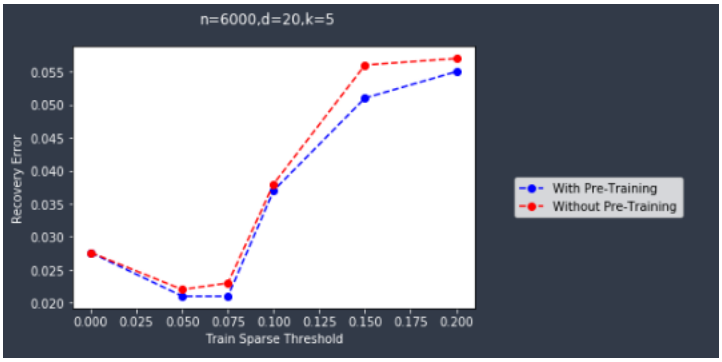


Figure 4: Recovery Error vs IHT Threshold

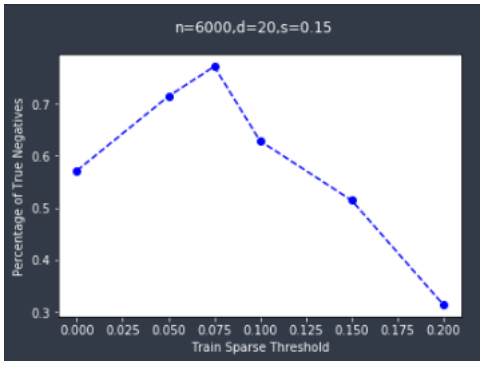


Figure 5: Percentage of True Negatives vs IHT Threshold

Figures 8-13 shows the recovery error and the percentage of true negatives vs various values of $\{n, d, k\}$ for both the types of IHT.

Figures 14 and 15 show comparision between the two types of IHT algorithms for various values of n and d respectively.

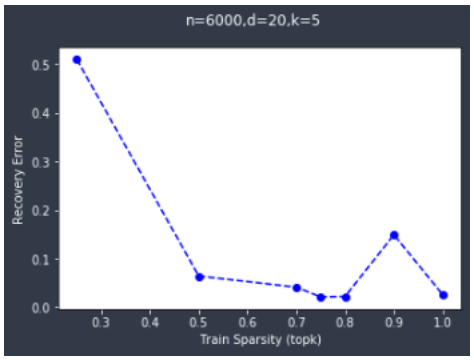


Figure 6: Recovery Error vs IHT Sparsity(topk)

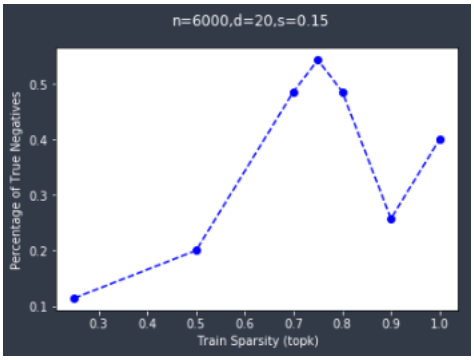
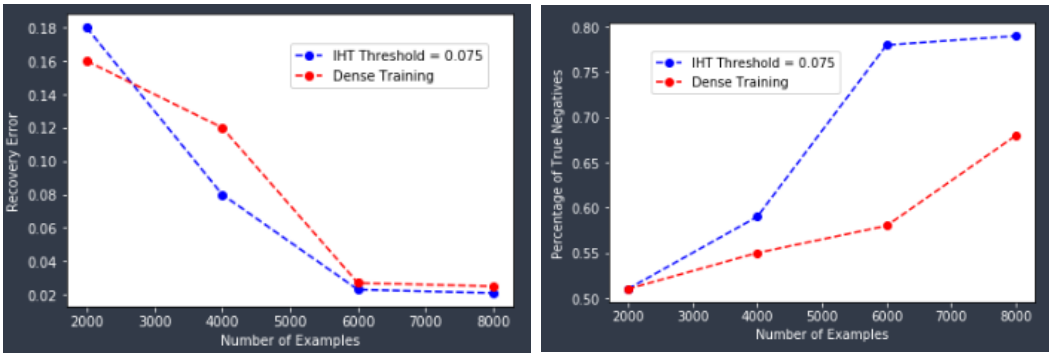


Figure 7: Percentage of True Negatives vs IHT Sparsity(topk)



