

# Relational Database as a Machine Learning Tool

John Nagel

[jknage01@louisville.edu](mailto:jknage01@louisville.edu)

University of Louisville CSE Department

Dr. Antonio Badia

# Introduction

- Relational databases fit the needs for large, scalable data storage solutions.
- Machine learning is used to infer trends or make predictions to inform decision making processes.
- Deploying modern machine learning libraries for production instances require implementing a separate API and backend interactions to transport data and use neural network models.
- This project will test how well the algorithms for regression and classification perform written purely in a relational database.

# Project Goals

- Implement gradient descent and backpropagation using a neural network written in SQL on data found in the database.
- Build a website to interface with the database to create, train, and test network models.
- Evaluate the performance of machine learning within a relational database and compare to established machine learning frameworks.
  - Loss - a measure of the difference between expected and actual result (“confidence”).
  - Accuracy - how well models can successfully output predictions.
  - Speed - the time it takes to collect, normalize, and train a network model.

# Requirements

- Extract columns from existing tables as datasets to use as training samples separated into features and label(s), where values are abstracted to be referenced by index and not column name.
- Train models with a single procedure that can use any dataset sample and model size.
- Create network models of a given size, shape, activation functions, and loss for a dataset.
- Train model with a given learning rate and batch size.
- Test model for final loss and accuracy.

# Systems

- Relational Database - PostgreSQL 12
- Web Server - Flask (Python)
  - Database Adapter - psycopg
  - Asynchronous communication - Socket IO

MLDB Home

Dataset Network

Datasets

	ID	Name
<input type="radio"/>	23	airfoil
<input type="radio"/>	18	wine_cultivars
<input type="radio"/>	16	test05
<input type="radio"/>	15	test04
<input type="radio"/>	12	test

Refresh

Delete

Insert

Source Table Name Dataset Name Normalize Features Normalize Labels

None

Feature Column List (comma delimited)

Label Column List (comma delimited)

