Journal of Machine Learning Research 25 (2024) 1-45 Submitted 8/21; Revised 11/23; Published 10/24

ENNS: Variable Selection, Regression, Classification and Deep Neural Network for High-Dimensional Data

Kaixu Yang KAIXUYANG@GMAIL.COM
LinkedIn Corporation,
Mountain View, California,
USA.

Arkaprabha Ganguli AGANGULI@ANL.GOV
Argonne National Laboratory
Lemont, Illinois,
USA

Tapabrata Maiti MAITI@MSU.EDU
Michigan State University
East Lansing, Michigan USA

Editor: Garvesh Raskutti

Abstract

High-dimensional, low-sample-size (HDLSS) data have been attracting people's attention for a long time. Many studies have proposed different approaches to dealing with this situation, among which variable selection is a significant idea. However, neural networks have been used to model complicated relationships. This paper discusses current variable selection techniques with neural networks. We showed that the stage-wise algorithm with the neural network suffers from some disadvantages, such as that the variables entering the model later may not be consistent. We also proposed an ensemble method to achieve better variable selection and proved that it has a probability tending to zero that a false variable will be selected. Moreover, we discussed further regularization to deal with over-fitting. Simulations and examples of real data are given to support the theory.

Keywords: Deep Neural Network, Variable Selection, Ensemble, High-dimensional data

1. Introduction

High-dimensional statistics modeling (Bühlmann and Van De Geer, 2011) has been popular for decades. Consider a high-dimensional regression or binary classification problem. Let $\boldsymbol{x} \in \mathbb{R}^p$ be the feature vector, and let $y \in \mathbb{R}$ for the regression problem and $y \in \{0, 1\}$ for the classification problem be the response. Our goal is to build a model based on the training sample $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\}$. We have more features than the sample size, i.e., $p > n$. Moreover, many data have complicated relationships among different variables, which is hard to capture through a linear model. A neural network is one of the best models for capturing complicated relationships. Considering a neural network structure between $\boldsymbol{x}$ and y is interesting.

In general, a high-dimensional model does not have consistent estimations since the systems have fewer constraints than a number of variables. Two major approaches can be used to deal with the high dimensionality. The first major approach is to assume that the feature space is sparse, i.e., only a small fraction of the variables are included in the modeling with y. A model with only a fraction of the original features enjoys simplicity and interpretability. Sparse solutions can be obtained using regularization Tibshirani (1996); Huang et al. (2010) or stage-wise algorithms Efron et al. (2004). Regularization obtains a sparse solution by shrinking the coefficients of the unimportant features to zero. The estimated coefficients are shrinkage estimators and thus have a smaller variance Copas (1983). However, regularization with multiple tuning parameters takes longer and may be sensitive to the tuning parameters in practice. Stage-wise algorithms add variables one by one and stop at a preferred time.

The second major approach is projection-based. One finds a lower-dimensional representation of the original feature space. Linear projection methods include the PCA Hotelling (1933) in the low-dimensional case and some of its variants Jolliffe et al. (2003); Zou et al. (2006) in the high-dimensional case. The PCA kernel PCA Schölkopf et al. (1998) performs PCA on a reproducing Hilbert kernel space to achieve nonlinearity. Manifold learning Lawrence (2012) embeds the original feature space in a low-dimensional manifold. Except for manifold learning algorithms that reduce the original dimensionality to 2 or 3 dimensions for visualization, a few manifold learning algorithms include multidimensional scaling (MDS) by Torgerson (1952), the local linear embedding (LLE) by Roweis and Saul (2000), and the Isomap by cites tenenbaum2000global. Applications of the manifold learning algorithms in the high-dimensional setup are studied for specific fields, but a general framework is not available. Another popular dimension reduction technique is the auto-encoder Kramer (1991), which uses a neural network to encode the feature space and decode the representation as close to the original feature space as possible. The above methods are unsupervised, and the lower-dimensional representation is no longer any of the original features. Therefore, we lose interpretability by doing so. Current manifold learning algorithms focus more on data visualization, which reduces the dimensionality to two or three; see, for example, Wang and Zhang (2019). These applications are not helpful in building models.

On the other hand, neural networks have been utilized to model complicated relationships since the 1940s Kleene (1951), and have gained much more attention since the significant improvement in computer hardware in this century. Specifically, Oh and Jung (2004) showed that the computation of neural networks can be significantly improved by GPU acceleration rather than running on the CPU, making it easy to train deeper network structures. Nowadays, variant neural networks are being applied worldwide, including the convolutional neural network (CNN), recurrent neural network (RNN), residual network (ResNet), and etc. In theory, neural networks represent complicated relationships mainly based on the universal approximation theorem Cybenko (1989); Barron (1993); Anthony and Bartlett (2009); Siegel and Xu (2019). The theorem states that a shallow neural network (neural network with one hidden layer) can approximate any continuous function with an arbitrarily small error given a vast number of hidden nodes under mild assumptions. In practice, to reach this good approximation, a huge set of training data is needed since the number of parameters in a neural network is much more than in other models. Moreover, the nonconvexity of a neural network structure makes it impossible to obtain a global optimum. Fortunately, a local optimum of the neural network provides a good approximation.

A deep Neural Network (DNN) is a neural network with a deep structure of hidden layers, which has better performance than a shallow neural network (neural network with only 1 hidden layer) in many aspects, including a broad field of subjects including pattern recognition, speech recognition, etc., see, for example, Mizumachi and Origuchi (2016); Li et al. (2019); Jiang et al. (2019). The deep structure has a greater approximation power than a shallow neural network. Several articles have been published in the literature on the approximation power of deep neural networks Bianchini and Scarselli (2014); Poggio et al. (2017); Shaham et al. (2018); Fan et al. (2020). The results suggest that the use of a deep neural network helps reduce the approximation error, which is useful in the cases where the approximation error dominates the total error. Therefore, it's necessary to consider a deep neural network model over a shallow model. However, finding a way to train the deep neural network well on a small sample size is necessary.

This paper will discuss the stage-wise variable selection algorithm with neural networks. We will show that the existing stage-wise algorithm performs well at the beginning and selects the correct variables. However, in later steps, the probability that it will select a correct variable decreases. Thus, we will propose an ensemble algorithm based on the stage-wise variable selection algorithm, named the ensemble neural network selection algorithm (ENNS). We will show that the new algorithm selects all correct variables with high probability and its false positive rate converges to zero. Moreover, instead of a regular neural network trained on select variables, we proposed a few methods to further reduce the variance of the final model. We will give a numerical comparison of these proposed algorithms and propose an algorithm for the l_1 penalized neural network with soft-thresholding operators.

In Section 2, we will discuss some major related works and the intrinsic dimensionality of a model. In Section 3, we will present the ideas and algorithms behind the ENNS algorithm and the methods of increasing stability during the estimation step. In Section 4, we will provide the theory for the arguments in this paper. In Section 5, we will present the results of some numerical studies to support our theorems and arguments. Section 7 will discuss some findings and future work.

2. Related works

In this section, we will discuss some major related works

2.1 The regularization approach

The rich literature has discussed achieving sparsity via regularization, for example, see (Tibshirani, 1996; Fan and Li, 2001; Zou, 2006; Yuan and Lin, 2006; Liu and Wu, 2007; Fan and Lv, 2008; Huang et al., 2010; Marra and Wood, 2011). Let $\boldsymbol{\theta}$ be the parameters of a model; a direct method is to add a zero norm of the parameters $\|\boldsymbol{\theta}\|_0$ to the loss function. However, optimizing a loss function with zero norms has been proved to be a non-deterministic polynomial-time hardness (NP-hard) problem Natarajan (1995), which requires exponential time to solve. However, it has been proved that instead of directly penalizing the number of nonzero coefficients by l_0 norm, l_1 type penalty can shrink some coefficients to zero, and thus the features corresponding to these coefficients are not included in the model (Tibshirani (1996), and this can be extended to adopt group-wise penalizationYuan and Lin (2006);

Huang et al. (2010)). The power of regularization is decided through the hyperparameter, which is also called the tuning parameter, by checking some criterion (BIC Schwarz et al. (1978), EBIC Chen and Chen (2008), GIC Zhang et al. (2010); Fan and Tang (2013)) or cross-validations.

Variations of the l_1 norm regularization are also widely used. It's known that l_1 norm regularization yields sparse solution (Tibshirani (1996)), while l_2 norm regularization controls the magnitude of coefficients and kills multicolinearity and overfitting. The l_1+l_2 norm regularization both yields sparse solution and encourages group effects(Zou and Hastie (2005)). The $l_{p,1}$ norm penalization (Yuan and Lin (2006), Argyriou et al. (2007)), where $p = 2$ matches the group lasso penalty, yields group sparsity. The $l_{p,1}, l_1$ norm penalty, known as the sparse group lasso penalty Simon et al. (2013), yields both group sparsity and in-group sparsity solutions. Adaptive methods (Zou (2006)) assign different weights to different coefficients using a data-driven method, which obtains oracle solutions. Some variations of norm regularization methods include the Dantzig selector (Candes et al., 2007), which is a variation of the lasso. The SCAD (Fan and Li, 2001) is defined from the derivative, instead of directly defined from the penalty term.

The regularization approach requires the assumption that only a few features are relevant in predicting the response, all other features either have exactly zero coefficients (strong sparsity assumption), for example, in the linear regression setup

$$\beta_j = 0, \quad for\ j \in A^0 \subset \{1, ..., p\} \qquad and \qquad \#\{\beta_j,\ j \notin A^0\} = q << p,$$

or the coefficients of irrelevant features are bounded from above by a negligible term (weak sparsity assumption)

$$\sum_{j \in A^0} |\beta_j| \leq \eta \qquad and \qquad \#\{\beta_j,\ j \notin A^0\} = q << p$$

(for example, see Zhang and Huang (2008)). If the sparsity assumption is satisfied, the regularization approach with property penalty will yield a sparse solution with a high probability that the true subset of relevant features is selected under some common mild conditions. An advantage of this approach is that we know which features are selected; thus, the model has rich interpretability.

2.2 Deep neural network approximation

It's well known as the universal approximation theorem by Cybenko (1989) that a shallow neural network with k hidden nodes, denoted $SN_k(x)$ can be used to approximate any continuous function $f(x)$ defined in a bounded domain with arbitrarily small error

$$|SN_k(x) - f(x)| < \epsilon$$

for all x in the bounded domain with a big enough k. This pioneering theorem encourages the use of neural networks in a wide way. Later, with the development of deep neural networks, people found limitations of the shallow neural network such that the number of neurons needed to achieve a desired error increases exponentially Chui et al. (1994, 1996). After that, people found that "the two hidden layer model may be significantly more promising than the single hidden layer model" Pinkus (1999). Sum neural networks, or equivalently, polynomial neural networks have been studied Delalleau and Bengio (2011); Livni et al. (2013), and universal approximation property has been established recently by Fan et al. (2020) that a continuous function in $\mathcal{F}_d^n$ can be approximated with error ϵ by a quadratic network who have depth

$$O\left(\log(\log(\frac{1}{\epsilon})) + \log(\frac{1}{\epsilon})\right)$$

and the number of weights

$$O\left(\log(\log(\frac{1}{\epsilon}))(\frac{1}{\epsilon})^{d/n} + \log(\frac{1}{\epsilon})(\frac{1}{\epsilon})^{d/n}\right)$$

where d is the dimension of the domain. The approximation theory for regular deep neural networks has also been established recently. Poggio et al. (2017) showed that a deep network need

$$O\left((n-1)\left(\frac{\epsilon}{L}\right)^{-2}\right)$$

model complexity to approximate a L-Lipshitz continuous function of n variables instead of

$$O\left(\left(\frac{\epsilon}{L}\right)^{-n}\right)$$

in a shallow neural network. Shaham et al. (2018); Siegel and Xu (2019) provides more detailed results for the approximation power of deep neural networks.

2.3 Variable selection and regularization in neural networks

In terms of variable selection in neural networks, Castellano and Fanelli (2000) proposes an algorithm to prune hidden nodes in a low-dimensional setup, and Srivastava et al. (2014) proposes the dropout technique to eliminate hidden nodes randomly. These methods set coefficients to zero and thus reduce the generalization variance but do not help in the high-dimensional setup, where one needs to eliminate input features. La Rocca and Perna (2005) studied a test procedure to select features in the time series scenario, but this cannot be generalized to a greater scenario.

In the high-dimensional setup, a neural network has even more parameters and thus is harder to train than in the low-dimensional setup. When we have a sample size compared to a huge number of parameters, a neural network usually has a high variance. A few literature is available in studying this property Feng and Simon (2017); Liu et al. (2017); Yang and Maiti (2020); Lemhadri et al. (2019). Via group lasso regularization Yuan and Lin (2006), one can shrink the whole connections of a specific variable to zero exactly,, with the development of deep neural networks, people found the limitations of shallow neural networks such that the number of neurons needed to achieve a desired sensitive to a small change in the tuning parameters in practice. Liu et al. (2017) also presents a stage-wise variable selection algorithm with neural networks called deep neural pursuit (DNP), which uses correlation to add new variables and enjoys faster speed. These methods are extensions of the high-dimensional linear models or additive models, which act as pioneers of this hot topic.

2.4 Algorithms

Regularization methods usually involve the penalty term l_1 of the norm, which is not easy to solve using regular gradient descent algorithms; see, for example, Wright (2015). This issue is general for all models with l_1 penalty. The regularization path for generalized linear models can be easily obtained from the coordinate descent algorithms (Wu et al., 2008; Friedman et al., 2010).

Various algorithms are used to obtain a path selection. The least angle regression Efron et al. (2004) provides a forward algorithm to add new features by looking at the correlation. The LARS algorithm, with a simple modification, can be used to obtain the lasso solution path. Tibshirani (2015) provides a stage-wise algorithm, which provides a very close solution path to the lasso solution path. Perkins et al. (2003) studied a stage-wise algorithm to incorporate the l_2, l_1 and l_0 norm penalty with the gradients with respect to the input weights. The gradient implicitly connects with the correlation studied in Efron et al. (2004).

As for the other penalties, Tewari et al. (2011) has shown an equivalence between using the stage-wise algorithm and the group lasso penalty. Liu et al. (2017) has applied the result on deep artificial neural networks for feature selection.

3. The two-step variable selection and estimation approach

Consider a feature vector $\boldsymbol{x} \in \mathbb{R}^p$ and a response $y \in \mathbb{R}$ for the regression setup and $y \in \{0, 1\}$ in the classification setup. We have data $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\}$ consisting of independent observations. Denote the design matrix $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)^T \in \mathbb{R}^{n\times p}$ and the response vector $\boldsymbol{y} = (y_1, ..., y_n)^T$. As mentioned before, we have more variables than observations, i.e., $p > n$. According to the previous discussion, variable selection is an important step in high-dimensional modeling. If one includes all variables in the model, there will be at least p parameters to estimate, which can not be done stably with the n observations. If a more complicated model is needed, the number of parameters will be tremendous, which will cause severe overfitting and high variance with a small training sample size.

Therefore, we hope that a feature selection step at the beginning can help to pick the import variables, and another estimation step could build a more accurate model based on the selected variables. Moreover, we will use deep neural networks as the structure since they can capture complicated relationships. We will consider a stage-wise algorithm in the variable selection step, performing a function similar to the DNP model in Liu et al. (2017). However, we will show that the stage-wise algorithm in DNP suffers from some disadvantages and propose an ensemble algorithm to relieve this situation. In the second step, we will discuss the methods to reduce variance and prevent over-fitting, since a deep neural network with only a few input variables can still have a huge number of parameters.

