Local Linear Approximation Algorithm for Neural Network

Anonymous Author(s)

Affiliation
Address
e-mail

Abstract

This paper is concerned with estimation of weights and biases in feed forward neural network (FNN). We propose using local linear approximation (LLA) for the activation function, and develop a LLA algorithm to estimate the weights and biases of one hidden layer FNN by iteratively linear regression. We further propose the layerwise optimized adaptive neural network (LOAN), in which we use the LLA to estimate the weights and biases in the LOAN layer by layer adaptively. We compare the performance of the LOAN with the commonly-used procedures in deep learning via analyses of four benchmark data sets. The numerical comparison implies that the proposed LOAN may outperform the existing procedures.

1 Introduction

Although deep learning has many successful applications, the estimation of the weights and biases are still challenging in the construction of deep neural network. This is partly because there is no algorithm to guarantee its resulting solution to be the optimizer of the objective function being optimized. Many heuristics and trial-and-errors methods are applied to get successfully trained network. This work aims to tackle this problem from a different point of view and develop a reliable training algorithm. Consider a $L$-hidden layer FNN:

$$h(l) = \sigma_l(W_l h(l-1) + b_l)$$  \hspace{1cm} (1)

for $l = 1, \cdots, L$ with $h(0) = x$, the $p$-dimensional input vector, where $\sigma_l(\cdot)$’s are activation functions, applied to each component of inputs in the previous layer. $W_l$ and $b_l$ are the weight matrix and the bias vector in the $l$-th hidden layer. Consider a linear regression model in the output layer

$$y = \beta_0 + \beta^T h(L) + \varepsilon.$$  \hspace{1cm} (2)

Denote $f(x; \theta) = \beta_0 + \beta^T h(L)$, where $\theta = \{\beta_0, \beta, W_1, \cdots, W_L, b_1, \cdots, b_L\}$. Suppose that we have a set of sample $\{x_i, y_i\}, i = 1, \cdots, n$. In the literature, the estimation of $\theta$ is to minimize the nonlinear least squares (LS) function

$$\ell(\theta) = \sum_{i=1}^{n} \{y_i - f(x_i, \theta)\}^2,$$

which is highly nonlinear in $\theta$, a high dimensional vector, and hence parameter estimation is challenging for deep neural network. Based on the chain rule, the gradient of $\ell(\theta)$ has an expressive form, while the Hessian matrix $\ell''(\theta)$ of $\ell(\theta)$ is hard to compute, and computational cost for evaluating the inverse of $\ell''(\theta)$ is too expensive to be afforded since the dimension of $\theta$ can be huge. As a result, the back propagation with gradient decent or its variations became the most popular parameter estimation method for deep neural network [6, 7, 8, 9]. Also see more recent developments.

We first observe that we could estimate the weights \( W_L \) and biases \( b_L \) without difficulty if we knew \( h^{(L-1)} \). Thus, instead of seeking the optimal weights and biases in all layers simultaneously, we propose to assign the weights and biases layerwise by starting with estimating the weights \( W_1 \) and \( b_1 \). To this end, we propose to locally approximate the activation function in FNN by a linear function, and develop a LLA algorithm to estimate \( W_i \)'s and \( b_i \)'s layer by layer based on \( h^{(i-1)} \) with \( h^{(0)} = x \), the input predictor vector. We refer the neural network constructed by this strategy as Layerwise Optimized Adaptive neural Network (LOAN for short) by the nature of its construction process. The LLA algorithm is distinguished from existing gradient descent algorithms in that it utilizes the Hessian matrix \( \ell''(\theta) \) in the same spirit of Fisher scoring algorithm for nonlinear regression models with normal error. By empirical analyses of four benchmark data sets in the literature of deep learning, we compare the performance of the LOAN with the commonly-used deep learning procedures including multi-layer perceptron (MLP) [15], AdaBoost [4], gradient boosting algorithm (GBM) [5], random forest (RF) [2], and XGBoost [3]. Our numerical comparison implies that the one-hidden layer LOAN may outperform these existing procedures in terms of prediction accuracy. Compared with the deep network, the one-hidden layer LOAN enjoys model interpretability and model parsimony. The multiple-hidden layer LOAN can be used to further improve the one-hidden layer LOAN in terms of prediction accuracy.

