Project description & preliminary results: Neural network causal discovery

Anne Helby Petersen

Aug 26, 2021

1 Introduction

This project aims to address causal discovery as a supervised machine learning problem. More specifically, we use neural networks to construct a function that takes in a correlation matrix and returns a CPDAG adjacency matrix. Hence we obtain a mapping from observational data (represented through the correlation matrix) to its causal data generating mechanism (represented by its Markov equivalence class via its CPDAG).

The motivation for doing so is threefold:

1. We expect that this causal discovery approach may have better small sample properties than other approaches because the full data generating mechanism is learned jointly, rather than sequentially one edge at a time, and hence error propagation should be limited.
2. We naturally obtain quantification (probabilities) of edge presence rather than just a single final graph. This also implies that we will be able to easily adjust the false negative (or false positive) rate by adjusting the classification threshold for the neural networks.
3. With the right amount of data and the right neural network architecture we should be able to obtain of precision that is *only* limited by how well we estimate the sufficient statistic (in our case: correlation matrix). This is due to the facts that 1) causal discovery can be formulated as a continuous optimization problem (Zheng et al. 2018) and 2) a neural network can be used to estimate any continuous function (universal approximator property) (Hornik et al. 1989). Hence, with oracle correlation matrix information, it should in principle be possible to estimate adjacency matrices at any level of precision.

Initially, we limit the scope of the project to linear Gaussian data generating mechanisms. In this case, the correlation matrix is a sufficient statistic and hence it is meaningful to only consider these as input for the discovery method. Moreover, we only consider data generating mechanisms that can be described by DAGs. Because DAGs are not identifiable from observational data in the linear Gaussian case, we aim to retain only the Markov equivalence class of the DAG, which is represented by its CPDAG.

We use the following notation throughout: The terms variable and node are used interchangeably. We use p to denote the number of nodes in the DAG/CPDAG and n to denote the number of iid observations simulated according to this DAG. We refer to the neural network-based causal discovery method as *NNdisco*.

2 Methods

In this section we describe the specific methods used to construct the NNdisco method. Note that the full procedure below must be carried out for each combinations of values of p and n.

Our proposed method follows classic supervised machine learning, except for the fact that the training data and testing data are simulated. The simulated training data are used to construct a classifier using a neural network, and finally this classifier is evaluated on independent test data.

2.1 Data simulation

The data are simulated in a three-step procedure:

1. Randomly construct a DAG adjacency matrix.
2. Simulate from a linear Gaussian structural equation model according to the adjacency matrix.
3. Construct feature data (correlation matrix) and label (CPDAG adjacency matrix) and permute their variable orderings.

Each step is described in more detail below. We denote the p variables by $X_1, ..., X_p$.

2.1.1 Construction of DAG adjacency matrix

The DAG adjacency matrices are constructed in the following way:

1.1. Let M be a $p \times p$ lower triangular matrix of 1s. Let $n_{\text{lowtri}} = \frac{p \cdot (p-1)}{2}$ be the number of non-zero entries.

1.2. Draw a sparsity s from Unif$[0, 0.8]$.

1.3. Sample $\lfloor s \cdot n_{\text{lowtri}} \rceil$ lower triangular entries from M and set these to zero (where $\lfloor \cdot \rceil$ denotes rounding).

The resulting adjacency matrix M will then correspond to a DAG with varying degree of sparsity, as determined by s.

2.1.2 Simulate linear Gaussian data

To ease notation, we here describe the procedure for simulating a single observation, but the full dataset can be constructed by independently repeating the following procedure n times.

The adjacency matrix is used as a data generating mechanism, i.e. $M[i,]$ determines the parents of X_i such that:

$$\text{pa}(X_i) = \{j \mid M[i, j] = 1\}$$

We simulate X_i according to the following data generating mechanism:

$$X_i := \sum_{X_j \in \text{pa}(X_i)} \beta_{j,i} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma_i^2)$ independently. The standard deviations are drawn independently from uniform distributions:

$$\sigma_i \sim \text{Unif}[0.5, 2]$$

The regression parameters are constructed as follows (independently):

$$\beta_{i,j} := b_{\text{val}} \cdot b_{\text{sign}}$$

where b_{val} is drawn from Unif$[0.1, 2]$ and b_{sign} is sampled from $\{-1, 1\}$ with probabilities $\{0.4, 0.6\}$, respectively. We use non-uniform sampling to avoid imposing unrealistic symmetry in the simulation setup, which might increase the risk of almost-nonfaithfulness. Note that we do not allow for $\beta_{ij} = 0$ to avoid actual nonfaithfulness.

In practice, this simulation is performed one variable at a time, noting that the causal ordering of the variables is $(X_1, X_2, ..., X_p)$ due to the lower triangular matrix being used as a starting point for creating the DAG adjacency matrix.

2.1.3 Construct features and labels

Let π be a random permutation of the integers $\{1, ..., p\}$.

In order to construct the feature data, we compute the Pearson correlation matrix for the full dataset $\mathbf{X} = (X_1, ..., X_p)$. Let C denote this correlation matrix. We then permute both rows and columns of C according to π and store the resulting correlation matrix as the feature data.

In order to obtain the label, we construct the CPDAG corresponding to the Markov equivalence class of the original DAG with adjacency matrix M (using the `pcalg` R package function `dag2cpdag()`). Let A denote the adjacency matrix corresponding to this CPDAG. We then permute rows and columns of A according to π and store the resulting CPDAG adjacency matrix as the label.

2.2 Classifier estimation

We construct a classifier $f : \mathbb{R}^{p\times p} \to [0,1]^{p\times p}$ which takes in a correlation matrix and outputs a matrix of probabilities of adjacency matrix entries being 1. This matrix of probabilities may then be converted into a pseudo adjacency matrix by choosing a threshold $\psi \in [0,1]$. Note that the resulting matrix may not be proper CPDAG adjacency matrix, which is why refer to it as a pseudo adjacency matrix.

The classifier is constructed by training a neural network on the simulated data described in the previous section. The aim is to find one or two 'one-size-fits-all' architectures that perform reasonably well at varying input sizes (p), as it is not realistic to build custom optimal network architectures for each possible choice of p.

We will start by investigating the performance of three different network architectures:

N1: A simple fully connected neural network:

- 6 fully connected layers of p^2 nodes with increasing regularization via dropout.

N2: A simple convolutional neural network:

- 2 convolutional layers each with p^2 filters and varying kernel sizes
- One pooling layer (max pooling)
- 2 fully connected layers with dropout regularization

N3: An advanced convolutional neural network:

- The Inception architecture. This is a complicated network architecture that is considered state-of-the-art for image classification.

A comparison of the performance of these networks will then guide further development of a useful network architecture. We should also consider using a network architecture that makes use of the fact that the input and output matrices are permutation equivariant for the variable orderings, as proposed by Li et al. (2020).

The networks output $p \times p$ matrices using a sigmoid activiation function. This means that the output will be a matrix of numbers in $[0,1]$ which can be interpreted as probability of that adjancency matrix entry being a one. For evaluation, we consider a sequence of different classification thresholds used for obtaining a pseudo adjacency matrix from this matrix of probabilities.

The models are trained and evaluated on batches of 100 observations. We evaluate the networks after training for 1000 epochs. We use a binary cross-entropy loss function, but should consider other loss functions at later stages of the project.

2.3 Evaluation

We evaluate the neural network discovery methods by comparing predicted pseudo adjacency matrices with actual adjacency matrices on the test data set. We consider the following metrics:

- Structural Hamming Distance
- Number of false positive entries (with direction)
- Number of false negative entries (with direction)
- Number of spurious adjacencies (false positive without direction)
- Number of missing adjacencies (false negative without direction)

- Percentage of correctly determined entries
- Percentage exactly correctly classified adjacency matrices
- Percentage of adjacency matrices that are valid CPDAGs
- Percentage of adjacency matrices that are valid PDAGs
- OVERVEJ: Diff in degree

These numbers are reported as means, both over the full test data and by average graph degree (number of neighbors). At a later stage, we should also consider reporting the distributions of selections of these measures to give a more detailed insight into the performance and its stability.

