

DYNAMIC PAIRWISE SPARSE TUNING (DPST) VS. STATIC TWO-PREDICTOR  
SELECTION: A NEURAL NETWORK APPROACH

by

Raymond Dacosta Azadda

B.Sc. University Cape Coast, Ghana, 2022

A Project Submitted in Partial Fulfillment of the Requirements  
for the Degree of

Master of Science

in

Statistics and Data Science

University of Alaska Fairbanks

May 2025

APPROVED:

Scott Goddard, Committee Chair

Julie McIntyre, Committee Member

Margaret Short, Committee Member

Leah Berman, Chair

Department of Mathematics and Statistics



© Copyright by Raymond Dacosta Azadda  
All Rights Reserved

## **Dedication**

This research project is dedicated to Almighty God for His guidance and blessings, to my beloved family for their love and support, and to my supervisor, Dr. Scott Goddard, for his invaluable mentorship throughout this journey.

## Abstract

Identifying two active predictors driving a response variable is critical in fields like genomics, medicine, and finance, yet standard methods, such as penalized regression, often fail to isolate these pairs consistently. We propose Dynamic Pairwise Sparse Tuning (DPST), a novel feedforward neural network (FNN) method that enhances sparse predictor selection by augmenting standard backpropagation with custom weight updates using adaptive thresholding, smoothed refinement, and pruning. Through simulations across predictor counts ( $P = 3, 4, 5$ ) and sample sizes ( $N = 1800, 3600$ ) using a controlled sparse coefficient matrix defining pair relationships, DPST consistently outperforms our static FNN baseline, also developed by us, which uses only backpropagation. For instance, DPST achieves an accuracy of 0.732 versus 0.609 at  $P = 5$ ,  $N = 3600$ , across  $C = \binom{P}{2}$  pairs (3, 6, 10). The baseline excels at  $P = 3$ ,  $N = 3600$  (accuracy 0.960 vs. 0.684), where DPST’s updates limit generalization, and trains faster (e.g., 3.57 s vs. 20.66 s). DPST’s precision suits applications like gene-pair detection and financial risk modeling, while the static baseline supports rapid analyses. Our results highlight DPST’s potential to advance sparse modeling.

## **Acknowledgements**

I express my profound gratitude to God for His protection and blessings throughout this academic journey, enabling me to complete this research project successfully. I extend my sincere gratitude, to my supervisor, Dr. Scott Goddard, for his ideas, insightful guidance, and feedback, which were pivotal to this project's success. I also extend a heartfelt thank you to the faculty of the Department of Mathematics and Statistics at the University of Alaska Fairbanks, especially Dr. Julie McIntyre, Dr. Ronald Barry, and Dr. Margaret Short, for their exceptional mentorship and teaching. My thanks go to my family for their constant prayers, support, and inspiration. I acknowledge my colleagues, and also the missionaries from The Church of Jesus Christ of Latter-day Saints, for their uplifting prayers and spiritual support.

## Table of Contents

Copyright .....	iii
Dedication .....	iv
Abstract .....	v
Acknowledgements .....	vi
Table of Contents .....	vii
List of Figures .....	x
List of Tables .....	xi
Chapter 1: Introduction .....	1
1.1 Research Background .....	1
1.2 Problem Statement.....	1
1.3 Objectives of the Study.....	2
1.3.1 General Objective .....	2
1.3.2 Specific Objectives .....	2
1.4 Research Questions .....	3
1.5 Significance of the Study .....	3
1.6 Limitation of the Study.....	3
Chapter 2: Literature Review .....	4
2.1 Overview .....	4
2.2 Static Methods for Selecting Predictors .....	4
2.3 Neural Network Approaches .....	5
2.4 Dynamic Methods .....	5
2.5 Research Gaps .....	6
Chapter 3: Methodology .....	7

3.1 Overview .....	7
3.2 Data Generation .....	7
3.2.1 Case 1: Independent Predictors .....	8
3.2.2 Case 2: Correlated Predictors .....	8
3.3 Sparse Coefficient Matrix .....	8
3.4 The Linear Predictors.....	10
3.5 Reshaping and Labeling.....	10
3.5.1 Block Partitioning .....	11
3.5.2 Response Reshaping.....	11
3.5.3 Predictor Reshaping .....	12
3.5.4 Matrix-Label Integration .....	13
3.5.5 Purpose of Reshaping .....	13
3.6 Feedforward Artificial Neural Networks .....	14
3.6.1 FNN Architecture .....	14
3.7 Static Baseline .....	15
3.8 Dynamic Pairwise Sparse Tuning (DPST).....	16
3.8.1 Weight Extraction .....	16
3.8.2 Weight-Distribution Thresholding.....	16
3.8.3 Smoothed Refinement .....	16
3.8.4 Dual-Mode Pruning .....	17
3.8.5 Implementation Summary .....	18
3.8.6 Simulation Role of the Sparse Matrix $A$ .....	19
3.9 Performance Evaluation .....	19
3.9.1 Confusion Matrix Metrics .....	19
3.9.2 Accuracy .....	20
3.9.3 Sensitivity (Recall) .....	20
3.9.4 Specificity.....	20
3.9.5 Precision .....	21
3.9.6 Negative Predictive Value (NPV).....	21
3.9.7 F1-Score .....	21

3.9.8	Cohen’s Kappa.....	21
3.10	Computational Metrics .....	22
3.10.1	Training Time.....	22
3.10.2	Memory Usage .....	22
Chapter 4:	Results .....	23
4.1	Results.....	23
4.2	Overview of Simulation Designs and Data Conditions .....	23
4.2.1	Predictor Correlation Structures .....	23
4.2.2	Coefficient Magnitude Ranges of Design Matrix $A$ .....	23
4.2.3	Results Interpretations.....	24
4.2.4	Predictive and Computational Results at $P = 3$ , $C = \binom{3}{2} = 3$ Pair Combinations .....	24
4.2.5	Predictive and Computational Results at $P = 4$ , $C = \binom{4}{2} = 6$ Pair Combinations .....	27
4.2.6	Predictive and Computational Results at $P = 5$ , $C = \binom{5}{2} = 10$ Pair Combinations .....	30
4.2.7	Discussion .....	33
Chapter 5:	Conclusions .....	34
Chapter 6:	Reference .....	35
6.1	References.....	35

## List of Figures

Figure 3.1: Feedforward Neural Network Architecture .....	15
Figure 4.1: Performance comparison at $P = 3$ .....	25
Figure 4.2: Performance comparison at $P = 4$ .....	28
Figure 4.3: Performance comparison at $P = 5$ .....	31

## List of Tables

Table 4.1: Performance comparison between DPST and static baseline across sample sizes at $P = 3$ .....	24
Table 4.2: Computational comparison between DPST and static baseline across sample sizes at $P = 3$ .....	26
Table 4.3: Performance comparison between DPST and static baseline across sample sizes at $P = 4$ .....	27
Table 4.4: Computational comparison between DPST and static baseline across sample sizes at $P = 4$ .....	29
Table 4.5: Performance comparison between DPST and static baseline across sample sizes at $P = 5$ .....	30
Table 4.6: Computational comparison between DPST and static baseline across sample sizes at $P = 5$ .....	32

## Chapter 1: Introduction

### 1.1 Research Background

In predictive modeling, it is common to have a large set of candidate variables, but often, only a small number drive the outcome. Identifying these active predictors from a larger set presents a significant challenge since they typically work together. This task is important in fields like genomics, where specific gene pairs might determine treatment outcomes, or finance, where two factors might dominate risk assessment. Traditional statistical methods, such as penalized regression, select significant predictors by shrinking irrelevant ones to zero. However, they are not designed to identify exactly two active predictors. Feedforward neural networks (FNNs) excel at prediction by capturing complex patterns, but they treat all predictors equally during training, complicating the identification of the two most influential pairs without additional guidance.

We propose **Dynamic Pairwise Sparse Tuning (DPST)**, a simulation framework using FNNs that refines the selection of two active predictors during training. Unlike static methods with fixed selections, DPST dynamically updates neural network weights to enforce sparse predictor selection, prioritizing the most relevant pair. The underlying relationships between predictors and responses remain fixed in the simulation data, with DPST's innovation lying in weight adjustments to identify these relationships. To manage larger predictor sets efficiently, DPST groups variables into blocks, processing within-block relationships to highlight the key pair. This approach uses FNNs' flexibility while addressing computational challenges through structured sparsity. We compare DPST to a static baseline, an FNN with fixed weights for sparse selection, to assess its ability to identify the true active pair.