3.1 The ensemble neural network selection (ENNS) algorithm

Consider the feature selection approach in Liu et al. (2017). Let $\mathcal{D} : \mathbb{R}^p \to \mathbb{R}$ be a deep neural network function that maps the original feature space to the output space. We don't specifically mark the number of hidden layers and node sizes in the notation but assume the deep neural network has m hidden layers with sizes $h_1, ..., h_m$. Denote the weight matrices in each layer to be $\boldsymbol{W}_0, ..., \boldsymbol{W}_m$, where $\boldsymbol{W}_0 \in \mathbb{R}^{p\times h_1}$, $\boldsymbol{W}_i \in \mathbb{R}^{h_i \times h_{i+1}}$ for $i = 1, ..., m-1$ and $\boldsymbol{W}_m \in \mathbb{R}^{h_m \times 1}$. Denote $\boldsymbol{t}_i$ the intercept for the i^{th} hidden layer and b the intercept of the output layer. Let $\boldsymbol{\theta} = (\boldsymbol{W}_0, ..., \boldsymbol{W}_m, \boldsymbol{t}_1, ..., \boldsymbol{t}_m, b) \in [-W, W]^{|\boldsymbol{\theta}|}$ be the parameters in the neural network model. For an input $\boldsymbol{x} \in \mathbb{R}^p$, denote the output

$$\eta_{\boldsymbol{\theta},\boldsymbol{x}} = \mathcal{D}_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{1}$$

where in the regression setup, the output is from a linear layer and $\eta \in \mathbb{R}$, while in the classification, an extra sigmoid layer is added and $\eta \in (0, 1)$. Moreover, we assume sparse features, i.e., only a small fraction of the variables are significantly related to the response. Without loss of generality, we assume

$$\mathcal{S}_0 = \{1, ..., s\}$$

of the variables are truly non-zero variables.

Define the loss function for regression to be the squared error loss

$$l(\boldsymbol{\theta}) = \mathbb{E}\left[(y - \eta)^2\right] \tag{2}$$

In practice, we work with the empirical loss

$$l(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{n}\|\boldsymbol{y} - \boldsymbol{\eta}\|_2^2 \tag{3}$$

where $\boldsymbol{\eta} \in \mathbb{R}^n$ with $\eta_i = \eta_{\boldsymbol{\theta},\boldsymbol{x}_i}$, $i = 1, ..., n$. Define the loss function for classification to be the negative log-likelihood, which is known as the cross-entropy loss

$$l(\boldsymbol{\theta}) = \mathbb{E}\left[y \log \eta + (1 - y) \log(1 - \eta)\right] \tag{4}$$

In practice, we work with the empirical loss

$$l(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \eta_i + (1 - y_i) \log(1 - \eta_i)\right] \tag{5}$$

Let $\boldsymbol{G}_i$ be the gradient of the loss function with respect to $\boldsymbol{W}_i$ in the back propagation process for $i = 0, ..., m$, i.e.,

$$\boldsymbol{G}_i = \frac{\partial}{\partial \boldsymbol{W}_i} l(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}), \quad i = 0, ..., m \tag{6}$$

The DNP algorithm starts with the null model and adds one variable at a time. Let $\mathcal{S}$ be the selected set and $\mathcal{C}$ be the candidate set. At the beginning, we have

$$\mathcal{S} = \{intercept\} \quad and \quad \mathcal{C} = \{1, ..., p\} \tag{7}$$

The model is trained on $\mathcal{S}$ only and the submatrix of $\boldsymbol{W}_0$ corresponding to the features in $\mathcal{C}$ is kept zero. After training, one chooses a l_q norm (usually with $q = 2$) and computes the gradient's norm for each $j \in \mathcal{C}$ of $\boldsymbol{W}_0$.

$$\boldsymbol{G}_{0j} = \frac{\partial}{\partial \boldsymbol{W}_{0j}} l(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}), \quad j \in \mathcal{C} \tag{8}$$

The next variable that enters the model, j_+ is

$$j_+ = \arg\max_{j \in \mathcal{C}} \|\boldsymbol{G}_{0j}\|_q \tag{9}$$

Then $\mathcal{S} = \mathcal{S}/j_+$ and $\mathcal{C} = \mathcal{C} \cup \{j_+\}$ To increase the stability, instead of computing $\boldsymbol{G}_{0j}$ directly, the DNP algorithm computes $\boldsymbol{G}_{0j}$ through the average over multiple dropouts. Let B_1 be the number of dropouts, the next variable is

$$j_+ = \arg\max_{j \in \mathcal{C}} \frac{1}{B_1} \sum_{b=1}^{B_1} \|\boldsymbol{G}_{b0j}\|_q \tag{10}$$

where $\boldsymbol{G}_{b0j}$ denotes the gradient of the loss function with respect to the first layer's j^{th} weight vector after the b^{th} random dropout.

The algorithm works because $\|\boldsymbol{G}_{0j}\|_q$ describes how much the loss function will change when the next update is performed on the weight of the corresponding variable Perkins et al. (2003). Tewari et al. (2011) also indicates that the selection of the variable by comparing $\|\boldsymbol{G}_{0j}\|_q$ is equivalence to applying the group lasso penalization; see also Liu and Wu (2007).

The algorithm works well at the very beginning, which is described by proposition 4.1 and proposition 4.2 in section 4. However, it suffers from a few disadvantages. First, as we include more correct variables in the model, the probability that we select another correct variable decreases. A simulation study in section 5 also provides numeric support for this argument. Second, one needs to pre-specify how many variables must be selected before stopping, denoted s_0. If this number is more than the number of true variables, denoted s, there will be $s_0 - s$ variables that should not be included but were included, i.e., the false positive rate could be high. Finally, the model does not use dropout or regularization during prediction, which has potential over-fitting risk. Here, we propose the ensemble neural network selection (ENNS) algorithm to remedy these issues, and we will discuss possible solutions for preventing over-fitting in the prediction step.

One could observe that when a fraction of $\mathcal{S}_0$ is already involved in the model, i.e., in $\mathcal{S}$, the model is trained such that these variables are used to explain the variations by all variables in $\mathcal{S}_0$. This weakens the effect of those truly nonzero variables in $\mathcal{C}$. These variables become less important than when no variable in $\mathcal{S}$. Moreover, there are fewer truly nonzero zero variables in $\mathcal{C}$ than at the very beginning, the probability that we select a correct variable in the next step is

$$\mathbb{P}(j_{next} \in \mathcal{S}_0) = \sum_{j \in \mathcal{S}_0 \cap \mathcal{C}} \mathbb{P}(j_{next} = j) \tag{11}$$

which will be even lower as $|\mathcal{S}_0 \cap \mathcal{C}|$ decreases. Therefore, there will be a nonzero probability that at one stage the selected variable does not belong to $\mathcal{S}_0$. This is described in section 4. We consider an ensemble method to remedy this issue.

The idea behind this ensemble method is similar to bagging Breiman (1996); Bühlmann et al. (2002). Assume that we want to add s_j variables in one step. Consider a bootstrap sample of size n_b from the original sample. The DNP with random initialization is trained on this sample, which yields a selection set $\mathcal{S}_1 = \{j_1, ..., j_{s_j}\}$. Rather than just making one pass, we propose that for b_2 in $1, ..., B_2$ and a bootstrap sample size n_r, we perform the feature selection on a random selection of n_r observations. Denote the features being selected in all B_2 rounds as

$$\mathcal{J}_1 = \{j_{11}, ..., j_{1s_j}\}, ..., \mathcal{J}_{B_2} = \{j_{B_21}, ..., j_{B_2s_j}\}$$

We will only allow a variable to enter the model if it appears at least $[B_2p_s]$ times in the B_2 rounds, for a fixed proportion p_s. Mathematically,

$$\mathcal{J} = \{j \ in \ at \ least \ [B_2p_s] \ of \ \mathcal{J}_1, ..., \mathcal{J}_{B_2}\}$$

is the set of variables that will enter the model in this step.

The reason that this ensemble will improve the selection lies in three points. First, the algorithm is an averaging of different bootstrapping results thus the effect of some extreme observations could be averaged out. The final selection result represents the common part of the whole sample. Secondly, neural network uses random initialization. Though the predictions seem similar in two different training, the estimated parameters are actually from different local minimums of the loss function. Therefore, these different training results represent different aspects of the model. Combining the two reasons, the selection results are closer to independence if we select a smaller n_b compared to n. However, n_b can not be too small to avoid misleading the neural network. Finally, if a variable is selected by mistake in some round, this is possibly due to the specific bootstrap sample making the relationship between the variable and the response stronger, which is not general in all bootstrap samples. In practice, one will observe that though false selection happens, those false variables are different in different rounds. Therefore, this ensemble will actually make the probability of false selection tend to zero. This result is described by theorem 4.1 in section 4. Moreover, if two variables' interaction effect is important in the model, they are likely to be included in the same step.

The proposed method may select fewer or more variables than the number of variables we specified, s_j. If the sample is not enough to represent the true relationship between the variables and the response, it's possible that the number of selected variables denoted $\hat{s}_j$, is less than s_j. In this case, we exclude the variables that are already in C from the neural network and perform another round of variable selection with $\mathcal{S} = \{1, ..., p\}/\mathcal{C}$ and then $\mathcal{C} = \{intercept\}$. The number of variables selected in this round will be $s_j - \hat{s}_j$. On the other hand, $\hat{s}_j$ being more than s_j occurs when the selection proportion p_s is specified too small. In this case, the variables would be sorted by their proportions that appeared and only select the first s_j variables in the list.

In summary, we specify a number s_0 at the beginning, which means the final model will include s_0 variables. In the j^{th} iteration, let s_j be the number of variables to be selected. Right now there are $|\mathcal{S}_j|$ variables in the model, denoted $\mathcal{S}_j$. Let $\boldsymbol{X}_{-n}$ be the sub-matrix of $\boldsymbol{X}$ where the columns with indices in $\mathcal{S}_j$ removed. Train the ensemble on $\boldsymbol{X}_{-n}$ and obtain selection result $\hat{\mathcal{S}}_j$. Let $s_{j+1} = s_j - |\hat{\mathcal{S}}_j|$. The algorithm is repeated until the model has selected s_0 variables. An algorithm is given in Algorithm 1. Under mild assumptions, the algorithm will finally reach selection consistency. This argument is described in theorem 4.1 in section 4. Moreover, a comparison of the variable selection performances of different modeling is presented in section 5.

The computation complexity of the ENNS algorithm on a single machine is the number of bagged neural networks times the computation complexity of training a single neural

```
Initialize number of selected variables $S = \emptyset$, $s = 0$ and target $s_0$;
while $|S| < s_0$ do
    for $b = 1, ..., B$ do
        Bootstrap sampling;
        Random initialization with zero feature;
        Run the DNP algorithm and obtain selection set $\mathcal{J}_b$;
    end
    Obtain $\mathcal{J} = \bigcup_{b=1}^{B} \mathcal{J}_b$;
    Compute $\mathcal{J}_T$ by filtering number of appearance;
    if $\mathcal{J}_T <= s_0 - |S|$ then
        $S = S \cup \mathcal{J}_T$;
        Remove the columns in $\mathcal{J}_T$ from training data;
    else
        $\mathcal{J}_T$ is the $m - s$ elements with highest number of appearance;
        $S = S \cup \mathcal{J}_T$;
    end
end
```

Algorithm 1: Algorithm for feature selection ENNS

network, which is equal to $O(Bhsnp)$. Here B is the number of bagged neural networks, h is the neural network structure complexity, s is the number of variables to be selected, n is the sample size and p is the variable dimension. However, since the bagging algorithm has independent elements, it's easy to parallelize the bagged neural networks by submitting different jobs. In this case, the computation complexity reduces to $O(hsnp)$, which is the same as that of DNP in Liu et al. (2017). As a comparison, Liu et al. (2017) also mentioned the computation complexity of HSIC-Lasso in Yamada et al. (2014), which grows cubicly with the sample size as $O(sn^3p)$.

3.2 Estimation with regularization

In this subsection, we will discuss possible procedures to prevent over-fitting. After feature selection, the deep neural network can be trained on the selected features. However, the number of parameters in the neural network model is still huge. A 4 hidden layer neural network with s selected variables and hidden layer sizes $h_1, ..., h_4$ has $sh_1 + h_1h_2 + h_2h_3 + h_3h_4 + h_4$ parameters (without counting the intercepts). For example, if we use the common hidden layer sizes $[50, 30, 15, 10]$ with the number of selected variables being 5, this brings 2466 parameters. As a comparison, the linear model has 6 parameters, while the GAM with 4 knots and degree 3 has 36 parameters. Compared to the number of parameters, the small sample size is still a challenging issue. Therefore, we need to be careful in training the neural network on the selected variables. A few methods are discussed below. Moreover, the Xavier initialization Glorot and Bengio (2010) is used here to ensure that the initial weights are in a proper range.

3.2.1 Dropping out and bagging

In the variable selection step, over-fitting is overcome by dropout layers, where we randomly set parameters to zero in the later layers. However, using dropout layers in prediction is risky, since we cannot measure the performance of doing a random dropout. One way is to use bagging again in this step. First, the connections in the estimated neural network, denoted $\mathcal{N}$, are randomly cut off, i.e., the weights are set to zero. By doing this, we obtained $\mathcal{N}_r$, where r stands for reduced. Then a prediction is made on model $\mathcal{N}_r$, denoted $\hat{y}_r$. This process is repeated for K times. Denote the reduced neural networks to be $\mathcal{N}_{kr}$ and their predictions with $\hat{y}_{kr}$. In the regression set up, the final prediction is defined as

$$\hat{\boldsymbol{y}} = \frac{1}{n}\sum_{k=1}^{K} \hat{y}_{kr}$$

In the classification set up, the final prediction is defined as

$$\hat{y}_i = \begin{cases} 1, & if\ \hat{p}_i > p_c \\ 0, & if\ \hat{p}_i < p_c \end{cases}, \quad for\ i = 1, ..., n$$

where p_c is some pre-specified threshold.

$$\hat{p} = \frac{1}{n}\sum_{k=1}^{K} \hat{y}_{kr}$$

and $\hat{p}_i$ is the i^{th} element of $\hat{p}$. A simulation study is performed in Section 5.

3.2.2 Stage-wise training

The stage-wise training idea comes from Liu et al. (2017), where the authors used it as a step-wise variable selection technique. However, here we adopt the idea of training the final model on the selected variables. The intuition behind this is that at each step, the information that is already trained remains in the training process. Therefore, adding a new variable adjusts the previous trained weights. Moreover, training with adaptive gradient algorithm (Adagrad, Duchi et al. (2011)) allows adaptive learning rates for different parameters and thus ensures faster and more accurate convergence. In detail, assume that we have selected m variables $\mathcal{J}$ from the ENNS algorithm. Let $\boldsymbol{X}_{\mathcal{J}}$ be the sub-matrix of $\boldsymbol{X}$ whose columns' indices are in $\mathcal{J}$. Then, the DNP algorithm in Liu et al. (2017) is trained on $\boldsymbol{X}_{\mathcal{J}}$ with $|\mathcal{J}|$ being the target number of variables. A simulation study is performed in Section 5.

3.2.3 l1 norm regularization

It's mentioned in section 2 that l_1 regularization gives sparse neural network and controls over-fitting by shrinking parameters towards zero, and some parameters can be shrunk to zero exactly. Therefore, we choose to use l_1 norm regularization to control the parameter size and the number of nonzero parameters.