The rest of this paper is organized as follows. In section 2, we develop the LLA algorithm and an algorithm for construction of the LOAN. Section 3 presents numerical comparisons. Conclusion and discussion are given in section 4.

## 2 New method for constructing neural network

Let us start with one-hidden layer neural network in order to get insights into the LLA algorithm.

### 2.1 One-hidden layer neural network

Note that \( h^{(0)} = x \). Then one hidden-layer FNN with \( J_1 \) nodes can be expressed as follows

\[
h^{(1)} = \sigma(W h^{(0)} + b) = \sigma(w_j^T x + b_j), \quad (j = 1, \cdots, J_1)
\]

where \( W \) is the weight matrix and \( b \) is the bias vector. The activation function is \( \sigma(z) \), which is the ReLU activation function. To estimate the weights and biases, we minimize a nonlinear LS function

\[
\ell_1(\theta) = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{J_1} \beta_j \sigma(w_j^T x_i + b_j))^2.
\]  

(3)

We next propose an algorithm to approximate (3). Given \( w_j^{(c)} \) and \( b_j^{(c)} \) in the current step, we propose to approximate \( \sigma(x^T w_j + b_j) \) by a linear function based on the first-order Taylor expansion of \( \sigma(z) \):

\[
\sigma(x^T w_j + b_j) \approx \sigma(x^T w_j^{(c)} + b_j^{(c)}) + ((x^T w_j + b_j) - (x^T w_j^{(c)} + b_j^{(c)})) I(x^T w_j^{(c)} + b_j^{(c)} > 0)
\]

(4)

for \( j = 1, \cdots, J_1 \). We refer [4] to as local linear approximation (LLA). Thus,

\[
\beta_j \sigma(x^T w_j + b_j) \approx \beta_j \sigma(x^T w_j^{(c)} + b_j^{(c)}) + \gamma_j I(x^T w_j^{(c)} + b_j^{(c)} > 0) + \eta_j^T x I(x^T w_j^{(c)} + b_j^{(c)} > 0)
\]

(5)

where \( \gamma_j = \beta_j (b_j - b_j^{(c)}) \) and \( \eta_j = \beta_j (w_j - w_j^{(c)}) \). Define \( z_{1ij} = \sigma(x_i^T w_j^{(c)} + b_j^{(c)}) \), \( z_{2ij} = I(x_i^T w_j^{(c)} + b_j^{(c)} > 0) \), and \( z_{3ij} = x_i I(x_i^T w_j^{(c)} + b_j^{(c)} > 0) \), \( j = 1, \cdots, J_1 \). Further define

\[
z_{1i} = [z_{11i}, \cdots, z_{1li}], \quad z_{2i} = [z_{21i}, \cdots, z_{2li}], \quad z_{3i} = [z_{31i}, \cdots, z_{3li}], \quad \text{and} \quad z_i = [z_{1i}, z_{2i}, z_{3i}]^T,
\]

which is a \( J_1 (p + 2) \)-dimensional vector. With the aid of approximation (5), the objective function in (3) is approximated by

\[
\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{J_1} (\beta_j z_{1ij} + \gamma_j z_{2ij} + \eta_j^T z_{3ij}))^2,
\]

(6)

which is the LS function of linear regression with the response \( y_i \) and predictors \( z_i \). Denote the resulting LS estimate of \( \beta_j, \gamma_j \) and \( \eta_j \) by \( \hat{\beta}_j, \hat{\gamma}_j \) and \( \hat{\eta}_j \), respectively. By the definition of \( \gamma_j \) and \( \eta_j \),

\[
\hat{\gamma}_j = \sum_{i=1}^{n} (\beta_j z_{1ij} + \gamma_j z_{2ij} + \eta_j^T z_{3ij}), \quad \hat{\eta}_j = \sum_{i=1}^{n} (\beta_j z_{1ij} + \gamma_j z_{2ij} + \eta_j^T z_{3ij}) - \beta_j (z_{1i} \otimes z_{2i} - \gamma_j z_{3i})
\]
we can update $b_j$ and $w_j$ as follows
\[ b_j^{(c+1)} = b_j^{(c)} + \gamma_j / \hat{\beta}_j, \quad \text{and} \quad w_j^{(c+1)} = w_j^{(c)} + \hat{\eta}_j / \hat{\beta}_j. \] (7)