We apply two different strategies for testing:

1. We match training- and test data sample sizes. For example, when evaluating the performance on test data with $n_{\text{test}} = 100$, we also use a model trained on datasets of size $n_{\text{train}} = 100$.
2. We use a model trained on the maximal sample size, $n_{\text{train}} = 50000$, but test the performance on varying smaller sample sizes.

At a later stage, we may consider training first on a larger sample size and using the weights thereby obtained as a starting point for a training on a smaller sample size.

3 Project plan summary

We will vary the following aspect of the procedure and compare the resulting performances:

- The size of each simulated training dataset, n_{train}. We will consider $n_{\text{train}} \in \{50, 100, 500, 1000, 5000, 10000, 50000\}$.
- The number of variables in the data generating mechanism, p. We will consider $p \in \{2, 5, 10, 20\}$.
- The choice of network architecture: Simple fully connected (N1), simple convnet (N2), advanced convnet (N3).
- The testing procedure: Either matching n_{train} and n_{test} or using $n_{\text{train}} = 50000$ for all values of n_{test}.

Note that $p = 2$ is a special case, where only marginal independence between the two variables can be determined.

The list above results in a total of 28 different training data sets to be simulated, 84 different neural networks to be trained, and 156 evaluation scenarios (2 times the number of neural network minus the 12 networks with $n = 50000$ where only $n_{\text{train}} = n_{\text{test}}$ is considered). We compute each of the metrics outlined in section 2.3 for each of these evaluation scenarios.

We will compare the neural network discovery method with the PC algorithm, as implemented in the R package `pcalg`. We consider a sequence of test significance levels, namely $\alpha \in \{10^{-8}, 10^{-4}, 10^{-3}, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8\}$.

3.1 Training and test data sizes

For each combination of these settings, we will consider training data of $b_{\text{train}} = 1000000$ correlation matrix/adjacency matrix pairs, and we test the procedure on $b_{\text{test}} = 5000$ correlation matrix/adjacency matrix pairs. Note that in practice, the necessary number of training data pairs will of course depend strongly on the number of nodes in the data generating DAG.

The number of different possible DAGs of size p can be computed using the recursive formula (Robinson 1973):

$$G(p) = \sum_{k=1}^{p} (-1)^{k+1} \binom{p}{k} 2^{k(p-k)} G(p-k)$$

For $p = 5$ there exists approx. 30,000 different DAGs, for $p = 10$ there are approx. 10^{18} different DAGs and for $p = 20$ there are approx. 10^{72} different DAGs.

However, there are fewer CPDAGs for each p (since several DAGs belong to the same Markov equivalence class and hence the same CPDAG). Therefore it may be that $b_{\text{train}} = 1000000$ is sufficient for learning the correlation matrix/CPDAG relationship for e.g. $n = 10$, even though the number of DAGs over 10 nodes is much larger. I am not aware of a formula for counting the number of CPDAGs for a given node size, but such a result will be useful for choosing appropriate values of b_{train}.

Note that there is no strict practical upper limit for possible b_{train}. The training data can be loaded in batches, so the full training data does not have to be loaded at once, and hence limited computer memory is not an issue. However, the larger b_{train} is, the longer it will of course take to train the model.

4 Preliminary results

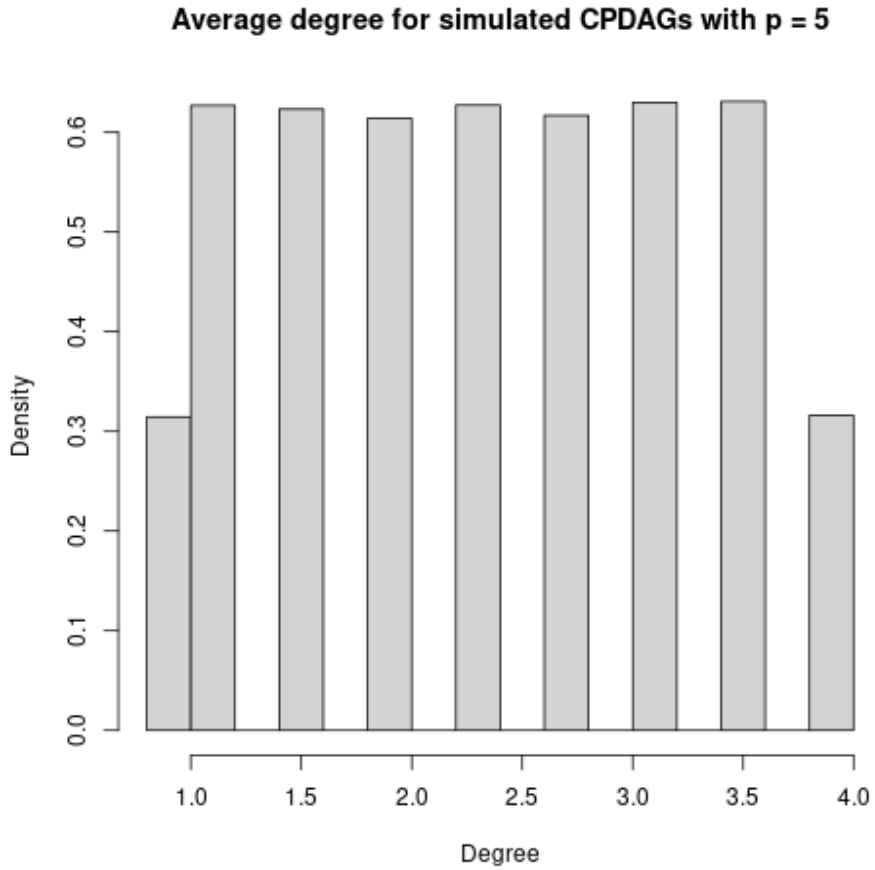
We here provide preliminary results regarding the performance of NNdisco. These results are all obtained using the N2 network. We have preliminary results only for $p = 5$ and $p = 10$. The $p = 5$ models have been trained on $b_{\text{train}} = 40000$ correlation/adjancency matrix pairs, while the $p = 10$ models have been trained on $b_{\text{train}} = 100000$ matrix pairs.

First, we present descriptions of the sparsity levels in the used training data sets. Secondly, we turn to performance evaluation for the neural network discovery methods. Note that we provide slightly different performance metrics than what is outlined in the above.

The data used here were simulated as explained in Section 2.1, except for one difference: The regression parameters in the data used here were drawn from Unif$[0.1, 5]$. This produces data with unrealistically large degrees of correlations between variables. Moreover the correlations are all positive. The latter should not be a problem at this stage, but will be an issue for real life applications as the neural networks would not have seen any negative correlations.

