# Cardinality Estimation Using Neural Networks

Henry Liu*
University of Waterloo
200 University Ave W
Waterloo, Canada
h228liu@uwaterloo.ca

Mingbin Xu*
York University
4700 Keele Street
Toronto, Canada
xmb@cse.yorku.ca

Ziting Yu*
University of Waterloo
200 University Ave W
Waterloo, Canada
z44yu@uwaterloo.ca

Vincent Corvinelli
IBM Canada Ltd.
8200 Warden Ave
Markham, Canada
vcorvine@ca.ibm.com

Calisto Zuzarte
IBM Canada Ltd.
8200 Warden Ave
Markham, Canada
calisto@ca.ibm.com

## ABSTRACT

Database query optimizers benefit greatly from accurate cardinality estimation; however, this is hard to achieve on tables with correlated and/or skewed columns. We present a novel approach using neural networks to learn and approximate selectivity functions that take a bounded range on each column as input, effectively estimating selectivities for all relational operators. Experimental results with a simplified prototype show a significant improvement over state-of-the-art cardinality estimators on constructed datasets in terms of accuracy, efficiency, and amount of user input required.

## Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.2.4 [**Database Management**]: Systems

## General Terms

Theory, Algorithms

## Keywords

Relational Database, Cardinality Estimation, Machine Learning, Neural Network

## 1. INTRODUCTION

In order for a database query optimizer to properly evaluate the cost of different access plans, it must accurately estimate the cardinality at each stage of an access plan. The application of a predicate reduces the cardinality by a factor called the selectivity. It is common to treat the predicates of a query independently when computing their net selectivity. However, the predicates can be statistically correlated, and thus the net selectivity of the query is not always the product of the individual selectivities. This is a known problem in the field of relational query optimization.

Database systems typically collect a set of statistics to estimate the selectivity of predicates. These statistics can be collected just-in-time on a sample on the table, through automatic background collection whenever the tables have changed significantly, etc. In spite of this, cardinality estimation continues to be one of the major reasons why database query optimizers sometimes choose non-optimal access plans.

DB2®, for example, uses number of rows, distinct values, high and low values, frequent values, and histograms to compute estimates for basic predicates. For equality predicates, such as `c1 = 5`, DB2 uses frequent values and distinct values. For range predicates, such as `c2 > 10`, histograms or high and low values are used to interpolate the fraction of rows selected. `BETWEEN` predicates are also specially treated to compute the range of values of interest. The selectivity for disjunctive predicates, such as `c1 = 5 OR c2 > 1`, is derived by appropriately combining estimates for the individual subterms. `IN` predicates, such as `c1 IN (2, 5, 7)`, have their estimates computed in similar way as `OR` predicates on the same column. For `LIKE` predicates of the form `c1 LIKE 'Can%'` , DB2 may exploit the starting string to compute estimates similar to a `BETWEEN` predicate. There is also special treatment for common expressions used in predicates, such as `MONTH(datecol) = 6`, where the semantics help produce better statistics.

With multiple conjunctive predicates, making assumptions using column value independence frequently results in cardinality underestimation. In order to handle statistical correlation between columns when there are multiple equality join predicates, DB2 may use column group (CG) statistics, if defined. This statistic has information about the number of distinct values in a combination of columns. For example, a CG on `(make, model)` in a car database captures the number of distinct make and model combinations. Estimation for the predicate combination of `make = 'HONDA'` and `model = 'ACCORD'` uses the CG statistic to capture the fact that ACCORD as a model would typically not be the name used by another car manufacturer. The user can specify a reasonable number of column combinations as part of a

*Work performed while author was an intern at IBM®

group and the selectivity estimates are adjusted using multiple groups. DB2 does not handle statistical correlation for range predicates[7].

To estimate the selectivity of join predicates, DB2 uses appropriate formulas that assume independence and inclusion; that is, the values in the column of the table that has the smaller number of distinct values are all included in the table that has the column with the larger number of distinct values. CG statistics can also be used to capture statistical correlation between multiple join predicates between two tables. When there is skew in the values of the join columns, the selectivity is better estimated using statistical views where statistics are collected on the result of tables that are joined [11].