Insert

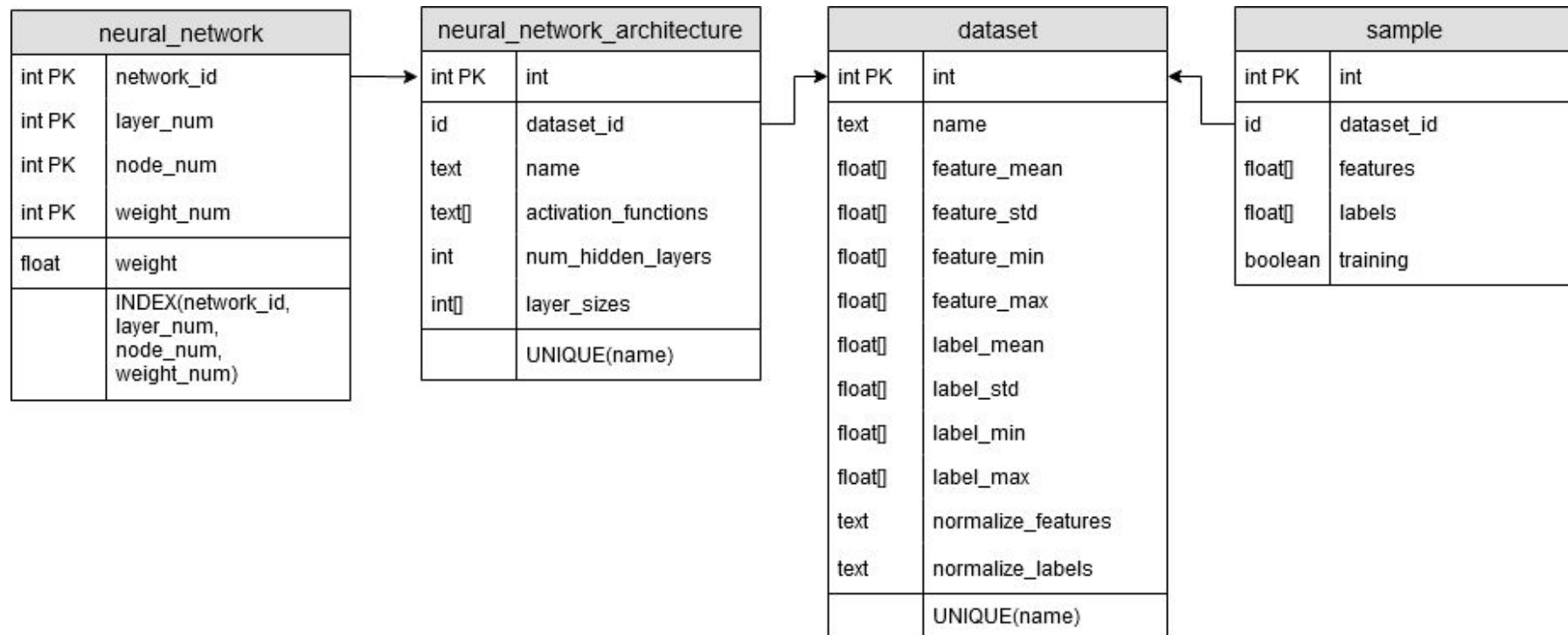
Messages

Connected to Server

Test

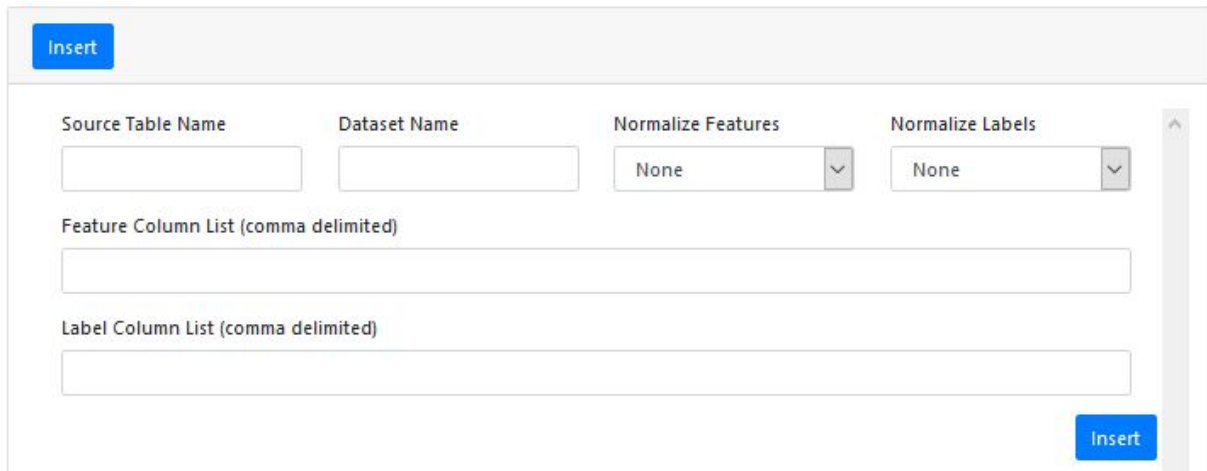
Clear

# Database



# Data Import

- Select data from an existing table or view to be used as a dataset for training.
- List columns to be used as features or labels and how to normalize them.
  - None, Z-Score, Min-Max
- Data inserted into dedicated “sample” table to not lock table while training.



The form is titled "Data Import" and contains the following fields and controls:

- Source Table Name:** A text input field.
- Dataset Name:** A text input field.
- Normalize Features:** A dropdown menu with "None" selected.
- Normalize Labels:** A dropdown menu with "None" selected.
- Feature Column List (comma delimited):** A text input field.
- Label Column List (comma delimited):** A text input field.
- Buttons:** There are two blue "Insert" buttons, one at the top left and one at the bottom right.