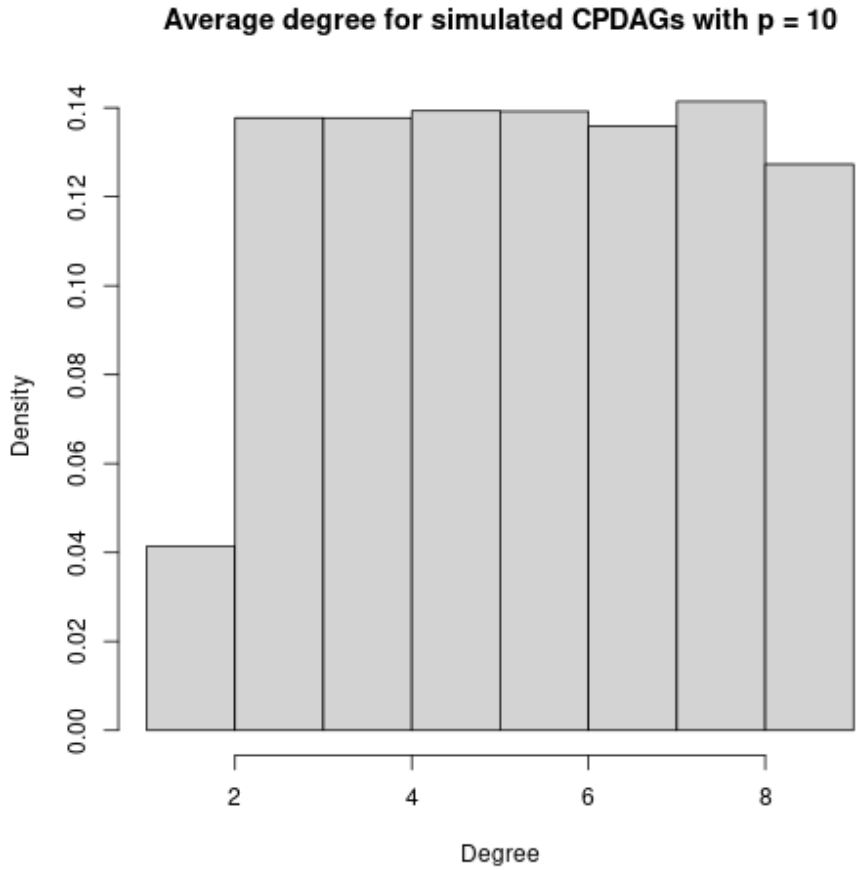
4.1 Sparsity of CPDAGs

We here show the average degree (number of neighbors per node) for the skeletons corresponding to the CPDAGs generated for $p = 5$. The average is taken over $n_{\text{train}} = 40000$ CPDAGs simulated according to the description in Sections 2.1 and 2.2.



We see that the sparsity levels presented in Section 2.1 result in graphs that vary a lot in their degree, and cover both very sparse and very dense graphs.

Similarly, we present the average degree of simulated CPDAG skeletons with $p = 10$ (based on $n_{\text{train}} = 100000$ CPDAGs):



We also here see that we succeed in producing graphs with varied degrees of sparsity, and that we cover the full range from very sparse to very dense graphs. No degrees are particularly over represented here.

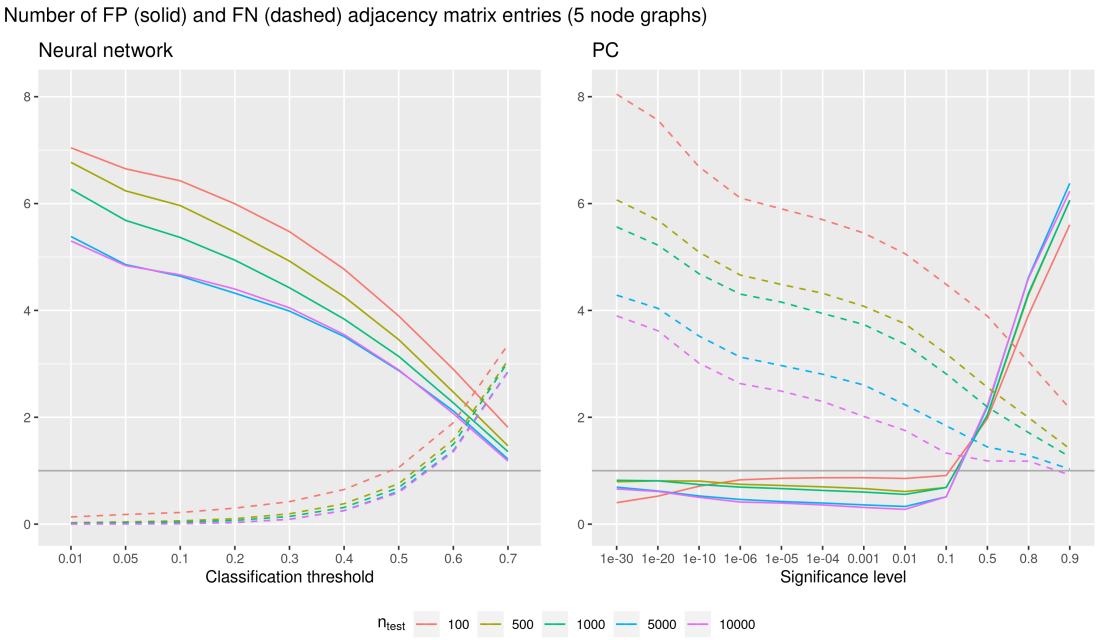
4.2 Results for p = 5, all graph degrees

For all of these models, we have used $b_{\text{train}} = 40000$ data points for training and $b_{\text{test}} = 2000$ data points for testing. The models were all trained on correlation/adjacency matrix pairs based on $n_{\text{train}} = 10000$ simulated observations, while we have varied the number of data points for testing, and evaluated performance for each such n_{test} option. The neural networks were trained for 500 epochs.

NNdisco is compared to PC with varying choices of significance level for tests.

4.2.1 False positives and false negatives

We present the number of false positives and false negatives in the figure below. Note that these are absolute numbers. The adjacency matrices are constructed such that a undirected edge is encoded as a bidirected edge. This means that a false positive can both mean that an edge that was supposed to be directed was mistakenly made undirected, or that a directed edge is completely absent.

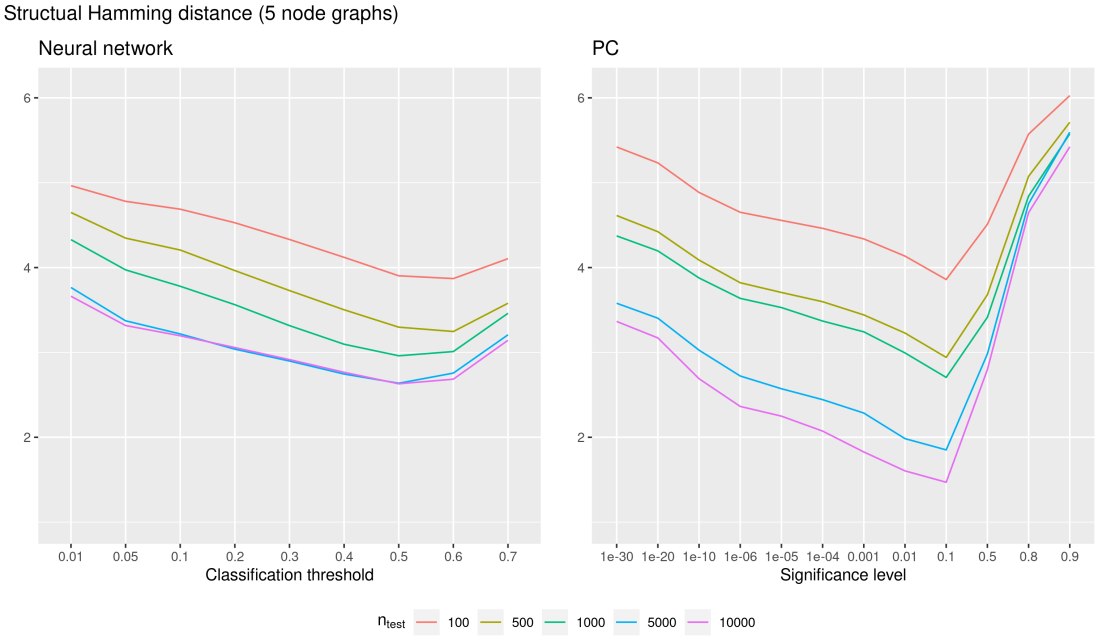


We find that NNdisco provides poor control of false positives, but good control of false negatives. PC has the opposite property: while the number of false positives is on average less than 1 for all significance levels of 0.1 or below, the number of false negatives can be rather large.

For PC, the number of false negatives seem to be largely influenced by the choice of n_{test}, while NNdisco seems more robust towards varying n_{test}. This may be a first indication that NNdisco will indeed be have good small sample properties, as discussed above.