If $|\hat{\beta}_j|$ is very close to zero, one may simply set $b_j^{(c+1)} = b_j^{(c)}$ and $w_j^{(c+1)} = w_j^{(c)}$. Thus, we may estimate $W$ and $b$ by iteratively updating (7) and regressing $y_i$ on the updated $z_i$. The procedure can be summarized as the following algorithm.

**Algorithm 1: Local Linear Approximation (LLA) Algorithm**

**Step 1** Set initial value for $W^{(0)} = [w_1^{(0)}, \ldots, w_j^{(0)}]^T$ and $b_j^{(0)}$, and let $c = 0$.

**Step 2** Calculate $z_i$ defined in the text based on $w_j^{(c)}$ and $b_j^{(c)}$, obtain the least squares estimate (LSE) $\beta_j$s, $\hat{\gamma}_j$s and $\hat{\eta}_j$s by running a linear regression $y_j$ on covariate $z_i$, and update the biases and weights
\[ b_j^{(c+1)} = b_j^{(c)} + \hat{\gamma}_j / \hat{\beta}_j, \quad \text{and} \quad w_j^{(c+1)} = w_j^{(c)} + \hat{\eta}_j / \hat{\beta}_j. \]

Set $c = 1, 2, \ldots$, and repeat Step 2 until the criterion of algorithm convergence meets.

We propose how to construct initial values for the iterative updates in section 2.2.

**2.2 Layer-wise Optimized Adaptive Neural Network**

The back-propagation with gradient decent algorithm is to seek optimal weights and biases in all layers of FNN via minimizing $\ell(\theta)$ in (2). This leads to a high-dimensional, nonconvex minimization problem. We propose a new procedure to assign the weights and biases as follows. We start with $h_1^{(0)} = x$, fit the data with one-hidden layer network as described in section 2.1, and obtain an estimate $\hat{W}$ and $\hat{b}$ of $W$ and $b$. We set $W_1 = \hat{W}$ and $b_1 = \hat{b}$ for the weight matrix and bias vector in FNN. Then we define $\tilde{x}_i = h_1^{(1)} = \sigma(W_1 h_0^{(0)} + b_1)$. We fit data $\{\tilde{x}_i, y_i\}$ with one-hidden layer network, and obtain $W$ and $b$ by the proposed LLA. Then we set $W_2 = W$ and $b_2 = b$. Thus, we construct a LOAN by estimating $W_i$’s and $b_i$’s by running one-hidden layer network on data $\{\tilde{x}_i, y_i\}$ with $\tilde{x}_i = h_1^{(l-1)}$. This procedure can be summarized as the following algorithm.

**Algorithm 2: Layerwise Optimized Adaptive Network (LOAN)**

**Step 1** Input LOAN structure $[J_1, \ldots, J_L]$ with $J_l$ being the number of nodes for the $l$-th layer.

**Step 2** Obtain $W_1$ and $b_1$ in the 1st hidden layer by fitting the data $\{x_i, y_i\}$ to a one-hidden layer neural network by the LLA algorithm. Set $h_i^{(1)} = \sigma(W_1 x_i + b_1)$.

**Step 3** For $l = 1, \ldots, L - 1$, set $\tilde{x}_i = h_i^{(l)}$, and obtain $W_{l+1}$ and $b_{l+1}$ by fitting the data $\{\tilde{x}_i, y_i\}$ to one-hidden layer neural network by the LLA algorithm. Set $h_i^{(l+1)} = \sigma(W_{l+1} \tilde{x}_i + b_{l+1})$.