Known solutions addressing statistical correlation between columns in selectivity estimation involve collecting statistics across different sets of columns, such as in [1], [2], and [5]. However, these solutions involve user input to collect the appropriate level of statistics, which is often difficult to determine, and are generally aimed toward equality predicates. Furthermore, statistics must also be regularly updated, a computationally expensive task. Other solutions such as [10], which involves materialized query tables, suffer from the same drawback of needing extensive user analysis and manual input. These solutions have in common inaccuracies when applied to particular datasets.

We propose a novel solution using neural networks (NNs). Preliminary work done on an NN-based approach have indicated some success, but are limited to specific predicates [3] or uniformly-distributed data in columns [8]. There has also been work done using other related machine learning techniques, such as Probabilistic Relational Models in [6]. In this paper, we generalize an NN-based approach to a neural network architecture that successfully learns a function estimating the selectivity of any query containing only relational predicates. This method requires almost no user input, and is capable of automatically adjusting its estimates as the database changes and more queries are run.

## 2. NEURAL NETWORKS

A neural network is a weighted graph with a layered architecture (Figure 1). Each node is called a neuron, and is a functional abstraction of a real neuron. Each layer is composed of several neurons, including a bias neuron whose value is always 1, and successive layers are completely connected. The neurons in each layer take as input the values of the neurons in the previous layer, and computes a function of those values and the connecting weights as its output.

Formally, let $x_{n,j}$ denote the value of the $j$-th neuron in the $n$-th layer, and $W^n_{i,j}$ denote the weight of the connection from $x_{n,i}$ to $x_{n+1,j}$. Then

$$z_{n+1,j} = \sum_i W^n_{i,j} x_{n,i} \tag{1}$$

$$x_{n+1,j} = \sigma\left(z_{n+1,j}\right) \tag{2}$$

where $\sigma$ is the activation function, generally chosen to be a sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{3}$$

The NN learns by adjusting its weights in a process called reinforcement learning, where the network learns and generalizes from example inputs and expected outputs presented

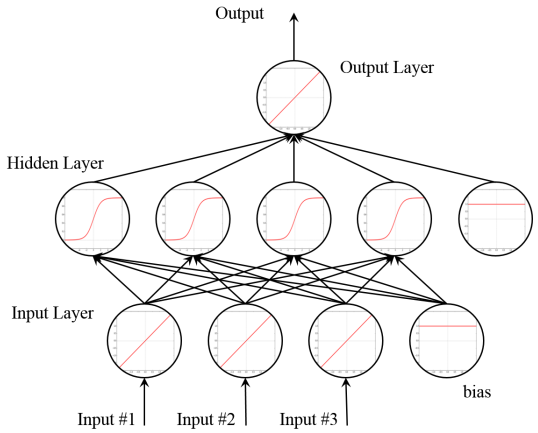


**Figure 1: A simple three-layer $3 \times 4 \times 1$ NN**

to it. An algorithm called backpropagation is used to train the NN as follows. Suppose we have a 3-layer NN as shown in Figure 1, and we have already calculated the output given by the NN for an example input. Let $E(y,t)$ be an error metric that measures how incorrect the output $y$ is with respect to the expected output $t$. Then for each weight in each layer, we may calculate

$$\frac{\partial E}{\partial W^n_{i,j}} = \frac{\partial E}{\partial \sigma}\frac{\partial \sigma}{\partial z_{i+1,j}}\frac{\partial z_{i+1,j}}{\partial W^n_{i,j}}. \tag{4}$$

Thus each weight may be adjusted such that the error for each training example slowly decreases, and hence the NN learns to give certain outputs for certain inputs. This is accomplished by the following update rule, where $\alpha$ is called the learning rate:

$$W^n_{i,j} := W^n_{i,j} - \alpha\frac{\partial E}{\partial W^n_{i,j}}. \tag{5}$$

On a higher level, this process generally trains the NN to recognize patterns in the input, and allows it to generalize and extrapolate to new inputs that it has not seen before using the training examples that it has already seen.

