

Effective Query Size Estimation Using Neural Networks

Hongjun Lu

Rudy Setiono¹

School of Computing

National University of Singapore

Kent Ridge, Singapore 119260

{luhj, rudys}@comp.nus.edu.sg

Abstract

This paper describes a novel approach to estimate the size of database query results using neural networks. Using the proposed approach, three layer neural networks are constructed and trained to learn the cumulative distribution functions of attribute values in relations. With a trained network, the estimation of the query result size could be obtained instantly by simply computing the network output from the given query predicates. The basic computational model using a cumulative distribution function to compute the query result size is described. The network construction and training is discussed. Comprehensive experiments were conducted to study the effectiveness of the proposed approach. The results indicate that the approach produces estimates with accuracies that are comparable with or higher than those reported in the literature.

Keywords: Query processing, relational algebra operations, cost based optimization, query size estimation, neural networks.

1 Introduction

Efficient and accurate estimation of query result size has been one of the important topics in database research. Without accurate size estimation, a query optimizer cannot effectively compare different query execution plans to select the optimal one. Along with the evolution of DBMS technology, accurate size estimation of query results assumes greater and greater importance. Such information can help to evenly distribute load among processors in parallel and distributed database systems. This is probably the major reason that research work on the topic has always been active [3, 6, 16, 8, 14, 15, 13, 7, 12, 9, 18, 2, 11]. As a result, various methods have been proposed for query size estimation. In their papers, Sun *et al.* and Chen *et al.* summarized these methods and grouped them into different categories [18, 2].

¹Corresponding author

In order to obtain an accurate estimation of query sizes, one has to rely on certain information. The existing methods differ in

1. what kind of information is required;
2. how and when the information is collected; and
3. in what form the information is stored.

Almost all of these methods use statistical distributions of the attribute values as the base for estimation. Sampling methods collect such information by sampling a portion of the database, often at runtime when an estimation is required. With sufficient samples, these methods can provide accurate estimation as the information collected is the most recent one. The major drawback of these methods is their relatively high runtime costs. Research work on these methods are focused on how to make sampling more effective, i.e., how to obtain the highest possible accuracy with minimum number of samples. To reduce runtime overhead, a number of methods collect statistics of the attribute values periodically at non-peak hours. The collected information is stored in the system. When an estimation is to be made, the computational effort needed to derive the estimation from the stored information is minimal. There are two basic ways to store the collected statistics: in the form of tables or in the form of approximating functions. With the statistics stored in the tabular form, size estimation only requires table look-up, and runtime overhead is negligible. However, this kind of information is non-parametric and inflexible. Large estimation errors may be introduced. Using functions to approximate the data distribution provides a more flexible way of storing the collected statistics. Most approximating functions are polynomial and only their coefficients need to be stored. Regression techniques are normally used to obtain these coefficients. Computing the values of an approximating function to obtain an estimation for a query size incurs a small runtime cost.

The work reported in this paper is motivated by Sun *et al.* [18]. They proposed a method for estimating size of query results using a regression model. Using their approach, the distribution of the data is described by a series function. The series function is derived by scanning the entire relation and computing the optimal coefficients of the series function using regression such that the series function best approximates the actual data distribution. While the computation of the series function can be conducted off-line during non-peak time, the estimation of result size is almost instantaneous with the regression function. Their experimental results demonstrated that the method provides size estimation with accuracy that is comparable to that of the

sampling method which is believed to provide the most accurate estimation. However the following deficiencies are found in their approach:

- The data distribution is approximated by a polynomial function. To compute a query size, Quadrature Formulas are used to find the area under the function. That is, the method computes two approximates, which may limit the accuracy of the estimation.
- Different series functions for different query types are used. Functions for selections involving up to two variables and equijoin are given. Functions for queries that involve more variables can be expected to be substantially more complex and difficult to compute, if not impossible.

In this paper, we propose a novel approach that overcomes the above problems. Instead of deriving the coefficients of polynomial functions that approximate the data distribution, our approach uses neural networks to learn the distribution functions. It is well known that three layer neural networks are capable of approximating any function arbitrarily well [4, 10]. To learn the distribution function of the data, a three layer neural network is constructed and then trained using data samples off-line. During runtime, parameters such as the constants in the selection predicate are fed into the trained network and the predicted result size is obtained from the output unit of the network. The method has been tested using data with various probability distributions. The results indicate that it provides accuracy higher than those reported by other methods. In addition to high accuracy, this approach has the following desirable features:

- Instead of approximating the probability density functions, our approach employs neural networks to approximate the cumulative distribution function directly. Approximations of the areas under the density function to estimate query size are not necessary, hence, less computation will be required to obtain query size estimation from the trained networks.
- Simple three layer neural networks are used for all types of queries. For different query types, the network topology and the training process are the same. One possible difference is in the number of input units of the neural networks. It depends on the number of attributes referenced by the queries in question. Complex queries do not pose any difficulties to the approach.
- The runtime cost is very low since only a few simple computations are needed. Even for off-line training, the time required is not high as the topology of the network used is

rather simple.