**2.3 Initial values for the LLA**

Motivated by model fitting for a single index model, we propose an initial value for using the LLA algorithm to estimate $W_l$ and $b_l$ at the $l$-th layer with $J_l$ nodes as follows. Set $\tilde{x} = h_{l-1}^{(l-1)}$, we fit data $\{\tilde{x}_i, y_i\}$ to a linear regression model and obtain the LS estimate of coefficients of $\tilde{x}$, denoted by $\hat{\alpha}$. Let $\tilde{x}_j, j = 1, \ldots, J_l$ be quantile points of $\tilde{x}_i^T \hat{\alpha}, i = 1, \ldots, n$. We may set initial value of $b_j^{(0)} = -\tilde{x}_j$, and $w_j^{(0)} = \hat{\alpha}$ for $j = 1, \ldots, J_l$.

Since the objective function $\ell_1(\theta)$ is nonconvex, numerical minimization algorithm may depend on the initial value. We find the proposed initial value strategy in the last paragraph work reasonably well in practice, but the performance of the LOAN can be further improved by integrating model averaging and cross-validation strategy. Specifically, we randomly partition data into a pre-specified $K$ subsets with approximately the same sample size, and then for $k = 1, \ldots, K$, we fit a LOAN with data excluding the $k$-th data subset, and obtain the mean squared error (MSE) for the entire data set, denoted by $\text{MSE}_k$. Rank the resulting LOANs by their MSEs from smallest to largest. Set the final model to be the average of the top $K_s$ LOANs with smaller MSEs. Here $K_s \leq K$ is a pre-specified integer. In our numerical study, we set $K = 20$ and $K_s = 15$, and find this work well.
Table 1: Comparison with commonly-used procedures with default hyper-parameter value.

<table>
<thead>
<tr>
<th>Methods</th>
<th>AF 100*mean(std)</th>
<th>BS 100*mean(std)</th>
<th>CHP 100*mean(std)</th>
<th>PK 100*mean(std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAN</td>
<td>4.3317(0.5895)</td>
<td>7.8386(0.4148)</td>
<td>18.5801(0.6955)</td>
<td>42.6082(4.9884)</td>
</tr>
<tr>
<td>MLP</td>
<td>25.4913(3.4600)</td>
<td>9.9173(2.0827)</td>
<td>24.1495(1.0754)</td>
<td>31.4595(2.3832)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>5.1042(0.7245)</td>
<td>7.8751(0.3137)</td>
<td>16.4491(0.6905)</td>
<td>3.7880(0.4386)</td>
</tr>
<tr>
<td>RF</td>
<td>6.5254(0.7346)</td>
<td>10.2843(1.0754)</td>
<td>21.2067(0.9217)</td>
<td>2.8041(0.4835)</td>
</tr>
</tbody>
</table>

Table 2: Comparison of the impact of PCA on LOAN and existing procedures.

<table>
<thead>
<tr>
<th>Methods</th>
<th>6 PCs 100*mean(std)</th>
<th>9 PCs 100*mean(std)</th>
<th>13 PCs 100*mean(std)</th>
<th>All (19) PCs 100*mean(std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAN</td>
<td>29.3152(2.7984)</td>
<td>18.3863(2.8369)</td>
<td>27.2601(7.1044)</td>
<td>43.0542(3.6137)</td>
</tr>
<tr>
<td>MLP</td>
<td>41.7293(2.6673)</td>
<td>34.7943(2.3511)</td>
<td>30.6869(2.1018)</td>
<td>30.6617(2.9831)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>50.8823(3.2462)</td>
<td>47.1367(2.7843)</td>
<td>42.9631(2.6408)</td>
<td>44.4751(2.1000)</td>
</tr>
<tr>
<td>RF</td>
<td>41.2433(2.8571)</td>
<td>39.6862(2.6231)</td>
<td>37.7282(2.5413)</td>
<td>39.9511(2.5392)</td>
</tr>
</tbody>
</table>

3 Numerical comparison

This section provides a brief summary of our numerical comparison. A complete and detailed description of data sets including the sample size, dimension of predictors, and the version of existing procedures are given in the Appendix of this paper. In this section, we compare the performance of the proposed LOAN with MLP [15], AdaBoost [4], GBM [5], RF [2], and XGBoost [3] by empirical analyses of four benchmark data sets: airfoil (AF, for short) data, bikesharing (BS, for short) data, california house price (CHP, for short) data, and parkinson (PK, for short) data. XGBoost performs better than AdaBoost and GBM. Thus, we present results of XGBoost only to save space. Results of AdaBoost and GBM are given in the Appendix.