## 3. LEARNING DISCONTINUOUS SELECTIVITY FUNCTIONS

Consider a single-table query with range predicates on each of its columns, where each $l_i$ is a pre-specified lower bound and $u_i$ an upper bound:

```
select * from t where t.c1 > l1 and t.c1 < u1 and
                      t.c2 > l2 and t.c2 < u2 and
                      t.c3 > l3 and t.c3 < u3 ...
```

Let $N$ be the number of columns in this table `t`. We may view the selectivity of queries of this sort as a function of the values $l_i$ and $u_i$. Formally, we have the selectivity function

$$\text{Sel} : \mathbb{R}^{2N} \to \mathbb{R}, \quad (l_1, u_1, \ldots, l_N, u_N) \mapsto c \tag{6}$$

where $c$ is the selectivity of the query represented by the values $l_i$ and $u_i$.

Our insight involves using NNs to learn to produce the correct output for this function given its inputs. It is known already that a typical NN exploits the continuity and smoothness of its activation function to learn a continuous function.

The Universal Approximation Theorem given in [4] essentially states that a NN with a single hidden layer is capable of learning and approximating any continuous function to an arbitrary degree of accuracy as more and more neurons are added. Given any continuous and analytic function $f : S \to \mathbb{R}$, where $S \subseteq \mathbb{R}$ is compact, there exists $N \in \mathbb{N}$, $a_i, b_i \in \mathbb{R}$, $W_i \in \mathbb{R}^m$ such that

$$\left| \sum_{i=1}^{N} a_i \sigma(W_i^T x + b_i) - f(x) \right| < \epsilon \tag{7}$$

for all $\epsilon > 0$ and $x \in S$. Note that the summatory term is the mathematical representation of a 3-layer NN, where the $b_i$ is the weight of the bias unit. We call this the architecture of the NN.

However, the selectivity function is almost never continuous. In fact, we expect it to have up to $M$ jump discontinuities, where $M$ is the number of rows in the table `t`. Hence a standard activation function is insufficient to accurately learn a discontinuous selectivity function.

Selmic and Lewis detail a solution to this problem in [9], where they use the set of "jump" activation functions

$$\varphi_k(x) = \begin{cases} 0 & x < 0 \\ (1 - e^{-x})^k & x \geq 0 \end{cases} \tag{8}$$

for an "augmented" NN architecture that can learn a smooth function $g : \mathbb{R} \to \mathbb{R}$ of one input with a single, known point of discontinuity at $d$:

$$\left| \left( \sum_{i=1}^{N} a_i \sigma(W_i^T x + b_i) + \sum_{i=1}^{N} c_i \varphi_i(x - d) \right) - g(x) \right| < \epsilon. \tag{9}$$

We extend this result to a multi-input NN that can learn piecewise continuous functions with multiple discontinuities $d_j$. Firstly, observe that the proof of Equation 9 given in [9] extends to $x \in \mathbb{R}^n$ , and therefore to architectures with more than one input neuron. Furthermore, we may deal with multiple points of discontinuity $d_j$ by composing multiple single-discontinuity approximators together. Thus we have, for some $M \in \mathbb{N}$, an approximator for the selectivity function Sel:

$$\left| \left( \sum_{i=1}^{N} a_i \sigma(W_i^T x + b_i) + \sum_{j=1}^{M} \sum_{i=1}^{N} c_{i,j} \varphi_i(x - d_j) \right) - \text{Sel}(x) \right| < \epsilon. \tag{10}$$

To simplify this, it turns out that the original Universal Approximation Theorem given in Equation 7 is applicable to NNs with any activation function that is nonconstant, bounded, continuous, and monotonically increasing, not just $\sigma$. But $\varphi_i$ satisfies these conditions on the domain $x \geq 0$, and hence we have

$$\left| \left( \sum_{i=1}^{N} a_i \sigma(W_i^T x + b_i) - \sum_{i=1}^{N} c_i \varphi_i(V_i^T x + d_i) \right) - \text{Sel}(x) \right| < \epsilon. \tag{11}$$

where $V$ is another set of weights, and $d_i$ is such that $V_i^T x + d_i \geq 0$ for all valid inputs $x$. Note that this requires the input $x$ be bounded.