### 1.2 Problem Statement

The primary challenge is accurately identifying the two active predictors from a larger set when their influence on the response is unknown. Penalized regression methods select predictors but

often fail to identify exactly two, lacking adaptability to complex relationships. While FNNs excel at prediction, they lack a mechanism to select only two predictors, often leading to computationally costly models as candidate numbers grow. This difficulty increases when relationships between predictors and responses are complex, rendering rigid approaches less effective.

A method that refines predictor selection during training is valuable for efficiently handling large predictor sets, where non-adaptive methods struggle to balance accuracy and computational demands. DPST addresses this by dynamically adjusting neural network weights for sparse predictor selection within blocks, balancing accuracy and efficiency against a static baseline. This is critical for applications requiring precise models, such as identifying gene pairs for therapies in genomics or key risk factors in finance.

### **1.3 Objectives of the Study**

#### **1.3.1 General Objective**

Develop and evaluate **Dynamic Pairwise Sparse Tuning (DPST)**, an FNN-based framework that refines the selection of two active predictors by dynamically updating neural network weights for sparse predictor selection during training, aiming to improve accuracy over a static baseline without weights updates.

#### **1.3.2 Specific Objectives**

1. Build a simulation with within-block structuring for DPST and a static baseline to process candidate predictors and identify the two active predictors in FNNs.
2. Apply adaptive thresholding and pruning in DPST to dynamically update neural network weights, targeting the two most influential predictors during training.
3. Create a static FNN baseline with fixed weights for sparse selection to compare with DPST.
4. Measure the accuracy and computational efficiency of DPST and the static baseline in identifying the true active pair.

## 1.4 Research Questions

1. Under what conditions (e.g., independent vs. correlated predictors) does DPST outperform the static baseline in identifying the two active predictors through adaptive weight adjustments?
2. How do accuracies of DPST and the static baseline vary as observations ( $N$ ) increase, and what does this imply for scaling to larger datasets?
3. How do accuracy and computational time differ between DPST and the static baseline as predictors ( $P$ ) grow, and what does this suggest for larger candidate sets?

## 1.5 Significance of the Study

DPST enhances the ability to select two active predictors from a larger set by dynamically updating neural network weights for sparse predictor selection, improving model precision. Its within-block approach efficiently targets the key pair. Comparing DPST to a static FNN baseline with fixed weights highlights adaptive vs. fixed strategies, offering insights for neural network-based selection methods and addressing computational challenges in decision-critical tasks like medical treatment or financial strategy optimization, where precision and efficiency are important.

## 1.6 Limitation of the Study

This study uses synthetic datasets, so DPST and the static baseline may not fully capture real-world variability in genomics or finance. Grouping predictors into blocks, treated as subsets of predictors processed together, reduces computational demands but not entirely for large sets. Focusing on FNNs excludes other network types (e.g., convolutional, recurrent), narrowing scope. Future work addressing these limits will strengthen the approach.

## Chapter 2: Literature Review

### 2.1 Overview

In many fields, identifying exactly two active predictors from a larger set is crucial for understanding what drives a response variable. This task is valuable but challenging, as non-adaptive methods often struggle to efficiently handle complex predictor relationships. Existing approaches typically lack the flexibility to refine their focus during modeling, particularly when capturing intricate patterns in large datasets. This chapter reviews traditional and neural network methods for selecting two active predictors, highlighting their limitations in adaptability and precision. It identifies a gap that is the absence of methods that adjust during training to isolate the key pair. Our approach, **Dynamic Pairwise Sparse Tuning (DPST)**, addresses this by using feedforward neural networks (FNNs) to adaptively update weights for sparse predictor selection, improving accuracy over static methods.

### 2.2 Static Methods for Selecting Predictors

Traditional statistical methods use penalties to identify important predictors. Lasso (Tibshirani, 1996) applies a penalty to shrink less relevant predictors to zero, effectively selecting a small subset. Elastic Net (Zou & Hastie, 2005) improves on this by combining penalties to better handle related predictors. While these methods excel at picking individual predictors, they are not tailored to select exactly two active predictors across diverse datasets.

Other approaches, such as those with pairwise terms (Bien *et al.*, 2013), aim to model effects between two predictors. These are applied in genomics (Cordell, 2009) to detect gene pairs and in finance (Tsai & Hsu, 2010) to identify risk factors. However, these methods lack mechanisms to adapt predictor selection during modeling, limiting their ability to refine focus on the most relevant pair. This highlights the need for a method like DPST that adaptively adjusts neural network weights during training to target the key pair.

### 2.3 Neural Network Approaches

Feedforward neural networks (FNNs) are strong at prediction by learning complex patterns (Rudin, 2019), but they process all predictors equally during training, making it difficult to single out two active ones without extra steps. Techniques like group sparsity (Scardapane *et al.*, 2017) penalize groups of predictors to simplify the model, yet they don't specifically target two predictors. Pruning methods (Frankle & Carbin, 2018) remove less important weights after training, boosting efficiency but not refining predictor selection during the process.

Structured sparsity (Han *et al.*, 2015; Wang & Chen, 2021) also reduces network size for faster computation, but it focuses on overall trimming rather than isolating two predictors. These approaches either commit to selections early or adjust too broadly during training, missing the chance to focus on the key pair as learning progresses. DPST addresses this by adaptively updating weights for sparse predictor selection within FNNs during training.

### 2.4 Dynamic Methods

Some methods attempt to adapt predictor selection during modeling. Neural pruning (Frankle & Carbin, 2018) cuts weights after training based on their magnitude, but this post-training adjustment doesn't help focus on two predictors during learning. Graph Neural Networks (GNNs) (Zhou & Zhang, 2022) can update relationships as they train, which could suit pair selection, but they rely on predefined structures, reducing their flexibility for this task.

These dynamic methods either miss the mark on selecting two active predictors or demand excessive setup, leaving space for a simpler, training-time solution. DPST fills this gap by adaptively adjusting weights for sparse predictor selection within FNNs, offering a direct way to identify the two most influential predictors as the model trains.

## 2.5 Research Gaps

Static methods like Lasso (Tibshirani, 1996) and pairwise terms (Bien *et al.*, 2013) are easy to use but cannot adapt their selections to capture complex predictor relationships, such as those in genomics (Cordell, 2009). Neural techniques like pruning (Frankle & Carbin, 2018) and structured sparsity (Wang & Chen, 2021) adjust broadly but fail to target two predictors during training. This reveals a need for a method that refines the selection of two active predictors within the training process. DPST meets this need by adaptively adjusting weights for sparse predictor selection, making it ideal for precision-driven fields like genomics and finance.

## Chapter 3: Methodology

### 3.1 Overview

This section outlines **Dynamic Pairwise Sparse Tuning (DPST)** and a static baseline, simulation frameworks developed to explore how feedforward neural networks (FNNs) can identify exactly two active predictors from a larger set driving a response variable. These frameworks assess whether an FNN can effectively perform sparse predictor selection, prioritizing the most influential pair based on fixed relationships in the simulated data.

### 3.2 Data Generation

Our simulation strategy evaluates the ability of FNNs to select two active predictors from a larger set, using two key components:

- Data simulation under two conditions: (i) independent predictors from univariate normal distributions, and (ii) correlated predictors from multivariate normal distributions.
- A sparse coefficient matrix that defines fixed relationships between predictors and the response in a controlled way.

This setup tests how well neural networks can identify the two active predictors under controlled conditions, supporting DPST's goal of sparse predictor selection.

Let  $P$  represent the number of predictors and  $N$  the number of observations. We construct a regression matrix  $X \in R^{N \times P}$ , where the  $i$ -th row is:

$$X_{i,\cdot} = (X_{i,1}, X_{i,2}, \dots, X_{i,P}).$$

We generate  $X$  under two distinct scenarios.

### 3.2.1 Case 1: Independent Predictors

Each predictor column  $X_{:,j}$  is drawn from an independent univariate normal distribution:

$$X_{i,j} \sim \mathcal{N}(\mu_j, \sigma^2), \quad j = 1, \dots, P, \quad i = 1, \dots, N,$$

where  $\mu_j$  is the mean of the  $j$ -th predictor and  $\sigma^2$  is the common variance. Here, predictors are mutually independent, with no correlation between columns.

### 3.2.2 Case 2: Correlated Predictors

To include dependence, we generate each row of  $X$  from a multivariate normal distribution:

$$X_{i,\cdot} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma),$$

where  $\boldsymbol{\mu} \in R^P$  is the mean vector, and  $\Sigma \in R^{P \times P}$  is a positive semidefinite covariance matrix. In this case, the columns of  $X$  are correlated according to  $\Sigma$ , reflecting realistic predictor relationships.