4.2.2 Structural Hamming distance

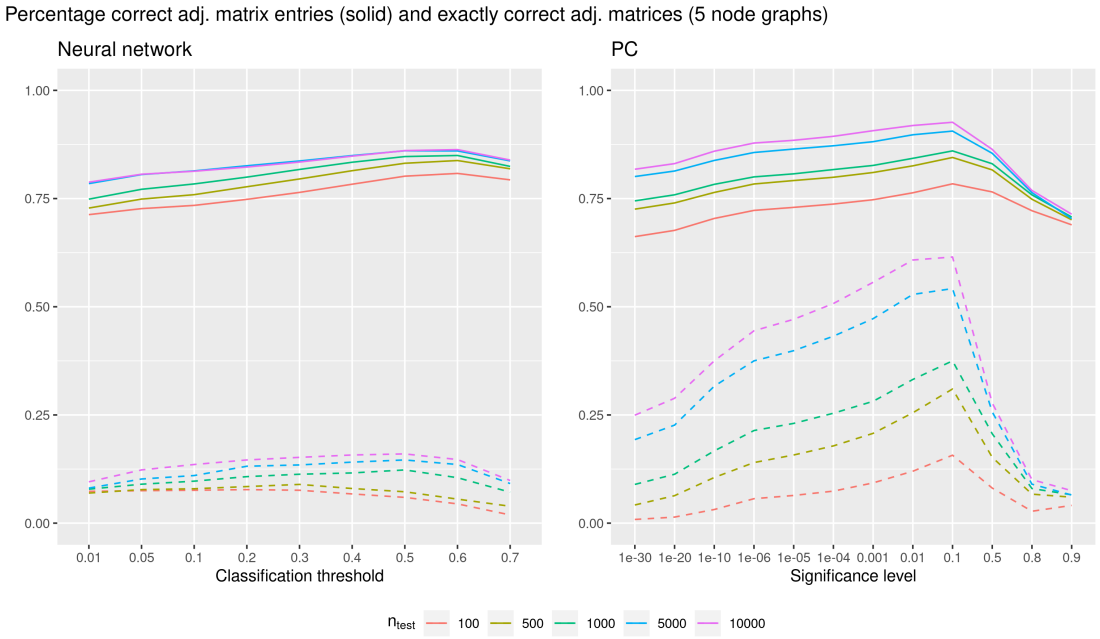
We here present the results concerning structural Hamming distances (SHD).



We find larger SHDs for NNdisco than PC when n_{test} is large. But when n_{test} is small, NNdisco produces smaller hamming distances than PC. This may be because PC tends to produce graphs that are too sparse in this scenario, as seen by the large false negative numbers in the previous figure.

4.2.3 Accuracy

Finally, we look at results for accuracy. The accuracies marked in solid lines are the average number of correct entries in the predicted adjacency matrices. The accuracies in dashed lines are the percentage of adjacency matrices that were completely correct.



We see that NNdisco seldom finds the completely correct adjacency matrix, no matter the value of n_{test}, while PC can produce completely correct adjacency matrices in approximately 60% of the cases for large n_{test} and significance level $\alpha = 0.1$ (right panel, dashed pink line, $\alpha = 0.1$).

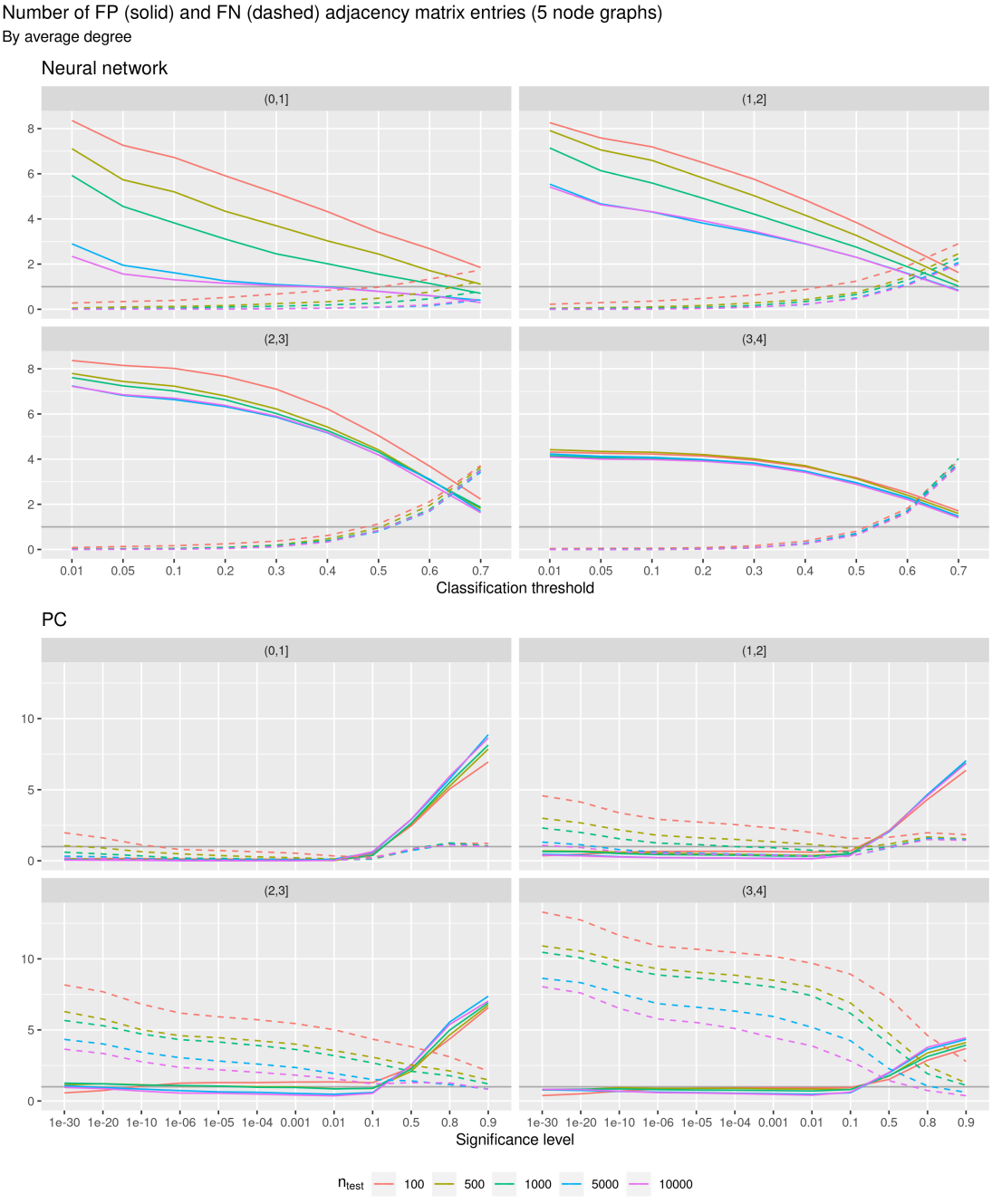
For the entry-wise accuracies, NNdisco obtains larger values than PC for small values of n_{test}, but is once again outperformed by PC for large n_{test}.

It thus again seems like NNdisco may have better small sample properties. Note also that no remedies are used to enforce that the output of NNdisco is actually a proper CPDAG. Therefore, it will perhaps be possible to increase accuracy substantially by adding host-poc corrections to the output or by identifying a more informative loss function to use for training the NNdisco models.

4.3 Results for p = 5, stratified by graph degree

We now show the same measures as above, but with stratification by degree. The degrees are measured as the average number of neighbors for each node in the actual test data adjacency matrices. Each panel in the figures in this section correspond to an interval of average degrees.