Finally, we may combine Equations 10 and 11 to obtain a simpler NN architecture using only the jump activation functions as follows:

$$\left| \left( \sum_{j=1}^{M} \sum_{i=1}^{N} a_{i,j} \varphi_i(W_{i,j}^T x + b_{i,j}) \right) - \text{Sel}(x) \right| < \epsilon. \tag{12}$$

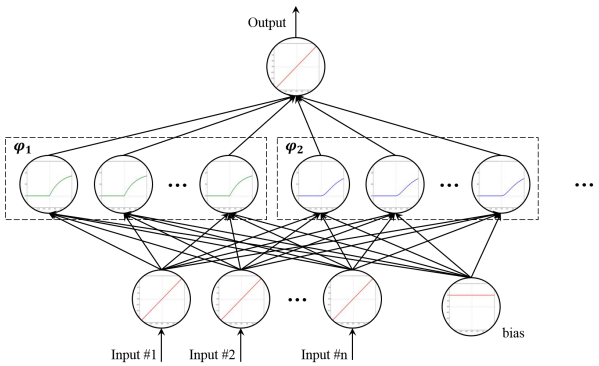


**Figure 2: Augmented NN architecture (hidden neurons grouped by activation)**

Figure 2 visually demonstrates this architecture. Neurons in the hidden layer are grouped by the jump function that they compute. Each group is fully connected to the input layer, i.e. each neuron in the hidden layer takes as input all the input neurons. Similarly, the output is connected to every neuron in the hidden layer. Note that we can map any non-numerical data type to the reals while preserving ordering. For the remainder of this paper, we assume all data is numerical.

# 4. IMPLEMENTATION DETAILS

Given this theoretical framework justifying the use of a 3-layer augmented NN to learn the selectivity function, we wish to expand its functionality. Primarily, we want to make the process of training it as automatic as possible, and we want it to be applicable to as many query operators as possible.

## 4.1 Training Query Generator

The set of queries that should be given to the NN as initial training examples must satisfy two dichotomous properties: it should be broad and varied enough that the NN is able to sample from most of the data space, but it should also be specific and precise enough that the NN can capture large changes in selectivity over a small change in input, e.g. when there are significantly many rows that share the same value in one or more columns. Furthermore, the task of generating varied queries of this sort becomes even more difficult in tables with a large number of columns which may be highly correlated or skewed, and there is no guarantee that a randomly formed query will even have a non-zero cardinality.

In this sense, training query generation is a chicken-and-egg problem, because we need to have some sense of the data distribution in order to generate effective queries, but the data distribution is what we are seeking to capture with the NN.

We propose the following general query generation algorithm that is reasonably effective at generating interesting queries for datasets with a small number of columns.

**Algorithm 1** Automatic Training Query Generator

1: **procedure** GENQUERY($numCol$)
2: $nShrink \leftarrow$ `randInt`$(1, numCol)$
3: $C \leftarrow$ a subset of $\{1, \ldots, numCol\}$ that contains $nShrink$ elements
4: **for all** $C_i \in C$ **do**
5: GENBOUNDS($C_i$) ▷ assign predicate bounds to each column
6: **end for**
7: **end procedure**

We do not explicitly specify an algorithm for predicate bound generation because various methods can work depending on the properties of the training queries needed. Some methods are to randomly choose valid bounds, or to begin with the minimum and maximum lower and upper bounds respectively and gradually shrink the bounds until a desired selectivity is achieved.

In our implementation we adopt a naive approach sampling each column proportionately to the number of distinct values within it. We apply predicates on up to $k$ columns at a time, where $k$ is bounded ensuring that the set of all possible $k$ column-tuples is not too large for a random generation process to generate a significant fraction of them.

Beyond this set of initial training examples, the NN can be continuously trained using the results from the database system's workload of queries.

## 4.2 Adjacent Value Estimator

Recall that our current NN architecture assumes that predicates only contain non-strict range operators, i.e. the operators $\geq$ and $\leq$. From these two operators, we may derive an equality operator by the equivalence

$$\texttt{t.c1} = x \iff x \leq \texttt{t.c1} \leq x. \tag{13}$$

However, we lack the strict range operators $>$ and $<$. To extend the functionality of the NN to predicates containing these operators, observe the equivalence between the predicates

$$l_1 < \texttt{t.c1} < u_1 \iff l_1^+ \leq \texttt{t.c1} \leq u_1^- \tag{14}$$

where $x^+$ denotes the smallest value greater than $x$ in its column, and $x^-$ the largest value smaller than $x$. Thus, a query containing predicates with strict operators may be rewritten into a query containing only predicates with non-strict operators, given a mechanism for finding $x^+$ and $x$ on any element of any column. Note that with this, we are able to handle all relational operators. Similar to existing functionality in database systems, this derivation can be extended to other, more complex predicates, such as `OR`, `IN`, `LIKE`, `NOT`, `IS NULL`, and `IS NOT NULL`.