(a) Recovery Error

(b) Percentage of True Negatives

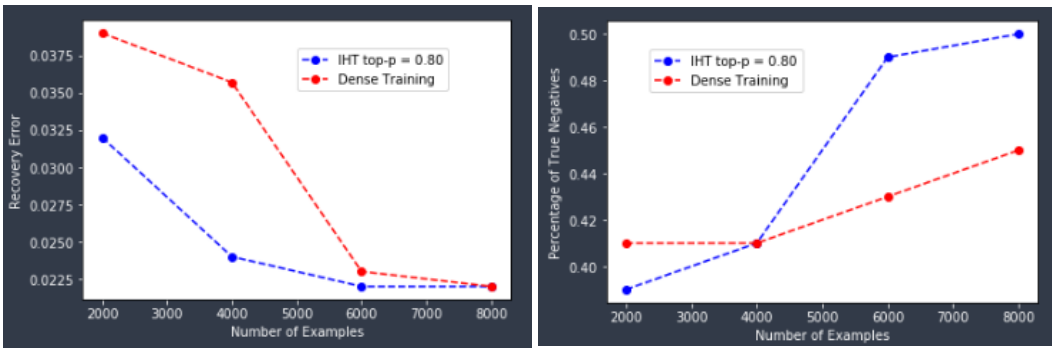
Figure 8: Number of Samples $=\{2000, 4000, 6000, 8000\}$ for IHT Threshold (Ground truth Threshold = 0.15)

References

[1] Prateek Jain, Ambuj Tewari, and Purushottam Kar. On iterative hard thresholding methods for high-dimensional m-estimation. *CoRR*, abs/1410.5137, 2014.

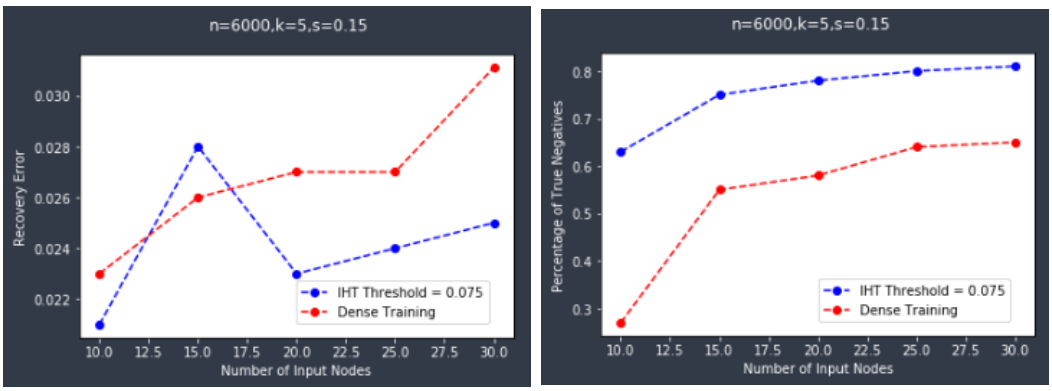
[2] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Generalization bounds for neural networks through tensor factorization. *CoRR*, abs/1506.08473, 2015.

[3] Volodymyr Kuleshov, Arun Tejasvi Chaganty, and Percy Liang. Tensor factorization via matrix factorization. *CoRR*, abs/1501.07320, 2015.



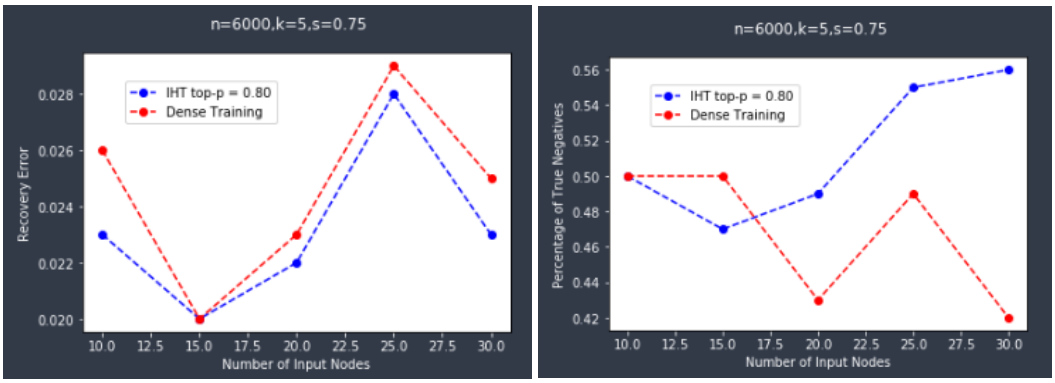
(a) Recovery Error (b) Percentage of True Negatives

Figure 9: Number of Samples $=\{2000, 4000, 6000, 8000\}$ for IHT $top-p$ (Ground truth Sparsity: 0.75)