The remainder of the paper is organized as follows: Section 2 describes the basic computation model of using cumulative density function to estimate query result size. Section 3 discusses how neural networks are used to implement the model. In Section 4, an experimental study is reported. Section 5 concludes the paper.

2 Estimation of query result size

Most database management systems support selection, projection and join as primitive operations. A query execution plan consists of a sequence of those primitive operations. The size of a projection is relatively easy to estimate. In this section, we describe the basic model that uses cumulative distribution functions to estimate the sizes of the results of selection and join operations.

2.1 Result size of selection

Selection is an operation that returns the tuples in a relation that satisfy the given selection predicate. A selection predicate can be written as a formula of the form

$$c_1\theta_1a_1 \text{ logop}_1 c_2\theta_2a_2 \text{ logop}_2 \dots c_i\theta_ia_i \text{ logop}_i \dots \text{ logop}_{n-1} c_n\theta_na_n$$

where a_i 's are attributes from the same relation, c_i 's are constants, θ_i 's are relational operators, $\theta_i \in \{=, \leq, <, >, \geq, \neq\}$, and logop_i 's are logical operators, $\text{logop}_i \in \{\text{and}, \text{or}, \text{not}\}$.

We define the cumulative distribution function (CDF) F_x for an attribute x of relation R as follows:

Definition 1 Let X be an attribute of relation R whose domain is $Dom(X)$, the cumulative distribution function F_x is a function of x , $x \in Dom(X)$, that returns the percentage of tuples in relation R satisfying $R.X \leq x$.

Figure 1 shows an example of the cumulative distribution function for an attribute that have been generated according to the standard normal distribution. Fifty thousand random variables were generated and mapped into discrete values in the interval $[0 \dots 10000]$. Let $|R|$ be the total number of tuples in relation R . From the definition, the result size of a selection involving a single attribute can be calculated as follows:

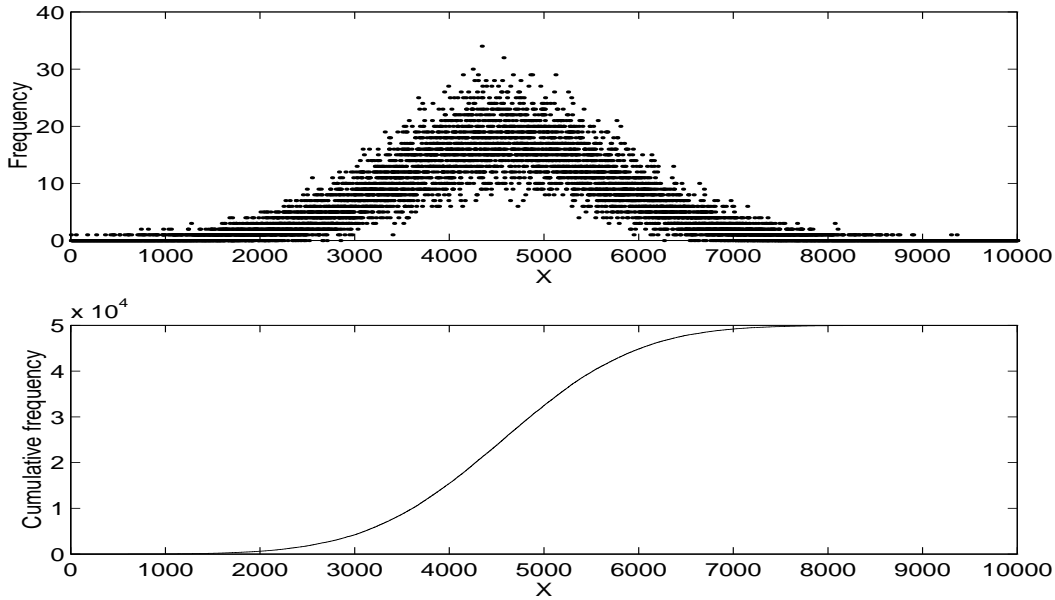


Figure 1: Fifty thousand normally distributed random variables with values in $[1 \dots 10000]$ (top) and its cumulative frequency (bottom).

Selection predicate	Result size
$R.X \leq a$	$ R F_x(a)$
$R.X = a$	$ R (F_x(a) - F_x(a - \Delta a))$
$R.X > a$	$ R (1 - F_x(a))$
$a < R.X \leq b$	$ R (F_x(b) - F_x(a))$

In the table, Δa is the unit value of attribute A . It is reasonable to assume that for any type of attributes, we can find such unit values in real database applications. For continuous numerical attributes, the unit can be either defined by the semantics of the attribute (*age, salary, etc.*) or by the precision in which the attribute value is stored in database and used in applications.

We now discuss how we can estimate the size of a selection involving more than one attribute or complex selection using a cumulative distribution function.

Definition 2 Let X, Y be two attributes of relation R whose domains are $Dom(X)$ and $Dom(Y)$, the cumulative distribution function F_{xy} is a function of $x, y, x \in Dom(X), y \in Dom(Y)$ that returns the percentage of tuples in relation R satisfying $R.X \leq x \wedge R.Y \leq y$.