Instead of keeping an index on every column, which becomes inefficient for larger tables, we may accurately estimate $^+$ and $^-$ via 2N additional small augmented NNs(depicted in Figure 3), two on each column of the table: one for approximating $^+$, and one for $^-$. We refer to these addditional NNs as estimators, and they may be trained simultaneously with the main NN, using the bounds from the training query generator.

# 5. PERFORMANCE ON DATASETS

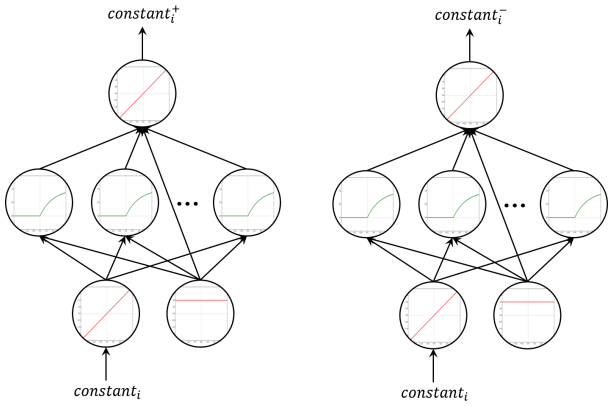


**Figure 3: Two separate NNs for $^+$ and $^-$ estimation**

| Column | DataType | Number of Distinct Values |
|---|---|---|
| seatbelton | character | 2 |
| with | character | 3 |
| driver | character | 3 |
| damage | character | 4 |
| year | integer | 74 |
| carid | integer | 5516 |
| ownerid | integer | 7755 |

**Table 1: Table Description of Car Accidents**

In order to test the predictive capacity of the NN, we run the NN against multiple datasets designed to contain highly skewed and/or correlated columns. Each dataset contains 40000 rows. In addition, we create a special single-column table with multiple unique numbers to test our NN-based estimator.

We use only a first-order approximation to the augmented NN for our prototype, for ease of implementation: all higher-order $\varphi_k$ neurons, for $k > 1$, are omitted, and all neurons, including the output, have $\varphi_1$ as their activation function.

## 5.1 Datasets Description

*Strongly Correlated Columns (StrongCor).*

This dataset may have a variable number of columns. Let `randInt(a,b)` pick a random integer between $a$ and $b$, inclusive. Denoting the value of the $c$-th column for the $i$-th row as $x_{i,c}$, the dataset is as follows:

$$x_{i,1} = \texttt{randInt}(0, 100)$$
$$x_{i,n} = (x_{i,n-1} + \texttt{randInt}(2, 3)) \mod 100$$

*Car Accidents.*

This synthetic dataset contains seven columns as in Table 2. When we fabricate the data, we follow the facts that (1) the usage of seatbelt decreases the injury level of the driver and (2) the object that the car hits or gets hit by affects the damage level of the car. These are described by four columns, namely, 'seatbelton', 'with', 'driver' and 'damage'.