(a) Recovery Error (b) Percentage of True Negatives

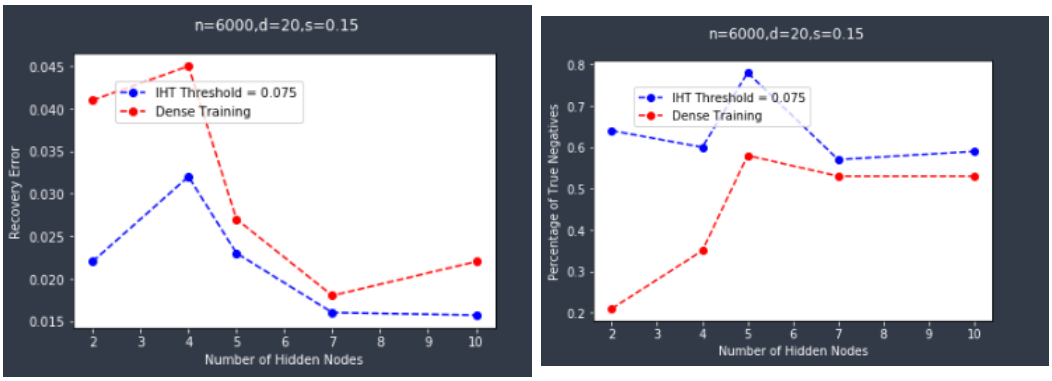
Figure 10: Number of Input Nodes: IHT Threshold (Ground truth Threshold = 0.15)



(a) Recovery Error (b) Percentage of True Negatives

Figure 11: Number of Input Nodes: IHT $top-p$ (Ground truth Sparsity: 0.75)

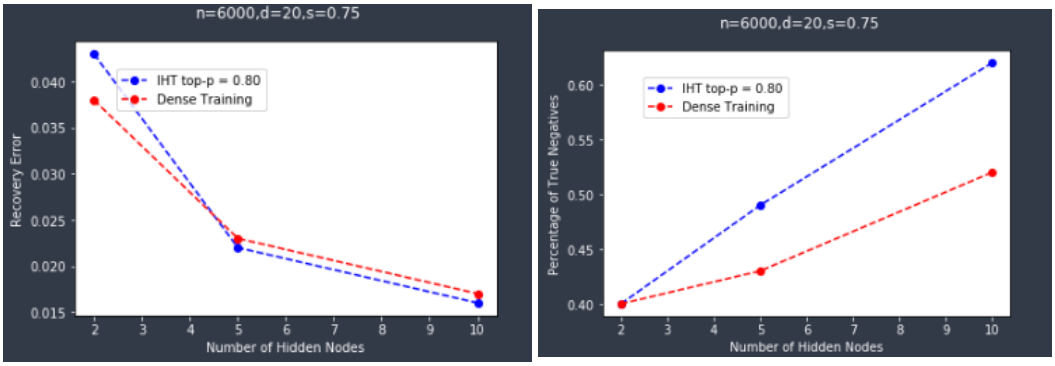
[4] Kai Zhong, Zhao Song, Prateek Jain, Peter L. Bartlett, and Inderjit S. Dhillon. Recovery guarantees for one-hidden-layer neural networks. *CoRR*, abs/1706.03175, 2017.



(a) Recovery Error

(b) Percentage of True Negatives

Figure 12: Number of Hidden Nodes: IHT Threshold (Ground truth Threshold = 0.15)



(a) Recovery Error

(b) Percentage of True Negatives

Figure 13: Number of Hidden Nodes: IHT $top - p$ (Ground truth Sparsity: 0.75)

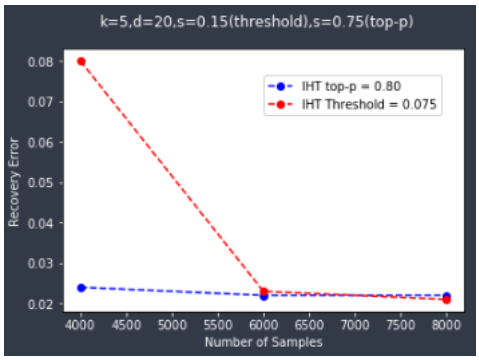


Figure 14: IHT Threshold and $top - p$ for various values of n

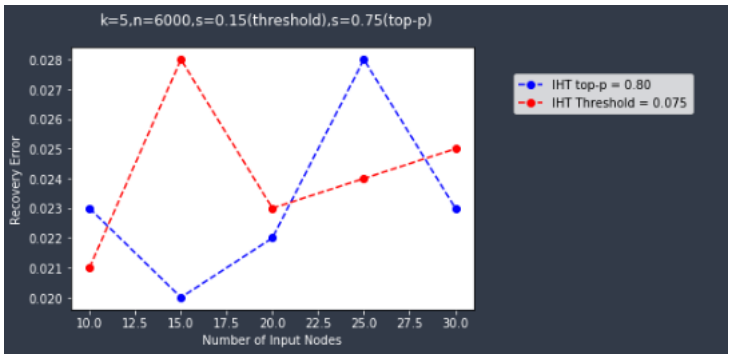


Figure 15: IHT Threshold and $top - p$ for various values of d