# Model Creation

- Create neural network models for a dataset.
- Specify name, loss function, layer sizes, and activation functions.

Insert

Train

Test

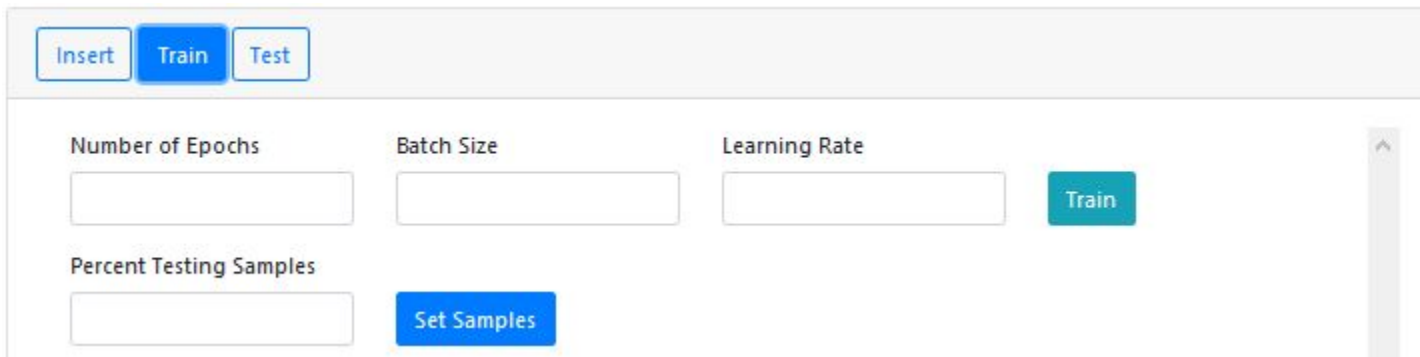
Name	Dataset	Loss Function	
<input type="text"/>	ID:2 (iris) 	Cross Entropy 	
Number of Input Nodes	Number of Hidden Layers	Number of Hidden Nodes	Number of Output Nodes
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Input Layer Activation Function		Hidden Layer Activation Function	Output Layer Activation Function
Linear 		Linear 	Linear 

Insert



# Model Training

- Train a selected model (not shown).
- First separate a random percent of the samples for testing.
- Specify number of epochs, batch size, and learning rate.
- Evaluate change in loss for each epoch from messages (not shown).



The image shows a user interface for model training. At the top, there is a light gray bar containing three buttons: 'Insert', 'Train' (highlighted in blue), and 'Test'. Below this bar, the interface is divided into two rows of input fields. The first row contains three text input fields labeled 'Number of Epochs', 'Batch Size', and 'Learning Rate', followed by a blue 'Train' button. The second row contains a text input field labeled 'Percent Testing Samples' and a blue 'Set Samples' button. A vertical scrollbar is visible on the right side of the interface.

Insert Train Test

Number of Epochs Batch Size Learning Rate

Percent Testing Samples

Train Set Samples

# Model Testing

- Test a selected model for loss and accuracy.
- Specify what samples to use for testing (only training or testing).
- Individual samples shown in table if checkbox checked.
- Average loss and accuracy always shown.

InsertTrainTest

Number of Samples

Show Individual Samples

Test Samples Only

☐

☐

Test

Results

Sample ID	Output Number	Label Value	Output Value	SM Output Value	Loss	Accuracy
-----------	---------------	-------------	--------------	-----------------	------	----------

# Example Usage

<u>iris</u>	
sepal_width	float
sepal_length	float
petal_width	float
petal_length	float
species	text

Insert

Source Table Name

Dataset Name

Normalize Features

Normalize Labels

iris

iris

Z-Score

None

Feature Column List (comma delimited)

sepal\_width,sepal\_length,petal\_width,petal\_length

Label Column List (comma delimited)

species

Insert

# Example Usage (cont.)

Insert

Train

Test

Name

iris\_model\_1

Dataset

ID:2 (iris)