Let $\hat{S}$ be the set of indices of the variables that are selected from the first step. Let $\boldsymbol{\Theta} = \theta_1, ..., \theta_L$ be the hidden layer weights and $\boldsymbol{T} = t_1, ..., t_L$ be the hidden layer intercepts (including the output layer). Let $f(x;\mathbf{\Theta},\boldsymbol{T})$ be the neural network structure with such parameters that maps the original input to the output. In the classification problem, define

$$\hat{\mathbf{\Theta}},\hat{\boldsymbol{T}} = \underset{\mathbf{\Theta},\boldsymbol{T}}{\arg\min} -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i f(\boldsymbol{x}_{\hat{S},i}) - \log(1+\exp(f(\boldsymbol{x}_{\hat{S},i})))\Big] + \sum_{l=1}^{L}\lambda_{nl}|\theta_L|, \tag{12}$$

where $\boldsymbol{x}_{\hat{S},i}$ denotes the i^{th} observation with only the selection variables included.

Direct training of the loss function 12 with the built-in loss penalty l_1 directly added to the loss of cross-entropy does not work well in current neural network libraries, including tensorflow and pytorch. Therefore, a coordinate descent algorithm is needed to obtain sparsity in the neural network. Define the soft-thresholding operator $S(\cdot,\cdot):\mathbb{R}^d\times\mathbb{R}\to\mathbb{R}^d$ as

$$(S(\boldsymbol{x},c))_i = sign(x_i)(|x_i|-c)_+, \quad i=1,...,d. \tag{13}$$

The algorithm consists of an iterative process of updating the weights of the neural network without the penalty l_1 and then applying the soft-thresholding operator 13. The number of epochs is pre-specified. However, the performance on the validation set can be monitored, and an early stopping criterion can be specified. The training will be stopped if the performance on the validation set does not improve for a pre-specified number of patience level. It worth noting that instead of selecting the tuning parameter, a sparsity level of each layer can be specified. Assuming there are M hidden layers with sizes $h_1,h_2,...,h_M$. One may specify percentile p_m for $m=1,...,M$. Denote W_m the weight of layer m and W_{p_m} the p_m^{th} percentile of W_{p_m}. Then for layer m, the soft-thresholding operator can be applied as $S(W_m,W_{p_m})$. For example, choosing a percentile of 50 will make a certain layer have 50% sparsity level. An algorithm is given in Algorithm 2. A simulation that compares the built-in l_1 penalty and the soft-thresholding operator is given is Section 5.

```
Initialize the weights with Xavier initialization;
while Early stopping False OR epochs < k do
    One step gradient descent for the neural network part;
    for weights in layers do
        Apply the soft-thresholding function with a pre-specified percentile;
    end
    Check early stopping criterion;
end
```

Algorithm 2: Algorithm for l_1 norm estimation using coordinate descent

4. Theoretical Guarantee

In this section, we will develop theoretical support for the proposed methodology. The methodology supports the intuitions used in the method. A few assumptions are made in the derivations of the theorems. The first famous assumption in high-dimensional modeling is sparsity.

Assumption 1 (Sparsity) *The features are sparse, i.e., only $s < o(n)$ of the $p = o(c_1 e^{n^{c_2}})$, $c_1 < 0, 0 < c_2 < 1$ variables are strongly related to the response. Specifically, we assume $y = f^*(X_S) + \epsilon, \epsilon \sim N(0, \sigma^2)$, where $f^*(\cdot)$ is the true signal, which is a bounded and continuously differentiable function depending on only the relevant features $X_S, |S| = s$.*

Assumption 2 (Hierarchy principle) *The signal strength of a feature X_j can be defined as $\beta_j = X_j' f^*(X_S)$. It prioritizes the linear aspect of the signal over the nonlinear component and employs it to steer feature sparsity. Additionally, to distinguish the null and non-null features, we assume, $\min_{j \in S} |\beta_j| \geq \gamma_n$ and $\max_{j \in S^c} |\beta_j| \leq o(e^{-p})$, where γ_n is a sequence that may go to zero, as nm goes to infinity.*

Assumption 2 accentuates the predominance of the linear aspect within the signal over the nonlinear counterpart, utilizing it as a guide for inducing sparsity among features. This approach closely parallels the hierarchy principle, extensively explored in the field of statistics (Choi et al. (2010), Yan and Bien (2017)).

Assumption 3 (Design matrix) *Considering a fixed design case, we assume the features are bounded in $[-\gamma, \gamma] \in \mathcal{R}$, centered, and $||X_j||_2^2 \leq \tau, j \in \{1, 2, \ldots, p\}$. Additionally, to maintain the correlation strength among the features, assume that $|X_1'\epsilon|, |X_2'\epsilon|, \ldots, |X_p'\epsilon|$ are associated; i.e. for any non-decreasing function $f(\cdot), g(\cdot) : \mathcal{R}^p \to \mathcal{R}$,*

$$cov(f(|X_1'\epsilon|, |X_2'\epsilon|, \ldots, |X_p'\epsilon|), g(|X_1'\epsilon|, |X_2'\epsilon|, \ldots, |X_p'\epsilon|)) \geq 0 \tag{14}$$

We acknowledge that in most instances, the predictors exhibit interdependence, or at least a subtle correlation. Nonetheless, even a faint correlation introduces significant intricacy to the methodology. Thus, we hypothesize that while the features might exhibit correlation among themselves, their correlation with the random noise should be close to zero and thus they should be associated (Property P4 of associated random variables in Esary et al. (1967)). Moreover, the assumption outlined in 2 regarding the minute signal strength β_j of null features indirectly constrains the relationship between null and non-null features. To comprehensively explore correlated predictor scenarios, we will conduct an extensive simulation study in 5.

For the remainder of our theoretical investigation, we adopt the aforementioned assumptions for ease of the demonstration. The following two propositions illustrate how the probability of choosing one variable over another in the first step is decided. The first proposition gives the probability that we select one variable over another, and the second proposition gives the probability that we will select a correct variable in the first step.

Proposition 4.1 *Recall that we will select predictor j if $j = \arg\max_j c_j$, where c_j is the L_2 norm of the gradient with respect to the j^{th} input. Hence, c_j being the criterion to select predictor j, we have*

$$\mathbb{P}(c_j < c_k) = 2L(h_1, -\delta, \rho_1) + 2L(h_2, \delta, \rho_2) + \Phi(h_1) + \Phi(h_2) - 2 \tag{15}$$

where

$$L(a,b,\rho) = \mathbb{P}(Z_1 > a, Z_2 > b), (Z_1, Z_2) \sim N_2(0,0,1,1,\rho), \tag{16}$$
$$h_1 = \frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 - 2\rho\sigma_1\sigma_2 + \sigma_2^2}}, h_2 = \frac{\mu_1 + \mu_2}{\sqrt{\sigma_1^2 + 2\rho\sigma_1\sigma_2 + \sigma_2^2}} \tag{17}$$
$$\rho_1 = \frac{\sigma_2 - \sigma_1\rho}{\sqrt{\sigma_1^2 - 2\rho\sigma_1\sigma_2 + \sigma_2^2}}, \rho_2 = \frac{\sigma_2 + \sigma_1\rho}{\sqrt{\sigma_1^2 + 2\rho\sigma_1\sigma_2 + \sigma_2^2}}, \tag{18}$$
$$\delta = \frac{\mu_2}{\sigma_2} \tag{19}$$

with $\mu_1 = X_j' f^*(X_S), \mu_1 = X_k' f^*(X_S), \sigma_1^2 = \sigma^2 ||X_j||_2^2, \sigma_2^2 = \sigma^2 ||X_k||_2^2, \rho = X_j' X_k$ *and* $\Phi(\cdot)$ *is the standard normal distribution CDF.*

Proposition 4.2 *Under assumptions 1, 2 and 3, the probability that we select a nonzero predictor at the very beginning using the stage-wise neural network selection is*

$$\mathbb{P}(\textit{A nonzero predictor enters the model first}) = \sum_{k=1}^{s} \int_0^\infty f_k(x) \prod_{j \neq k}^{p} F_j(x) dx \tag{20}$$

where

$$F_k(x) = \frac{1}{2}\left[erf\left(\frac{x + |\beta_k|}{\sqrt{2\sigma^2}}\right) + erf\left(\frac{x - |\beta_k|}{\sqrt{2\sigma^2}}\right)\right] \quad \textit{and}$$
$$f_k(x) = \frac{\partial}{\partial x} F_k(x) = \sqrt{\frac{2}{\pi\sigma^2}} e^{-\frac{x^2+\beta_k^2}{2\sigma^2}} \cosh\frac{\beta_k x}{\sigma^2} \tag{21}$$

and $erf(\cdot)$ *is the error function. Moreover, if* $\beta_{max} = \max_{j=1,\ldots,s}$ *is bounded, as* $s \to \infty$, *the probability is bounded from above*

$$\mathbb{P} \leq 1 - \delta$$

where δ *is a nontrivial quantity.*

Proofs of the two propositions are given in the Appendix. Proposition 4.1 and 4.2 describe the behavior of neural network stage-wise algorithm at the very beginning. The probability that we select one predictor over another depends on the sum of their signal strength and the difference in their signal strength. The greater the difference, the higher the probability that we will select the predictor with higher signal strength. The probability that we will select a correct predictor at the very beginning is described by the error function and standard normal density functions. Though the form of the probability looks complicated, since the error function can be approximated by an exponential function with proper constants, we can show that in some cases, it is not guaranteed that a variable entering the model first is a nonzero variable. Specifically, this happens when we have a low signal strength or a huge number of candidate variables.

So there is a concern that a wrong variable will mistakenly enter the model due to a special training case of the neural network model, as shown in the previous proposition. With the bagging algorithm, we are able to eliminate the false positive selections with probability tending to 1. The intuition is that false positive selection of a certain predictor happens due to a specific observation of the design matrix, which appears to be more correlated to the response or residual. However, with different sub-samplings, it's very unlikely that they yield the same wrong selection. In variable selection algorithms, the most important property is to be able to select the correct predictors consistently. Here we show that ENNS enjoys this property in the following theorem.

Theorem 4.1 *Assuming sufficient signal strength for the non-null features as specified in 1,2, 3, assuming the estimated parameters $\hat{\boldsymbol{\theta}}$ share the same bounded parameter space $[-W, W]^{car(\boldsymbol{\theta})}$, then*

1. *the probability that the same null predictor appears in B_2-bagging rounds tends to zero asymptotically as B_2 increases.*

2. *the probability that the model is able to include all the non-null predictors in the selected set tends to one asymptotically.*

i.e.

$$\mathbb{P}(\hat{\mathcal{S}} = \mathcal{S}) \to 1 \quad \text{as } n \to \infty \text{ and } B_2 \to \infty \tag{22}$$

The proof is given in the appendix. In theorem 4.1, we showed that with strong enough signal strength in the true nonzero variables, the algorithm will be able to select all nonzero variables with probability tending to 1. The conditions are not verifiable in practice, however, extensive examples in section 5 show that the ENNS algorithm reaches selection consistency easier than the other algorithms.

For the estimation step, there has been some discussion about the asymptotic properties such as Feng and Simon (2017); Yang and Maiti (2020), where the results of using sparse group lasso penalty are given. The l_1 norm penalty is actually a special case of the sparse group lasso with the lasso parameter being 1 and the group lasso parameter being 0. Therefore, the results of these papers hold as long as we have $\hat{\mathcal{S}} = \mathcal{S}_0$, which has probability tending to 1 by theorem 4.1. Here we will adapt the theory in Györfi et al. (2006) and will provide the following result.

Theorem 4.2 *Under the assumptions 1, 2, and 3, consider the variables selected from the ENNS algorithm and the estimation with the l_1 regularization method. Denote the l_1 regularization tuning parameter with λ_n and the corresponding Lagrangian parameter K_n. Denote the hidden layer size with k_n. In the regression set up, assume $\mathbb{E}(Y^2) < \infty$, if $K_n \to \infty$, $k_n \to \infty$ and $k_n s^2 K_n^4 \log(k_n s K_n^2)/n \to 0$, we have*

$$\lim_{n\to\infty, B_2\to\infty} \mathbb{P}\left(\mathbb{E}\int |f_n(\boldsymbol{x}) - f(\boldsymbol{x})|^2 \mu(dx) \to 0\right) = 1$$

where f_n is the estimated neural network and f is the true function. In the classification set up, assuming that the probability of response being 1 is bounded away from 0 and 1 by $\tilde{\epsilon}$, denote with Q the maximum number of equivalent neural network classes, choosing tuning parameter $\lambda_n \geq c\sqrt{k_n \log n/n}(\sqrt{\log Q} + \sqrt{k_n \log s}\log(nk_n))$, if $\log(n)/(n\tilde{\epsilon}^2) \to 0$, $s^2 k_n \lambda_n^2/(n\tilde{\epsilon}^2) \to 0$ and $n^{-1}k_n^{9/2}s^{5/2}\sqrt{\log(s_n)} \to 0$ as $n \to \infty$, we have

$$\lim_{n\to\infty, B_2\to\infty} \mathbb{P}\left(R(f_n) - R(f^*) \to 0\right) = 1$$

where $R(f_n)$ is the risk of neural network classifier and $R(f^*)$ is the risk of Bayes classifier.

Theorem 4.2 states that under the previously discussed conditions, the regression reaches weak estimation consistency of the non-parametric function defined in Györfi et al. (2006). For the classification, the neural network classifier's risk tends to the optimal risk, Bayes risk, see for example Devroye et al. (2013). The theorem is a direct result from the existing results of the low dimension neural network regression model and classifiers. Conditioning on the fact that we can select all correct variables with probability tending to 1, applying the full probability formula, the consistency of the two-step approach can be derived with the low dimensional consistency plus the probability of non-selection-consistency.

The consistency error comes from two aspects: the variable selection error and the estimation error. The intuition behind this is that with a wrong selection result, the estimation error may be big, however, this happens with probability tending to zero. With a correct selection result, the estimation error behaves the same as in the low dimensional case, which converges to zero.

5. Simulation study

In this section, we use a few examples as numerical supports to our arguments in the previous sections. The code for DNP is composed according to the algorithm outline in Liu et al. (2017), and the code of ENNS is composed according to the algorithm in this paper. Both codes can be found at `https://github.com/KaixuYang/ENNS`.

5.1 Stage-wise correct selection probability decreasing study

In this subsection, we use an example to demonstrate that the chance of selecting a correct variable in a stage-wise neural network decreases as we have more correct variables in the model. Consider a design matrix $\boldsymbol{X}$ that is drawn from a uniform distribution between -1 and 1. The sample size is set to $n = 1000$ and the number of predictors is set to $p = 10000$. The first $s = 5$ predictors are related with the response. We consider three different true structures of the relationship between the predictors and the response: linear, additive non-linear and neural network. For the response, we consider two different cases: regression and classification. In the linear case, the coefficients are drawn from a standard normal distribution. In the additive non-linear case, the functions are set to

$$\eta = \sin(x_1) + x_2 + \exp(x_3) + x_4^2 + \log(x_5 + 2) - 2 \tag{23}$$

where $y = \eta + \epsilon$ in the linear case and $prob = \sigma(\eta)$ in the classification case. In the neural network case, we set hidden layers as $[50, 30, 15, 10]$ and weights from a standard normal distribution.

For each of the cases, we test the cases when we start from 0 to 4 correct predictors. In order to eliminate the effect of different signal strengths from different predictors, we random sample i indices from $1, ..., 5$ as the selected variables, for $i = 0, ..., 4$, and include these i indices predictors as the initially selected variables. We run a repetition of 1000 times and report the proportion that the next variable that enters the model is a correct predictor. The results are summarized in table 1.