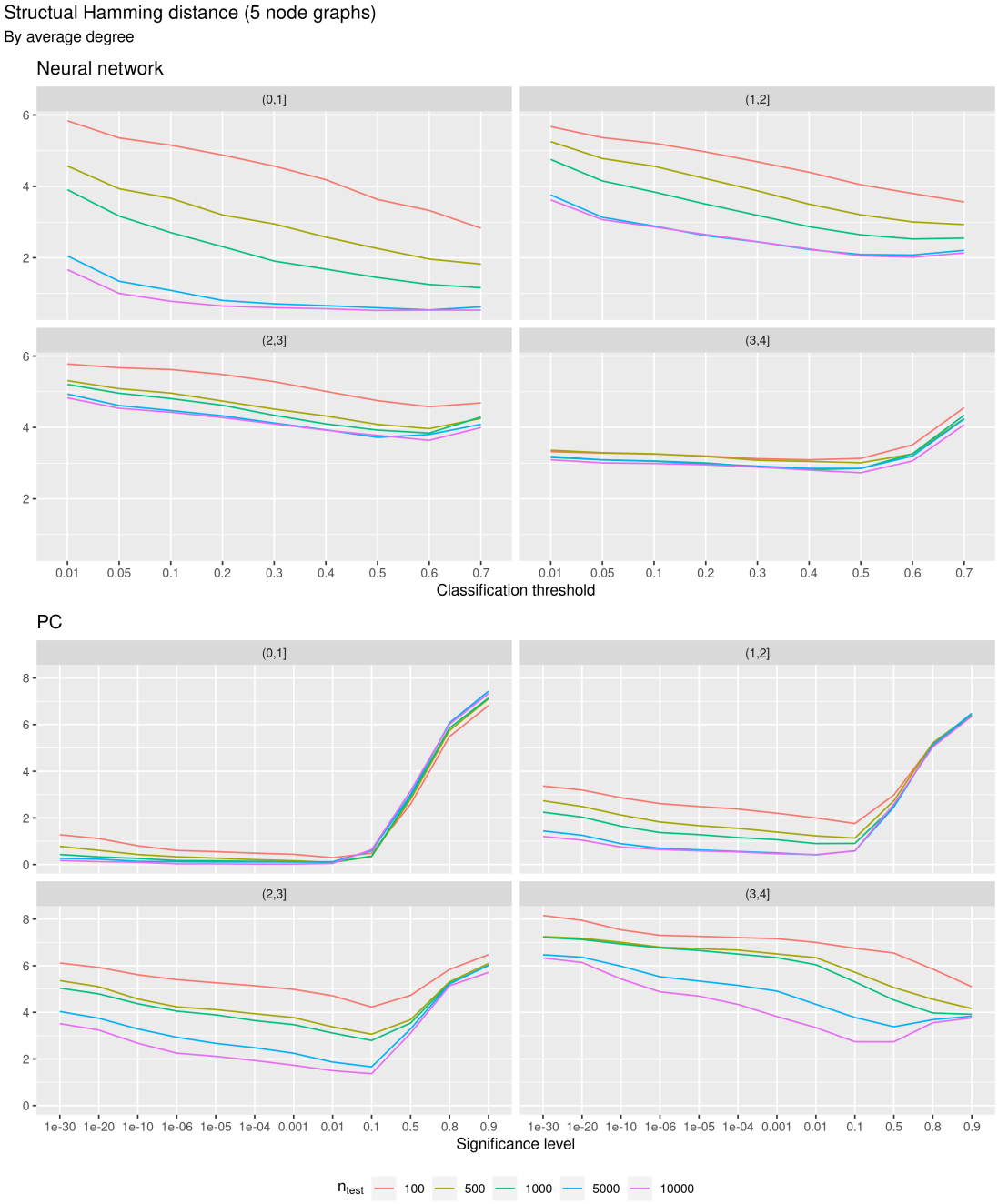
4.4 False positives and false negatives



We see that NNdisco has large numbers of false positives, especially for sparse graphs for almost all choices of classification threshold. The method generally performs best for dense graphs, and in this case, the method is quite robust towards the value of n_{test}. The varying performance for different graph densities may be an indication that NNdisco requires a larger learning data set in order to describe the relationship between correlation matrices and adjacency matrices for all sparsity levels. This will be explored in the project plan outlined above, where we will use $b_{\text{train}} = 1000000$ training data points (here, we use only 40000).

For PC, one notable observation is that $\alpha = 0.1$ seems to be the empirically optimal choice of significance level for controlling false positives, no matter the degree. I have no good explanation for this results, and I am therefore worried that it is a statistical artifact that results from the specific simulation setup.

4.5 Structural Hamming distance

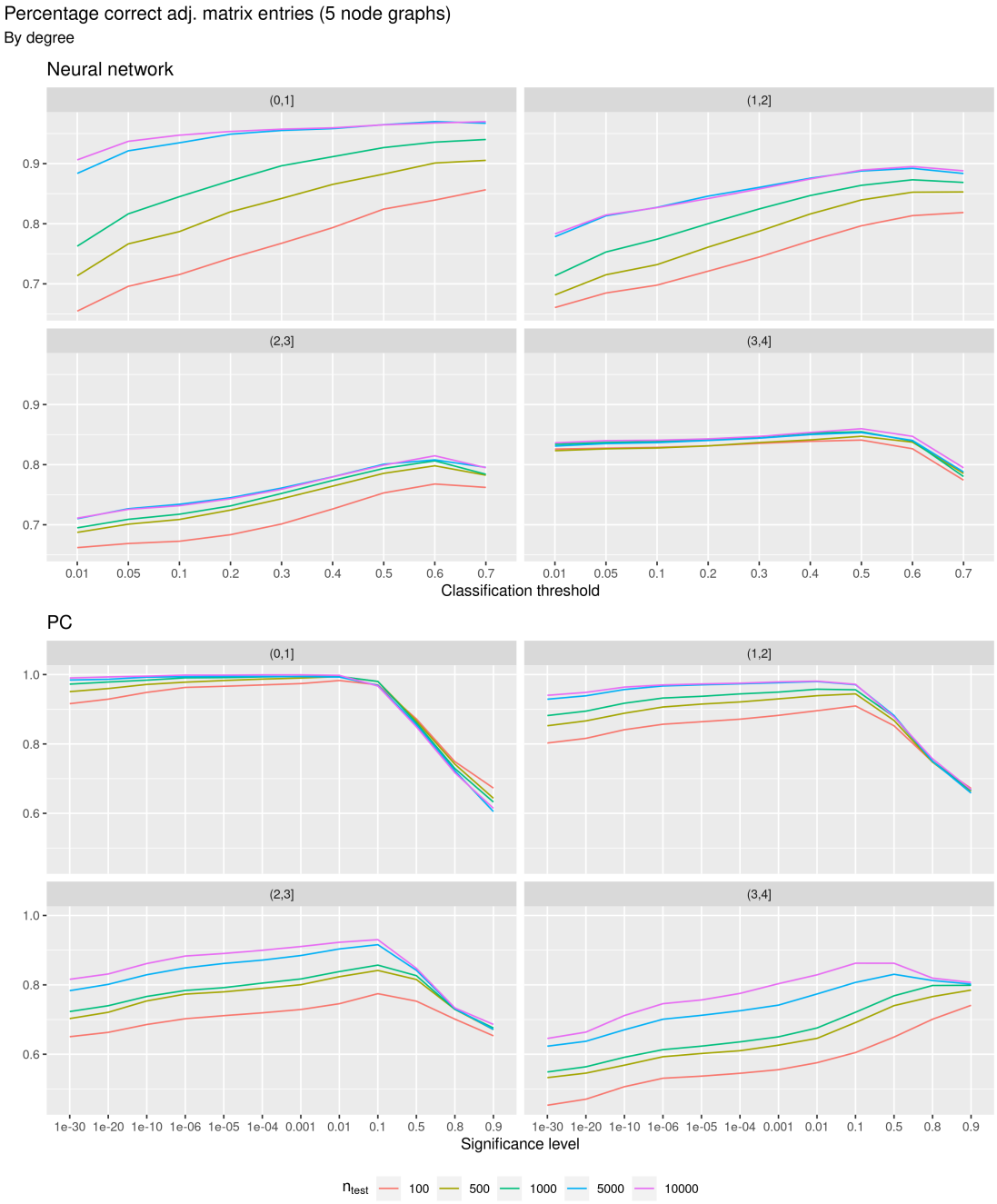


NNdisco shows increasing sensitivity towards the value of n_{test} as the degree decreases; while dense graphs obtain almost identical SHDs across all choices of n_{test}, the performance on sparse graphs depends strongly on the value of n_{test}.

PC shows the opposite tendency: more dependency on n_{test} for larger graph degrees. Note also that we again find best performance (as measured by lowest SHD) at significance level 0.1 for almost all combinations of n_{test} and average degree.

