Relational Database as a Machine Learning Tool

John Nagel
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

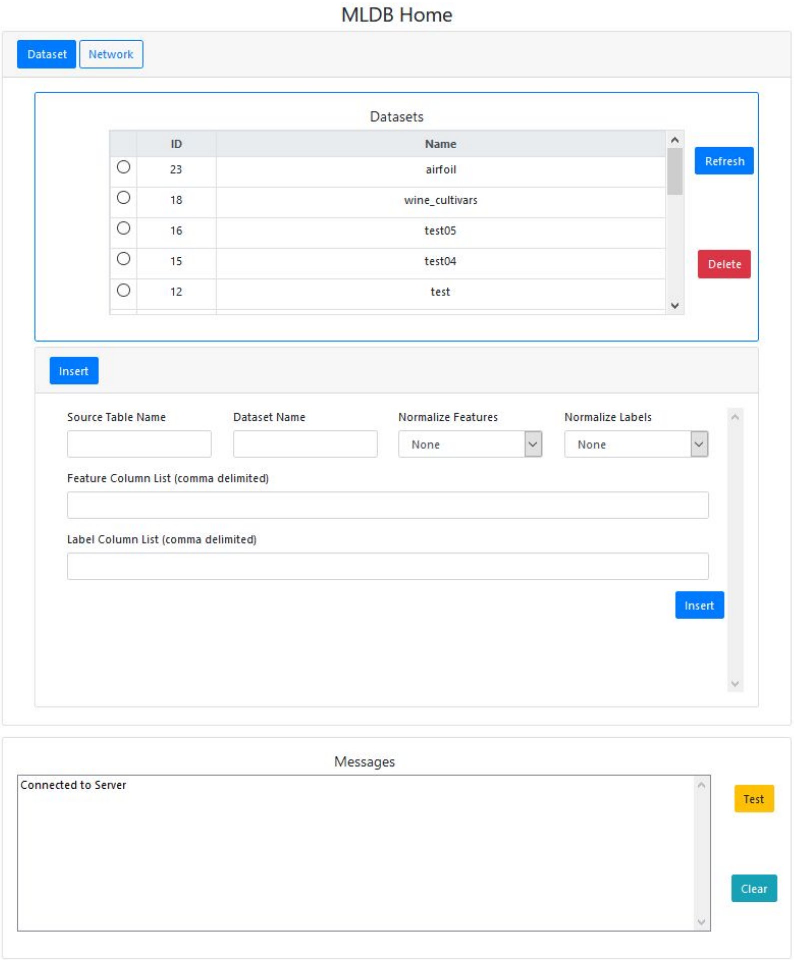
- Implement gradient descent and backpropagation using a neural network written in SQL on data found in the database.
- Build a website to interface with the database to create, train, and test network models.
- Evaluate the performance of machine learning within a relational database and compare to 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

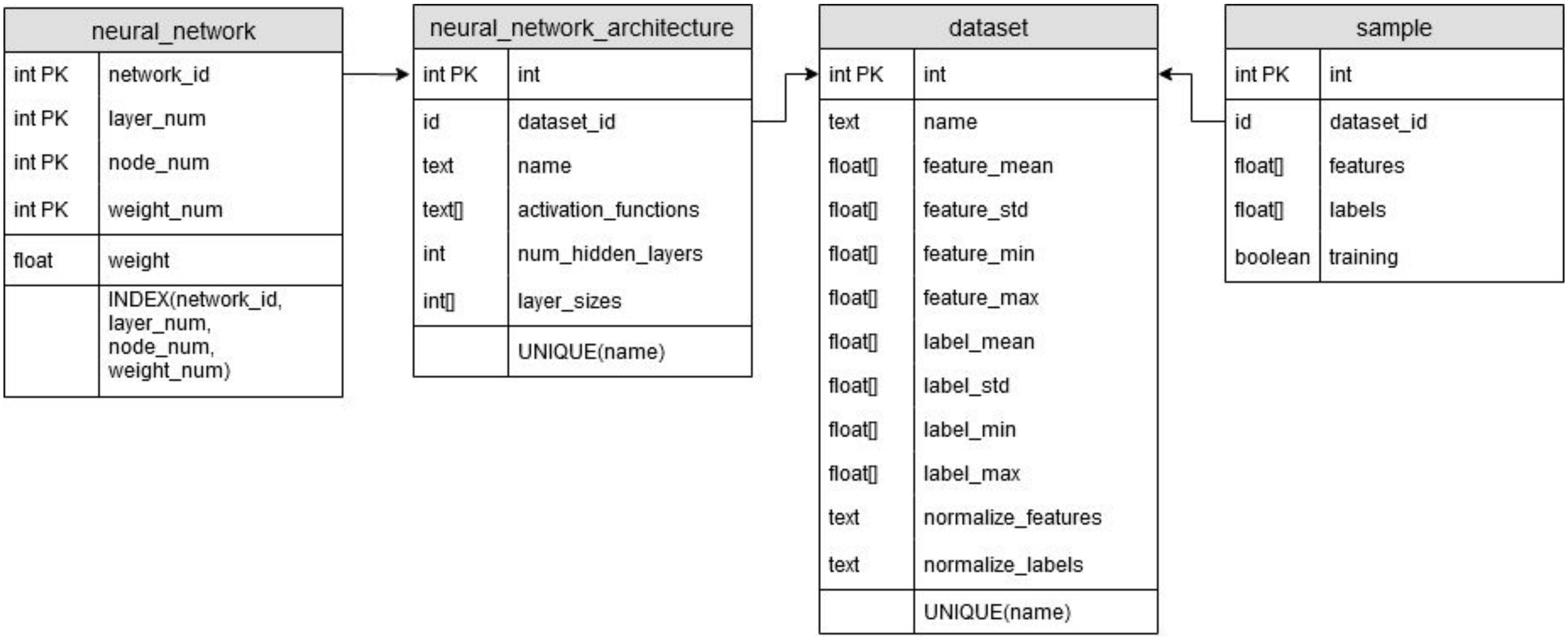
- Extract columns from existing tables as datasets to use as training samples separated into features and label(s), where values are abstracted to be referenced by index and not column name.
- Train models with a single procedure that can use any dataset sample and model size.
- Create network models 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