## 5.2 Datasets Results

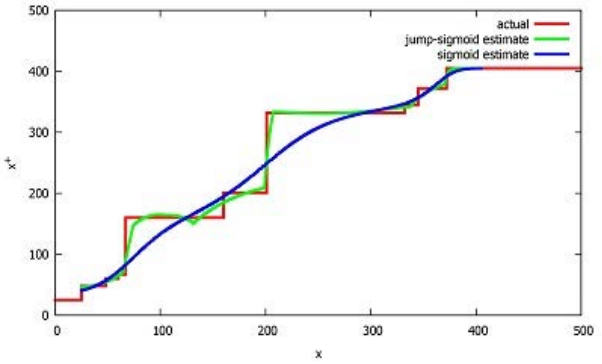


**Figure 4: Estimator Learned by Augmented NN**

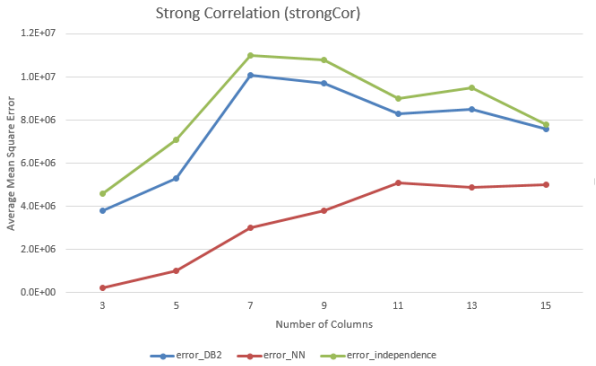


**Figure 5: Error of strongCor**

To start off, we demonstrate the capacity of an augmented NN to learn piecewise-continuous functions, and its advantage over a standard sigmoidal NN. Figure 4 plots the function learned by an augmented NN for the estimator. This estimator is trained on a sparse set of integers $x_i \in [0, 500]$. For comparison, the actual $x_i^+$ function is plotted along with the function learned by a sigmoidal NN trained in exactly the same way.

To test the performance of the main NN, we generate a test set of 10000 queries on `strongCor` which are uniformly distributed in terms of their resultant cardinalities when applied to a particular dataset. Each dataset is generated and inserted into a DB2 database as a table. For each query in the test set, we find the selectivity predicted by the NN, the DB2 estimator, and the independence/uniformity assumption, i.e. that the distribution of values in a column is uniform and independent of the distribution of values in any other column. The focus of this test is on the accuracy of the NN as a selectivity estimator and not the cost of the initial training set. Further study is needed to evaluate this cost.

Figure 5 illustrates these errors, averaged, for the `strongCor` dataset. Note that it does not fully demonstrate the capacity of the NN, because as the number of columns increases, the training query generator and test set query generator cannot generate as non-trivial queries as when the number of columns are low. We call a query non-trivial if it exposes correlation and/or skew, and therefore is not estimated well by an independence/uniformity assumption. However, when these kinds of queries are constructed manually, the NN gives cardinality estimates to within an order of magnitude while DB2 and the independence/uniformity assumption gives sub-one estimations.

The results of equality predicates and range predicates are reported in Table 4 and Table 5 respectively for `Car Accidents`. We contrast our augmented NN with the optimizer estimates from DB2. To ensure fairness, we collect CG (column groups) statistics, i.e. the number of distinct tuples, that covers the aforementioned correlations, as in Table 3. Each row in Table 4 and Table 5 represents the results of a query. For example, Row 1 of table Table 4 is expressed in SQL as:

```
SELECT   *
FROM     CARS.ACCIDENTS
WHERE    seatbelton = 'N' and
         with = 'animal' and
         driver = 'dead'
```

while Row 1 of Table 5 is:

```
SELECT   *
FROM     CARS.ACCIDENTS
WHERE    'ANIMAL' <= with <= 'CAR' and
         'DEAD' <= driver <= 'INJURED' and
         'MINOR' <= damage <= 'SEVERE'
```

In Table 5, the first half rows and the second half rows capture the lower bounds and the upper bounds of queries respectively. Cases in which lower bound is identical to upper bound or the range covers the entire column are not considered, because the former are effectively equality predicates while the latter provides no filtering effect. Thus they are not shown in the result set. The rightmost three columns are NN estimate, DB2 estimate and the actual cardinality respectively.

The primary cardinality adjustment in DB2 is CG statistics. The disadvantage is that it assumes uniform distribution for infrequent tuples. As displayed in Table 4, both infrequent and non-existent tuples are assigned an identical cardinality `1/|CGS| = 2505`, which is off more than an order of magnitude. On the contrary the augmented NN accurately estimate the cardinality in most cases. NN learns from the data instead of assuming any data distribution. When it comes to range predicates, DB2 is significantly outperformed because CGS doesn't help and it falls back to independence assumption (Table 5). The NN's self-adaptation to unknown data distribution is well demonstrated. However, attention must be paid that NN accepts only numerical inputs. Therefore a hash technique which perseveres the sort order of character sequences is required. In our implementation, we take advantage of DB2's hash mechanism and it works reasonably well.