Using the function F_{xy} , the result size of selections referencing the two attributes in relation R can be computed as follows:

Selection predicate	Result size
$R.X \leq a \wedge R.Y \leq c$	$ R F_{xy}(a, c)$
$a < R.X \leq b \wedge R.Y \leq c$	$ R (F_{xy}(b, c) - F_{xy}(a, c))$
$R.X < a \wedge c < R.Y \leq d$	$ R (F_{xy}(a, d) - F_{xy}(a, c))$
$a < R.X \leq b \wedge c < R.Y \leq d$	$ R (F_{xy}(b, d) - F_{xy}(a, d) - F_{xy}(b, c) + F_{xy}(a, c))$

It is obvious that the above definitions can be generalized to selections with n attributes ($n > 2$).

2.2 Result size of join operation with selections

Join operation is a binary operation. Let R and S be two relations. Join $R \bowtie_{X\theta Y} S$ returns tuple pairs in R and S satisfying the join condition $R.X\theta S.Y$ where θ is a relational operator, $\theta \in \{\leq, <, =, \neq, >, \geq\}$. Similarly, we define the cumulative distribution functions as follows:

Definition 3 Let X and Y be two attributes of relation R and S , respectively. The domains of $R.X$ and $S.Y$ are $Dom(X)$ and $Dom(Y)$. Cumulative distribution function F_x^{join} is a function of x, y , $x \in Dom(X), y \in Dom(Y)$, that returns the percentage of tuples in $R \times S$, the cross product of relation R and S satisfying $R.X \leq x \wedge R.X = S.Y$.

The above definition is motivated by the following observations. An equijoin operation is in fact an operation that finds matching tuples in R and S with the same attribute values on the specified attributes. For a given value of $R.X$ there may be a number of matching tuples in S . For example, if R is a student relation and S is a relation of students' grades, then the result of a join on *student.id* is the number of courses taken by each student. F_x^{join} defined above is the cumulative distribution of the number of matching tuples.

If F_x^{join} defined above is available, the result size of equijoin with selection can be estimated as follows:

Query predicate	Result size
$R \bowtie_{x=y} S$	$ R S F_x^{join}(max)$
$R \bowtie_{x=y} S \wedge a < R.X \leq b$	$ R S (F_x^{join}(b) - F_x^{join}(a))$

where *max* is the maximum value of $R.X$.

3 Using neural networks to learn CDF

3.1 Training feedforward neural networks

Artificial neural networks are densely interconnected networks of simple computational elements, *neurons*. There exist many different network topologies. Among them, the *multi-layer perceptron* is especially useful for *supervised learning*. In supervised learning, a network is *trained* using a set of training data which contains the input patterns and the corresponding expected output or target value of each of these patterns. After training, it is hoped that the network generalizes well, that is, it can predict the output when given a new pattern which may not exist in the training data set with an acceptable accuracy. Applying this concept to query size estimation, we train a network using different queries with known result sizes. The trained network can then be used to estimate the result sizes of similar queries.

Figure 2 shows a three layer feedforward network. It consists of an input layer, a hidden layer and an output layer. A unit (neuron) in the network has a number of inputs and a single output. For example, a neuron H_j in the hidden layer has $x_1^i, x_2^i, \dots, x_n^i$ as input and α^j as output. A link in the network is associated with a weight. The input links of H_j have weights $w_1^j, w_2^j, \dots, w_n^j$. A unit in the hidden layer computes its output, i.e. the *activation value* by summing up its weighted inputs, subtracting a threshold, and passing the result to a non-linear function f , the *activation function*. The outputs from neurons in one layer are fed as inputs to neurons on the next layer. In this manner, when an input tuple is applied to the input layer, an output value is obtained at the output layer.

Networks used to learn cumulative distribution function are rather simple. The number of input units is equal to the number of attributes in the queries. The number of output units is always one. There must be a sufficient number of hidden units in the network so that the network can be trained to an acceptable accuracy rate. The weights for links in the network are given initial values that are randomly generated in the interval $[-1,1]$. The network training aims to adjust these weights so that the output of the network can predict the function value when given input values. This is done as follows.

The activation value of a unit in the hidden layer is first computed by passing the weighted sum of input values to a non-linear activation function. Let w_ℓ^m be the weights for the connections from input unit ℓ to hidden unit m . Given an input pattern $x^i, i \in \{1, 2, \dots, k\}$, where k is the