The regression matrix  $X$  takes the form:

$$X = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,P} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,P} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N,1} & X_{N,2} & \cdots & X_{N,P} \end{bmatrix},$$

where columns are independent (Case 1) or correlated (Case 2).

### 3.3 Sparse Coefficient Matrix

To define the true relationships in our simulation, we create a sparse coefficient matrix  $A \in R^{P \times C}$ , where  $C = \binom{P}{2}$  is the number of unique predictor pairs among the  $P$  predictors. Each column of  $A$  corresponds to one pair, with exactly two nonzero entries indicating the two active predictors

influencing the response  $Y$ . The elements of  $A$  are:

$$A_{i,k} = \begin{cases} a_{i,k}, & \text{for exactly two indices } i \in \{1, \dots, P\}, \text{ for each } k \in \{1, \dots, C\}, \\ 0, & \text{otherwise,} \end{cases}$$

where  $a_{i,k} \sim \mathcal{U}([-1, 1])$  by default, or from another range  $\mathcal{U}([b_{\min}^k, b_{\max}^k])$  to control pair strength variability. This matrix  $A$  encodes the fixed contribution of each predictor pair to the response.

The structure of  $A$ , with columns indexed by pairs  $(j, l)$  for  $1 \leq j < l \leq P$  is:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 & \cdots & 0 & 0 \\ a_{2,1} & 0 & a_{2,3} & 0 & \cdots & 0 & 0 \\ 0 & a_{3,2} & 0 & a_{3,4} & \cdots & 0 & 0 \\ 0 & 0 & a_{4,3} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{P-1,C-1} & a_{P-1,C} \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_{P,C} \end{bmatrix} \in R^{P \times C},$$

where each column has exactly two nonzero entries, representing the pair's joint effect.

We also consider the transpose  $A^\top \in R^{C \times P}$ , where each row corresponds to a pair with two nonzero entries:

$$A^\top = \begin{bmatrix} a_{1,1} & a_{2,1} & 0 & 0 & \cdots & 0 & 0 \\ a_{1,2} & 0 & a_{3,2} & 0 & \cdots & 0 & 0 \\ 0 & a_{2,3} & 0 & a_{4,3} & \cdots & 0 & 0 \\ 0 & 0 & a_{3,4} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{P-1,C} & a_{P,C} \end{bmatrix} \in R^{C \times P}.$$

This sparse design ensures only two predictors per pair contribute to  $Y$ .

The simulation data generated here allows us to test how well FNNs identify these two active predictors. The choice of independent or correlated predictors explores different data structures, while the sparse matrix  $A$  defines the true relationships, keeping them fixed for evaluation.

### 3.4 The Linear Predictors

Given the regression matrix  $X \in R^{N \times P}$  and sparse coefficient matrix  $A \in R^{P \times C}$ , we define the linear predictor matrix  $B \in R^{N \times C}$  as:

$$B = X \cdot A.$$

Each column  $B_{.,k}$ , corresponding to the  $k$ -th predictor pair  $(j, l)$  for  $1 \leq j < l \leq P$ , is a linear combination of that pair's predictors across  $N$  observations.

Let  $\beta_0$  be an intercept, and  $\epsilon \in R^{N \times C}$  a noise matrix with entries  $\epsilon_{i,k} \sim \mathcal{N}(0, \sigma_\epsilon^2)$ , where  $\sigma_\epsilon^2 = 1$  by default, adding variability. The response matrix  $Y \in R^{N \times C}$  is:

$$Y = \beta_0 \mathbf{1}_{N \times C} + B + \epsilon,$$

where  $\mathbf{1}_{N \times C}$  is a matrix of ones. This links each pair's fixed contribution to  $Y$ , enabling evaluation of selection accuracy.

### 3.5 Reshaping and Labeling

This reshaping process prepares the simulated data for neural network modeling by converting the response matrix  $Y$  and regression matrix  $X$  into a format suitable for feedforward neural networks (FNNs). Unlike structured objects like matrices, FNNs require input features and targets as vectors. Here, we reshape  $Y$  and  $X$  into a single input vector per block, with targets as labels identifying the two active predictors, ensuring alignment between inputs and labels is preserved.

### 3.5.1 Block Partitioning

To organize the data, we partition the observations into non-overlapping blocks, which can be thought of as samples for modeling. Let  $Y \in R^{N \times C}$  be the response matrix and  $X \in R^{N \times P}$  the regression matrix, where:

- $N$  is the total number of observations.
- $P$  is the number of predictors.
- $C = \binom{P}{2}$  is the number of unique predictor pairs.

We divide the  $N$  observations into  $M = \frac{N}{m}$  blocks of size  $m$ , ensuring that each block contains  $m$  consecutive observations. This structure allows the neural network to model patterns across groups of observations while preserving the sequence.

### 3.5.2 Response Reshaping

The response matrix  $Y$  is reshaped into a structured matrix  $Y_{\text{block},k} \in R^{M \times m}$  for each target pair index  $k$ . We select one column of  $Y$  for the target predictor pair in each block, matching  $L$ , defined below. This transformation aligns the response values with the block structure of the regression matrix while preserving the sequence of observations:

$$Y_{\text{block},k} = \begin{bmatrix} Y_{1,k} & Y_{2,k} & \cdots & Y_{m,k} \\ Y_{m+1,k} & Y_{m+2,k} & \cdots & Y_{2m,k} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N-m+1,k} & Y_{N-m+2,k} & \cdots & Y_{N,k} \end{bmatrix},$$

where:

- Each row corresponds to a block of  $m$  consecutive response values.
- The columns preserve the sequential ordering within each block.

- $k$  is the index of the target pair for the block, matching  $L$ .

This structure ensures that each block retains continuity, allowing the neural network to process meaningful local dependencies.

### 3.5.3 Predictor Reshaping

The regression matrix  $X$  is reshaped into  $X_{\text{block}} \in R^{M \times (P \times m)}$ , ensuring that predictors are correctly aligned with their respective response values across blocks:

$$X_{\text{block}} \in R^{M \times (P \times m)},$$

where:

- Each row represents a block containing  $m$  consecutive observations.
- The total number of columns is  $P \times m$ , as each of the  $P$  predictors is represented over  $m$  observations within each block.

Mathematically, the reshaped matrix is structured as:

$$X_{\text{block}} = \begin{bmatrix} X_{1,1} & \cdots & X_{1,P} & X_{2,1} & \cdots & X_{2,P} & \cdots & X_{m,1} & \cdots & X_{m,P} \\ X_{m+1,1} & \cdots & X_{m+1,P} & X_{m+2,1} & \cdots & X_{m+2,P} & \cdots & X_{2m,1} & \cdots & X_{2m,P} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ X_{N-m+1,1} & \cdots & X_{N-m+1,P} & X_{N-m+2,1} & \cdots & X_{N-m+2,P} & \cdots & X_{N,1} & \cdots & X_{N,P} \end{bmatrix}.$$

This reshaping ensures that the structured input retains its original dependencies while enabling the neural network to process within-block relationships.

### 3.5.4 Matrix-Label Integration

To create the final input for FNN modeling, we concatenate  $Y_{\text{block},k}$  and  $X_{\text{block}}$  horizontally:

$$Z_{\text{input}} = [Y_{\text{block},k} \ X_{\text{block}}] \in R^{M \times (m+P \times m)}.$$

We also generate a target vector  $L \in R^{M \times 1}$ , where each entry is a string label  $(X_i, X_j)$  identifying the active predictor pair for that block. The final input-target matrix is:

$$Z = [Z_{\text{input}} \ L] \in R^{M \times (m+P \times m+1)},$$

with properties:

1. **Alignment:** Each row of  $Z$  represents a block, ensuring  $Y_{\text{block},k}$ ,  $X_{\text{block}}$ , and  $L$  are correctly matched.
2. **Dimensionality:**  $Z$  has dimensions  $M \times (m + P \times m + 1)$ , where  $M = \frac{N}{m}$  is the number of blocks,  $m$  is the block size,  $P \times m$  is the number of observations made on each of the  $P$  predictors in the block, and 1 is the label column.
3. **Block Consistency:** The observation sequence within each block is preserved, supporting accurate predictor pair identification.

This matrix  $Z$  is ready for FNN training, enabling the network to predict the active predictor pair label from the combined features.

### 3.5.5 Purpose of Reshaping

We reshape the data into blocks, treated as samples, and vectors because FNNs require a single input feature vector and target per sample, not matrices. This structure preserves the relationships between observations within blocks, ensuring the network can accurately identify the two active predictors while maintaining data organization and alignment for effective modeling.