## 6. CONCLUSION

These results demonstrate that NNs have the capacity to accurately estimate the selectivity of relational queries for datasets of highly skewed or correlated values, an improvement over statistics-based estimation methods. In particular, the augmented NN architecture improves significantly over a traditional sigmoid architecture even when only a first-order approximation is used. We show that with highly

| Column | DataType | Number of Distinct Values |
|---|---|---|
| seatbelton | character | 2 |
| with | character | 3 |
| driver | character | 3 |
| damage | character | 4 |
| year | integer | 74 |
| carid | integer | 5516 |
| ownerid | integer | 7755 |

**Table 2: Table Description of Car Accidents**

| CG | CG Cardinality |
|---|---|
| ownderid, carid | 12767 |
| seatbelton, with | 6 |
| with, dirver | 9 |
| with, damage | 12 |
| dirver, damage | 12 |
| with, driver, damage | 34 |
| seatbelton, with, dirver | 16 |
| seatbelton, with, driver, damage | 62 |

**Table 3: CG statistics of Car Accidents**

| seatbelton | with | driver | NN | DB2 | ACTUAL |
|---|---|---|---|---|---|
| N | ANIMAL | DEAD | 70 | 2505 | 14 |
| N | ANIMAL | INJURED | 1 | 2505 | 229 |
| N | ANIMAL | UNHARMED | 1067 | 2505 | 973 |
| N | CAR | DEAD | 3872 | 2505 | 4937 |
| N | CAR | INJURED | 2275 | 2505 | 1824 |
| N | CAR | UNHARMED | 11109 | 3458 | 349 |
| N | PEOPLE | DEAD | 237 | 2505 | 32 |
| N | PEOPLE | INJURED | 1111 | 2505 | 669 |
| N | PEOPLE | UNHARMED | 3197 | 2505 | 2907 |
| Y | ANIMAL | DEAD | 1 | 2505 | 0 |
| Y | ANIMAL | INJURED | 883 | 2505 | 291 |
| Y | ANIMAL | UNHARMED | 2728 | 2505 | 2535 |
| Y | CAR | DEAD | 3560 | 3531 | 3412 |
| Y | CAR | INJURED | 6378 | 5173 | 8457 |
| Y | CAR | UNHARMED | 5468 | 8157 | 5032 |
| Y | PEOPLE | DEAD | 1 | 2505 | 0 |
| Y | PEOPLE | INJURED | 1608 | 3127 | 827 |
| Y | PEOPLE | UNHARMED | 6951 | 4087 | 7595 |

**Table 4: Result of Queries with Equality Predicates**

| with | driver | damage | NN | DB2 | ACTUAL |
|---|---|---|---|---|---|
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 11385 | 2946 | 9761 |
| ANIMAL CAR | INJURED UNHARMED | MINOR SEVERE | 13295 | 2946 | 11498 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 13207 | 2946 | 10634 |
| CAR PEOPLE | INJURED UNHARMED | MINOR SEVERE | 21548 | 2946 | 18703 |
| ANIMAL CAR | DEAD INJURED | NONE SEVERE | 7902 | 1726 | 7710 |
| ANIMAL CAR | INJURED UNHARMED | NONE SEVERE | 8485 | 1726 | 7268 |
| CAR PEOPLE | DEAD INJURED | NONE SEVERE | 8605 | 1726 | 8195 |
| CAR PEOPLE | INJURED UNHARMED | NONE SEVERE | 15792 | 1726 | 11310 |
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 6190 | 2727 | 11454 |
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 7338 | 2727 | 12422 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 7082 | 2727 | 11963 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 12326 | 2727 | 16350 |
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 6748 | 3328 | 11676 |
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 7979 | 3328 | 12950 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 8000 | 3328 | 12720 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 13793 | 3328 | 21280 |
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 6384 | 1821 | 2273 |
| ANIMAL CAR | DEAD INJURED | MINOR SEVERE | 7581 | 1821 | 4758 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 7607 | 1821 | 3196 |
| CAR PEOPLE | DEAD INJURED | MINOR SEVERE | 13338 | 1821 | 12323 |

**Table 5: Result of Queries with Range Predicates**

skewed and correlated datasets containing a large number of columns, an augmented NN can reliably predict the selectivity of relational queries to within an order of magnitude, which is sufficient for the generation of good access plans; with less skewed and correlated datasets, we often see accuracy down to the single digits. However, this is based on the assumption that a good training workload can be provided to or automatically generated for the NN.