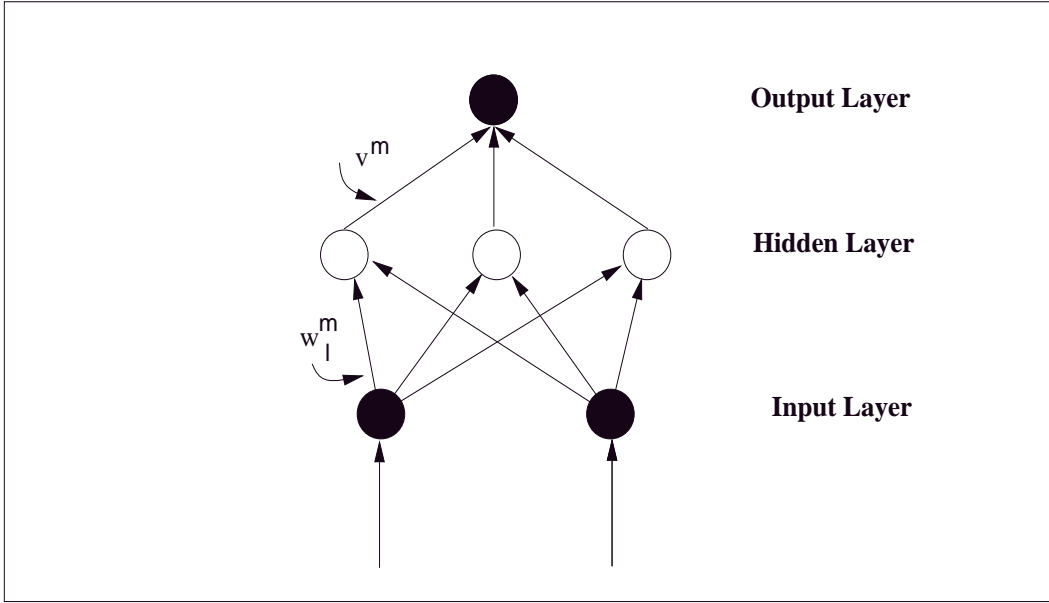


Figure 2: A three layer feedforward neural network.

number of samples in the data set, the activation value of the m -th hidden unit is

$$\alpha^m = \delta \left(\sum_{\ell=1}^n (x_{\ell}^i w_{\ell}^m) - \tau^m \right),$$

where $\delta(\cdot)$ is an activation function. In our study, we use the hyperbolic tangent function

$$\delta(x) := (e^x - e^{-x}) / (e^x + e^{-x})$$

as the activation function for the hidden units, which makes the range of activation values of the hidden units $[-1, 1]$. The real-valued parameter τ^m is commonly called the bias or threshold of hidden unit m .

Once the activation values of all the hidden units have been computed, the output of the network for input x^i is computed as

$$S^i = \sum_{m=1}^h \alpha^m v^m$$

where v^m is the weight of the connection between hidden unit m and output unit and h is the number of hidden units in the network. The prediction error is computed as

$$e^i = S^i - t^i, \tag{1}$$

where t^i is the target value for input x^i .

The ultimate objective of the training phase is to obtain a set of weights that produces an acceptable error level. To measure the prediction error, an error function is needed so that the training process becomes a process to adjust the weights (w, v) to minimize this function. An obvious choice for this function is the sum of the squared errors:

$$E(w, v) = \sum_{i=1}^k (e^i)^2 = (S^i - t^i)^2. \quad (2)$$

The network training starts with an initial set of weights $(w, v)^{(0)}$ and iteratively updates the weights to minimize $E(w, v)$. Any unconstrained minimization algorithm can be used for this purpose. In particular, the gradient descent method is used in the training algorithm known as the backpropagation algorithm. Because of the slow convergence of the standard backpropagation method, many researchers have looked for alternative algorithms for finding the minimum of the error function. A number of alternative algorithms for neural network training have been proposed [1].

To reduce the network training time, which is very important when the data set is large, we employed a variant of the quasi-Newton algorithm [19], the BFGS method. This algorithm has a superlinear convergence rate, as opposed to the linear rate of the gradient descent method. The BFGS method approximates the nonlinear function $f(z)$ at z^k by the quadratic function

$$f_k(z) = f(z^k) + \nabla f(z^k)(z - z^k) + \frac{1}{2}(z - z^k)B^k(z - z^k), \quad (3)$$

where B^k is a symmetric positive definite matrix that approximates the Hessian matrix of the function $f(z)$ at z^k . The search direction can then be generated by minimizing this quadratic function. We define the search direction

$$d^k := z - z^k = -(B^k)^{-1}\nabla f(z^k), \quad (4)$$

and generate the next approximate solution

$$z^{k+1} = z^k + \lambda^k d^k, \quad (5)$$

where λ^k is an optimal solution of the line search problem

$$\min_{\lambda > 0} f(z^k + \lambda^k d^k). \quad (6)$$

Given a search direction d^k , an iterative one-dimensional optimization method can be applied to find a step length λ that solves the line search problem (6). However, this procedure may

require a large number of function and/or gradient evaluations. A step length $\lambda^k > 0$ is considered acceptable if it satisfies the following two conditions.

$$f(z^k + \lambda^k d^k) \leq f(z^k) + c_1 \lambda^k (d^k)^T \nabla f(z^k), \quad (7)$$

$$(d^k)^T \nabla f(z^k + \lambda^k d^k) \geq c_2 (d^k)^T \nabla f(z^k), \quad (8)$$

where c_1 and c_2 are two constants such that $0 < c_1 \leq c_2 < 1$ and $c_1 < 0.5$. The condition (7) is to ensure that the step length λ^k produces a sufficient decrease in the value of the function $f(z)$ at the new point z^{k+1} , while the second condition is to ensure that the step length is not too small. Detailed description of an algorithm for finding a step length λ^k that satisfies both conditions (7) and (8) can be found in [17].