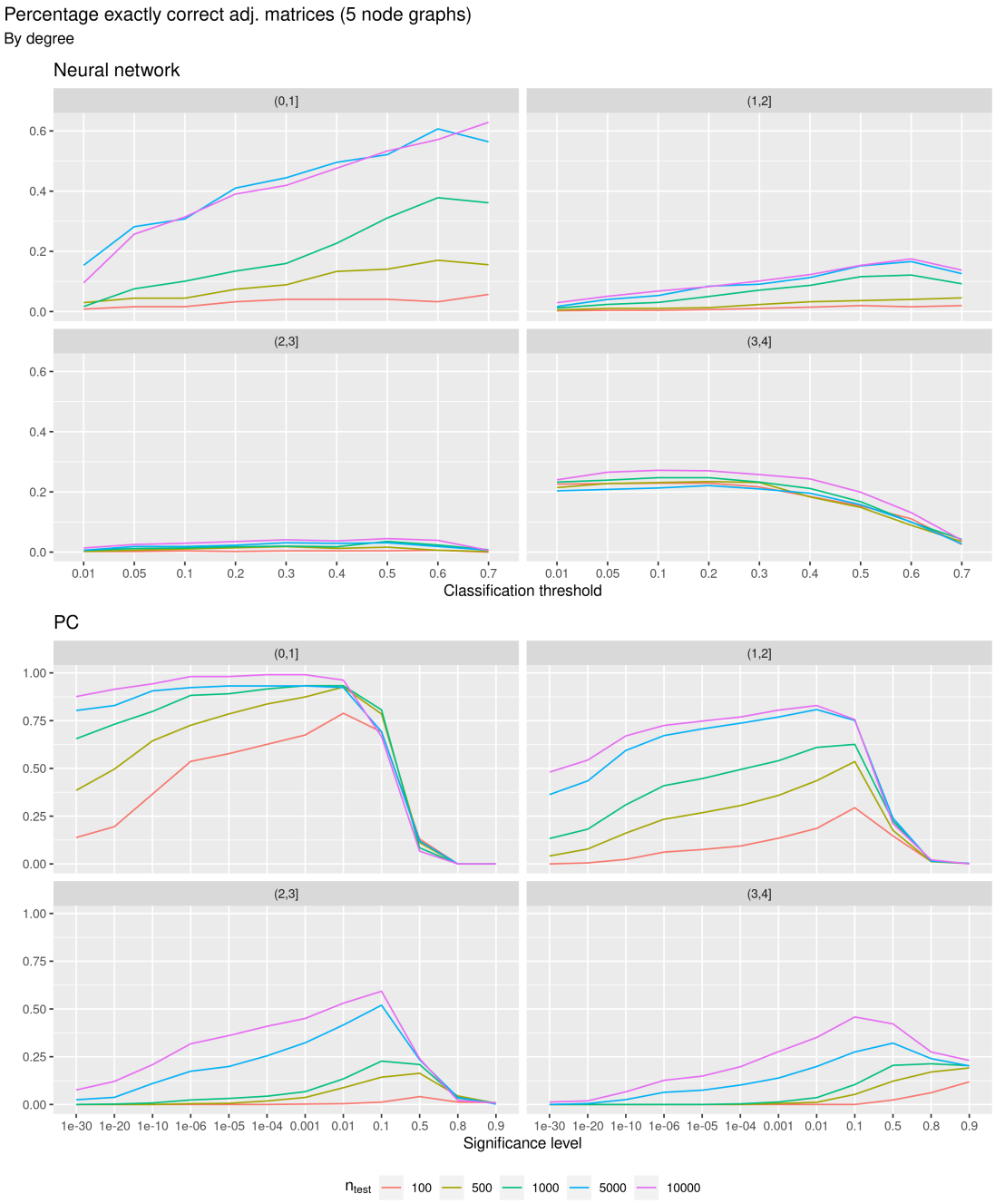
4.6 Accuracy

In the interest of readibility, we plot the two accuracy measures separately here.



We here find patterns similar to what we have seen for the other performance measures: NNdisco is sensitive towards n_{test} for small degrees, while PC is sensitive towards n_{test} for large degrees.

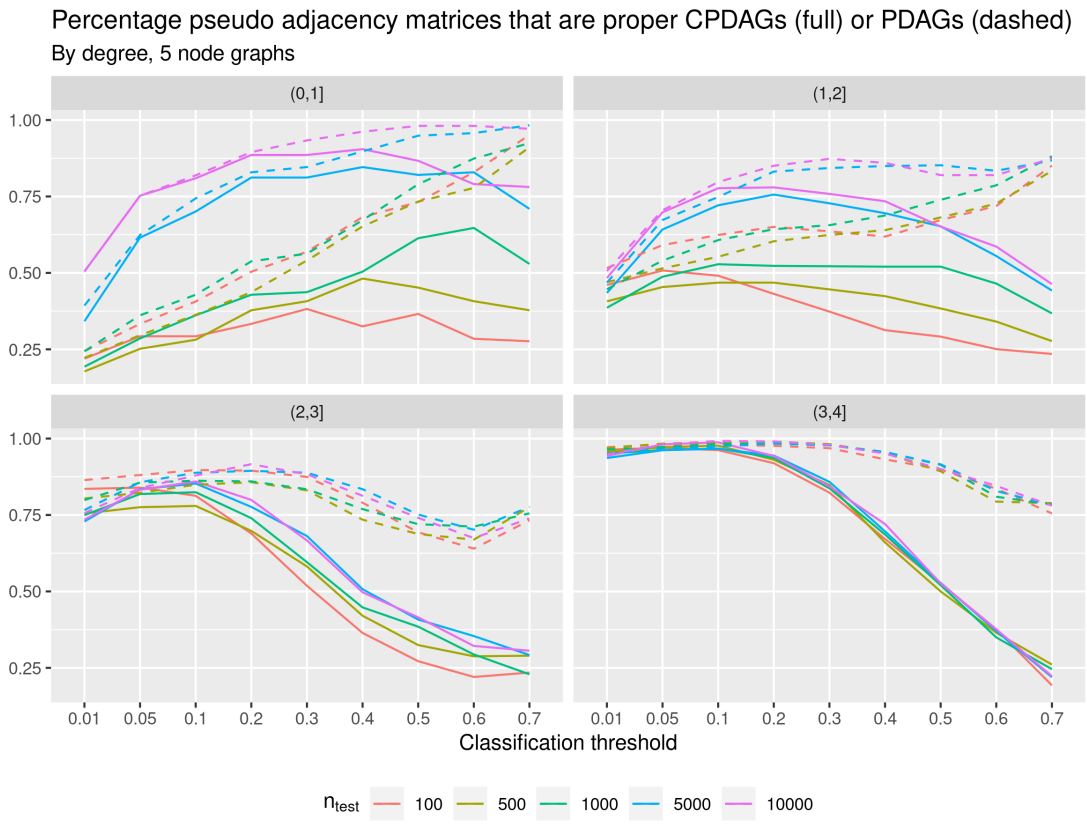
Once again, we note that PC curiously performs best at $\alpha = 0.1$ across most combinations of n_{test} and degree.



We then turn to the percentage exactly correctly predicted adjacency matrices. Here, NNdisco generally performs poorly. We find little variation across n_{test} values, except for in the most sparse case.

PC again shows a general sensitivity towards n_{test} and in most cases, $\alpha = 0.1$ is yet again the optimal significance level choice, except for the very largest values of n_{test} in combination with the sparsest graphs (degree in (0,2]). Here, smaller significance levels result in slightly larger percentage correctly predicted matrices.

4.6.1 Number of proper CPDAGs and PDAGs



Finally, we provide an overview of the number of pseudo adjacency matrices returned by NNdisco that are in fact proper CPDAGs or PDAGs. Note that PC always outputs a proper CPDAG.

We find that NNdisco rather often returns PDAGs, but seldom CPDAGs in most of the evaluated scenarios. Especially for low values of n_{test}, the output is almost never a proper CPDAG, unless the classification threshold is set extremely low.