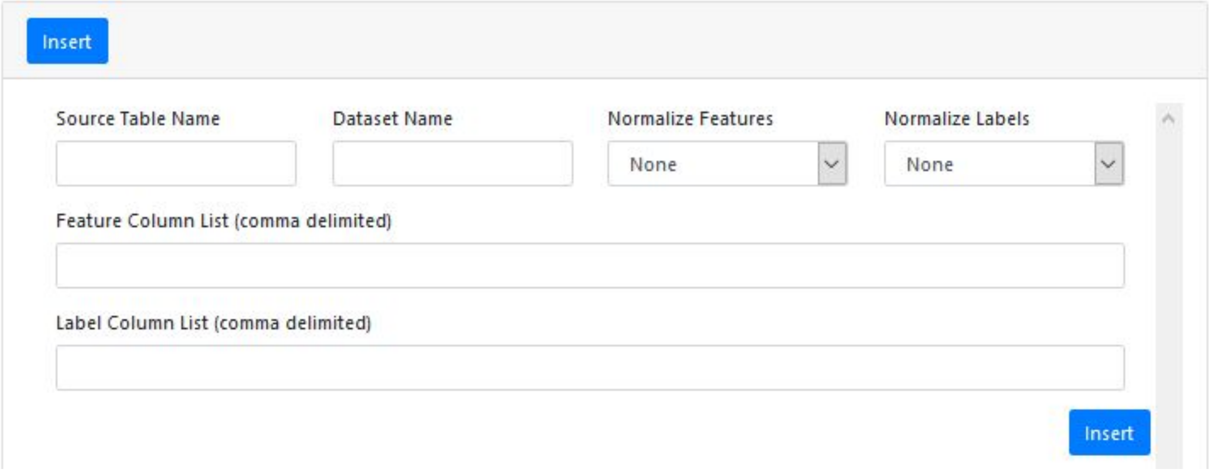


Database


neural_network
int PK network_id
int PK layer_num
int PK node_num
int PK weight_num
float weight
INDEX(network_id, layer_num, node_num, weight_num)
neural_network_architecture
int PK int
id dataset_id
text name
text[] activation_functions
int num_hidden_layers
int[] layer_sizes
UNIQUE(name)
dataset
int PK int
text name
float[] feature_mean
float[] feature_std
float[] feature_min
float[] feature_max
float[] label_mean
float[] label_std
float[] label_min
float[] label_max
text normalize_features
text normalize_labels
UNIQUE(name)
sample
int PK int
id dataset_id
float[] features
float[] labels
boolean training