## 3.6 Feedforward Artificial Neural Networks

Feedforward neural networks (FNNs) provide a framework for predictive modeling by processing input features through layered computations to learn complex patterns. We use FNNs to predict active predictor pair labels from the reshaped data, using their simplicity, fully connected layers, and scalability to capture intricate relationships in the simulation data.

### 3.6.1 FNN Architecture

The FNN architecture processes the structured data  $Z \in R^{M \times (m+P \times m+1)}$ , combining reshaped responses and predictors per block. The architecture consists of:

- **Input Layer:** Receives  $Z_{\text{input}} \in R^{M \times (m+P \times m)}$ , where each neuron corresponds to a response or predictor value from a block, preserving the sequence of  $m$  observations across  $P$  predictors.
- **Hidden Layers:** One or more fully connected layers with logistic activation functions capture patterns in the predictor-response relationships. Layer sizes are adjustable to adapt to task complexity.
- **Output Layer:** Predicts the predictor pair label  $(X_i, X_j)$  for each block, aligning with the simulation's pair structure.

Figure 3.1 shows the FNN designed to extract relationships from  $Z$ , supporting accurate prediction of the active predictor pair label.

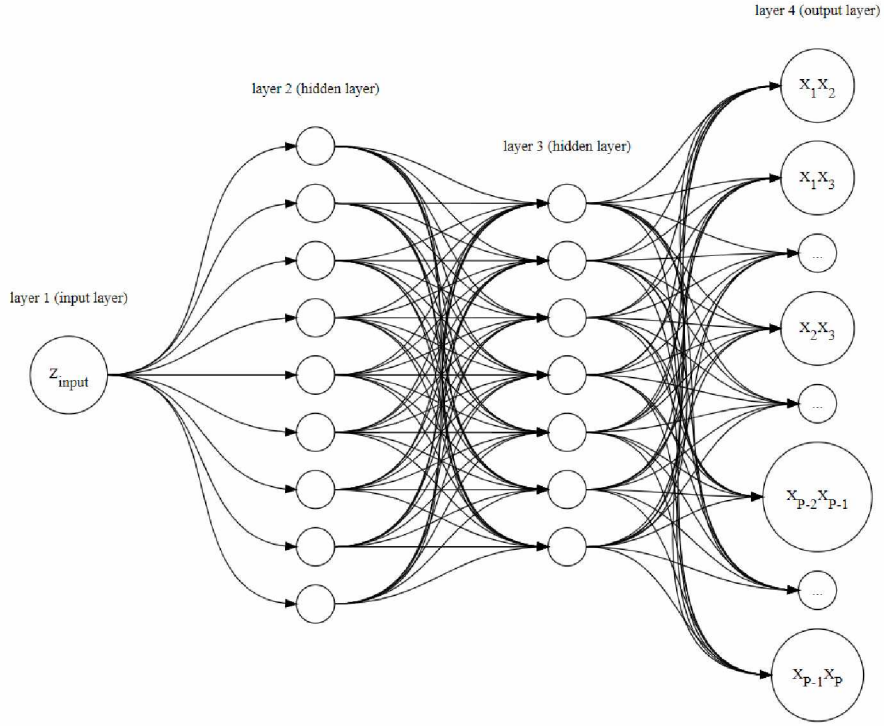


Figure 3.1: Feedforward Neural Network Architecture

### 3.7 Static Baseline

Up to this stage, the static baseline and the DPST share the same data preparation and neural network architecture. Both utilize the regression matrix  $X \in R^{N \times P}$ , sparse coefficient matrix  $A \in R^{P \times C}$ , and reshaped input-target matrix  $Z \in R^{M \times (m + P \times m + 1)}$ , as defined in the preceding subsections. The static baseline process is as follows:

- Uses the FNN architecture to predict predictor pair labels  $(X_i, X_j)$ , training with standard backpropagation to estimate weights without thresholding, smoothing, or pruning.
- Applies the same  $X$ ,  $A$ , and  $Z$  as the DPST, ensuring identical input data and pair structure.
- Trains the FNN once to predict the active predictor pairs, relying on the fixed relationships encoded in  $A$ .

This baseline provides a reference for evaluating predictor pair selection, using the same simulation framework but differing in its weight estimation approach.

### 3.8 Dynamic Pairwise Sparse Tuning (DPST)

Up to the training stage, DPST and the static baseline share the same regression matrix  $X \in R^{N \times P}$ , sparse coefficient matrix  $A \in R^{P \times C}$ , reshaped input-target matrix  $Z \in R^{M \times (m+P \times m+1)}$ , and FNN architecture, as defined previously. To enhance accuracy in identifying the two active predictors, DPST employs a custom method to dynamically update neural network weights during training for sparse predictor selection, unlike the static baseline, which relies on standard backpropagation without further adjustments. This custom method involves three components: Weight-Distribution Thresholding, Smoothed Refinement, and Dual-Mode Pruning, detailed below.

#### 3.8.1 Weight Extraction

At each training epoch  $t$ , we extract the weight matrix  $W^{(t)}$  from the FNN’s first layer. Each element  $W_{j,i}^{(t)}$  represents the influence of input feature  $j$  (from  $Z_{\text{input}}$ ) on neuron  $i$ . These weights drive DPST’s custom updates to refine predictor pair selection.

#### 3.8.2 Weight-Distribution Thresholding

We compute an adaptive threshold  $\tau^{(t)}$  based on the distribution of absolute weights:

$$\tau^{(t)} = \text{quantile}(\{|W_{j,i}^{(t)}|\}, q),$$

where  $q \in \{0.75, 0.80, 0.85\}$  is a quantile parameter, chosen based on preliminary experiments to balance sensitivity. Weights above  $\tau^{(t)}$  are prioritized, while those below are minimized, focusing the FNN on significant predictor pairs during training.

#### 3.8.3 Smoothed Refinement

To ensure stable weight updates, we apply a Smoothed Refinement term  $M^{(t)}$  with two steps:

1. **Compute Raw Update:** Calculate the update direction and size:

$$\Delta W_{j,i}^{(t)} = \begin{cases} +\alpha, & \text{if } |W_{j,i}^{(t)}| \geq \tau^{(t)}, \\ -\alpha, & \text{otherwise,} \end{cases}$$

where  $\alpha \in \{0.01, 0.05, 0.1\}$  is a step size, chosen based on preliminary experiments.

2. **Apply Cumulative Adjustment:** Combine with past updates:

$$M_{j,i}^{(t+1)} = \beta \cdot M_{j,i}^{(t)} + (1 - \beta) \cdot \Delta W_{j,i}^{(t)},$$

then update weights:

$$W_{j,i}^{(t+1)} = W_{j,i}^{(t)} \times (1 + M_{j,i}^{(t+1)}),$$

where  $\beta \in \{0.8, 0.9, 0.95\}$ , chosen based on preliminary experiments, balances stability and adaptation.

This process minimizes fluctuations, aiding convergence to the active predictor pair.

### 3.8.4 Dual-Mode Pruning

To enforce sparsity, we prune weights after refinement. Weights below a threshold  $\varepsilon \in \{0.01, 0.05, 0.1\}$ , chosen based on preliminary experiments, are set to zero:

$$W_{j,i}^{(t+1)} = \begin{cases} 0, & \text{if } |W_{j,i}^{(t+1)}| < \varepsilon, \\ W_{j,i}^{(t+1)}, & \text{otherwise.} \end{cases}$$

This keeps the model focused on the two active predictors, enhancing efficiency.

### 3.8.5 Implementation Summary

DPST operates over training epochs with the following structure: An epoch is a full sweep over all  $M = \frac{N}{m}$  blocks, while an iteration is a single forward and backward pass over one block, followed by DPST’s custom weight updates. For example, with  $N = 1800$  and  $m = 4$ , there are  $M = 450$  iterations per epoch, chosen based on preliminary experiments to balance computational efficiency. The process is as follows:

- **Initialization:** Weights  $W^{(0)}$  are obtained from a preliminary backpropagation training phase of the FNN on the entire dataset (all  $M = \frac{N}{m}$  blocks in  $\mathbf{Z}$ ), providing an informed starting point, with subsequent updates driven by DPST’s custom weight update method integrated into the backpropagation process.
- **Epoch-Based Process:** For each epoch  $t = 1, \dots, 5$ , selected based on preliminary experiments to ensure convergence:
  1. Compute the threshold  $\tau^{(t)}$  using quantile  $q$  at the start of the epoch.
  2. For each block (iteration):
    - (a) Extract weights  $W^{(t)}$  from the FNN’s first layer after standard backpropagation.
    - (b) Calculate  $\Delta W_{j,i}^{(t)}$  using  $\tau^{(t)}$  and apply Smoothed Refinement to update  $M_{j,i}^{(t+1)}$  and  $W_{j,i}^{(t+1)}$ .
    - (c) Apply Dual-Mode Pruning to enforce sparsity.
  3. Complete the epoch after processing all  $M$  blocks, then proceed to epoch  $t + 1$ , recomputing  $\tau^{(t+1)}$ .
  4. The refined weights  $W_{j,i}^{(t+1)}$  and cumulative adjustment  $M_{j,i}^{(t+1)}$  from epoch  $t$  are carried over to epoch  $t + 1$ , enabling progressive refinement of the active predictor pair across the five epochs.