In our numerical comparison, we first standardize the response and predictors so that their sample means and variances equal 0 and 1, respectively. We split each data set into 80% training data and 20% testing data with seed 1, \cdots, 30. Thus, we may obtain 30 MSEs for testing data and 30 MSEs for training data for each procedure. We report the sample mean and standard deviation (std) of the 30 MSEs for each procedure and data set. We first compare the performance of the one-hidden layer LOAN with 60 nodes with all other procedures with default settings. Table 1 depicts the values of 100 times of the sample mean and std of the 30 MSEs, and implies that the LOAN performs well in terms of prediction accuracy. Specifically the LOAN has the smallest MSE for AF and BS data sets. For CHD, XGBoost performs the best, and follows by the LOAN. The LOAN has slightly greater MSE than XGBoost since XGBoost has a deeper tree by default. For PK data, the RF performs the best, and then follows by XGBoost. It seems that both RF and XGBoost perform much better than the LOAN and MLP. This motivates us to examine the data further. By doing exploratory data analysis of PK data, we find that several of its 19 predictors are highly correlated. The poor performance of the LOAN may be due to the collinearity of predictors since the linear regression is used to update the weights and biases. We conduct a principal component analysis (PCA), and the first 6, 9 and 13 principal components (PC) can explain 95%, 99% and 99.9% of the total variance. We further apply all procedures to PK data with predictors being set to PCs. Table 2 depicts the MSEs for PK data with PC predictors. Compared with Table 1 the performance of XGBoost and RF becomes poor. The performance of MLP is quite stable across different numbers of PCs. The performance of the LOAN improves significantly via using PCs. In Tables 1 and 2 we use the one-layer LOAN for comparison. We have examine the performance of the multiple-hidden layer LOAN in the Appendix due to the space limit.

4 Conclusion and discussion

We develop the LLA algorithm and further develop the LOAN. Our comparison indicates that the LOAN may outperform commonly-used procedures. This paper focuses on regression with continuous responses. The LOAN can be extended to classification problems.
Broader Impact

FNN is a commonly-used model in deep learning, which has been used in our daily life. The proposed LLA algorithm can significantly improve the existing algorithm for FNN, and the proposed LOAN can outperform commonly-used procedures such as MLP and XGBoost in deep learning. Researchers in data mining and machine learning can certainly benefit from the proposed LLA algorithm and LOAN. People who use deep learning techniques and methods may also benefit this research in their daily life.

References


This appendix provides a complete section of Section 3 on numerical comparison in the main text. We compare the performance of the proposed LOAN with commonly-used procedures in deep learning by empirical analysis of four data sets:

1. **airfoil (AF, for short) data**, which consists of \( n = 1503 \) observations with \( p = 5 \) predictors and can be downloaded from
   
   
   The response is the scaled sound pressure level. The explanatory variables include frequency, angle of attack, chord length, free-stream velocity, suction side displacement thickness.

2. **bikesharing (BS, for short) data**, which consists of \( n = 17379 \) observations with \( p = 8 \) predictors and can be downloaded from
   
   https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset
   
   The response is the log-transformed count. The explanatory variables include normalized hour, normalized temperature, normalized humidity, normalized wind speed, season, holiday, weekday and weather situation.

3. **California house price (CHP, for short) data** [14], which consists of \( n = 20,640 \) observations with \( p = 8 \) predictors and can be downloaded from Carnegie-Mellon StatLib repository (http://lib.stat.cmu.edu/datasets/).
   
   The response variable is the median house value. The explanatory variables include longitude, latitude, housing median age, medium income, population, total rooms, total bedrooms and households.