Loss Function

Cross Entropy

Number of Input Nodes

4

Number of Hidden Layers

2

Number of Hidden Nodes

8

Number of Output Nodes

3

Input Layer Activation Function

ReLU

Hidden Layer Activation Function

ReLU

Output Layer Activation Function

Linear

Insert

Networks

	ID	Name	Dataset	Layers	Activation Fns	Loss Fn
<input checked="" type="radio"/>	112	iris_model_1	iris	IN:4, HDN:2x8, OUT:3	RELU,RELU,LINEAR	CROSS_ENTROPY

# Example Usage (cont.)

Insert

Train

Test

Number of Epochs

1

Batch Size

25

Learning Rate

0.1

Train

Percent Testing Samples

15

Set Samples

Messages

DB Activity finished successfully  
NOTICE: End (14:53:04.015459). Duration (00:00:00.44195)  
NOTICE: Epoch 1 Avg Loss (1.0993834474040165)  
NOTICE: Epoch 1 (14:53:03.645873), since last (<NULL>)  
NOTICE: Beginning Training (14:53:03.644469). Duration (00:00:00.071021)  
NOTICE: Begin (14:53:03.573544)  
NOTICE: Network:112, Dataset:2 Layers:3, [RELU:RELU:LINEAR] Loss:CROSS\_ENTROPY, Batches:6, NF:ZSCORE, NL:<NULL>  
Beginning DB Query - train\_network  
DB Activity finished successfully  
Beginning DB Query - set\_training\_samples

Test

Clear

# Example Usage (cont.)

InsertTrainTest

Number of Samples

4

Show Individual Samples☒

Test Samples Only☒

Test

Results

Sample ID	Output Number	Label Value	Output Value	SM Output Value	Loss	Accuracy
					1.46	0.5
9119	2	2	0.59	0.471	0.753	1
9073	3	3	0.227	0.42	0.867	1
9076	2	3	0.59	0.471	2.495	0
9079	2	3	0.297	0.424	1.724	0

# Experiment

- Tested different models using the datasets Iris, Wine Cultivar, and Airfoil.
- Recorded accuracy before and after training, loss during training.
- Compared to a Keras models with equivalent architecture and initialization.
- Results averaged across 5 runs.

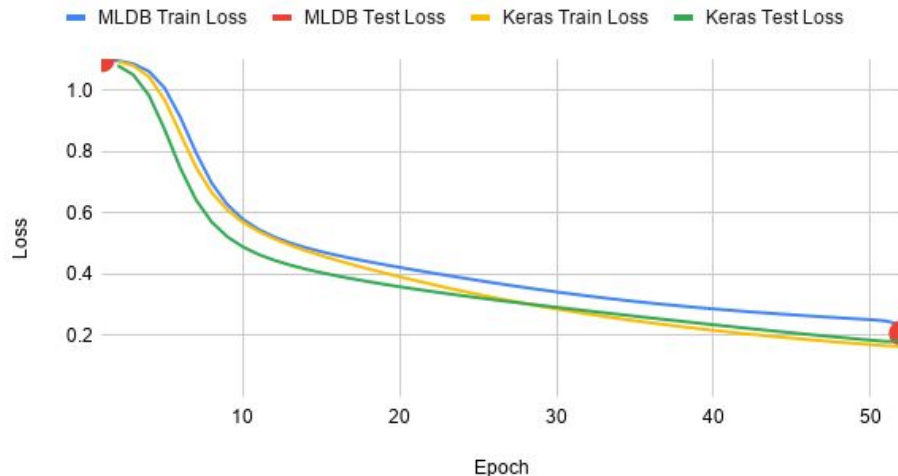
<u>Dataset</u>	<u>Features</u>	<u>Samples</u>	<u>Type</u>	<u>Tested Models</u>
Iris	4	150	Classification	(1x8 tanh), (2x6 tanh), (2x12 linear)
Wine Cultivar	13	178	Classification	(1x8 tanh), (2x6 tanh), (2x12 linear)
Airfoil	5	1503	Regression	(1x8 tanh), (1x16 tanh), (2x6 tanh)

\* model “(1x8 tanh)” indicates one hidden layer with 8 nodes using where all outputs use the tanh activation function. Input and Output layer sizes determined by dataset.