By keeping an approximate of the inverse of the Hessian matrix and updating it at each iteration of the method, the computation of the inverse of the matrix B^k can be avoided. Let us define $H^k = (B^k)^{-1}$, and hence the search direction $d^k = -H^k \nabla f(z^k)$. Since the matrix H^k is to approximate the inverse of the Hessian matrix at z^k , it needs to be updated as the method moves from one point to the next. The BFGS method updates the matrix H^k as follows

$$H^{k+1} = \left(I - \frac{\delta^k (\gamma^k)^T}{(\delta^k)^T \gamma^k} \right) H^k \left(I - \frac{\delta^k (\gamma^k)^T}{(\delta^k)^T \gamma^k} \right)^T + \frac{\delta^k (\delta^k)^T}{(\delta^k)^T \gamma^k}, \quad (9)$$

where

$$\begin{aligned} \delta^k &= z^{k+1} - z^k, \\ \gamma^k &= \nabla f(z^{k+1}) - \nabla f(z^k). \end{aligned}$$

The BFGS quasi-Newton method with inexact line search for minimizing the function $f(z)$ can be summarized as follows.

BFGS Algorithm

Step 0. Initialization.

Choose any z^0 as a starting point. Let $H^0 = I$, set $k = 0$. Let $\epsilon > 0$ be a small terminating scalar.

Step 1. Iterative step.

- Check for convergence:

If $\|\nabla f(z^k)\| \leq \epsilon \max\{1, \|z^k\|\}$ then Stop.

Otherwise

1. Compute the search direction

$$d^k = -H^k \nabla f(z^k).$$

2. Calculate a step length λ^k such that both conditions (7) and (8) are satisfied, let

$$z^{k+1} = z^k + \lambda^k d^k.$$

3. Update the matrix H by (9).
4. Set $k = k + 1$ and repeat Step 1.

3.2 Estimating query size using neural networks

The process of using neural networks to estimate query result size consists of two tasks. The first task is to train neural networks as described in the previous subsection. Before the training, sampling from the database is needed to obtain the training data set. As network training usually takes a significant amount of time, it ought to be done during non-peak hours. A trained network is represented by two sets of weights, one set is the weights between the input units and the hidden units and another set is the weights between the hidden units and output unit. If the network has I input units and H hidden units, total number of weights to be stored is $(I + 1) * H$. Since both I and H are not very large, the space required is not large. A database system can store the trained networks for most frequently asked query types.

The second task is to give inputs to the network in order to estimate the query size. When the result size of a query is to be estimated, the weights of the corresponding network are retrieved. The constants in the query predicates are used as inputs of the network. The activation values of the hidden units and then the output value at the output unit are computed using the network weights. As the computation of the activation values is rather simple, the runtime estimation can be done almost instantly.

The detailed process will be further elaborated in next section where the experimental results are described. These experiments were conducted to simulate a real application of the approach to a database.

4 Experimental results

To evaluate the proposed approach, three sets of experiments were conducted:

1. Simple selection query: $\sigma_{x_1 < X \leq x_2} R$.
2. Complex selection query: $\sigma_{x_1 < X \leq x_2 \wedge y_1 < Y \leq y_2} R$.
3. Equijoin query with selection: $\sigma_{x_1 < X \leq x_2} R \bowtie_{X=Y} S$.

Each experiment was conducted in four steps:

1. generate test data simulating a database;
2. generate training data set for network training;
3. construct networks and train them using the training data set;
4. generate queries to obtain the errors in size estimation.

Synthetic data were used in the experiments. For each data set generated, the number of table size N , the number of distinct attribute values K , and the distribution f , were specified. The values of these parameters are listed below:

Parameters	Description	Values
N	Number of tuples	10000, 20000, 50000
K	Number of distinct values	800, 1000, 10000
f	Distribution of attribute values	Exponential, Normal, χ^2

The training data set consists of tuples having two or more fields depending on the query types. Each field is a possible attribute value except the last field which is the size estimation of the result size. For an $n + 1$ -tuple, $(a_1, a_2, \dots, a_n, T)$, T is the number of tuples in the relation that satisfy the condition $A_1 \leq a_1 \wedge A_2 \leq a_2 \wedge \dots \wedge A_n \leq a_n$ where A_i 's ($1 \leq i \leq n$) are attributes of the data.

The absolute and relative errors are computed as follows:

$$abs_err = \frac{|actual - estimate|}{dbsize} \times 100\%$$

$$rel_err = \frac{|actual - estimate|}{actual} \times 100\%$$

actual is the actual result size and *estimate* is the estimation obtained from the network. For selection queries, *dbsize* is the number of tuples in the original database. For selections with join, *dbsize* is the number of tuples in the join result of two relations. Both error measures are used as we share the same view with Chen and Roussopoulos: for queries with small selectivity, the relative estimation error does not really reflect the goodness of an estimation [2]. When selectivity is small, a large relative error is mainly caused by the small number of actual tuples satisfying the query predicates. An extreme case is that, when a query returns no tuples, the relative error rate will be infinite.