4. **Parkinson (PK, for short) data**, which consists of \( n = 5874 \) observations with \( p = 19 \) predictors and can be downloaded from
   
   https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring
   
   The response variable is the total UPDRS scores, and the explanatory variables include subject age, subject gender, time interval from baseline recruitment date, and 16 biomedical voice measures.

In our numerical analysis, we compare the performance of the proposed LOAN with

1. **MLP**: multi-layer perceptron [15, 16] (python scikit-learn version 0.22.1),
3. **GBM**: gradient boosting algorithm [5, 16]
4. **AdaBoost** [4, 16]
5. **RF**: random forest [2, 16]

In our numerical comparison, we first standardize the response and predictors so that their sample means and variances equal 0 and 1, respectively. We split each data set into 80% training data and 20% testing data with seed 1, · · · , 30. Thus, we may obtain 30 MSEs for testing data and 30 MSEs for training data for each procedure. We report the sample mean and standard deviation (std) of the 30 MSEs for each procedure and data set.

We first compare the performance of the one-hidden layer LOAN with all other procedures with default settings. The number of nodes in the LOAN is set to be 60. Table [A.1] depicts the values of 100 times of the sample mean and std of the 30 MSEs of testing data (testing MSE for short) as well as the 30 MSEs of training data (training MSE for short). Table [A.1] implies that XGBoost performs better than other two boosting methods: AdaBoost and GBM, and the LOAN performs well in terms of prediction accuracy. Specifically the LOAN has the smallest testing MSEs for AF and BS data sets. For CHD, XGBoost performs the best, and follows by the LOAN in terms of testing MSE. The LOAN has slightly greater testing MSE than XGBoost, while XGBoost has a deeper tree by default. For AF, BS and CHD data sets, the LOAN has the smallest difference between testing MSE and training MSE. This implies that the LOAN is less likely to be over-fitting. Compared with other procedures, the LOAN has relatively small standard deviation of MSE. This implies that the LOAN has a fairly stable performance. For PK data, the RF performs the best, and then follows by XGBoost. It seems that both RF and XGBoost performs much better than the LOAN and MLP. This motivates us to examine the data further.
257 principal components (PC) can explain 95%, 99% and 99.9% of the variance. We further apply weights and biases. Thus, we conduct a principal component analysis (PCA), and the first 6, 9 and 258
Tables A.1 and A.2 clearly imply that the performance of RF and XGBoost are not robust under linear 259
apply all these procedures to CHD data. Table A.3 depicts the sample means and standard deviation 260
the performance of the LOAN with multiple-hidden layers. To this end, we set the MLP, XGBoost, 261
tables to PK data with predictors being set to PCs. Table A.2 depicts the MSEs for PK data 262
By doing exploratory data analysis of PK data, we find that several of the 19 predictors in PK data 263
are highly correlated. For example, the correlation between Shimmer and Shimmer(db) is 0.9923. 264
The correlation between Jitter(%) and Jitter(RAP) is 0.9841. The poor performance of the LOAN 265
may be due to the collinearity of predictors since the linear regression is applied for updating the 266
weights and biases. Thus, we conduct a principal component analysis (PCA), and the first 6, 9 and 267
13 principal components (PC) can explain 95%, 99% and 99.9% of the variance. We further apply 268
tables to PK data with predictors being set to PCs. Table A.2 depicts the MSEs for PK data 269
with PC predictors. Compared with Table A.1, the performance of XGBoost and RF becomes poor. 270
Tables A.1 and A.2 clearly imply that the performance of RF and XGBoost are not robust under linear 271
transformation on predictors. The performance of MLP is quite stable across different numbers of PC 272
variables. The performance of the LOAN improves significantly via using PCs.

In Tables A.1 and A.2, we use the one hidden-layer LOAN for comparison. It is of interest to examine 273
the performance of the LOAN with multiple-hidden layers. To this end, we set the MLP, XGBoost, 274
GBM and RF with the same number of layer or depth as L, the number of hidden layers used in the 275
LOAN. Other parameters of the MLP, XGBoost, GBM and RF are set to be the default values. We 276
apply all these procedures to CHD data. Table A.3 depicts the sample means and standard deviation

Table A.1: Comparison with commonly-used procedures with default hyper-parameter value.