The FNN trains over five epochs using the fixed  $A$  to generate  $Y$  once at initialization. This iterative process, with  $\tau^{(t)}$  computed per epoch and custom updates applied per block, refines the focus on the active predictor pair.

### 3.8.6 Simulation Role of the Sparse Matrix $A$

In our simulation, both the static baseline and **Dynamic Pairwise Sparse Tuning (DPST)** rely on a fixed sparse coefficient matrix  $A$  to define the true relationships between predictors and the response  $Y$ . This matrix remains constant across all experiments, ensuring a consistent ground truth for evaluating predictor selection. The static baseline estimates neural network weights using standard backpropagation without further adjustments, while DPST dynamically updates its weights for sparse predictor selection during training. By maintaining a fixed  $A$ , we isolate the impact of these weight update strategies, allowing us to assess whether DPST’s dynamic approach enhances accuracy in identifying the two active predictors compared to the static baseline’s standard backpropagation method.

## 3.9 Performance Evaluation

This section outlines the evaluation criteria for comparing **Dynamic Pairwise Sparse Tuning (DPST)** with a static approach, assessing predictive accuracy and computational feasibility. DPST iteratively refines neural network weights, while the static approach estimates weights using standard backpropagation without further adjustments, both applied to the fixed coefficient matrix  $A$ .

### 3.9.1 Confusion Matrix Metrics

We evaluate classification performance using a confusion matrix aggregated across all predictor pairs:

$$\begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix},$$

where:

- TP (True Positives): Number of active predictor pairs correctly identified, matching  $L$ .
- FP (False Positives): Inactive pairs misclassified as active.
- FN (False Negatives): Active pairs missed by the model.

- TN (True Negatives): Inactive pairs correctly excluded.

This aggregation assesses each method's ability to identify active predictors, providing a comprehensive view of classification performance across the entire predictor set.

### 3.9.2 Accuracy

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

measures the overall proportion of correct pair classifications across all predictions. It serves as a baseline metric to evaluate how well each method distinguishes active from inactive predictor pairs under varying simulation conditions.

### 3.9.3 Sensitivity (Recall)

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

quantifies the model's ability to detect all truly active predictor pairs. It reflects the proportion of actual active pairs (those with non-zero coefficients in  $A$ ) correctly identified, ensuring the model captures the full set of influential relationships.

### 3.9.4 Specificity

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

assesses the model's ability to correctly exclude inactive predictor pairs. It measures the proportion of pairs with zero coefficients in  $A$  that are accurately classified as non-influential, reducing false positives in sparse selection.

### 3.9.5 Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

evaluates the reliability of the model's predictions of active predictor pairs. It indicates the proportion of pairs classified as active that truly have non-zero coefficients in  $A$ , ensuring selected pairs are genuinely influential.

### 3.9.6 Negative Predictive Value (NPV)

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}}$$

assesses the reliability of the model's classifications of inactive predictor pairs. It measures the proportion of pairs predicted as non-influential that are correctly identified as having zero coefficients in  $A$ , complementing Specificity by focusing on the correctness of negative predictions.

### 3.9.7 F1-Score

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

balances Precision and Sensitivity to provide a single metric for model performance, particularly valuable when active predictor pairs are sparse. It combines the ability to detect true active pairs with the reliability of those detections.

### 3.9.8 Cohen's Kappa

$$\kappa = \frac{\text{Accuracy} - P_e}{1 - P_e},$$

where

$$P_e = \left( \frac{TP + FN}{\text{Total pairs}} \right) \cdot \left( \frac{TP + FP}{\text{Total pairs}} \right) + \left( \frac{FP + TN}{\text{Total pairs}} \right) \cdot \left( \frac{FN + TN}{\text{Total pairs}} \right),$$

adjusts Accuracy to account for agreement occurring by chance. It provides a robust measure of classification performance by comparing observed accuracy to expected random agreement, enhancing evaluation reliability.

### 3.10 Computational Metrics

#### 3.10.1 Training Time

Training time (seconds), measured via R’s `Sys.time()`, quantifies the duration required for model training as sample size ( $N$ ) and predictor count ( $P$ ) increase. For the static baseline, it reflects a training phase using standard backpropagation to estimate weights. For DPST, it sums across multiple epochs (e.g., 5), where each epoch is a full sweep over the data. During each epoch, the FNN processes all blocks in  $Z$ , adjusting weights using backpropagation and DPST’s custom update mechanism (Weight-Distribution Thresholding, Smoothed Refinement, and Dual-Mode Pruning). These iterations refine the weights to improve prediction of active predictor pairs in  $L$ , using the fixed input  $Y_{\text{block},k}$  and  $X_{\text{block}}$  from  $Z$ , while keeping  $A$  constant.

#### 3.10.2 Memory Usage

Memory usage (GB), estimated as peak consumption via `object.size()`, captures the computational resource footprint during training as  $N$  and  $P$  increase. This metric evaluates the practical viability of each method, comparing DPST’s pruning mechanism across epochs to the static baseline.

## Chapter 4: Results

### 4.1 Results

### 4.2 Overview of Simulation Designs and Data Conditions

This section evaluates **Dynamic Pairwise Sparse Tuning (DPST)** against a static baseline in identifying two active predictors, using the simulation framework and metrics from prior sections. DPST refines neural network weights across five epochs, each epoch being a full sweep over the data, while the static baseline estimates weights using standard backpropagation without further adjustments, both applied to a fixed coefficient matrix  $A$ . For each condition, we conducted simulation runs, one per block ( $M = \frac{N}{m}$ ), iterating over all blocks in  $Z$  to train the FNN. These iterations, repeated per epoch, refine DPST’s ability to detect active predictors.

#### 4.2.1 Predictor Correlation Structures

- Independent predictors (Models 1 and 2)
- Correlated predictors (Models 3 and 4)

#### 4.2.2 Coefficient Magnitude Ranges of Design Matrix $A$

- Standard intervals (Models 1 and 3): Coefficients drawn from  $\text{Uniform}[-1, 1]$
- Custom broader intervals (Models 2 and 4): Coefficients drawn from wider ranges, enhancing the signal strength of active predictors to test detection accuracy.

We varied predictor dimensions ( $P = 3, 4, 5$ ), yielding  $C = \binom{P}{2}$  pair combinations, and sample sizes ( $N = 1800, 3600$ ) to compare performance comprehensively.

### 4.2.3 Results Interpretations

### 4.2.4 Predictive and Computational Results at $P = 3$ , $C = \binom{3}{2} = 3$ Pair Combinations

#### *Predictive Performance*

Table 4.1 shows DPST outperforming static at  $n = 1800$  across all models, peaking in Model 2 with Accuracy 0.9279 [0.8629, 0.9684], Avg. F1 0.9278, and Kappa 0.8919 versus static’s 0.7387 [0.6468, 0.8175], Avg. F1 0.7441, and Kappa 0.6081. DPST’s tight confidence intervals reflect high precision, its Avg. F1 indicates a strong balance between detecting active pairs and avoiding false positives, and its elevated Kappa suggests substantial agreement beyond chance, far exceeding static’s moderate agreement. At  $n = 3600$ , static leads in Model 2 with Accuracy 0.9600 [0.9254, 0.9815], Avg. F1 0.9599, and Kappa 0.9400 which is near perfect agreement while DPST drops to 0.6844 [0.6194, 0.7446], Avg. F1 0.6734, and Kappa 0.5267. This suggests DPST’s iterative updates, optimized for smaller datasets ( $n = 1800$ ), may specialize to specific patterns, reducing generalization at larger sample sizes ( $n = 3600$ ), where static’s adapts more effectively.