4.1 Simple selection

Simple selection refers to the selection that involves only one attribute. Experiments were conducted to measure the error rates for estimating the result size of query $\sigma_{x_1 < X \leq x_2} R$. The steps of the experiments are as follows:

1. Generate the data set:
 - (a) Select the number of possible distinct values for the attribute K .
 - (b) Generate N random variables (RVs) according to a prespecified distribution function f .
 - (c) Normalize the RVs such that their range is $[0, 1]$.
 - (d) Map the normalized RVs into distinct tuples x_1, x_2, \dots, x_K and assign the value i to x_i for all i .
 - (e) For all $i = 1, 2, \dots, K$, compute $P(i)$, the cumulative distribution $P_i = \sum_{j=1}^i N(x_j)$, where $N(x_j)$ is the number of normalized RV's that have been mapped to x_j .
2. Randomly select p -percent from the data set for neural network training. Each tuple in the data set is a pair of normalized input value x_i/K and the associated target value P_i/K .
3. Test the accuracy of the network:

For $i = 1, 2, \dots, \#queries$:

 - (a) Generate two random numbers x_1 and x_2 , $0 \leq x_1 < x_2 \leq K$.

- (b) Compute the actual number of tuples in the data set satisfying $x_1 < X \leq x_2$.
- (c) Use the network to find the predicted number of tuples = $N * (S(x_2/K) - S(x_1/K))$.
- (d) Compute the prediction error.

We generated $N = 50000$ tuples and distributed them among $K = 10000$ distinct values. Three distribution functions were tested:

1. $\text{Exp}(\beta)$: exponential distribution with mean β ,
2. $N(0,1)$: standard normal distribution, and
3. χ_{10}^2 distribution with 10 degrees of freedom.

Each of the 50000 tuples in the data was selected with p -percent probability for neural network training. The value of p was 10, 25 and 50. The total number of selections was 100. If the number of selected tuples was less than 1 percent (500 tuples), the selection was discarded. Otherwise, the predicted size was estimated by the neural network. The relative errors computed were grouped into 5 groups according to the size of the selections: $s = [1\%, 5\%)$, $s = [5\%, 10\%)$, $s = [10\%, 25\%)$, $s = [25\%, 50\%)$, and $s = [50\%, 100\%]$ of the number of tuples in the data set, N .

The network had one output unit, 10 hidden units and 2 input units. One of the input units was used to represent the bias at the hidden units. Its input for all training tuples was 1. The second input was the normalized x_i , i.e., i divided by K . Neural network training was terminated if a point in the weight-space of the network that was sufficiently close to a local minimum of the error function had been obtained. The measure used is the gradient of the error function:

$$\|\nabla E(w, v)\| \leq \epsilon \times \max(1, \|w, v\|)$$

where $\|w, v\|$ is the Euclidean distance of (w, v) from 0, $\nabla E(w, v)$ is the gradient of $E(w, v)$ and $\epsilon = 10^{-10}$.

The results of the experiments are summarized in Tables 1- 3.

4.2 Complex selection

For complex selection, data sets with two columns were generated to simulate database relations. The procedures for data and query generation are similar to those for simple selection.

Table 1: Relative error rates for exponential distribution with $\beta = 100$

	$p = 10\%$	$p = 20\%$	$p = 50\%$
$1\% \leq s < 5\%$	1.65	1.61	1.63
$5\% \leq s < 10\%$	0.40	0.41	0.39
$10\% \leq s < 25\%$	0.33	0.32	0.32
$25\% \leq s < 50\%$	0.32	0.30	0.31
$50\% \leq s \leq 100\%$	0.08	0.08	0.09

Table 2: Relative error rates for the standard normal distribution

	$p = 10\%$	$p = 20\%$	$p = 50\%$
$1\% \leq s < 5\%$	1.08	1.08	1.11
$5\% \leq s < 10\%$	0.58	0.59	0.56
$10\% \leq s < 25\%$	0.45	0.43	0.45
$25\% \leq s < 50\%$	0.21	0.21	0.21
$50\% \leq s \leq 100\%$	0.04	0.04	0.04

Table 3: Relative error rates for χ_{10}^2 distribution

	$p = 10\%$	$p = 20\%$	$p = 50\%$
$1\% \leq s < 5\%$	0.79	1.12	1.17
$5\% \leq s < 10\%$	0.43	0.51	0.80
$10\% \leq s < 25\%$	0.37	0.33	0.62
$25\% \leq s < 50\%$	0.17	0.18	0.23
$50\% \leq s \leq 100\%$	0.04	0.04	0.11

Table 4: Estimation errors for selection $\sigma_{x_1 < R.X \leq x_2 \wedge y_1 < R.Y \leq y_2} R$, $|R| = 10000$