Table 1: The proportion of correct variable selection after 0-4 correct variables in the model, for different cases over 1000 repetitions. The results show the mean.

y	structure	0 variable	1 variable	2 variables	3 variables	4 variables
	Linear	.995(.002)	.952(.006)	.863(.010)	.774(.013)	.430(.016)
Reg	Additive	.993(.003)	.847(.011)	.905(.009)	.794(.012)	.531(.016)
	NN	.998(.001)	.971(.005)	.932(.007)	.788(.013)	.574(.016)
	Linear	.989(.003)	.918(.009)	.873(.009)	.813(.011)	.552(.016)
Cls	Additive	.992(.003)	.957(.006)	.911(.009)	.706(.014)	.633(.015)
	NN	.994(.002)	.968(.006)	.947(.004)	.895(.009)	.762(.013)

In the table, we see that the probability of selecting a correct predictor decreases as we have more correct predictors in the model, in all cases. The only exception is in the regression set up with additive non-linear structure from 1 variable to 2 variables, which may due to random error.

5.2 False positive rate study

In this subsection, we use an example to demonstrate that the false positive rate of ENNS (the probability of selecting a wrong variable) is superior than the pure stage-wise algorithm. Note that if one set the number of variables to be s, stage wise algorithm always select 5 variables, while ENNS will stop if there isn't any new variable that satisfy the condition to be added. Therefore, it's possible that ENNS selects less number of variables and avoid wrong selection. We used the same setup as Liu et al. (2017) to generate responses. Two different types of responses including regression and classification will be considered here. The input variable $\boldsymbol{X}$ was drawn from $U(-1,1)$, where the feature dimension p was fixed to be $10,000$. The corresponding labels were obtained by passing $\boldsymbol{X}$ into the feed-forward neural network with hidden layer sizes $\{50, 30, 15, 10\}$ and ReLU activation functions. Input weights connecting the first s inputs were randomly sampled from $N(1,1)$ for regression and $N(0,1)$ for classification. The remaining input weights were kept zero. For each $s = 2, 5, 10$, we generated 1000 training samples. In table 2, we report the false positive rate between the ENNS algorithm and the neural network stage-wise algorithm. Additionally, we implemented the LassoNet algorithm (Lemhadri et al., 2021) due to its comparative performances in many application domains.

It can be tested that the ENNS's false positive rate is significantly less than the false positive rate of DNP under significance level $\alpha = 0.05$. This provides strong evidence that the ENNS is useful in reducing the probability of selecting an incorrect variable.

5.3 Variable selection simulation study

In this subsection, we study the variable selection capability of the ensemble neural network selection (ENNS) algorithm in a complicated setup. We used a similar setup as in the last subsection to generate responses. Two different types of responses including regression and classification will be considered here. The input variable $\boldsymbol{X}$ was drawn from $U(-1,1)$, where

Table 2: Selection false positive rate average of the ENNS, DNP and LassoNet under different number of true variables in 101 repetitions. Standard deviations are given in parenthesis.

Response	Method	s=2	s=5	s=10
Regression	ENNS	10.4%(21.5%)	11.5%(22.1%)	12.8%(23.6%)
	DNP	22.5%(29.5%)	30.2%(28.7%)	41.4%(33.2%)
	LassoNet	30.7% (26.9%)	38.2% (24.6%)	53.1% (39.4%)
Classification	ENNS	4.7%(17.9%)	7.4%(18.6%)	9.8%(20.3%)
	DNP	16.5%(24.4%)	24.8%(29.7%)	40.5%(32.8%)
	LassoNet	25.3% (26.9%)	33.8% (21.6%)	47.7% (34.2%)

Table 3: Variable selection capacity of ENNS and other methods with low signal strength in the regression (top) and classification (bottom) setup. The numbers reported are the average number of selected variables that are truly nonzero. The standard errors are given in parentheses.

Response	Method	s=2	s=5	s=10
Regression	ENNS	1.73(0.52)	4.21(0.56)	9.25(1.11)
	DNP	1.61(0.50)	3.92(0.56)	8.77(1.13)
	LASSO	1.65(0.57)	3.87(0.62)	9.62(1.38)
	HSIC-LASSO	1.67(0.47)	3.80(0.66)	3.61(1.17)
	LassoNet	1.72(0.31)	4.45(0.51)	8.97(0.48)
Classification	ENNS	1.81(0.49)	4.24(0.87)	8.04(1.25)
	DNP	1.67(0.76)	3.76(1.06)	5.95(1.29)
	LASSO	1.67(0.56)	3.76(0.75)	5.76(1.38)
	HSIC-LASSO	1.67(0.47)	2.80(0.91)	3.61(1.17)
	LassoNet	1.73(0.23)	4.06(0.33)	8.39(0.38)

Table 4: Variable selection capacity of ENNS and other methods with normal signal strength. The numbers reported are the average number of selected variables that are truly nonzero. The standard errors are given in parentheses.

Method	s=2	s=5	s=10
ENNS	2.00(0.00)	4.71(0.55)	8.38(2.06)
DNP	1.86(0.35)	4.38(0.84)	7.43(2.36)
LASSO	1.81(0.39)	4.19(1.01)	7.47(2.40)
HSIC-LASSO	1.71(0.45)	3.71(1.12)	4.95(2.13)

the feature dimension p was fixed to be $10,000$. The corresponding labels were obtained by passing $\boldsymbol{X}$ into the feed-forward neural network with hidden layer sizes $\{50, 30, 15, 10\}$ and ReLU activation functions. Input weights connecting the first s inputs were randomly sampled from $N(0, 2)$ for regression and $N(0, 1)$ for classification. The remaining input weights were kept at zero. The DNP model was coded according to their algorithm outline in python with pyTorch. The ENNS algorithm is based on the DNP algorithm with an ensemble wrapper. The LASSO (Tibshirani, 1996) is implemented by the scikit learn library, and the HSIC lasso (Yamada et al., 2014) is implemented using the HSICLasso library. In all these algorithms, the number of selected variables is strictly restricted to the same as the true number of variables. In the ENNS, we run a bagging of 10 rounds with a selection proportion of 0.3. We report the average number of correct variables that are selected on 101 repetitions of the data generation in Table 3.

We observe that the ENNS outperforms the other variable selection algorithms in all three cases, and the difference is significant when $s = 10$ under a t-test. The ENNS performs better when there are more nonzero variables. None of the algorithms were able to reconstruct the original feature indices due to a few reasons: the sample size is relatively small compared to the number of variables; the data generation through neural network structures is complicated; and the signal strength is relatively low. Furthermore, it's worth observing that LassoNet, owing to its prediction-optimized performance, consistently opts for a significantly larger number of features. As a consequence, this choice leads to an increased count of false discoveries. This trend remains consistent across all our numerical experiments, and as a result, we have chosen not to include its results in the remaining experiments.

To thoroughly study the variable selection power of the ENNS algorithm, we implemented another simulation case in a classification where we have a higher signal strength while keeping all other conditions the same. In this simulation study, we increase the mean of the weights of the nonzero variables to 3.5. With the same implementations, we summarize the results in Table 4. Moreover, Table 5 summarizes the results for signal strength 10.

The ENNS reaches selection consistency when $s = 2$, while the other compared algorithms still do not have selection consistency. However, all algorithms have obvious improvements in all cases. We have to admit that selecting the correct subset of variables in all 101 repetitions is extremely challenging since the data is highly variable in different

Table 5: Variable selection capacity of ENNS and other methods with high signal strength. The numbers reported are the average number of selected variables which are truly nonzero. The standard errors are given in parenthesis.

Method	s=2	s=5	s=10
ENNS	2.00(0.00)	5.00(0.00)	9.90(0.29)
DNP	2.00(0.00)	5.00(0.00)	9.47(1.10)
LASSO	2.00(0.00)	4.90(0.29)	9.23(1.74)
HSIC-LASSO	2.00(0.00)	4.62(0.84)	7.76(2.76)

repetitions. Moreover, when s gets greater, the importance of a few variables are less likely to be observed from the data.

Moreover, as we know, the bagging algorithm can be paralyzed since different runs are independent of each other. Therefore, the computational efficiency of this variable selection algorithm is almost the same as the computation efficiency of a single run.

5.4 Estimation simulation study

In this subsection, we compare the estimation methods in section 3. To fully study the difference between these methods without the effects of other factors, in this subsection we assume correct selection and perform the estimation on the correct subset of variables. The data are generated according to the same scheme as in the last subsection. We will compare the performance of these different estimation methods for $s = 2, 5, 10$ assuming that we know the correct subset of variables. The simulation is run on 101 repetitions of data generation using different seeds. In the results, we report the RMSE, the MAE and the MAPE for regression, and the accuracy, the auc score and the f1 score for classification. These are in Table 6.

On average, we see l_1 norm regularization gives best performance, except for the MAPE of $s = 10$ in regression. Moreover, we observe that both built-in l_1 and soft-thresholding gives smaller standard errors, which coincides with the shrinkage estimator's properties. However, soft-thresholding provides better performance in average than built-in. The reason is that sparsity is not well supported with most libraries, thus a manual operation is needed to obtain sparsity.

5.5 Variable selection and estimation

In this subsection, we study the prediction capability of the two-stage approach – ENNS algorithm with l_1 neural network, and compare it with the DNP model, the logistic regression and the HSIC-lasso with SVM. We use the same neural network structure to generate data as in this section. Over 101 repetitions, we report the average RMSE (rooted mean squared error), MAE (mean absolute error) and MAPE (mean absolute percent error) for regression and average accuracy, AUC and F1 Score for classification, as well as their standard errors. The results are summarized in table 7.

Table 6: Prediction results on the testing set using neural networks with and without l_1 norm regularization for $s = 2, 5, 10$. RMSE is rooted mean squared error, MAE is mean absolute error, and MAPE is mean absolute percent error. Accuracy is the percentage of correct prediction, auc is area under the ROC curve, and f1 score is the inverse of inverse precision plus the inverse recall.

Response	Metric	Method	s=2	s=5	s=10
Regression	RMSE	Neural Network	31.24(13.32)	69.46(37.40)	136.64(60.54)
		Xavier initialization	18.64(11.07)	58.89(27.73)	136.58(65.57)
		l_1 built-in	20.47(9.62)	59.37(23.61)	129.55(50.48)
		l_1 soft-thresholding	**5.97(4.18)**	**45.83(33.06)**	**109.31(47.24)**
		Stage-wise	10.59(11.20)	47.64(22.69)	117.65(43.96)
		Bagging	25.48(10.89)	59.49(26.53)	133.45(59.72)
	MAE	Neural Network	16.45(10.91)	52.85(28.47)	103.76(45.99)
		Xavier initialization	13.65(8.06)	45.34(22.18)	105.17(53.66)
		l_1 built-in	15.56(7.76)	45.32(18.34)	98.54(38.36)
		l_1 soft-thresholding	**4.37(3.02)**	**35.49(26.21)**	**83.85(36.45)**
		Stage-wise	7.91(8.23)	38.86(20.00)	89.82(33.82)
		Bagging	14.77(7.92)	43.16(20.51)	99.38(41.66)
	MAPE	Neural Network	0.012(0.015)	0.030(0.026)	0.033(0.026)
		Xavier initialization	0.009(0.009)	0.027(0.022)	0.029(0.017)
		l_1 built-in	0.011(0.012)	0.029(0.023)	0.032(0.021)
		l_1 soft-thresholding	**0.005(0.007)**	**0.017(0.010)**	0.029(0.023)
		Stage-wise	0.007(0.007)	0.019(0.015)	**0.027(0.016)**
		Bagging	0.010(0.010)	0.026(0.024)	0.030(0.022)
Classification	Accuracy	Neural Network	0.944(0.026)	0.886(0.037)	0.841(0.041)
		Xavier initialization	0.952(0.026)	0.891(0.034)	0.831(0.036)
		l_1 built-in	0.927(0.031)	0.844(0.085)	0.752(0.110)
		l_1 soft-thresholding	**0.964(0.028)**	**0.908(0.029)**	**0.855(0.031)**
		Stage-wise	0.945(0.030)	0.886(0.038)	0.804(0.042)
		Bagging	0.877(0.069)	0.806(0.068)	0.753(0.087)
	AUC	Neural Network	0.942(0.027)	0.882(0.038)	0.837(0.042)
		Xavier initialization	0.951(0.027)	0.891(0.034)	0.825(0.037)
		l_1 built-in	0.924(0.031)	0.833(0.100)	0.734(0.123)
		l_1 soft-thresholding	**0.964(0.029)**	**0.905(0.029)**	**0.851(0.032)**
		Stage-wise	0.943(0.031)	0.884(0.038)	0.800(0.041)
		Bagging	0.877(0.065)	0.803(0.063)	0.751(0.084)
	F1 Score	Neural Network	0.943(0.027)	0.887(0.045)	0.841(0.049)
		Xavier initialization	0.952(0.026)	0.892(0.041)	0.832(0.048)
		l_1 built-in	0.927(0.031)	0.824(0.192)	0.732(0.200)
		l_1 soft-thresholding	**0.965(0.026)**	**0.908(0.036)**	**0.857(0.033)**
		Stage-wise	0.944(0.031)	0.883(0.042)	0.806(0.049)
		Bagging	0.870(0.077)	0.792(0.060)	0.748(0.088)

Table 7: Model performance of the combination of ENNS algorithm and l_1 thresholding estimation, compared with DNP, Lasso and HSIC-Lasso for $s = 2, 5, 10$ cases in both regression and classification. The average performance of 101 repetitions with their standard errors in parenthesis are presented.

Response	Metric	Method	s=2	s=5	s=10
Regression	RMSE	**ENNS+l1**	**15.67(30.35)**	**48.14(21.16)**	**174.08(65.38)**
		DNP	25.42(33.16)	62.63(29.02)	**178.91(60.15)**
		Lasso	79.44(67.31)	104.19(49.38)	192.04(77.34)
		HSIC-Lasso	56.32(59.41)	86.77(47.51)	188.35(56.48)
	MAE	**ENNS+l1**	**12.03(23.68)**	**40.12(19.95)**	**132.07(44.99)**
		DNP	20.15(27.15)	47.85(22.31)	**136.06(45.95)**
		Lasso	64.11(54.63)	81.97(39.76)	147.86(60.21)
		HSIC-Lasso	42.89(34.66)	70.04(41.23)	144.37(48.15)
	MAPE	**ENNS+l1**	**0.012(0.025)**	**0.028(0.036)**	**0.041(0.037)**
		DNP	0.017(0.028)	0.032(0.032)	**0.042(0.041)**
		Lasso	0.042(0.029)	0.046(0.035)	0.046(0.025)
		HSIC-Lasso	0.033(0.021)	0.036(0.025)	0.048(0.024)
Classification	Accuracy	**ENNS+l1**	**0.967(0.029)**	**0.848(0.025)**	**0.756(0.067)**
		DNP	0.933(0.076)	0.822(0.068)	0.736(0.064)
		Lasso	0.732(0.103)	0.726(0.071)	0.692(0.075)
		HSIC-Lasso	0.805(0.094)	0.798(0.094)	0.706(0.081)
	AUC	**ENNS+l1**	**0.959(0.036)**	**0.834(0.024)**	**0.709(0.058)**
		DNP	0.898(0.148)	0.780(0.100)	0.699(0.052)
		Lasso	0.652(0.121)	0.640(0.102)	0.625(0.068)
		HSIC-Lasso	0.774(0.125)	0.748(0.121)	0.677(0.061)
	F1-Score	**ENNS+l1**	**0.962(0.037)**	**0.859(0.036)**	**0.708(0.089)**
		DNP	0.903(0.208)	0.761(0.199)	0.705(0.100)
		Lasso	0.590(0.299)	0.604(0.250)	0.634(0.192)
		HSIC-Lasso	0.744(0.206)	0.731(0.242)	0.666(0.208)

Our proposed algorithm obtained a slight performance boost via the ensemble method. Moreover, the standard errors of these results are slightly greater than the standard errors in the last subsection, where the estimation was done assuming correct selection. The increase in standard errors is mainly due to the selection variations.