Table 4.1: Performance comparison between DPST and static baseline across sample sizes at  $P = 3$

Sample Size ( $n$ )	Model	Condition	Accuracy	95% CI	Avg. F1	Kappa
$n = 1800$	1	Static	0.8378	[0.7559, 0.9010]	0.8374	0.7568
		DPST	0.9099	[0.8406, 0.9559]	0.9098	0.8649
	2	Static	0.7387	[0.6468, 0.8175]	0.7441	0.6081
		DPST	0.9279	[0.8629, 0.9684]	0.9278	0.8919
	3	Static	0.8018	[0.7154, 0.8714]	0.8046	0.7027
		DPST	0.9279	[0.8629, 0.9684]	0.9278	0.8919
	4	Static	0.8108	[0.7255, 0.8789]	0.8149	0.7162
		DPST	0.9189	[0.8517, 0.9623]	0.9184	0.8784
$n = 3600$	1	Static	0.8889	[0.8404, 0.9268]	0.8898	0.8333
		DPST	0.7778	[0.7177, 0.8303]	0.7769	0.6667
	2	Static	0.9600	[0.9254, 0.9815]	0.9599	0.9400
		DPST	0.6844	[0.6194, 0.7446]	0.6734	0.5267
	3	Static	0.9156	[0.8713, 0.9484]	0.9164	0.8733
		DPST	0.6222	[0.5554, 0.6858]	0.6178	0.4333
	4	Static	0.9289	[0.8871, 0.9588]	0.9289	0.8933
		DPST	0.6933	[0.6286, 0.7529]	0.6918	0.5400

### Visual Insights

Figure 4.1 shows DPST's performance at  $n = 1800$ , consistently achieving higher Accuracy and Avg. F1 across all models compared to static, reflecting its effectiveness with moderate data sizes. At  $n = 3600$ , static overtakes DPST, particularly in Model 2, with superior scores and tighter CIs, while DPST's decline indicates that its training, tuned for smaller datasets, may capture specific patterns less effectively as sample size grows.

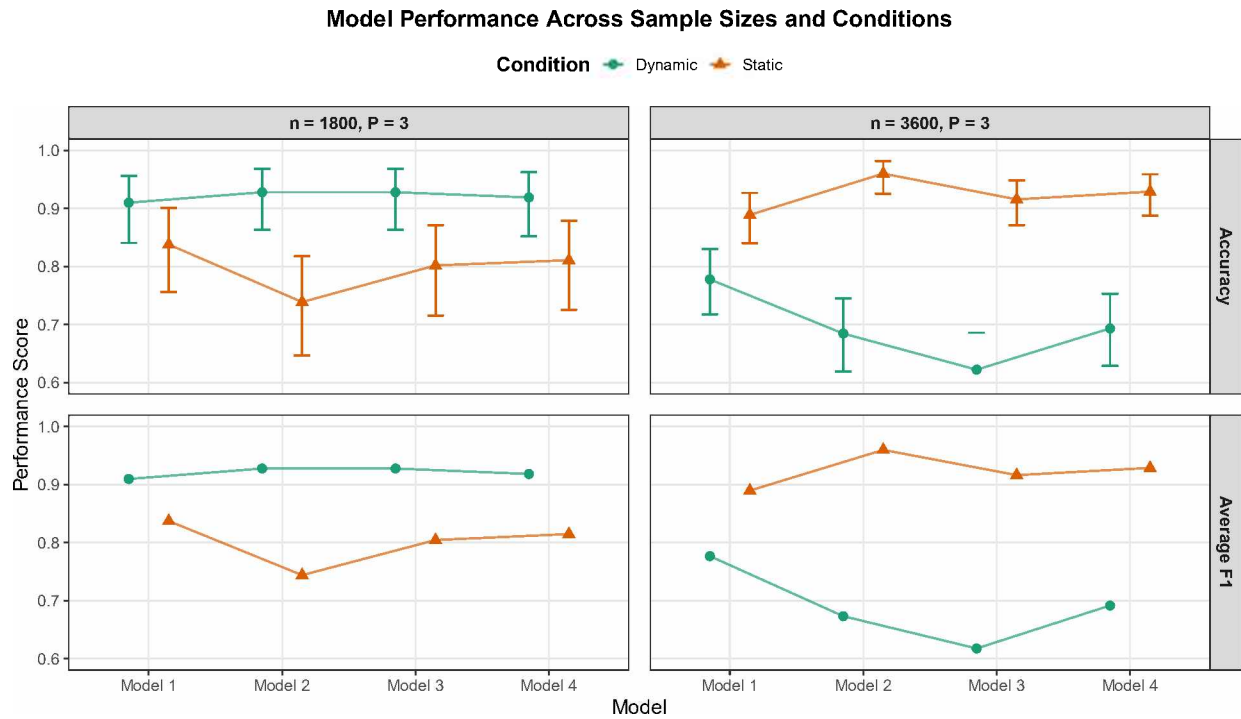


Figure 4.1: Performance comparison at  $P = 3$

### Computational Metrics

Table 4.2 details DPST’s five-epoch training at  $n = 1800$ , averaging 1.1231 s (0.2246 s/epoch) across models, 5.8 times static’s 0.1920 s, with ranges of 0.9579–1.3190 s (DPST) vs. 0.1345–0.2571 s (static), and memory remains low and equal ( 0.000455 GB). At  $n = 3600$ , DPST averages 4.6687 s (0.9337 s/epoch), 13.3 times static’s 0.3497 s, ranging 2.8359–7.9702 s (DPST) vs. 0.2078–0.5250 s (static), with memory at 0.000930 GB. DPST’s iterative cost supports its accuracy advantage at  $n = 1800$ , though static’s efficiency becomes more pronounced at larger  $n$ .

Table 4.2: Computational comparison between DPST and static baseline across sample sizes at  $P = 3$

Sample Size ( $n$ )	Model	Condition	Training Time (s)	Memory Usage (GB)
$n = 1800$	1	Static	0.2571	0.0004550964
		DPST	0.9579	0.0004553050
	2	Static	0.2378	0.0004550964
		DPST	1.0707	0.0004553050
	3	Static	0.1387	0.0004550964
		DPST	1.3190	0.0004553050
	4	Static	0.1345	0.0004550964
		DPST	1.1449	0.0004553050
$n = 3600$	1	Static	0.5250	0.0009296983
		DPST	4.5885	0.0009299070
	2	Static	0.2214	0.0009296983
		DPST	3.2802	0.0009299070
	3	Static	0.4446	0.0009296983
		DPST	7.9702	0.0009299070
	4	Static	0.2078	0.0009296983
		DPST	2.8359	0.0009299070

#### 4.2.5 Predictive and Computational Results at $P = 4$ , $C = \binom{4}{2} = 6$ Pair Combinations

##### *Predictive Performance*

Table 4.3 shows DPST’s consistent advantage over static at  $n = 1800$  across all models, peaking in Model 1 with Accuracy 0.8704 [0.7921, 0.9273], Avg. F1 0.8704, and Kappa 0.8444, surpassing static’s 0.7778 [0.6876, 0.8521], Avg. F1 0.7766, and Kappa 0.7333. DPST’s high Avg. F1 reflects a robust balance of precision and recall, while its Kappa indicates strong agreement beyond chance, contrasting with static’s wider CIs and moderate agreement. At  $n = 3600$ , DPST maintains its lead, e.g., Model 1 Accuracy 0.7207 [0.6568, 0.7787], Avg. F1 0.7205, and Kappa 0.6649 versus static’s 0.6396 [0.5727, 0.7028], Avg. F1 0.6381, and Kappa 0.5676, showing sustained reliability unlike  $P = 3$  where static surged ahead.

Table 4.3: Performance comparison between DPST and static baseline across sample sizes at  $P = 4$

Sample Size ( $n$ )	Model	Condition	Accuracy	95% CI	Avg. F1	Kappa
$n = 1800$	1	Static	0.7778	[0.6876, 0.8521]	0.7766	0.7333
		DPST	0.8704	[0.7921, 0.9273]	0.8704	0.8444
	2	Static	0.6667	[0.5695, 0.7545]	0.6533	0.6000
		DPST	0.8148	[0.7286, 0.8831]	0.8139	0.7778
	3	Static	0.7593	[0.6675, 0.8363]	0.7631	0.7111
		DPST	0.8519	[0.7706, 0.9129]	0.8531	0.8222
	4	Static	0.7315	[0.6376, 0.8122]	0.7299	0.6778
		DPST	0.8056	[0.7183, 0.8754]	0.8030	0.7667
$n = 3600$	1	Static	0.6396	[0.5727, 0.7028]	0.6381	0.5676
		DPST	0.7207	[0.6568, 0.7787]	0.7205	0.6649
	2	Static	0.6441	[0.5773, 0.7071]	0.6320	0.5730
		DPST	0.7252	[0.6615, 0.7828]	0.7266	0.6703
	3	Static	0.6261	[0.5589, 0.6900]	0.6264	0.5514
		DPST	0.7117	[0.6473, 0.7704]	0.7104	0.6541
	4	Static	0.6351	[0.5681, 0.6985]	0.6360	0.5622
		DPST	0.7072	[0.6426, 0.7662]	0.7084	0.6486

### Visual Insights

Figure 4.2 shows DPST's consistent superiority at  $n = 1800$ , with higher Accuracy and Avg. F1 across all models, demonstrating its effectiveness in moderate-sized datasets compared to static. At  $n = 3600$ , DPST retains its lead, though with a slight decline from  $n = 1800$ , still outperforming static, which contrasts with  $P = 3$  where static dominated at larger  $n$ .