Distribution of $R.X - R.Y$	$s < 10\%$		$10 \leq s < 20\%$		$20 \leq s < 30\%$		$30 \leq s < 40\%$		$40 \leq s < 50\%$		$50 \leq s \leq 100\%$	
	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
exp-exp	0.42	10.04	0.84	6.23	1.87	7.54	1.74	4.97	2.43	5.34	3.97	6.71
exp-norm	0.64	13.88	1.05	7.55	0.71	3.12	0.52	1.48	0.59	1.32	0.54	0.85
exp- χ^2	0.58	19.50	0.65	4.40	0.92	3.97	0.91	2.53	0.88	1.94	1.13	1.78
norm-norm	0.31	11.15	0.44	3.31	0.53	2.15	0.46	1.41	0.47	1.03	0.60	0.89
norm- χ^2	0.56	29.76	0.30	2.17	0.42	1.73	0.59	1.68	0.60	1.31	0.31	0.46
$\chi^2 - \chi^2$	0.72	32.91	0.54	3.65	0.45	1.82	0.61	1.78	0.38	0.84	0.65	0.99
Average	0.53	19.54	0.63	4.55	0.81	3.38	0.80	2.30	0.89	1.96	1.20	1.94

Table 5: Estimation errors for selection $\sigma_{x_1 < R.X \leq x_2 \wedge y_1 < R.Y \leq y_2} R$, $|R| = 20000$

Distribution of $R.X - R.Y$	$s < 10\%$		$10 \leq s < 20\%$		$20 \leq s < 30\%$		$30 \leq s < 40\%$		$40 \leq s < 50\%$		$50 \leq s \leq 100\%$	
	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
exp-exp	1.42	28.14	1.39	10.93	1.37	5.34	1.53	4.39	1.66	3.69	2.98	4.59
exp-norm	0.59	15.67	0.57	4.25	0.25	0.97	0.44	1.23	0.29	0.65	0.49	0.76
exp- χ^2	1.06	24.24	0.96	7.49	1.61	6.22	1.88	5.42	1.63	3.67	2.02	3.20
norm-norm	0.57	15.99	0.63	4.58	0.57	2.34	0.81	2.32	1.05	2.36	0.60	0.78
norm- χ^2	0.68	19.00	0.39	2.38	0.46	1.78	0.43	1.22	0.54	1.18	0.66	0.81
$\chi^2 - \chi^2$	0.73	17.86	1.06	7.78	0.65	2.55	0.64	1.87	1.09	2.41	0.49	0.62
Average	0.84	20.15	0.83	6.23	0.88	3.20	0.95	2.90	1.04	2.32	1.20	1.79

We chose two relation sizes, 10000 and 20000. With three probability distributions for each attribute, six relations of each size were generated.

The networks for complex selection has one more input unit since two attribute values are needed for training. The training tuples have three fields, x_1, y_1 and N , where N is the number of tuples in R that satisfy the condition $R.X \leq x_1 \wedge R.Y \leq y_1$. The number of distinct values for $R.X$ and $R.Y$ and the number of neural network training samples are summarized below.

Relation Size	No of Distinct Values	No of Training Samples
10000	800	2000
20000	1000	5000

To evaluate the accuracy of the approach, 120 queries were randomly generated. In order to get an insight on the accuracy of the networks, the queries were grouped according to their selectivity s . The error rates of estimating the query size using the networks are listed in Table 4 and Table 5.

From these tables, it can be seen that the relative error rates are reasonably small, especially for large selectivities. The higher error rate for queries with low selectivity partly comes from the number of tuples satisfying the condition is relatively small. A small difference between the predicted number and the actual number of tuples produces a large relative error. The results

also show that for highly skewed data the estimation error is higher than for other data sets. For example, when the two attributes are exponentially distributed, or one is exponentially distributed and the other has a χ^2 distribution, the error rates are higher. The reason could be that with a relatively small number of samples, the network may not be able to predict with good accuracy around the area where the distribution changes quickly. Note that, in the experiments where the relation size is 10000 and there are 800 distinct values, there could be up to 640000 sampling points. Only 2000 samples, which is a very small portion, are used for training the network.

Comparing our results shown with those reported by Sun *et al.* [18], it can be seen that our approach does produce higher accuracy than theirs. They reported error rates ranging from 8% to 20%.

4.3 Join with selections

Experiments were also conducted for join with selections. The data generation and network training for joins were similar to those for complex selections. Six pairs of relations were generated. The sizes of the join results are listed in Table 6. It can be seen that the sizes of the join are rather representative with respect to the join selectivities.

Table 6: Relations used to test join with selection

Distribution of R and S	10000 \bowtie 10000	20000 \bowtie 20000	50000 \bowtie 50000
exp-exp	971,996	3,206,106	7,827,699
exp-norm	5,538	3,016	25,317
exp- χ^2	101,084	389,739	2,099,011
norm-norm	280,890	828,781	2,630,472
norm- χ^2	65,322	380,084	82,888
$\chi^2 - \chi^2$	306,686	812,257	2,726,096

After a network had been trained, the following procedure was used to test the accuracy of the neural network's estimates:

1. One hundred queries were generated.
2. For each query, two random numbers, x_1 and x_2 , $0 \leq x_1 < x_2 \leq K$ were generated.
3. For each pair of x_1 and x_2 , the result size of query $\sigma_{x_1 < R.X \leq x_2} R \bowtie_{x=y} S$ using the original data was computed.