# 7. FURTHER RESEARCH

In this final section, we outline some avenues of research that should be pursued to improve this preliminary experiment at replacing the current statistics-based cardinality estimators with a more flexible and intelligent one.

## 7.1 Relational Operators Across Columns

While an NN-based cardinality estimator is capable of handling relational op- erators, it only does so when the operator compares values in the column with a constant, i.e. we can handle `t.c1 > 3` but not `t.c1 > t.c2`. A simple, naive solution to this problem is to estimate `t.c1 > t.c2` by decomposing it into constant chunks, as follows,

$$\bigvee_i (\texttt{t.c1} > x_i) \wedge (x_i \leq \texttt{t.c2} < x_{i+1}) \tag{15}$$

where $x_i$ is some ordered set of constants. Note that the larger the set, the more accurate the approximation. In fact, there should be more $x_i$ where the selectivity function has higher gradient, and less where it has lower gradient.

We have yet to test this naive decomposition method. Regardless, it seems inefficient and highly prone to error, and there appears to be no good procedure for choosing $x_i$ without knowing the selectivity function beforehand. Hence, we hope to find a more efficient method, perhaps based on modifying the architecture of the NN, that does not depend on

a decomposition approximation.

## 7.2 Neural Network Implementation

In section 3, we justified the use of a 3-layer augmented NN to learn the selectivity function. While our implementation is capable of accurately estimating the selectivity function, we did not exhaustively evaluate the various types of NN that can be used, such as recurrent NNs or deep NNs.

# 8. ACKNOWLEDGEMENTS

The student authors would like to acknowledge their IBM counterparts for the freedom to pursue their solution to the cardinality estimation problem that they were challenged with.

# 9. TRADEMARKS



# 10. REFERENCES

[1] T. Beavin, B. Iyer, A. Shibamiya, H. Tie, and M. Wang. Query optimization through the use of multi-column statistics to avoid the problems of column correlation. (U.S. Patent 5995957A), November 1999.

[2] T. Beavin, B. Iyer, A. Shibamiya, H. Tie, and M. Wang. Query optimization through the use of multi-column statistics to avoid the problems of non-indexed column correlation. (U.S. Patent 6272487B1), August 2001.

[3] J. Boulos and Y. Viemont. Selectivity estimation using neural networks.

[4] G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2:303–314, February 1989.

[5] J. Freytag, G. Lohman, and C. Zuzarte. Query optimization method for incrementally estimating the cardinality of a derived relation when statistically correlated predicates are applied. (U.S. Patent 6738755B1), May 2004.

[6] B. T. L. Getoor and D. Koller. Selectivity estimation using probabilistic models. In *Proceedings of the ACM SIGMOD*, pages 461–472, 2001.

[7] S. Kapoor and V. Corvinelli. (2006, December 21). "Understand column group statistics in DB2". *IBM developerWorks*. Available from `http://www.ibm.com/developerworks/data/library/techarticle/dm-0612kapoor/`. Accessed 24 August 2015.

[8] S. Lakshmi and S. Zhou. Selectivity estimation in extensible databases — a neural network approach. In *Proceedings of the 24th VLDB Conference*, pages 623–627, 1991.

[9] R. Selmic and F. Lewis. Neural network approximation of piecewise continuous functions: Application to friction compensation. *IEEE Transactions on Neural Networks*, 13:745–751, July 2000.

[10] D. Simmen. Query optimization technique for obtaining improved cardinality estimates using statistics on pre-defined queries. (U.S. Patent 20040181521A1), September 2004.

[11] "Statistical views". *IBM Knowledge Center, DB2 for Linux UNIX and Windows 10.5.0*. Available from `http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.perf.doc/doc/c0021713.html`. Accessed 24 August 2015.