5.6 Correlated predictors

In this subsection, we use an example to study the numerical performance of the proposed algorithm in a correlated predictor situation. We will consider two different correlations: $\rho = 0.3$ and $\rho = 0.7$. The results for $\rho = 0$ will also be included as a comparison. Let $u_1, ..., u_n$ be i.i.d. standard normal random variables, x_{ij} be i.i.d. standard normal random variables, which are independent of $u_1, ..., u_n$, for $i = 1, ..., n$ and $j = 1, ..., p$. Do the transformation $x_{ij} = (x_{ij} + tu_i)/\sqrt{1+t^2}$ for some t, then we obtained standard normal correlated variables

$$cor(x_{ij}, x_{ik}) = \frac{t^2}{1+t^2},\ i = 1, ..., n; j = 1, ..., p$$

Taking $t = \sqrt{3/7}$ gives correlation 0.3 and taking $t = \sqrt{7/3}$ gives correlation 0.7. Then we truncate the random variables to interval $[-1, 1]$. The structure to generate response is kept the same as in the last subsection. The results of variable selection and estimation is given in table 8.

From the table, we see that in the correlated cases, the model works almost as well as when there is no correlation. All models select fewer variables when the correlation is higher, and this is a well-known symptom of variable selection with correlated predictors. However, this does not affect the estimation step and sometimes even improves the estimation results. The reason could be that we have fewer variables; thus, the model is simpler. Since the predictors are correlated, we do not lose too much information by not selecting some of them. Moreover, some results not in the table include the false positive rate, where the average for ENNS is 0.05 ± 0.03, while that of the DNP is 0.29 ± 0.14. Therefore, ENNS includes fewer redundant variables in the estimation step and achieves better performance.

6. Real data examples

In this section, we evaluate the performance of the two-step model on some real-world data sets.

6.1 Variable selection: MRI data

In this example, we evaluate the variable selection capability with other variable selection models and compare the results with the biological ground truth. The data used in this example come from Alzheimer's disease neuroimaging initiatives (ADNI), see `http://adni.loni.usc.edu/`. The data includes neuroimaging results from $n = 265$ patients, including 164 Alzheimer's (AD) patients and 101 cognitively normal (CN) individuals. 145 regions of interest (ROI) that span the entire brain were calculated using Multi-Atlas ROI segmentation, and 114 ROIs were derived by combining single ROIs within a tree hierarchy to obtain volumetric measurements from larger structures. Therefore,

Table 8: Selection and estimation comparison for predictors with correlation 0, 0.3 and 0.7. The number of nonzero predictors is set to 5. For selection, the average number of correct selected variables with its standard error is given. For estimation the average RMSE or AUC with their standard errors is given. The results are averaged over 101 repetitions.

Response	Model	selection		
		$\rho = 0.0$	$\rho = 0.3$	$\rho = 0.7$
Regression	ENNS+l_1	3.81(0.79)	3.27(0.75)	2.29(0.70)
	DNP	3.48(0.96)	2.95(0.79)	2.14(0.56)
	LASSO	3.38(0.90)	2.85(0.79)	2.11(1.12)
Classification	ENNS+l_1	3.66(1.05)	3.25(0.76)	2.38(0.72)
	DNP	3.62(1.09)	3.43(0.91)	2.71(1.03)
	LASSO	3.55(0.79)	2.90(1.31)	1.95(0.72)
Response	Model	estimation		
		$\rho = 0.0$	$\rho = 0.3$	$\rho = 0.7$
Regression	ENNS+l_1	40.82(19.46)	37.17(27.29)	43.18(44.47)
	DNP	81.43(46.00)	92.91(65.25)	101.15(90.63)
	LASSO	131.37(74.22)	151.16(108.88)	113.30(97.54)
Classification	ENNS+l_1	0.856(0.040)	0.875(0.061)	0.907(0.030)
	DNP	0.774(0.100)	0.766(0.106)	0.793(0.092)
	LASSO	0.598(0.083)	0.634(0.117)	0.683(0.116)

$p = 259$ ROIs were used in this example. Details of the pre-processing method can be found in `https://adni.bitbucket.io/reference/docs/UPENN_ROI_MARS/Multi-atlas_ROI_ADNI_Methods_mod_April2016.pdf`. Among these ROIs, biologically important features are picked, see Table 9, where red indicates the most important, yellow means second importance, and green means third importance. The combinations and all other ROIs are not listed.

The full data set is used for variable selection, and the selection result is chosen such that a 3-fold cross-validation performs best. We run the ENNS algorithm along with the LASSO and the DNP. The selection results are presented in Table 9. Note here if a model selects a simple combination of some features, these features are also marked as selected. Moreover, Table 10 shows the number of combined features selected for the models and the number of false positive selections. We observe that LASSO misses a lot of important features and selected only 1/4 of the combined features as neural networks. This indicates that the features may have a complicated relationship with the response. ENNS performs better than the shallow DNP in terms of the metrics in table 10, where IS is a weighted average score with the weights for red, yellow and green being 3, 2 and 1, respectively, NI is the number of selected important variables; and NU is the number of selected unimportant variables. As a property of the ENNS, it selects fewer false positive variables. It's hard to track the combined features since a lot are involved. However, the combinations represent biological intuitions. Neural networks select more combined features and perform better in this sense.

6.2 Regression: riboflavin production data

In this example, we consider the riboflavin production with bacillus subtilis data, which is publicly available in the 'hdi' package in R. The data set contains a continuous response, which is the logarithm of the riboflavin production rate, and $p = 4088$ predictors which are the logarithm of the expression level of 4088 genes. There are $n = 71$ observations available. All predictors are continuous with positive values.

We perform 50 repetitions of the following actions. The data is split into training (56) and testing (15) observations. The training data is further split into training (49) and validation (7). The training data is normalized with mean zero and standard deviation one. We train the ENNS algorithm to select variables and perform the l_1 neural network to make prediction. Along with our proposed algorithms, we compare the performance with the lasso penalized linear regression, which is implemented by the scikit-learn library in python; the group lasso penalized generalized additive model in Yang and Maiti (2018), where the code can be found at `https://github.com/KaixuYang/PenalizedGAM`; and the sparse group lasso penalized neural network in Feng and Simon (2017). Figure (1) shows the average testing mean squared error (MSE) along with the number of selected features for different models. Our proposed algorithm converges fast and performs competitive. Table 11 shows the average prediction accuracy with standard error in parenthesis and the median number of variables selected. Our proposed method has competitive mean performance but lower standard error.

The final model of small sample utilizes only 2 hidden layers, with over 90% sparsity to prevent over-fitting, which is necessary for this small training sample size, 49. Training with

Table 9: Variable selection result for the AD data. The table includes all biologically important variables with three levels: red (very important), yellow (secondly important) and green (thirdly important). The non-important variables are not included in the model. Checkmarks indicate whether the corresponding algorithm selected the variable or not.

Gene code	🟨 R31	🟨 R32	🟨 R36	🟨 R37	🟩 R38	🟩 R39	🟩 R40	🟩 R41	🟥 R47	🟥 R48	🟨 R49	🟨 R50	🟨 R51
Lasso		✓	✓							✓	✓	✓	
DNP	✓	✓				✓	✓		✓	✓	✓		
ENNS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Gene code	🟨 R52	🟩 R55	🟩 R56	🟩 R57	🟩 R58	🟩 R59	🟩 R60	🟥 R81	🟥 R82	🟥 R85	🟥 R86	🟥 R87	🟥 R88
Lasso		✓							✓				
DNP		✓						✓	✓	✓	✓	✓	
ENNS			✓					✓	✓	✓	✓	✓	✓
Gene code	🟥 R100	🟥 R101	🟩 R102	🟩 R103	🟩 R106	🟩 R107	🟥 R116	🟥 R117	🟨 R118	🟨 R119	🟨 R120	🟨 R121	🟨 R122
Lasso		✓				✓	✓					✓	
DNP		✓		✓		✓	✓	✓			✓		
ENNS	✓	✓			✓	✓	✓						
Gene code	🟨 R123	🟨 R124	🟨 R125	🟨 R132	🟨 R133	🟨 R136	🟨 R137	🟥 R138	🟥 R139	🟥 R140	🟥 R141	🟨 R142	🟨 R143
Lasso	✓	✓			✓	✓	✓	✓		✓		✓	
DNP		✓	✓		✓	✓	✓	✓	✓			✓	
ENNS		✓	✓		✓	✓	✓	✓	✓	✓		✓	
Gene code	🟨 R146	🟨 R147	🟨 R152	🟨 R153	🟨 R154	🟨 R155	🟥 R162	🟥 R163	🟥 R164	🟥 R165	🟥 R166	🟥 R167	🟥 R168
Lasso					✓	✓	✓		✓	✓		✓	
DNP					✓	✓	✓					✓	
ENNS					✓	✓	✓	✓	✓	✓	✓	✓	
Gene code	🟥 R169	🟨 R170	🟨 R171	🟩 R178	🟩 R179	🟩 R186	🟩 R187	🟨 R190	🟨 R191	🟩 R194	🟩 R195	🟥 R198	🟥 R199
Lasso		✓	✓	✓	✓		✓	✓					
DNP		✓	✓		✓			✓	✓				
ENNS		✓	✓		✓			✓	✓		✓		
Gene code	🟨 R200	🟨 R201	🟩 R202	🟩 R203	🟩 R204	🟩 R205	🟩 R206	🟩 R207					
Lasso		✓		✓	✓								
DNP	✓		✓	✓		✓							
ENNS	✓	✓		✓		✓							

Table 10: Variable selection result for the AD data. The reported numbers include IS, the weighted average of selected important variables with the weights being 3, 2 and 1 for red (most important), yellow (secondly important) and green (thirdly important), respectively; NI, number of important variables selected; and NU, number of unimportant variables selected.

Variable selection method	IS	NI	NU
LASSO	1.094	32/86	25/59
DNP	1.428	40/86	15/59
ENNS	1.624	49/86	6/59

Table 11: Test MSE with standard error in parentheses and median of number of features for different models in the riboflavin gene data example.

Model	Test MSE	Number of features
ENNS+l_1 neural network	0.268(0.115)	42
Regularized neural network	0.273(0.135)	44
Linear model with LASSO	0.286(0.124)	38
Generalized additive model with group lasso	0.282(0.136)	46

large batch size, small learning rate, huge number of epochs and early stopping help the model learn better and prevent over-fitting. We admit that tuning the network structure and learning parameters are hard, but we obtain better and stabler results once we have the right numbers.

6.3 Classification: prostate cancer data

In this example, we consider prostate cancer gene expression data publicly available in `http://featureselection.asu.edu/datasets.php`. The data set contains a binary response with 102 observations on 5966 predictor variables. The data set is a high-dimensional data set. The responses have values 1 (50 sample points) and 2 (52 sample points), where 1 indicates normal and 2 indicates tumor. All predictors are continuous predictors with positive values.

We perform 50 repetitions of the following actions. The data is split into training (81) and testing (21) observations. The training data are divided into training (70) and validation (11). In each split, the number of class 0 and class 1 observations are kept roughly the same. We train the ENNS algorithm to select variables and perform the l_1 neural network to make predictions. Along with our proposed algorithms, we compare the performance with the l_1 norm penalized logistic regression, the l_1 support vector machine (SVM), both of which are implemented with the scikit-learn library in Python; the group lasso penalized generalized additive model in Yang and Maiti (2018), where the code can be found at `https://github.com/KaixuYang/PenalizedGAM`; and the sparse group lasso penalized neural network in Feng and Simon (2017). Figure (2) shows the average testing accuracy over the 20 repetitions and the number of selected features for different models. Our proposed algorithm converges fast and performs competitively. Table 12 shows the average prediction accuracy with standard error in parenthesis and the median number of variables selected. Our proposed methods has competitive mean performance but lower standard error. One needs to notice that the mean performance is hard to improve further since the results are already good and reach the bottleneck of the current explanatory variables. The reason that GAM performs worse than the other models is that the range of predictor variables is relatively small and skewed. Thus, the basis expansion on GAM does not work well.

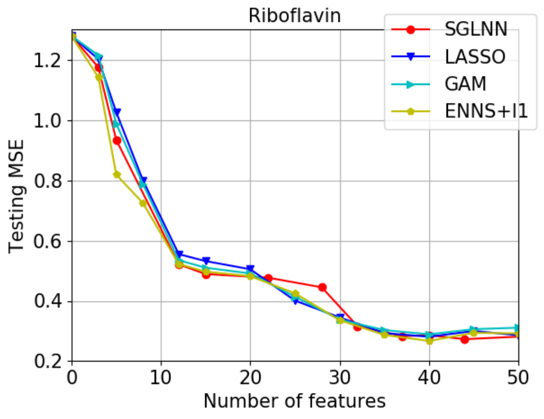

Figure 1: Testing mean squared error (MSE) for different models on the riboflavin data.

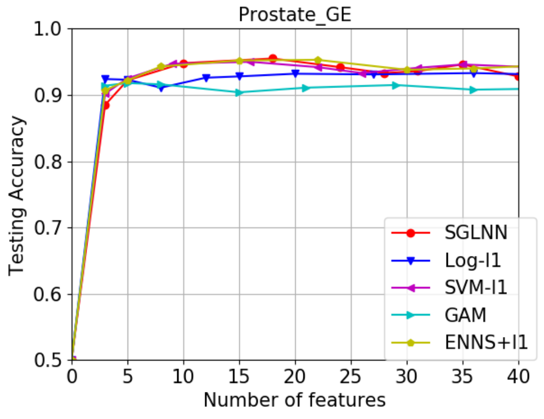

Figure 2: Testing accuracy for different models on the prostate cancer data.

Table 12: Test accuracy with standard error in parentheses and median of number of features for different classifiers in the Prostate gene data example.

Classifier	Test accuracy	Number of features
ENNS+l_1 neural network	0.956(0.053)	15
Regularized neural network	0.955(0.066)	18
Logistic Regression with Lasso	0.933(0.058)	36
l1 penalized Linear SVM	0.950(0.052)	16
Generalized additive model with group lasso	0.918(0.061)	5

7. Conclusion

This paper discusses existing methods for dealing with high-dimensional data and how to apply the stage-wise algorithm to neural networks. We addressed the shortage of current stage-wise neural network variable selection algorithms and proposed the ENNS to overcome this shortage. Moreover, we also compared different methods to reduce the over-fitting issue further after the variable selection step. The methodology was given to support the argument and new algorithm, and simulation studies were given as additional evidence for the algorithm to work.

Although some simulations and methodologies have been used to select neural network variables, the theory of neural networks still deserves much more investigation. We hope that this paper can work as a pioneer and attract more people's attention to the field of neural network theory.

8. Acknowledgement

The authors appreciate the reviewers' comments, which improved the manuscript. Grants NSF-1924724, NSF-1952856, and NSF-2124605 partially supported the work.