Table 7: Estimation errors for equijoin with selection $\sigma_{x_1 < R.X \leq x_2} R \bowtie S$

Distribution of R - S	10000 \bowtie 10000		20000 \bowtie 20000		50000 \bowtie 50000	
	Abs Err.	Rel. Err.	Abs. Err.	Rel Err.	Abs. Err.	Rel. err.
exp-exp	0.13	3.40	0.01	3.82	0.01	3.28
exp-norm	0.45	0.99	0.45	0.55	0.33	0.98
exp- χ^2	0.10	4.28	0.09	4.61	0.33	3.74
norm-norm	0.06	6.07	0.03	4.36	0.02	3.21
norm- χ^2	0.26	5.36	0.10	6.00	0.23	1.49
$\chi^2 - \chi^2$	0.05	3.46	0.04	6.20	0.04	5.68
Average	0.17	3.93	0.12	4.26	0.11	3.06

4. For each pair of x_1 and x_2 , the size of the query result was predicted using the trained network.
5. The absolute and relative errors of the prediction were computed. For absolute error, the join size listed in Table 6 was used as the data size.

The results obtained are shown in Table 7.

It can be seen from this table that the estimates from the network are quite accurate with the maximum error rate of about 6%.

5 Conclusions

In this paper, we reported our study on using neural networks to estimate the result size of relational queries, including selection and selection with equijoin. In the proposed approach, neural networks are trained to learn the cumulative distribution function of attribute values. The network structure and the training process are discussed. Comprehensive experiments were conducted to evaluate the effectiveness of the neural network approach. The results indicate that neural networks can provide estimates with accuracy rates that are comparable with or better than those reported in the literature.

The accuracy of the proposed approach largely depends on the accuracy of the trained network. To obtain a network that can predict well, the selection of the training data plays a vital role. One of the possible future work is to incorporate the sampling techniques proposed in the literature into the network training process so that the network training can be more effective. The approach can be generalized to handle more complex queries. The structure

of the networks for the more complex queries and the network training method are similar to those described in this paper.

References

- [1] R. Battiti. First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation* 4 (1992) 141–166.
- [2] C.M. Chen and N. Roussopoulos. Adaptive selectivity estimating using query feedback. In *Proceedings of ACM SIGMOD Conference on Data Management*, Minneapolis, MN, (May 1994) 161–172.
- [3] S. Christodoulakis. Estimating block transfers and join sizes. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, New York, (1983) 40–54.
- [4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2 (1989) 303-314.
- [5] J.E. Dennis Jr. and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations* (Prentice Hall, Englewood Cliffs, NJ, 1983).
- [6] J. Fedorowicz. Database evaluation using multiple regression techniques. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Boston, MA (1984) 70–76.
- [7] W-C. Hou and G. Ozsoyoglu. Statistical estimator for aggregate relational algebra queries. *ACM Transactions on Database Systems*, 16:4 (1991) 600–654.
- [8] W-C. Hou, G. Ozsoyoglu, and B.K. Taneja. Statistical estimators for relational algebra expressions. In *Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Austin, TX (March 1988) 276–287.
- [9] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of ACM SIGMOD Conference on Data Management*, San Diego, CA (May 1992) 341–350.
- [10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks* 2 (1989) 359-366.

- [11] B. Harangsri, J. Shepherd and A.H.H. Ngu. Query size estimation using machine learning. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, Melbourne, Australia (1997), 97–106.
- [12] Y.E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Denver, CO (1991) 268–277.
- [13] R.J. Lipton and J.F. Naughton. Practical selectivity estimation through adaptive sampling. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ (1990) 1–11.
- [14] C.A. Lynch. Selectivity estimation and query optimization in large databases with highly skewed distribution of column values. In *Proceedings of 14th International Conference on Very Large Data Bases*, Los Angeles, CA (1988) 240–251.
- [15] M. Muralikrishna and D. DeWitt. Equidepth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Chicago, IL (1988) 28–36.
- [16] G. Piatesky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Boston, MA (1984) 256–275.
- [17] D.F. Shanno and K.H. Phua. Algorithm 500: Minimization of unconstrained multivariate functions. *ACM Transaction on Mathematical Software*, 2(1) (1976) 87–96.
- [18] W. Sun, Y. Ling, N. Rishe, and Y. Deng. An instant and accurate size estimation method for joins and selection in a retrieval-intensive environment. In *Proceedings of ACM SIGMOD Conference on Data Management*, Washington, DC (1993) 79–88.
- [19] R.L. Watrous. Learning algorithms for connectionist networks: Applied gradient methods for nonlinear optimization. In *Proceedings of IEEE First International Conference on Neural Networks*, IEEE Press, New York (1987) 619–627.