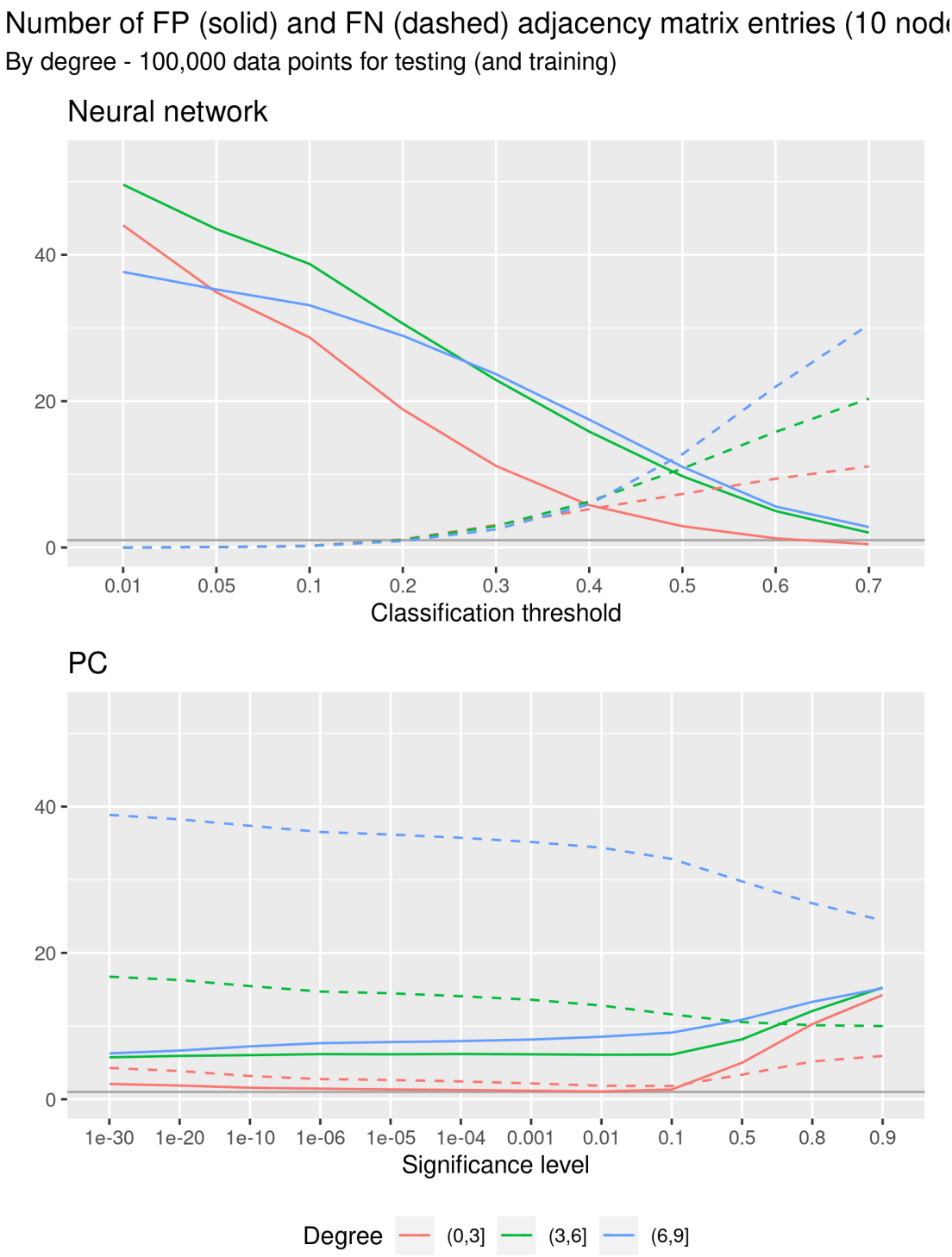
An interesting next step would be to consider constructing the encompassing CPDAG for each pseudo adjacency matrix that is a PDAG but not a CPDAG, and investigate whether this will increase performance.

4.7 Results for p = 10

We here present preliminary results for $p = 10$. We only consider the case $n_{\text{train}} = n_{\text{test}} = 100000$. The NNdisco models have been trained using the N2 network architecture (simple convnet) on $b_{\text{train}} = 100000$ correlation/adjacency matrix pairs and they have been evaluated on $b_{\text{test}} = 2000$ matrix pairs. The neural networks were trained for 200 epochs.

We only present the results stratified by degree, and note that the line color in the plots below now represents average graph degree.

4.7.1 False positives and false negatives



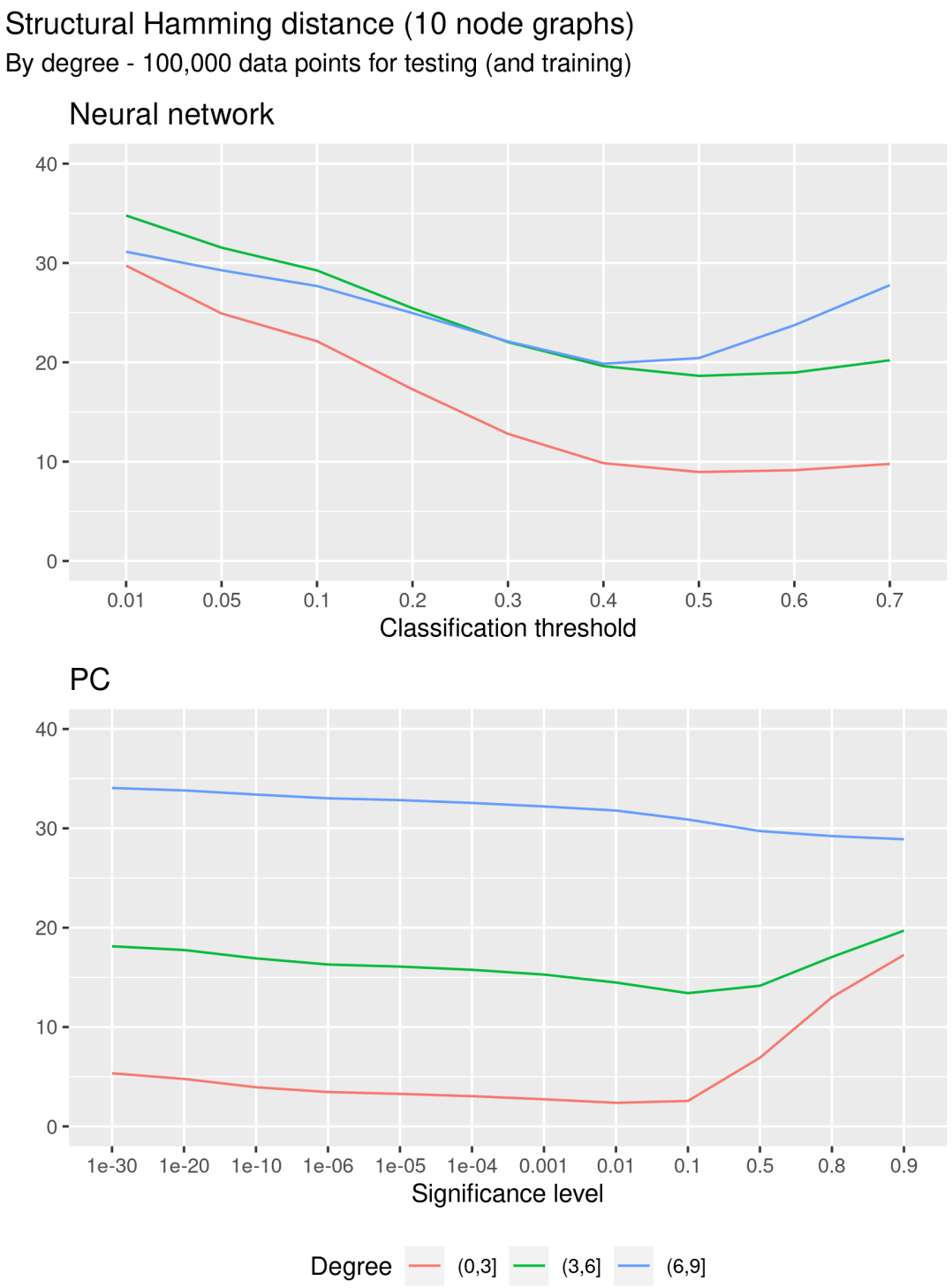
We find very poor control of false positives for NNdisco for all graph degrees. At the classification threshold values where we are still able to obtain low numbers of false negatives, the number of false positives makes the NNdisco methods essentially miss the target.