References

Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations.* cambridge university press, 2009.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48, 2007.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565, 2014.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Peter Bühlmann and Sara Van De Geer. *Statistics for high-dimensional data: methods, theory and applications.* Springer Science & Business Media, 2011.

Peter Bühlmann, Bin Yu, et al. Analyzing bagging. *The Annals of Statistics*, 30(4):927–961, 2002.

Emmanuel Candes, Terence Tao, et al. The dantzig selector: Statistical estimation when p is much larger than n. *The annals of Statistics*, 35(6):2313–2351, 2007.

Giovanna Castellano and Anna Maria Fanelli. Variable selection using neural-network models. *Neurocomputing*, 31(1-4):1–13, 2000.

Jiahua Chen and Zehua Chen. Extended bayesian information criteria for model selection with large model spaces. *Biometrika*, 95(3):759–771, 2008.

Nam Hee Choi, William Li, and Ji Zhu. Variable selection with the strong heredity constraint and its oracle property. *Journal of the American Statistical Association*, 105(489): 354–364, 2010. doi: 10.1198/jasa.2010.tm08281. URL `https://doi.org/10.1198/jasa.2010.tm08281`.

Charles K Chui, Xin Li, and Hrushikesh Narhar Mhaskar. Limitations of the approximation capabilities of neural networks with one hidden layer. *Advances in Computational Mathematics*, 5(1):233–243, 1996.

CK Chui, Xin Li, and Hrushikesh Narhar Mhaskar. Neural networks for localized approximation. *Mathematics of Computation*, 63(208):607–623, 1994.

John B Copas. Regression, prediction and shrinkage. *Journal of the Royal Statistical Society: Series B (Methodological)*, 45(3):311–335, 1983.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in neural information processing systems*, pages 666–674, 2011.

Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul): 2121–2159, 2011.

Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

J. D. Esary, F. Proschan, and D. W. Walkup. Association of Random Variables, with Applications. *The Annals of Mathematical Statistics*, 38(5):1466 – 1474, 1967. doi: 10.1214/aoms/1177698701. URL `https://doi.org/10.1214/aoms/1177698701`.

Fenglei Fan, Jinjun Xiong, and Ge Wang. Universal approximation with quadratic deep networks. *Neural Networks*, 124:383–392, 2020.

Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

Jianqing Fan and Jinchi Lv. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5): 849–911, 2008.

Yingying Fan and Cheng Yong Tang. Tuning parameter selection in high dimensional penalized likelihood. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):531–552, 2013.

Jean Feng and Noah Simon. Sparse-input neural networks for high-dimensional nonparametric regression and classification. *arXiv preprint arXiv:1711.07592*, 2017.

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

Evarist Giné and Joel Zinn. Bootstrapping general empirical measures. *The Annals of Probability*, pages 851–869, 1990.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A distribution-free theory of nonparametric regression*. Springer Science & Business Media, 2006.

Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

Jian Huang, Joel L Horowitz, and Fengrong Wei. Variable selection in nonparametric additive models. *Annals of statistics*, 38(4):2282, 2010.

Bing'er Jiang, Tim O'Donnell, and Meghan Clayards. A deep neural network approach to investigate tone space in languages. *The Journal of the Acoustical Society of America*, 145(3):1913–1913, 2019.

Ian T Jolliffe, Nickolay T Trendafilov, and Mudassir Uddin. A modified principal component technique based on the lasso. *Journal of computational and Graphical Statistics*, 12(3):531–547, 2003.

H. J. Kim. A class of ratio distributions of dependent folded normals and its applications. *Statistics*, 50(4):791–811, 2015. doi: 10.1080/02331888.2015.1094071. URL `https://doi.org/10.1080/02331888.2015.1094071`.

Stephen Cole Kleene. Representation of events in nerve nets and finite automata. Technical report, RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Michele La Rocca and Cira Perna. Variable selection in neural network regression models with dependent data: a subsampling approach. *Computational statistics & data analysis*, 48(2):415–429, 2005.

Neil D Lawrence. A unifying probabilistic perspective for spectral dimensionality reduction: Insights and new models. *Journal of Machine Learning Research*, 13(May):1609–1638, 2012.

Ismael Lemhadri, Feng Ruan, and Robert Tibshirani. A neural network with feature sparsity. *arXiv preprint arXiv:1907.12207*, 2019.

Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *J. Mach. Learn. Res.*, 22(1), jan 2021. ISSN 1532-4435.

You L Li, Jessica Ducey-Wysling, Aurélie D'Hondt, Dongwoon Hyun, Bhavik Patel, and Jeremy J Dahl. Vector flow imaging using a deep neural network. *The Journal of the Acoustical Society of America*, 146(4):2901–2902, 2019.

Bo Liu, Ying Wei, Yu Zhang, and Qiang Yang. Deep neural networks for high dimension, low sample size data. In *IJCAI*, pages 2287–2293, 2017.

Yufeng Liu and Yichao Wu. Variable selection via a combination of the l 0 and l 1 penalties. *Journal of Computational and Graphical Statistics*, 16(4):782–798, 2007.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. A provably efficient algorithm for training deep networks. *CoRR, vol. abs/1304.7045*, 2013.

Giampiero Marra and Simon N Wood. Practical variable selection for generalized additive models. *Computational Statistics & Data Analysis*, 55(7):2372–2387, 2011.

Mitsunori Mizumachi and Maya Origuchi. Superdirective non-linear beamforming with deep neural network. *The Journal of the Acoustical Society of America*, 140(4):3167–3167, 2016.

Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.

Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

Simon Perkins, Kevin Lacker, and James Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of machine learning research*, 3 (Mar):1333–1356, 2003.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.

Uri Shaham, Alexander Cloninger, and Ronald R Coifman. Provable approximation properties for deep neural networks. *Applied and Computational Harmonic Analysis*, 44(3): 537–557, 2018.

Jonathan W Siegel and Jinchao Xu. On the approximation properties of neural networks. *arXiv preprint arXiv:1904.02311*, 2019.

Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Ambuj Tewari, Pradeep K Ravikumar, and Inderjit S Dhillon. Greedy algorithms for structurally constrained high dimensional problems. In *Advances in Neural Information Processing Systems*, pages 882–890, 2011.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Ryan J Tibshirani. A general framework for fast stagewise algorithms. *The Journal of Machine Learning Research*, 16(1):2543–2588, 2015.

Warren S Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17 (4):401–419, 1952.

Wen-Jen Tsay, Cliff J Huang, Tsu-Tan Fu, and I-Lin Ho. A simple closed-form approximation for the cumulative distribution function of the composite error of stochastic frontier models. *Journal of Productivity Analysis*, 39(3):259–269, 2013.

Rongrong Wang and Xiaopeng Zhang. Capacity preserving mapping for high-dimensional data visualization. *arXiv preprint arXiv:1909.13322*, 2019.

Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1): 3–34, 2015.

Tong Tong Wu, Kenneth Lange, et al. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

Makoto Yamada, Wittawat Jitkrittum, Leonid Sigal, Eric P Xing, and Masashi Sugiyama. High-dimensional feature selection by feature-wise kernelized lasso. *Neural computation*, 26(1):185–207, 2014.

Xiaohan Yan and Jacob Bien. Hierarchical Sparse Modeling: A Choice of Two Group Lasso Formulations. *Statistical Science*, 32(4):531 – 560, 2017. doi: 10.1214/17-STS622. URL `https://doi.org/10.1214/17-STS622`.

Kaixu Yang and Tapabrata Maiti. Ultra high-dimensional generalized additive model: consistency and tuning parameter selection. *Technical report, Michigan State University*, 2018.

Kaixu Yang and Tapabrata Maiti. Statistical aspects of high-dimensional sparse artificial neural network models. *Machine learning and knowledge extraction*, 2(1):1–19, 2020.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Cun-Hui Zhang and Jian Huang. The sparsity and bias of the lasso selection in high-dimensional linear regression. *The Annals of Statistics*, pages 1567–1594, 2008.

Yiyun Zhang, Runze Li, and Chih-Ling Tsai. Regularization parameter selections via generalized information criterion. *Journal of the American Statistical Association*, 105(489): 312–323, 2010.

Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, 2006.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.

Appendix A. Proof

In this section, we will provide the proof of the theorems in section 4.

Proof of Proposition 4.1

Proof Consider independent observations $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\}$. In the regression set up, we have

$$y|x_1, ..., x_p \sim \mathcal{N}(f^*(x_S), \sigma^2).$$

Let $\hat{S}$ be the set of variables included in the current model. The algorithm computes

$$\boldsymbol{G}_{0j} = \frac{\partial}{\partial \boldsymbol{W}_{0j}} l(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) := (G_{0j1}, ..., G_{0jK})$$

where K is the size of the first hidden layer. Without loss of generality, we may consider a shallow network in this part, since there isn't any predictor x involved in this section, all estimates can be treated as constants, which are universal for all $j's$. We have

$$G_{0jk} = -\frac{2}{n}\sum_{i=1}^{n} y_i \hat{a}_k \sigma'(\sum_{j=1}^{p} x_{ij}\hat{\theta}_{jk} + \hat{t}_k)x_{ij}, \quad k = 1, ..., K$$

where $\hat{a}_k, \hat{t}_k$ are estimated parameters for the initial model and $\hat{\theta}_{jk}$ is set to zero for all input variables at the very beginning. Thus we have

$$\begin{aligned}\|\boldsymbol{G}_{0j}\|_2 &= \sqrt{\sum_{k=1}^{K}[-\frac{2}{n}\sum_{i=1}^{n} y_i \hat{a}_k \sigma'(\sum_{j=1}^{p} x_{ij}\hat{\theta}_{jk} + \hat{t}_k)x_{ij}]^2} \\ &= \frac{2}{n}\sqrt{\sum_{k=1}^{K} \hat{a}_k^2 \sigma'(\hat{t}_k)} |\boldsymbol{x}_{(j)}^T \boldsymbol{y}|\end{aligned}$$

Since the leading constant is independent of j, it's easier to consider the different part, denoted

$$c_j = |\boldsymbol{x}_{(j)}^T \boldsymbol{y}|$$

for $j \in \mathcal{C}$, where $\mathcal{C}$ is the candidate set. The first variable selected is

$$j_+ = \arg\max_{j \in \mathcal{C}} c_j.$$

At the very beginning, we have for $x \geq 0$ that

$$\begin{aligned}\mathbb{P}(c_1 \leq x) &= \mathbb{P}\left(|\boldsymbol{x}_{(1)}^T \boldsymbol{y}| \leq x\right) \\ &= \mathbb{P}\left(-x \leq \boldsymbol{x}_{(1)}^T \boldsymbol{y} \leq x\right) \\ &= \mathbb{P}\left(-x \leq \boldsymbol{x}_{(1)}^T (f^*(X_S) + \epsilon) \leq x\right) \\ &= \mathbb{P}\left(-x \leq \beta_1 + \boldsymbol{x}_{(1)}^T \epsilon \leq x\right) \\ &= \Phi\left(\frac{x - \beta_1}{\sigma\|\boldsymbol{x}_{(1)}\|_2}\right) - \Phi\left(\frac{-x - \beta_1}{\sigma\|\boldsymbol{x}_{(1)}\|_2}\right) \qquad (24)\end{aligned}$$

where $\beta_j = X_{(j)}^T f * (X_S)$ according to Assumption 2. This result implies that greater β leads to a higher probability of large c_1. Then for any two features $X_{(j)}$ and $X_{(k)}$

$$\mathbb{P}(c_j > c_k) = \mathbb{P}\left(|\boldsymbol{x}_{(j)}^T \boldsymbol{y}| > |\boldsymbol{x}_{(k)}^T \boldsymbol{y}|\right) \tag{25}$$

Let

$$W_j = \boldsymbol{x}_{(j)}^T \boldsymbol{y} \qquad and \qquad W_k = \boldsymbol{x}_{(k)}^T \boldsymbol{y} \tag{26}$$

which are both normally distributed. Therefore, c_j and c_k follow folded normal distribution

$$c_j \sim FN(\beta_j, \sigma^2 \|\boldsymbol{x}_{(j)}\|_2^2) \qquad and \qquad c_2 \sim FN(\beta_2, \sigma^2 \|\boldsymbol{x}_{(k)}\|_2^2) \tag{27}$$

We can calculate

$$Cov(W_j, W_k) = Cov(\beta_j + \boldsymbol{x}_{(j)}^T \epsilon, \beta_k + \boldsymbol{x}_{(k)}^T \epsilon) = \sigma^2 \boldsymbol{x}_{(j)}^T \boldsymbol{x}_{(k)} \tag{28}$$

Since both c_j and c_k are positive, the probability is equivalent to

$$\mathbb{P}(c_j > c_k) = \mathbb{P}\left(\frac{c_j}{c_k} > 1\right) \tag{29}$$

Let

$$c_{jk} = \frac{c_j}{c_k}$$

Then we have

$$c_{jk} \sim RFN(\beta_j, \beta_k, \sigma^2 \|\boldsymbol{x}_{(j)}\|_2^2, \sigma^2 \|\boldsymbol{x}_{(k)}\|_2^2, \rho) \tag{30}$$

where RFN stands for the ratio of folded normal distributions. By theorem 3.1 in Kim (2015), we have the CDF of c_{jk}

$$F_{jk}(x) = 2L\left(h_1, -\delta, \rho_1\right) + 2L\left(h_2, \delta, \rho_2\right) + \Phi(h_1) + \Phi(h_2) - 2 \tag{31}$$

where

$$L(a, b, \rho) = \mathbb{P}(Z_1 > a, Z_2 > b), (Z_1, Z_2) \sim N_2(0, 0, 1, 1, \rho), \tag{32}$$

$$h_1 = \frac{\mu_j - \mu_k}{\sqrt{\sigma_j^2 - 2\rho\sigma_j\sigma_k + \sigma_k^2}}, h_2 = \frac{\mu_j + \mu_k}{\sqrt{\sigma_j^2 + 2\rho\sigma_j\sigma_k + \sigma_k^2}} \tag{33}$$

$$\rho_1 = \frac{\sigma_k - \sigma_j\rho}{\sqrt{\sigma_j^2 - 2\rho\sigma_j\sigma_k + \sigma_k^2}}, \rho_2 = \frac{\sigma_k + \sigma_j\rho}{\sqrt{\sigma_j^2 + 2\rho\sigma_j\sigma_k + \sigma_k^2}}, \tag{34}$$

$$\delta = \frac{\mu_k}{\sigma_k} \tag{35}$$

with $\mu_j = X'_{(j)} f^*(X_S), \mu_k = X'_{(k)} f^*(X_S), \sigma_j^2 = \sigma^2 ||X_{(j)}||_2^2, \sigma_k^2 = \sigma^2 ||X_{(k)}||_2^2, \rho = X'_j X_k$ and $\Phi(\cdot)$ is the standard normal distribution CDF.