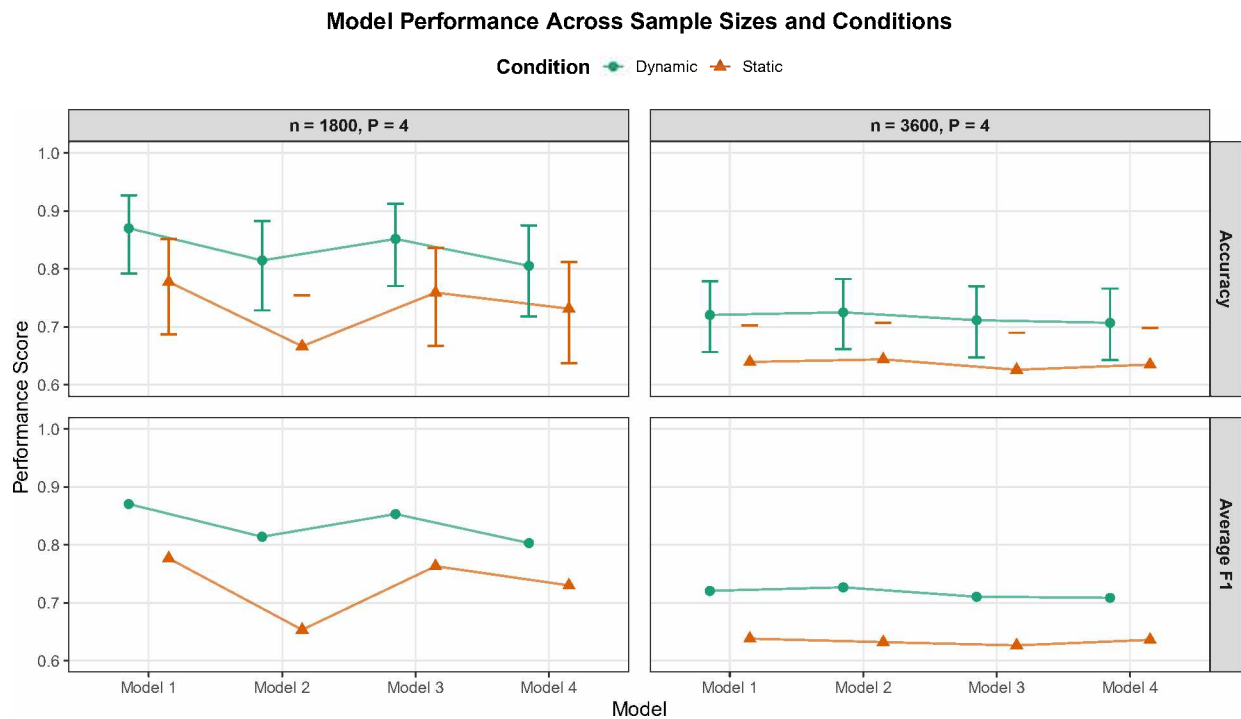


Figure 4.2: Performance comparison at  $P = 4$

### Computational Metrics

Table 4.4 reveals DPST’s five-epoch training at  $n = 1800$  averages 1.3513 s (0.2703 s/epoch), 4.7 times static’s 0.2905 s, ranging 1.1869–1.6553 s (DPST) vs. 0.2526–0.3258 s (static), with memory steady at 0.001220 GB. At  $n = 3600$ , DPST averages 16.0356 s (3.2071 s/epoch), 5.6 times static’s 2.8395 s, ranging 12.6078–21.6423 s (DPST) vs. 2.5699–3.5102 s (static), memory 0.002126 GB. DPST’s iterative cost underpins its accuracy gains, while static’s lower runtime highlights efficiency at larger  $n$ .

Table 4.4: Computational comparison between DPST and static baseline across sample sizes at  $P = 4$

Sample Size ( $n$ )	Model	Condition	Training Time (s)	Memory Usage (GB)
$n = 1800$	1	Static	0.3258	0.001219690
		DPST	1.2808	0.001220159
	2	Static	0.2526	0.001219690
		DPST	1.1869	0.001220159
	3	Static	0.2901	0.001219690
		DPST	1.6553	0.001220159
	4	Static	0.2937	0.001219690
		DPST	1.2822	0.001220159
$n = 3600$	1	Static	2.7015	0.002125919
		DPST	13.6387	0.002126388
	2	Static	2.5699	0.002125919
		DPST	12.6078	0.002126388
	3	Static	3.5102	0.002125919
		DPST	16.2537	0.002126388
	4	Static	3.3862	0.002125919
		DPST	21.6423	0.002126388

#### 4.2.6 Predictive and Computational Results at $P = 5$ , $C = \binom{5}{2} = 10$ Pair Combinations

##### *Predictive Performance*

Table 4.5 highlights DPST’s lead over static at  $n = 1800$ , peaking in Model 3 with Accuracy 0.6273 [0.5299, 0.7176], Avg. F1 0.6111, and Kappa 0.5859, compared to static’s 0.5455 [0.4477, 0.6407], Avg. F1 0.5346, and Kappa 0.4949. DPST’s Avg. F1 reflects a balanced ability to identify active pairs accurately, and its Kappa indicates moderate to substantial agreement beyond chance, outpacing static’s fair agreement. At  $n = 3600$ , DPST sustains its edge, e.g., Model 2 Accuracy 0.7318 [0.6681, 0.7891], Avg. F1 0.7323, and Kappa 0.7020 versus static’s 0.6091 [0.5412, 0.6740], Avg. F1 0.6015, and Kappa 0.5657, demonstrating resilience at higher complexity compared to  $P = 3$ .

Table 4.5: Performance comparison between DPST and static baseline across sample sizes at  $P = 5$

Sample Size ( $n$ )	Model	Condition	Accuracy	95% CI	Avg. F1	Kappa
$n = 1800$	1	Static	0.5091	[0.4120, 0.6057]	0.5001	0.4545
		DPST	0.6182	[0.5207, 0.7092]	0.6159	0.5758
	2	Static	0.4636	[0.3680, 0.5612]	0.4528	0.4040
		DPST	0.6158	[0.5114, 0.7007]	0.6079	0.5657
	3	Static	0.5455	[0.4477, 0.6407]	0.5346	0.4949
		DPST	0.6273	[0.5299, 0.7176]	0.6111	0.5859
	4	Static	0.5545	[0.4567, 0.6493]	0.5450	0.5051
		DPST	0.6000	[0.5022, 0.6922]	0.5979	0.5556
$n = 3600$	1	Static	0.5682	[0.4999, 0.6346]	0.5617	0.5202
		DPST	0.6182	[0.5505, 0.6827]	0.6168	0.5758
	2	Static	0.6091	[0.5412, 0.6740]	0.6015	0.5657
		DPST	0.7318	[0.6681, 0.7891]	0.7323	0.7020
	3	Static	0.5909	[0.5228, 0.6565]	0.5878	0.5455
		DPST	0.6136	[0.5458, 0.6783]	0.6156	0.5707
	4	Static	0.5545	[0.4567, 0.6493]	0.5450	0.5051
		DPST	0.7136	[0.6490, 0.7724]	0.7090	0.6818

### Visual Insights

Figure 4.3 confirms DPST's sustained superiority at both  $n = 1800$  and  $n = 3600$  for  $P = 5$ , with higher Accuracy and Avg. F1 across models, peaking in Model 2 at  $n = 3600$ . Static lags behind, underscoring DPST's capability to handle increased complexity effectively.