<table>
<thead>
<tr>
<th>Methods</th>
<th>AF 100*mean(std)</th>
<th>BS 100*mean(std)</th>
<th>CHP 100*mean(std)</th>
<th>PK 100*mean(std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE for Testing Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>4.3317(0.5895)</td>
<td>7.8386(0.4148)</td>
<td>18.5801(0.6955)</td>
<td>42.6082(4.9884)</td>
</tr>
<tr>
<td>MLP</td>
<td>25.4913(3.4600)</td>
<td>9.9173(2.0827)</td>
<td>24.1495(1.0754)</td>
<td>31.4595(2.3832)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>5.1042(0.7245)</td>
<td>7.8751(0.3137)</td>
<td>16.4491(0.6905)</td>
<td>3.7880(0.4386)</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>31.7242(2.6197)</td>
<td>27.7419(0.7384)</td>
<td>56.3744(5.3267)</td>
<td>60.9171(1.8358)</td>
</tr>
<tr>
<td>GBM</td>
<td>14.7338(1.5846)</td>
<td>16.5968(0.5592)</td>
<td>21.2866(0.7254)</td>
<td>21.0340(0.7685)</td>
</tr>
<tr>
<td>RF</td>
<td>6.5254(0.7346)</td>
<td>10.2843(0.4672)</td>
<td>21.2067(0.9217)</td>
<td>2.8041(0.4835)</td>
</tr>
<tr>
<td>MSE for Training Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>2.3213(0.1634)</td>
<td>6.3305(0.2549)</td>
<td>15.7216(0.1275)</td>
<td>28.8462(2.5983)</td>
</tr>
<tr>
<td>MLP</td>
<td>24.2824(0.0268)</td>
<td>7.7488(0.7496)</td>
<td>23.7055(1.1570)</td>
<td>23.7447(1.3502)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>0.2623(0.0268)</td>
<td>4.3713(0.0946)</td>
<td>5.7634(0.1456)</td>
<td>0.3115(0.0264)</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>29.2881(1.1124)</td>
<td>27.2715(0.6059)</td>
<td>55.8631(5.5463)</td>
<td>60.4119(1.9148)</td>
</tr>
<tr>
<td>GBM</td>
<td>11.2280(0.6303)</td>
<td>15.8681(0.1972)</td>
<td>19.5493(0.1864)</td>
<td>19.0881(0.7516)</td>
</tr>
<tr>
<td>RF</td>
<td>0.9366(0.0388)</td>
<td>1.4413(0.0179)</td>
<td>3.8432(0.0766)</td>
<td>0.3897(0.0356)</td>
</tr>
</tbody>
</table>

Table A.2: Comparison of the impact of PCA on LOAN and existing procedures based on PK data