# Results

- Loss decreased similarly to Keras loss for Iris, slower with Wine and Airfoil.

MLDB/Keras Iris (1x8) Loss



MLDB/Keras Wine (2x6) Loss





# Results (cont.)

- Final accuracy comparable to Keras.

Iris (1x8)

<u>Epoch</u>	<u>MLDB Train Acc.</u>	<u>Keras Train Acc.</u>	<u>MLDB Test Acc.</u>	<u>Keras Test Acc.</u>
0	22.34	8.66	22.73	8.69
25	88.91	95.28	99.09	93.91

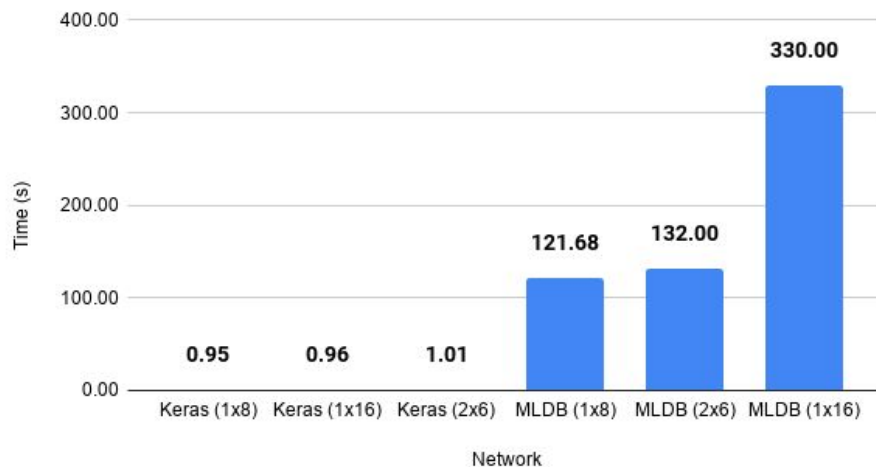
Airfoil (1x16)

<u>Epoch</u>	<u>MLDB Train Acc.</u>	<u>Keras Train Acc.</u>	<u>MLDB Test Acc.</u>	<u>Keras Test Acc.</u>
0	83.18	86.5	82.48	82.2
25	97.12	95.71	97.25	94.72

# Results (cont.)

- Major flaw in this implementation is runtime.
- Time can be hundreds of times slower than Keras
- Database has difficulty materializing intermediate data between layers during forward and back propagation.
- This worsens for larger datasets and larger model sizes.
- Image to right uses only 1503 samples and has 113 parameters maximum.

Airfoil Data Load and Train Times



# Project Summary

- Implemented gradient descent and backpropagation using only a relational database.
- Designed an environment that can apply machine learning to any database schema with minimal effort.
- Created a user interface to easily create and test models only using the database.

# Conclusion

- Machine learning can successfully be applied to any environment with benefits and drawbacks.
- Accuracy outcomes matched those of an existing machine learning framework through an easy to use interface at the sacrifice of flexibility.
- SQL's declarative approach is not meant to process algorithms with many looping structures or recursion.
- This project was successful in its implementation and fulfilling requirements, but does not reach optimal performance outcomes to be viable in the real world.

# Source Code

<https://github.com/JKNags/MLDB>

# Report

[https://github.com/JKNags/MLDB/blob/master/report/mlldb\\_report.pdf](https://github.com/JKNags/MLDB/blob/master/report/mlldb_report.pdf)