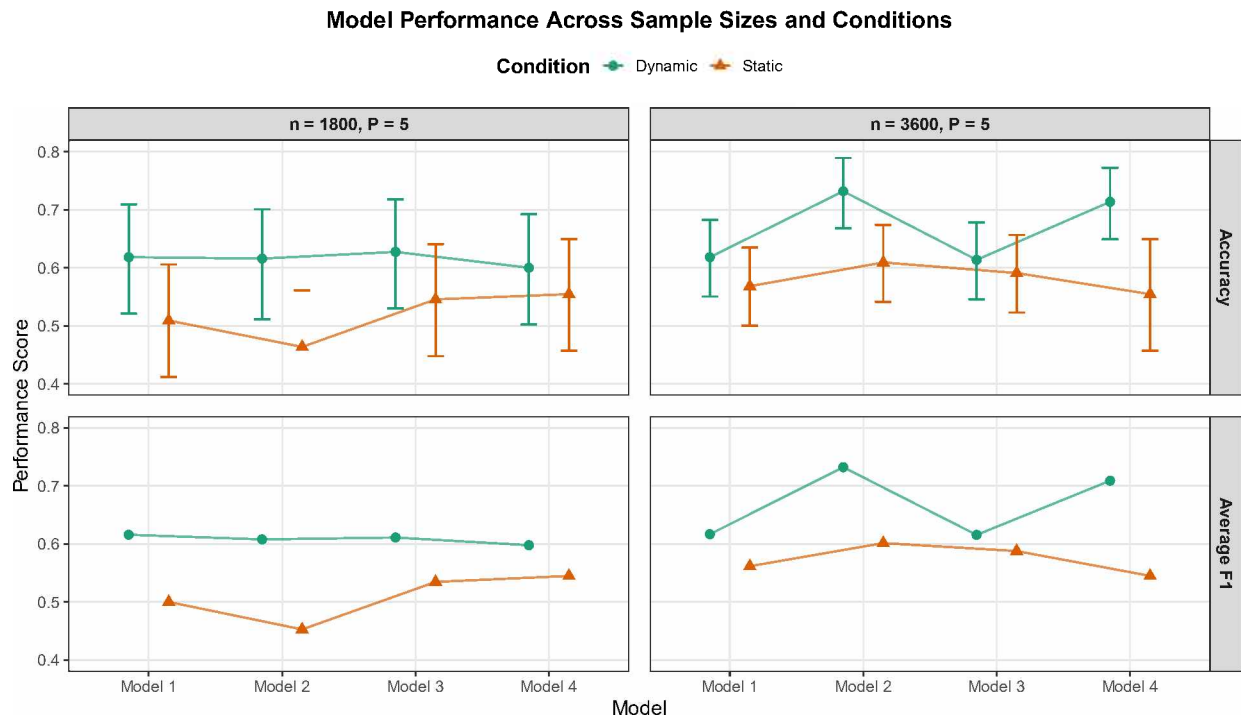


Figure 4.3: Performance comparison at  $P = 5$

### Computational Metrics

Table 4.6 outlines DPST’s five-epoch training at  $n = 1800$ , averaging 5.0052 s (1.0010 s/epoch), 4.4 times static’s 1.1402 s, ranging 4.4034–5.4399 s (DPST) vs. 1.0418–1.2364 s (static), with memory at 0.002489 GB. At  $n = 3600$ , DPST averages 20.6635 s (4.1327 s/epoch), 5.8 times static’s 3.5700 s, ranging 19.8553–22.1353 s (DPST) vs. 1.3799–4.8482 s (static), memory 0.004435 GB. DPST’s higher runtime reflects its iterative refinement, underpinning its accuracy gains, while static offers efficiency at larger scales.

Table 4.6: Computational comparison between DPST and static baseline across sample sizes at  $P = 5$

Sample Size ( $n$ )	Model	Condition	Training Time (s)	Memory Usage (GB)
$n = 1800$	1	Static	1.0484	0.002488568
		DPST	4.7470	0.002489038
	2	Static	1.0418	0.002488568
		DPST	4.4034	0.002489038
	3	Static	1.2344	0.002488568
		DPST	5.4304	0.002489038
	4	Static	1.2364	0.002488568
		DPST	5.4399	0.002489038
$n = 3600$	1	Static	3.2989	0.004434064
		DPST	19.8553	0.004434533
	2	Static	4.3527	0.004434064
		DPST	20.1379	0.004434533
	3	Static	4.8482	0.004434064
		DPST	22.1353	0.004434533
	4	Static	1.3799	0.004434064
		DPST	20.5257	0.004434533

### 4.2.7 Discussion

This study compared DPST and a static baseline across  $P = 3, 4, 5$  and  $n = 1800, 3600$ , addressing optimal conditions, scalability, and trade-offs. Simulations varied predictor correlations and coefficient ranges, revealing key insights. At  $n = 1800$ , DPST consistently outperformed static (e.g.,  $P = 3$ : 0.928 vs. 0.739;  $P = 5$ : 0.627 vs. 0.546), with tight CIs, high Avg. F1 reflecting balanced detection, and strong Kappa showing reliable agreement, excelling under custom ranges. At  $n = 3600$ , static led at  $P = 3$  (0.960 vs. 0.684), suggesting DPST overfitting, but DPST prevailed at  $P = 4$  (0.721 vs. 0.640) and  $P = 5$  (0.732 vs. 0.609), indicating scalability with complexity. DPST's five-epoch iterations raised runtimes (e.g.,  $P = 5$ : 20.664 s vs. 3.570 s at  $n = 3600$ ), 4.4–5.8 times static's, with memory equal (0.0044 GB), supporting accuracy at a computational cost.

Future work should test real-world data, explore  $P > 5$ , and optimize epochs to balance runtime and accuracy.

## Chapter 5: Conclusions

This study presents **Dynamic Pairwise Sparse Tuning (DPST)**, a novel method that refines neural network weights across five epochs, each a full training pass, to optimize the identification of two active predictors. Within each epoch, internal iterations drive convergence and enhance performance via hyperparameter tuning. Simulations with predictor dimensions ( $P = 3, 4, 5$ ) and sample sizes ( $n = 1800, 3600$ ) show DPST outperforming a static baseline, especially in moderate sized datasets with higher predictor counts, achieving, for instance, an accuracy of 0.732 versus 0.609 at  $P = 5$  and  $n = 3600$ , indicating a good detection of key predictor pairs.

However, DPST's iterative nature elevates computational costs, with training times longer than the static baseline (e.g., 20.664 s vs. 3.570 s at  $P = 5$ ,  $n = 3600$ ). This trade-off favors precision critical fields like genomics, where accurate predictor selection trumps computational concerns, while static methods excel in larger datasets with fewer predictors (e.g., 0.960 vs. 0.684 at  $P = 3$ ,  $n = 3600$ ), suiting time sensitive tasks like financial modeling.

Future research should test DPST on real world data, streamline its epoch structure to reduce runtime, and explore its potential with larger predictor sets ( $P > 5$ ).

## Chapter 6: Reference

### 6.1 References

- Bien, J., Taylor, J. & Tibshirani, R. (2013) A lasso for hierarchical interactions. *The Annals of Statistics* **41**, 1111–1141.
- Cordell, H.J. (2009) Detecting gene-gene interactions that underlie human diseases. *Nature Reviews Genetics* **10**, 392–404.
- Frankle, J. & Carbin, M. (2018) The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* .
- Fritsch, S. & Guenther, F. (2024) *neuralnet: Training of neural networks*. Comprehensive R Archive Network (CRAN), version 1.44.2, <https://CRAN.R-project.org/package=neuralnet>.
- Han, S., Mao, H. & Dally, W.J. (2015) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* .
- Kuhn, M. (2024) *caret: Classification and regression training*. Comprehensive R Archive Network (CRAN), version 6.0-94, <https://CRAN.R-project.org/package=caret>.
- Rudin, C. (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* **1**, 206–215.
- Scardapane, S., Comminiello, D., Hussain, A. & Uncini, A. (2017) Group sparse regularization for deep neural networks. *Neurocomputing* **241**, 81–89.
- Team, P. (2024a) *RStudio: Integrated development environment for R*. Posit Software, PBC, Boston, MA, <https://www.posit.co/>.
- Team, R.C. (2024b) *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, <https://www.R-project.org/>.

- Tibshirani, R. (1996) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* **58**, 267–288.
- Tsai, C.F. & Hsu, Y.F. (2010) Feature selection in financial distress prediction. *Expert Systems with Applications* **37**, 7146–7153.
- Wang, J. & Chen, L. (2021) Structured sparsity in deep neural networks for efficient inference. *IEEE Transactions on Neural Networks and Learning Systems* **32**, 1234–1245.
- Zhou, T. & Zhang, Q. (2022) Graph neural networks for relational data modeling. *Journal of Machine Learning Research* **23**, 1–25.
- Zou, H. & Hastie, T. (2005) Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **67**, 301–320.