<table>
<thead>
<tr>
<th>Methods</th>
<th>6 PCs 100*mean(std)</th>
<th>9 PCs 100*mean(std)</th>
<th>13 PCs 100*mean(std)</th>
<th>All (19) PCs 100*mean(std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE for Testing Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>29.3152(2.7984)</td>
<td>18.3863(2.8369)</td>
<td>27.2601(7.1044)</td>
<td>43.0542(3.6137)</td>
</tr>
<tr>
<td>MLP</td>
<td>41.7293(2.6673)</td>
<td>34.7943(2.3511)</td>
<td>30.6869(2.1018)</td>
<td>30.6617(2.9831)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>50.8823(3.2462)</td>
<td>47.1367(2.7843)</td>
<td>42.9631(2.6408)</td>
<td>44.4751(2.1000)</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>61.2930(2.6109)</td>
<td>78.9303(2.7339)</td>
<td>76.2363(2.2727)</td>
<td>76.5351(2.5682)</td>
</tr>
<tr>
<td>GBM</td>
<td>67.0713(2.3618)</td>
<td>62.4927(2.2980)</td>
<td>59.1466(1.8293)</td>
<td>59.7928(1.9610)</td>
</tr>
<tr>
<td>RF</td>
<td>41.2433(2.8571)</td>
<td>39.6682(2.6231)</td>
<td>37.7282(2.5413)</td>
<td>39.9511(2.5392)</td>
</tr>
<tr>
<td>MSE for Training Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAN</td>
<td>21.7594(1.5550)</td>
<td>11.8719(1.4716)</td>
<td>16.0485(2.7811)</td>
<td>29.2183(3.0557)</td>
</tr>
<tr>
<td>MLP</td>
<td>37.1915(1.8439)</td>
<td>27.9013(0.9417)</td>
<td>22.0252(0.9036)</td>
<td>20.5078(0.9063)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>10.0009(0.7694)</td>
<td>6.6056(0.4240)</td>
<td>4.0770(0.3162)</td>
<td>2.9568(0.2952)</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>79.1353(2.3975)</td>
<td>76.3020(2.1195)</td>
<td>73.5840(2.0781)</td>
<td>73.5073(2.3712)</td>
</tr>
<tr>
<td>GBM</td>
<td>59.1133(1.5346)</td>
<td>53.7506(1.1314)</td>
<td>49.5714(0.9834)</td>
<td>49.4500(0.9870)</td>
</tr>
<tr>
<td>RF</td>
<td>5.9341(0.2602)</td>
<td>5.6795(0.1998)</td>
<td>5.3804(0.2432)</td>
<td>5.6393(0.2185)</td>
</tr>
</tbody>
</table>
Table A.3: Comparison of the number of hidden layers in LOAN and existing procedures based on CHD data

<table>
<thead>
<tr>
<th>Methods</th>
<th>$L = 1$</th>
<th>$L = 2$</th>
<th>$L = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100*mean(std)</td>
<td>100*mean(std)</td>
<td>100*mean(std)</td>
</tr>
<tr>
<td>MSE for Testing Data</td>
<td>18.5801(0.6955)</td>
<td>18.0160(0.6742)</td>
<td>17.5237(0.6530)</td>
</tr>
<tr>
<td>LOAN</td>
<td>22.5382(0.9667)</td>
<td>20.6272(0.8958)</td>
<td>20.2324(0.8761)</td>
</tr>
<tr>
<td>MLP</td>
<td>28.5367(0.7766)</td>
<td>20.7961(0.6612)</td>
<td>18.3401(0.6515)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>35.5315(0.8693)</td>
<td>24.5913(0.7452)</td>
<td>21.2869(0.7255)</td>
</tr>
<tr>
<td>GBM</td>
<td>66.7451(1.4958)</td>
<td>53.6974(1.1509)</td>
<td>43.9402(1.0961)</td>
</tr>
<tr>
<td>MSE for Training Data</td>
<td>15.7216(0.1275)</td>
<td>14.3645(0.2425)</td>
<td>12.5553(0.1572)</td>
</tr>
<tr>
<td>LOAN</td>
<td>21.2712(0.6557)</td>
<td>18.3534(0.4721)</td>
<td>16.7369(0.5431)</td>
</tr>
<tr>
<td>MLP</td>
<td>27.9085(0.1824)</td>
<td>19.1496(0.2419)</td>
<td>15.1269(0.1666)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>35.1605(0.1962)</td>
<td>23.7616(0.2104)</td>
<td>19.5492(0.1863)</td>
</tr>
<tr>
<td>GBM</td>
<td>66.7071(0.6086)</td>
<td>53.5375(0.3329)</td>
<td>43.4837(0.3294)</td>
</tr>
</tbody>
</table>

of the 30 training and testing data splittings for $L = 1$, 2 and 3. The number of nodes in the LOAN with one hidden layer is set to be 60. The number of nodes in the LOAN with two hidden layer is set to be 60 and 10 in the first and second layer, respectively. The number of nodes in the LOAN with three hidden layers is set to be 60, 10 and 50 in the first, second and third layer, respectively. Since there is no option to set different depth levels for AdaBoost, we exclude it for this comparison. From Table A.3, we can see that both the LOAN and MLP have smaller MSE as $L$ increases, but there is no dramatic improvement, while XGBoost, GBM and RF seem to have more improvement as $L$ increases. Table A.3 indicates that the LOAN performs the best when $L = 1$, 2 and 3.