Then we have

$$\mathbb{P}(X_{(j)} \text{ will be selected before } X_{(k)}) = \mathbb{P}(c_j < c_k) = F_{12}(1) \tag{36}$$

fully characterized by the signal strength β_j, β_k, the noise variance σ^2 and the feature norms $||X_{(j)}||_2^2, ||X_{(k)}||_2^2$, as specified in equations 31, 32. ■

Proof of Proposition 4.2

Proof We continue from the proof of proposition 4.1. Let $\hat{S}$ be the set of variables included in the current model. At the very beginning, we have proved in the proof of proposition 4.1 that

$$c_j \sim FN(\beta_j, \sigma^2 ||X_{(j)}||_2^2) \qquad j = 1, ..., p \tag{37}$$

Denote the event

$$E_k = \{c_k > \max_{i \neq k} c_i\}, \;\; k \in S \tag{38}$$

It's easy to observe that E_k's are mutually exclusive. Therefore, we have

$$\begin{aligned}
&\mathbb{P}(\text{At least one of } \{c_j, j \in S\} \text{ is greater than all of } \{c_j, j \in S^c\}) \\
=&\mathbb{P}\left(\bigcup_{k \in S} E_k\right) \\
=&\sum_{k \in S} \mathbb{P}(E_k)
\end{aligned} \tag{39}$$

We may calculate

$$Pr(E_k) = Pr(c_k > c_{(-k,p-1)})$$

where $c_{(-k,p-1)}$ is the largest order statistic of $c_{(1)}, ...c_{(k-1)}, c_{(k+1)}, c_{(p)}$. Let $F_{(-k,p-1)}$ and $f_{(-k,p-1)}$ be the CDF and PDF of $c_{(-k,p-1)}$, respectively. Then we have

$$\begin{aligned}
&Pr(E_k) \\
=&Pr(c_k > c_{(-k,p-1)}) \\
=&\int_0^\infty Pr(c_k > x) f_{(-k,p-1)}(x) dx \\
=&\int_0^\infty [1 - F_k(x)] \frac{\partial}{\partial x} F_{(-k,p-1)}(x) dx \\
=&\left[[1 - F_k(x)] F_{(-k,p-1)}(x)\right]\Big|_0^\infty + \int_0^\infty f_k(x) F_{(-k,p-1)}(x) dx \\
=&\int_0^\infty f_k(x) F_{(-k,p-1)}(x) dx
\end{aligned} \tag{40}$$

where the second equality is by the convolution formula, and the fourth equality is by integration by parts. Therefore,

$$\begin{aligned}
&Pr(\text{At least one of } \{c_j, j \in S\} \text{ is greater than all of } \{c_j, j \in S^c\}) \\
=&\sum_{k \in S} \int_0^\infty f_k(x) F_{(-k,p-1)}(x) dx
\end{aligned} \tag{41}$$

Next, we will show that although this might be a high probability, this will not converge to one even for large n. Let

$$\begin{aligned}
p_k &= \int_0^\infty f_k(x)\mathbb{P}(\max_{j=\{1,2,\ldots,p\}/\{k\}} c_j \le x)dx \\
&\le \int_0^\infty f_k(x)\mathbb{P}(\min_{j=\{1,2,\ldots,p\}/\{k\}} c_j \le x)dx \\
&= \int_0^\infty f_k(x)\left(1-\mathbb{P}(\min_{j=\{1,2,\ldots,p\}/\{k\}} c_j \ge x)\right)dx \\
&= \int_0^\infty f_k(x)\left(1-\mathbb{P}(c_j \ge x, j\in\{1,2,\ldots,p\}/\{k\})\right)dx \\
&\le \int_0^\infty f_k(x)\left(1-\prod_{j=\{1,2,\ldots,p\}/\{k\}} \mathbb{P}(c_j \ge x)\right)dx \\
&= \int_0^\infty f_k(x)\left(1-\prod_{j=\{1,2,\ldots,p\}/\{k\}} (1-F_j(x)))\right)dx \qquad (42)
\end{aligned}$$

Now, from the properties of folded normal distribution, we have

$$F_k(x) = \frac{1}{2}\left[erf\left(\frac{x+|\beta_k|}{\sqrt{2\sigma^2}}\right) + erf\left(\frac{x-|\beta_k|}{\sqrt{2\sigma^2}}\right)\right]$$

and

$$f_k(x) = \frac{\partial}{\partial x}F_k(x) = \sqrt{\frac{2}{\pi\sigma^2}}e^{-\frac{x^2+\beta_k^2}{2\sigma^2}}\cosh\frac{\beta_k x}{\sigma^2}$$

where the $erf(\cdot)$ is the Gauss-error function; i.e., $erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$. For the ease of calculation, we assumed $||X_{(j)}||_2^2 = 1, \forall j = \{1,2,\ldots,p\}$. By the formulas for F_k and f_k, we have

$$\begin{aligned}
p_k &= \int_0^\infty \sqrt{\frac{2}{\pi\sigma^2}}e^{-\frac{x^2+\beta_k^2}{2\sigma^2}}\cosh(\frac{\beta_k x}{\sigma^2})\left(1-\prod_{j\neq k}\{1-\frac{1}{2}\left[erf\left(\frac{x+\beta_j}{\sqrt{2\sigma^2}}\}\right)+erf\left(\frac{x-\beta_j}{\sqrt{2\sigma^2}}\right)\right]\}\right)dx \\
&= \int_0^\infty \sqrt{\frac{1}{2\pi\sigma^2}}\left[e^{-\frac{(x+\beta_k)^2}{2\sigma^2}}+e^{-\frac{(x-\beta_k)^2}{2\sigma^2}}\right]\left(1-\prod_{j\neq k}\{1-\frac{1}{2}\left[erf\left(\frac{x+\beta_j}{\sqrt{2\sigma^2}}\right)+erf\left(\frac{x-\beta_j}{\sqrt{2\sigma^2}}\right)\right]\}\right)dx
\end{aligned} \qquad (43)$$

Do change of variable $z = x/\sigma$, we have

$$p_k = \int_0^\infty \frac{1}{\sqrt{2\pi}}\left[e^{-\frac{(z+\frac{\beta_k}{\sigma})^2}{2}}+e^{-\frac{(z-\frac{\beta_k}{\sigma})^2}{2}}\right]\left(1-\prod_{j\neq k}\{1-\frac{1}{2}\left[erf\left(\frac{z+\frac{\beta_j}{\sigma}}{\sqrt{2}}\right)+erf\left(\frac{z-\frac{\beta_j}{\sigma}}{\sqrt{2}}\right)\right]\}\right)dz \qquad (44)$$

Let $\tilde{\beta}_k = \beta_k/\sigma$, without loss of generality, assume that $\infty = \beta_0 \geq \beta_1 \geq ... \geq \beta_p \geq \beta_{p+1} = 0$, we have

$$\begin{aligned}
p_k =& \int_0^\infty \frac{1}{\sqrt{2\pi}} \left[e^{-\frac{(z+\tilde{\beta}_k)^2}{2}} + e^{-\frac{(z-\tilde{\beta}_k)^2}{2}} \right] \left(1 - \prod_{j\neq k} \{1 - \frac{1}{2} \left[erf\left(\frac{z+\tilde{\beta}_j}{\sqrt{2}}\right) + erf\left(\frac{z-\tilde{\beta}_j}{\sqrt{2}}\right) \right] \} \right) dz \\
=& \sum_{i=0}^{p} \int_{\beta_{i+1}}^{\beta_i} \frac{1}{\sqrt{2\pi}} \left[e^{-\frac{(z+\tilde{\beta}_k)^2}{2}} + e^{-\frac{(z-\tilde{\beta}_k)^2}{2}} \right] \\
& \left(1 - \prod_{j\neq k} \{1 - \frac{1}{2} \left[erf\left(\frac{z+\tilde{\beta}_j}{\sqrt{2}}\right) + \mathbb{1}_{\{j\geq i+1\}} erf\left(\frac{z-\tilde{\beta}_j}{\sqrt{2}}\right) - \mathbb{1}_{\{j\leq i\}} erf\left(\frac{\tilde{\beta}_j - z}{\sqrt{2}}\right) \right] \} \right) dz
\end{aligned} \tag{45}$$

By the exponential approximation of the error function, see for example Tsay et al. (2013), there exists c_1 and c_2 such that

$$\sup_{x>0} |erf(x) - (1 - \exp[-c_1 x - c_2 x^2])|$$

can be arbitrarily small, where approximately $c_1 \approx 1.095$ and $c_2 \approx 0.7565$. Consider this approximation, we have

$$\begin{aligned}
p_k =& \sum_{i=0}^{p} \int_{\beta_{i+1}}^{\beta_i} \frac{1}{\sqrt{2\pi}} \left[e^{-\frac{(z+\tilde{\beta}_k)^2}{2}} + e^{-\frac{(z-\tilde{\beta}_k)^2}{2}} \right] \\
& \times \left(1 - \prod_{j\neq k} \left\{ 1 - \frac{1}{2} \left[1 + \mathbb{1}_{\{j\geq i+1\}} - \mathbb{1}_{\{j\leq i\}} - e^{\frac{c_1^2}{4c_2}} \right. \right. \right. \\
& \times \left(e^{-\frac{c_2}{2}\left[z + \left(\tilde{\beta}_j + \frac{c_1}{\sqrt{2}c_2}\right)\right]^2} + \mathbb{1}_{\{j\geq i+1\}} e^{-\frac{c_2}{2}\left[z + \left(-\tilde{\beta}_j + \frac{c_1}{\sqrt{2}c_2}\right)\right]^2} \right. \\
& \left. \left. \left. \left. - \mathbb{1}_{\{j\leq i\}} e^{-\frac{c_2}{2}\left[z - \left(\tilde{\beta}_j + \frac{c_1}{\sqrt{2}c_2}\right)\right]^2} \right) \right] \right\} \right) dz
\end{aligned} \tag{46}$$

Here

$$e^{\frac{c_1^2}{4c_2}} \approx 1.48 >> 1$$

Observe that as when $i = s$, also observe that assumption 2 indicates $\max_{j=s+1,...,p} \beta_j \to 0$, we have

$$\prod_{j=1, j\neq k}^{s} \frac{1}{2} e^{\frac{c_1^2}{4c_2}} \left[e^{-\frac{c_2}{2}\left[z - \left(\beta_j + \frac{c_1}{\sqrt{2}c_2}\right)\right]^2} - e^{-\frac{c_2}{2}\left[z + \left(\beta_j + \frac{c_1}{\sqrt{2}c_2}\right)\right]^2} \right] \to 0 \; as \; s \to \infty$$

Therefore, the formula of p_k can be simplified to

$$\begin{aligned}
p_k = & \, o\left(\frac{1}{2^s} e^{\frac{sc_1^2}{4c_2}}\right) + \sum_{i=0}^{s} \int_{\beta_{i+1}}^{\beta_i} \frac{1}{\sqrt{2\pi}} \left[e^{-\frac{(z+\tilde{\beta}_k)^2}{2}} + e^{-\frac{(z-\tilde{\beta}_k)^2}{2}} \right] \\
& \times \left(1 - \prod_{j \neq k} \left\{ 1 - \frac{1}{2} \left[1 + \mathbb{1}_{\{j \geq i+1\}} - \mathbb{1}_{\{j \leq i\}} - e^{\frac{c_1^2}{4c_2}} \right. \right. \right. \\
& \times \left(e^{-\frac{c_2}{2}\left[z+\left(\tilde{\beta}_j+\frac{c_1}{\sqrt{2}c_2}\right)\right]^2} + \mathbb{1}_{\{j \geq i+1\}} e^{-\frac{c_2}{2}\left[z+\left(-\tilde{\beta}_j+\frac{c_1}{\sqrt{2}c_2}\right)\right]^2} \right. \\
& \left. \left. \left. \left. - \mathbb{1}_{\{j \leq i\}} e^{-\frac{c_2}{2}\left[z-\left(\tilde{\beta}_j+\frac{c_1}{\sqrt{2}c_2}\right)\right]^2} \right) \right] \right\} \right) \\
\leq & \, o\left(\frac{1}{2^s} e^{\frac{sc_1^2}{4c_2}}\right) + \sum_{i=0}^{s} \left(\frac{1}{2} e^{\frac{c_1^2}{4c_2}} \right)^{s-i} \\
& \times \frac{1}{2s} \left[\Phi(\tilde{\beta}_i - \tilde{\beta}_k) - \Phi(\tilde{\beta}_{i+1} - \tilde{\beta}_k) + \Phi(\tilde{\beta}_i + \tilde{\beta}_k) - \Phi(\tilde{\beta}_{i+1} + \tilde{\beta}_k) \right] \qquad (47)
\end{aligned}$$

where Φ is the normal CDF and the inequality is by observing

$$e^{-x^2} \leq 1$$

and the term in the bracket is less than 2 when $j \geq i+1$. Then summing up $p_k's$ and observing the double sum is not converging to zero since it consists of a geometric component, when β_{max} is not big enough and let $s \to \infty$, we have

$$\begin{aligned}
& 1 - \sum_{k=1}^{s} p_k \\
\geq & 1 - o\left(s \frac{1}{2^s} e^{\frac{sc_1^2}{4c_2}} \right) - \sum_{k=1}^{s} \sum_{i=0}^{s} \left(\frac{1}{2} e^{\frac{c_1^2}{4c_2}} \right)^{s-i} \\
& \frac{1}{2s} \left[\Phi(\tilde{\beta}_i - \tilde{\beta}_k) - \Phi(\tilde{\beta}_{i+1} - \tilde{\beta}_k) + \Phi(\tilde{\beta}_i + \tilde{\beta}_k) - \Phi(\tilde{\beta}_{i+1} + \tilde{\beta}_k) \right] \\
\geq & \sum_{i=1}^{s} \left(\frac{1}{2} e^{\frac{c_1^2}{4c_2}} \right)^{s-i} \sum_{k=1}^{s} \frac{1}{2s} (1 - \Phi(\beta_{max})) - o\left(s \frac{1}{2^s} e^{\frac{sc_1^2}{4c_2}} \right) \\
\geq & c - o(1) \qquad (48)
\end{aligned}$$

This implies, that there is always a positive probability that a null feature gets selected at the beginning even when $n \to \infty$. ■

Proof of Theorem 4.1

Proof In this proof, first, we will show the probability that the same zero predictor appears in k bagging rounds tends to zero as k increases. Then, we prove that the probability that the model is able to include all the non-null features tens to one asymptotically.