For PC, we find good control of both false positives and false negatives for sparse graphs, but poor control

for denser graphs. In the sparse scenario (degree (0,3]), $\alpha = 0.1$ again results in best performance, but other significance levels are more optimal for denser graphs.

The poor performance of NNdisco may be due to the training data set being too small. There are 10^5 data points in the training data, and approx. 10^{18} different DAGs of this size – though I am not sure about the number of CPDAGs. It may also be due to the choice of network architecture. The N2 estimates a total of 1741400 parameters, which is most likely not sufficient to express the relationship between correlation and adjacency matrices for 10 node graph.

4.7.2 Structural Hamming distance

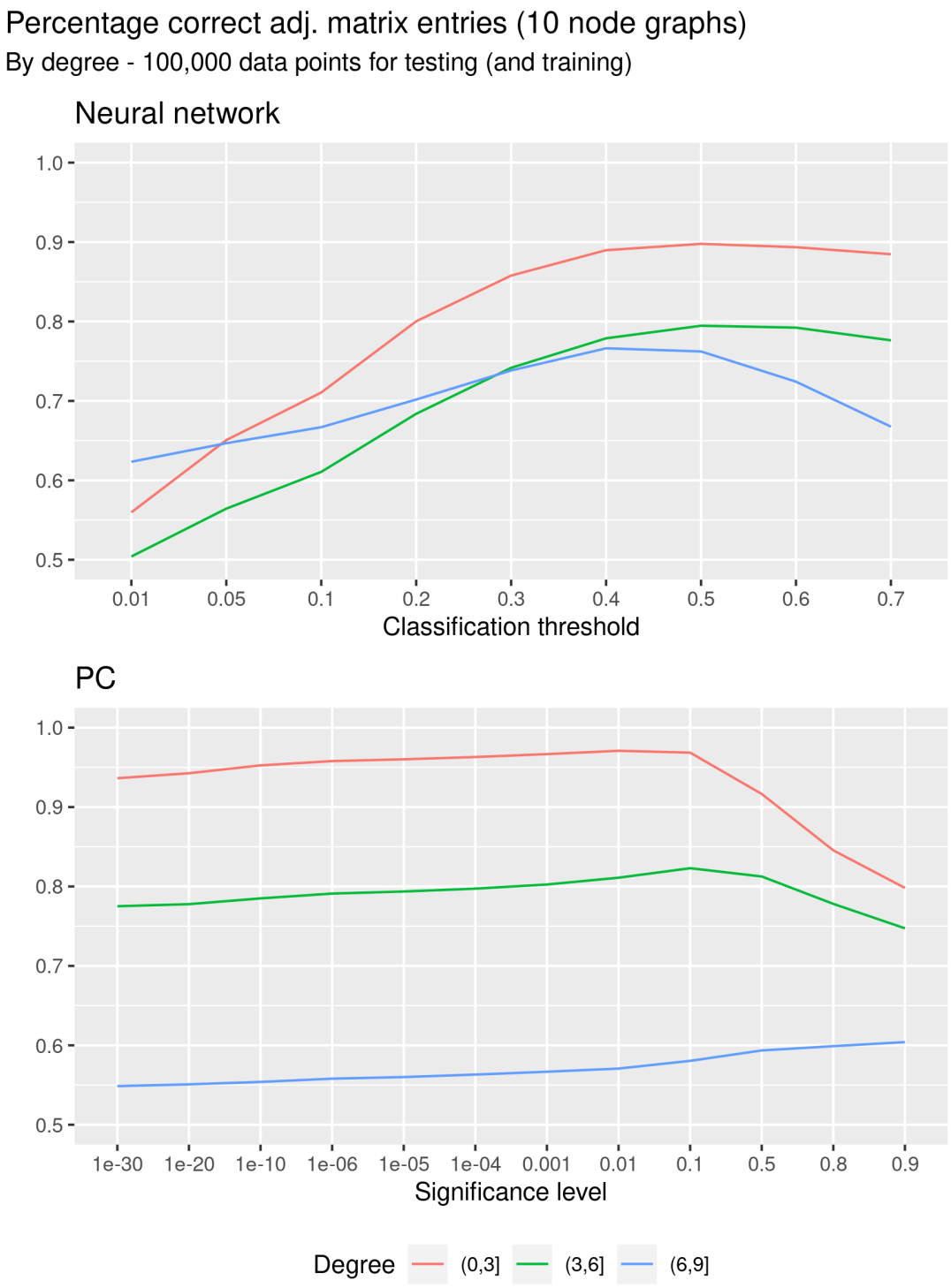


We again see poor performance for the NNdisco method. Even for sparse graphs, where the method performs best, we can not achieve SHD values that make the method useful.

For PC, small SHD values are found for sparse graphs, but not for the other scenarios. We again find $\alpha = 0.1$ to perform best for all but the most dense graphs.

4.7.3 Accuracy

We only provide results for average accuracy over adjacency matrix entries, as there are very few exactly correctly predicted adjacency matrices for $p = 10$.



This performance metric shows the same overall picture as the other two: PC performs well for sparse graphs, and less so for dense graphs. NNdisco overall performs worse, but shows less sensitivity towards graph degree.

5 Questions and further ideas

- How many CPDAGs exist for each p? This will help inform how large b_{train} needs to be.
- How to obtain a more useful measure of performance
  - Ideally, we would want a performance measure that reflects that faulty conclusions for weak and/or unimportant edges are not as problematic as faulty conclusions for stronger and/or more influential edges.
  - Suggestion: Consider (log) rank probability score.
- What loss function to use in order to better consider full output matrix structure, rather than just entry-by-entry
  - Needs to be differentiable (optimization is done using a type of gradient descent)
  - Further practical problems: Needs to be implemented in a way that allows for automatic differentiation in Tensorflow. In practice, this means that it should be specified as a (linear?) combination of basic functions already available from Tensorflow. But there may be a way to work around this.
- Should we be worried about PC performance almost universally being best for $\alpha = 0.1$? Does this reflect that the simulation setup is somehow not general enough?
- Idea: coefficient matrix classification. Instead of trying to estimate the correct CPDAG adjacency matrix, we could instead try to estimate the correct coefficient matrix (for linear structural equation models). A few unresolved questions:
  - How to deal with the fact that the coefficient matrix is not identified from correlation matrices (only up to Markov equivalence class)? We could aim for bounds for causal effects over Markov equivalence class (IDA-like approach), but are such bounds useful in practice and will empirical researchers be able to interpret them?
  - How to generalize from linear structural equation models?
  - How to evaluate performance?
- Idea: Aim to output supergraph of correct CPDAG and apply e.g. PC post hoc to prune spurious edges
- Idea: Try using full data set as input rather than correlation matrix. This will allow for easier generalizations to non-Gaussian data generating mechanism.
- Idea: Train on restricted subsets of DAGs to obtain a method that reflects background knowledge.
- Idea: Simulate data with latent variables to handle data that are not causally sufficient.
- Idea: Aim to learn causal ordering only rather than full adjacency matrix. May be used as helper for other learning algorithms or as a part of NNdisco.
- Idea: Train NNs on inverse of correlation matrix (precision matrix) instead.
  - This describes conditional independencies more directly, so maybe there could be a gain performance.
  - However, matrix inversion is somewhat expensive computationally and the precision matrix only carries information about independence given *all* other variables. I am not sure if we can easily go from the precision matrix of a full dataset to the precision matrix of a subset, which would be necessary.

6 References

Hornik K, Stinchcombe M, White H, et al. (1989) Multilayer feedforward networks are universal approximators. Neural networks 2(5):359–366

Li, H., Xiao, Q., & Tian, J. (2020). Supervised Whole DAG Causal Discovery. arXiv preprint arXiv:2006.04697.

Robinson, R. W. (1973), "Counting labeled acyclic digraphs", in Harary, F. (ed.), New Directions in the Theory of Graphs, Academic Press, pp. 239–273.

Zheng, X., Aragam, B., Ravikumar, P., & Xing, E. P. (2018). Dags with no tears: Continuous optimization for structure learning. arXiv preprint arXiv:1803.01422.