At the first step, we have $\mathcal{C} = \{1, ..., p\}$ and $\mathcal{S} = \{\}$ and include the variables sequentially. At the m^{th} step, denote the candidate set $\mathcal{C}^m$ and the selected set $\mathcal{S}^m$. First, assume that there is at least one non-null predictor in the the candidate set $\mathcal{C}^m$; i.e. $|\mathcal{C}^m \cap S_0| \neq \phi$. Consider the j-th feature and the following estimator: $\hat{s}_j^m = \mathbb{1}_{\{\|\boldsymbol{G}_{mj}\|_2 \leq t^m\}}$. Hence, conditioning on the observations, there exists a fixed t_m such that $\hat{s}_j^m$ indicates whether the j-th feature is not selected (=1) or selected (=0). The bagged estimator is defined as

$$\hat{s}_{j,B}^m = \mathbb{E}\left[\mathbb{1}_{\{\|\boldsymbol{G}_{mj}^*\|_2 \leq t_m\}}\right] \tag{49}$$

where $\boldsymbol{G}_{mj}^*$ is $\boldsymbol{G}_{mj}$ evaluated on a bootstrap sample. By the uniform law of large numbers, see for example Györfi et al. (2006), we have

$$\sup_{\boldsymbol{x},y}\left|\frac{1}{B_2}\sum_{b=1}^{B_2}\mathbb{1}_{\{\|\boldsymbol{G}_{mj,b}^*\|_2 \leq t_m\}} - \mathbb{E}\left[\mathbb{1}_{\{\|\boldsymbol{G}_{mj}^*\|_2 \leq t_m\}}\right]\right| \to 0 \quad as\ B_2 \to \infty \tag{50}$$

Let $\mathbb{P}_n^B$ be the empirical measure of the bootstrap sample. By algebra, in the m-th step,

$$||G_{mj}||_2 = \sqrt{\sum_{k=1}^{k}\left[-\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{\mu}_i)\hat{a}_k\sigma'(x_i'\hat{\theta}_k + \hat{t}_k)x_{ij}\right]^2} \tag{51}$$

Here, $\hat{\theta}_{jk}$ is estimated from data for $j \in \mathcal{S}^m$ and $\hat{\theta}_{jk} = 0$ for $j \in \mathcal{C}^m$, $\hat{\mu}_i)$ is the neural network estimate of y_i based on $X_{\mathcal{S}^m,i}$. Let $\hat{\epsilon}_i = y_i - \hat{\mu}_i$ be the prediction error for i-th data point. Also, let, $\sigma_{ik}' = \sigma'(x_i'\hat{\theta}_k + \hat{t}_k)$. Then,

$$\begin{aligned}||G_{mj}||_2 &= \frac{2}{n}\sqrt{\sum_{k=1}^{k}\hat{a}_k^2\left[\sum_{i=1}^{n}\hat{\epsilon}_i\sigma_{ik}'x_{ij}\right]^2} &(52)\\ &\leq \frac{2}{n}\sqrt{\sum_{k=1}^{k}\hat{a}_k^2\left[\max_{i=1,2,...,n}\sigma_{ik}'\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\hat{\epsilon}_i x_{ij}\geq 0} + \min_{i=1,2,...,n}\sigma_{ik}'\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\hat{\epsilon}_i x_{ij}<0}\right]^2 (\hat{\epsilon}'x_{(j)})^2} &(53)\end{aligned}$$

Similarly,

$$||G_{mj}||_2 \geq \frac{2}{n}\sqrt{\sum_{k=1}^{k}\hat{a}_k^2\left[\min_{i=1,2,...,n}\sigma_{ik}'\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\hat{\epsilon}_i x_{ij}\geq 0} + \max_{i=1,2,...,n}\sigma_{ik}'\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\hat{\epsilon}_i x_{ij}<0}\right]^2 (\hat{\epsilon}'x_{(j)})^2} \tag{54}$$

Now, we can decompose the last term in equations 52and 54 as

$$\begin{aligned}\hat{\epsilon}'x_{(j)} &= (y - \hat{\mu})'X_{(j)}\\ &= (f^*(X_S) + \epsilon - \hat{\mu})'X_{(j)}\\ &= X_{(j)}'f^*(X_S) + \epsilon'X_{(j)} - \hat{\mu}'X_{(j)}\\ &= \beta_j + \epsilon'X_{(j)} - \hat{\mu}'X_{(j)} &(55)\end{aligned}$$

Due to the bounded nature of the parameter space Θ and the features by assumption 3, the predicted values $\hat{\mu}$ would be bounded. Hence, by assumption 2 , for $j \in \mathcal{C}^m \cap S, E||G_{mj}||_2 \geq \frac{c\gamma_n}{\sqrt{n}}$. Additionally, for $j \in \mathcal{C}^m \cap S^c, E||G_{mj}||_2 \leq \frac{c'k}{n}$. Hence, if $\gamma_n \geq \frac{c'k}{c\sqrt{n}}$, $\min_{j\in\mathcal{C}^m\cap S} E||G_{mj}||_2 \geq \max_{\mathcal{C}^m\cap S^c} E||G_{mj}||_2$.

Hence, from now on, let us set the t_m in the bagging estimator 49 as $t_m = \frac{1}{2}\left(\frac{c'k}{n} + \frac{c\gamma_n}{n}\right)$. Later we will discuss how to estimate the abstract t_m by a data-adaptive procedure. Now, let $||G_{mj}||_2 = \frac{2}{n}\sqrt{J}$ where, for any arbitrary $\theta \in \Theta$,

$$J = \sum_{k=1}^{k} \hat{a}_k^2 \left[\sum_{i=1}^{n} \hat{\epsilon}_i \sigma'_{ik} x_{ij}\right]^2 = c_k \left(\sum_{i=1}^{n} (y_i - \mathcal{D}_\theta(x_i))\sigma_{ik} x_{ij}\right)^2 \tag{56}$$

Hence, for any fixed $\theta \in \Theta$,

$$(y_i - \mathcal{D}_\theta(x_i))\sigma_{ik}x_{ij} \sim N\left((f^*(X_{i,S}) - \mathcal{D}_\theta(x_i))\sigma_{ik}x_{ij}, \sigma^2(\sigma_{ik}x_{ij})^2\right) \tag{57}$$

Consequently, by the central limit theorem (CLT), for some increasing sequence b_n, $b_n(||G_{mj}||_2 - c_0) \rightarrow N(0, \sigma_\infty^2)$.

Now, $\sup_{\theta\in\Theta} J \leq KW^2(\sum_{i=1}^n (y_i + W)x_{ij})^2$ would also follow the CLT. Also, J is continuous wrt θ; hence $\sup_{\theta\in\Theta} J \in$ the domain of J. This implies that for an estimated $\hat{\theta}$, there exists an increasing sequence b_n, for which $b_n(||G_{mj}(\hat{\theta})||_2 - c_0) \rightarrow N(0, \sigma_\infty^2)$.

Also, as $\mathcal{D}_\theta$ is Lipsctitz continuous wrt θ, $||G_{mj}||_2$ is bounded. Hence, by Giné and Zinn (1990) and Bühlmann et al. (2002), the bootstrap consistency is preserved; i.e.,

$$\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{||G^*_{mj,b}||_2 \leq t_m\}} \rightarrow \Phi\left(\frac{b_n(t^m - c_0)}{\sigma_\infty} - Z\right) \quad \textit{as } n \textit{ and } B_2 \rightarrow \infty \tag{58}$$

Now, for $j \in \mathcal{C}^m \cap S^c$,

$$\begin{aligned}\mathbb{P}(j \in \mathcal{C}^m \cap \mathcal{S}^{m+1} \cap S^c) &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{||G^*_{mj,b}||_2 \geq t_m\}} \geq p_s, j \in \mathcal{C}^m \cap S^c\right) \\ &\approx \mathbb{P}\left(\Phi\left(\frac{b_n(t^m - c_0)}{\sigma_\infty} - Z\right) \leq 1 - p_s\right) \\ &\rightarrow 0\end{aligned} \tag{59}$$

Now, consider the other scenario, where $\mathcal{C}^m \cap S^c = \phi$. Here, we will show that the same null feature appears more than the threshold-limited number of times in all the bagging rounds with probability converging to zero. It can be shown as,

$$\begin{aligned}\mathbb{P}(j \in \mathcal{C}^m \cap \mathcal{S}^{m+1^c} \cap S^c) &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{||G^*_{mj,b}||_2 \geq t_m\}} \leq p_s, j \in \mathcal{C}^m\right) \\ &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{||G^*_{mj,b}||_2 \leq t_m\}} \geq 1 - p_s, j \in \mathcal{C}^m \cap S^c\right) \\ &\approx \mathbb{P}\left(\Phi\left(\frac{b_n(t^m - c_0)}{\sigma_\infty} - Z\right) \geq 1 - p_s\right) \\ &\rightarrow 1\end{aligned} \tag{60}$$

Hence, the same null feature appears in many bagging rounds only with probability tending to zero. This further implies that the overall false positive rate of ENNS goes to zero.

Now, to prove that the power of ENNS converges to one, it is sufficient to show that the model will be able to include all the non-null predictors. Here, WLG, we assume the first s features are the true non-null features; i.e. $S = \{1, 2, \ldots, s\}$. Then we consider the worst case where $\mathcal{S}^m = \{1, 2, \ldots, s-1\}, \mathcal{C}^m = \{s, s+1, \ldots, p\}$. Hence,

$$\begin{aligned}\mathbb{P}(s \in \mathcal{S}^{m+1} \cap \mathcal{C}^m) &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{\|G^*_{ms,b}\|_2 \geq t_m\}} \leq p_s\right) \\ &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{\|G^*_{ms,b}\|_2 \leq t_m\}} \geq 1 - p_s\right) \\ &\approx \mathbb{P}\left(\Phi\left(\frac{b_n(t^m - c_0)}{\sigma_\infty} - Z\right) \geq 1 - p_s\right) \\ &\to 0 \end{aligned} \tag{61}$$

Finally, we talk about the abstract form of $t_m = \frac{1}{2}\left(\frac{c'k}{n} + \frac{c\gamma_n}{n}\right)$. This choice was justified because it can achieve the nice separation between the null and non-null features; e.g. for $j \in \mathcal{C}^m \cap S, \mathbb{P}\left(||G_{mj}||_2 \geq t_m\right) = 1 - \Phi\left(\frac{b_n(t^m - E||G_{mj}||_2)}{\sigma_\infty} - Z\right) \to 1$; and for $j \in \mathcal{C}^m \cap S^c, \mathbb{P}\left(||G_{mj}||_2 \leq t_m\right) = \Phi\left(\frac{b_n(t^m - E||G_{mj}||_2)}{\sigma_\infty} - Z\right) \to 1$ However, instead of fixing the t_m at this abstract value, we propose to start each iteration by setting t_m as the maximum observed $||G_{mj}||_2$ in the candidate set $\mathcal{C}^m$. Hence, one after another all the non-null predictors will be selected. Next, as soon as t_m becomes the maximum $||G_{mj}||_2$ in the null feature set, i.e. $j \in \mathcal{C}^m \cap S$,

$$\begin{aligned}&\mathbb{P}(j_0 \in \mathcal{C}^m \cap \mathcal{S}^{m+1} \cap S^c), \text{ where } j_0 \in \mathcal{C}^m \cap S^c, ||G_{mj_0}||_2 = \max_{j \in \mathcal{C}^m} ||G_{mj}||_2 \\ &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{\|G^*_{mj_0,b}\|_2 \geq t_m\}} \geq p_s\right) \\ &= \mathbb{P}\left(\frac{1}{B_2}\sum_{b=1}^{B_2} \mathbb{1}_{\{\|G^*_{mj_0,b}\|_2 \leq t_m\}} \leq 1 - p_s\right) \\ &\approx \mathbb{P}\left(\Phi\left(\frac{b_n(t^m - ||G_{mj_0}||_2)}{\sigma_\infty} - Z\right) \leq 1 - p_s\right) \\ &= 1 - p_s, \text{ where } p_s = 1 - o(1/\sqrt{n}) \end{aligned} \tag{62}$$

■

Proof of Theorem 4.2

Proof In this subsection, we prove the estimation and prediction of regression and classification, respectively. In the regression set up, under assumption 1, we have

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon = f(\boldsymbol{x}_{\mathcal{S}}) + \epsilon$$

We have

$$\begin{aligned}
&\mathbb{P}\left(\mathbb{E}\int |f_n(\boldsymbol{x}_{\hat{\mathcal{S}}}) - f(\boldsymbol{x}_{\mathcal{S}})|^2\mu(dx) \to 0\right)\\
=&\mathbb{P}\left(\mathbb{E}\int |f_n(\boldsymbol{x}_{\hat{\mathcal{S}}}) - f(\boldsymbol{x}_{\mathcal{S}})|^2\mu(dx) \to 0 \middle| \hat{\mathcal{S}} = \mathcal{S}\right)\mathbb{P}\left(\hat{\mathcal{S}} = \mathcal{S}\right)\\
&+\mathbb{P}\left(\mathbb{E}\int |f_n(\boldsymbol{x}_{\hat{\mathcal{S}}}) - f(\boldsymbol{x}_{\mathcal{S}})|^2\mu(dx) \to 0 \middle| \hat{\mathcal{S}} \neq \mathcal{S}\right)\mathbb{P}\left(\hat{\mathcal{S}} \neq \mathcal{S}\right)\\
\geq&\mathbb{P}\left(\mathbb{E}\int |f_n(\boldsymbol{x}_{\hat{\mathcal{S}}}) - f(\boldsymbol{x}_{\mathcal{S}})|^2\mu(dx) \to 0 \middle| \hat{\mathcal{S}} = \mathcal{S}\right)\mathbb{P}\left(\hat{\mathcal{S}} = \mathcal{S}\right)\\
=&\mathbb{P}\left(\mathbb{E}\int |f_n(\boldsymbol{x}_{\mathcal{S}}) - f(\boldsymbol{x}_{\mathcal{S}})|^2\mu(dx) \to 0\right)\mathbb{P}\left(\hat{\mathcal{S}} = \mathcal{S}\right) \qquad (63)
\end{aligned}$$

Observe that

$$\left|\hat{\boldsymbol{\beta}}\right| \leq |\boldsymbol{\theta}|_1 \leq K_n$$

According to Györfi et al. (2006), when we perform a neural network estimation on the true subset of variables, we have that the total error is bounded by the approximation error, which is bounded according to Fan et al. (2020), plus the estimation error, which is bounded by the covering number, then by the packing number, then by the Vapnik-Chervonenkis dimension, and finally by the space dimension, i.e.

$$\begin{aligned}
&\mathbb{E}\int |f_n(\boldsymbol{x}_{\mathcal{S}}) - f(\boldsymbol{x}_{\mathcal{S}})|^2\mu(dx)\\
=&O\left(L\sqrt{\frac{k_n}{n-1}}\right) + \delta_n \qquad (64)
\end{aligned}$$

where L is the Lipshitz continuity coefficient, k_n is the first hidden layer size, and by Györfi et al. (2006) δ_n satisfies

$$\mathbb{P}\left\{\sup \delta_n > \epsilon\right\} \leq 8\left(\frac{384K_n^2(k_n+1)}{\epsilon}\right)^{(2s+5)k_n+1} e^{-n\epsilon^2/128\cdot 2^4 K_n^4}$$

Under theorem assumptions, the probability above is summable, thus we have the first probability in 63 converges to 1. On the other hand, by theorem 4.1, we have the second probability in 63 converges to 1. Therefore, the result for regression set up is proved.

In the classification set up, similarly, we have

$$\begin{aligned}
&\mathbb{P}\left(R(f_{n,\hat{\mathcal{S}}}) - R(f^*_{\mathcal{S}}) \to 0\right)\\
=&\mathbb{P}\left(R(f_{n,\hat{\mathcal{S}}}) - R(f^*_{\mathcal{S}}) \to 0 \middle| \hat{\mathcal{S}} = \mathcal{S}\right)\mathbb{P}\left(\hat{\mathcal{S}} = \mathcal{S}\right) + \mathbb{P}\left(R(f_{n,\hat{\mathcal{S}}}) - R(f^*_{\mathcal{S}}) \to 0 \middle| \hat{\mathcal{S}} \neq \mathcal{S}\right)\mathbb{P}\left(\hat{\mathcal{S}} \neq \mathcal{S}\right)\\
\geq&\mathbb{P}\left(R(f_{n,\hat{\mathcal{S}}}) - R(f^*_{\mathcal{S}}) \to 0 \middle| \hat{\mathcal{S}} = \mathcal{S}\right)\mathbb{P}\left(\hat{\mathcal{S}} = \mathcal{S}\right)\\
=&\mathbb{P}\left(R(f_{n,\mathcal{S}}) - R(f^*_{\mathcal{S}}) \to 0\right)\mathbb{P}\left(\hat{\mathcal{S}} = \mathcal{S}\right) \qquad (65)
\end{aligned}$$

By Devroye et al. (2013), we have

$$R(f_n) - R(f^*) \to 0 \quad as\ n \to \infty$$

and from theorem 4.1, we have the second probability in equation 65 tends to 1. Combine these two results, the consistency of classification case is proved. ■