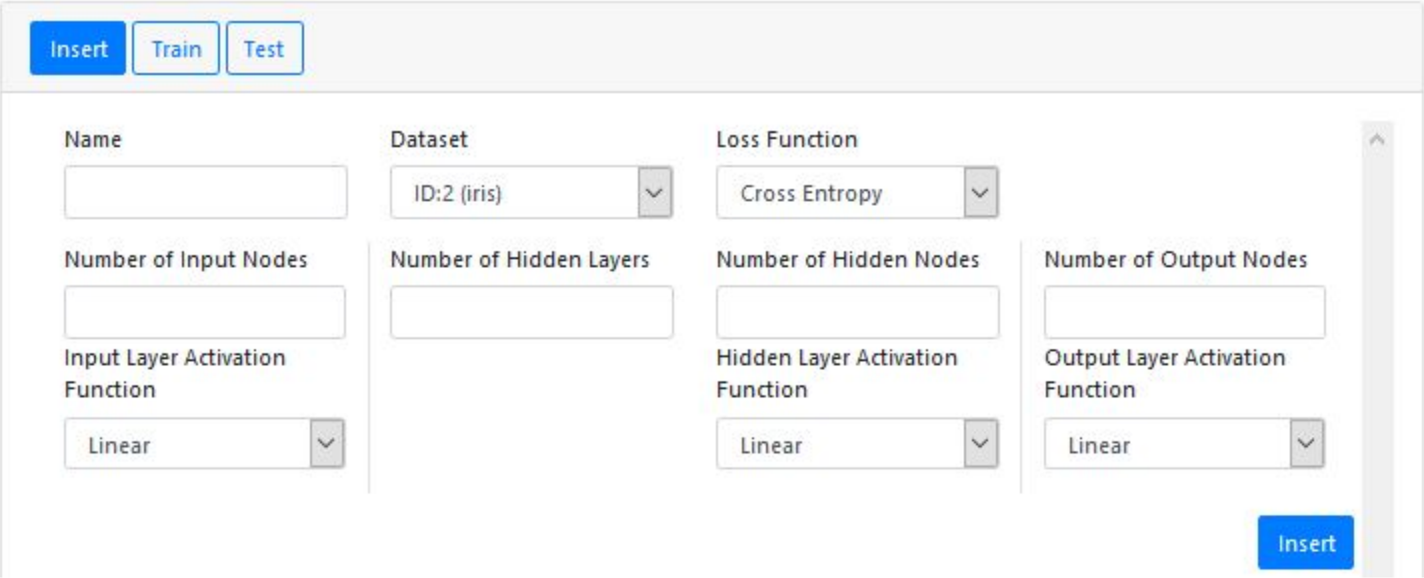
Data Import

- 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.



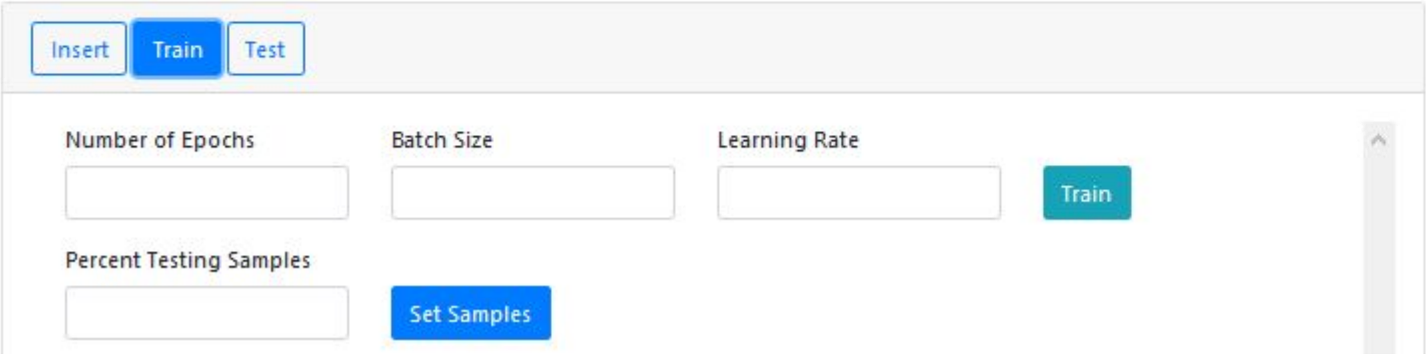
Model Creation

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



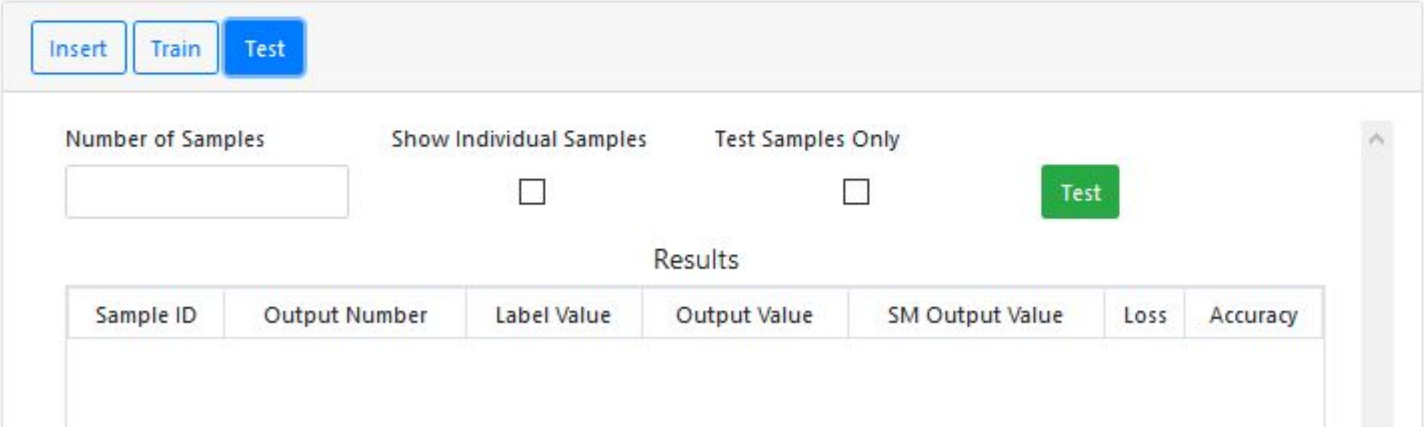
Model Training

- 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).



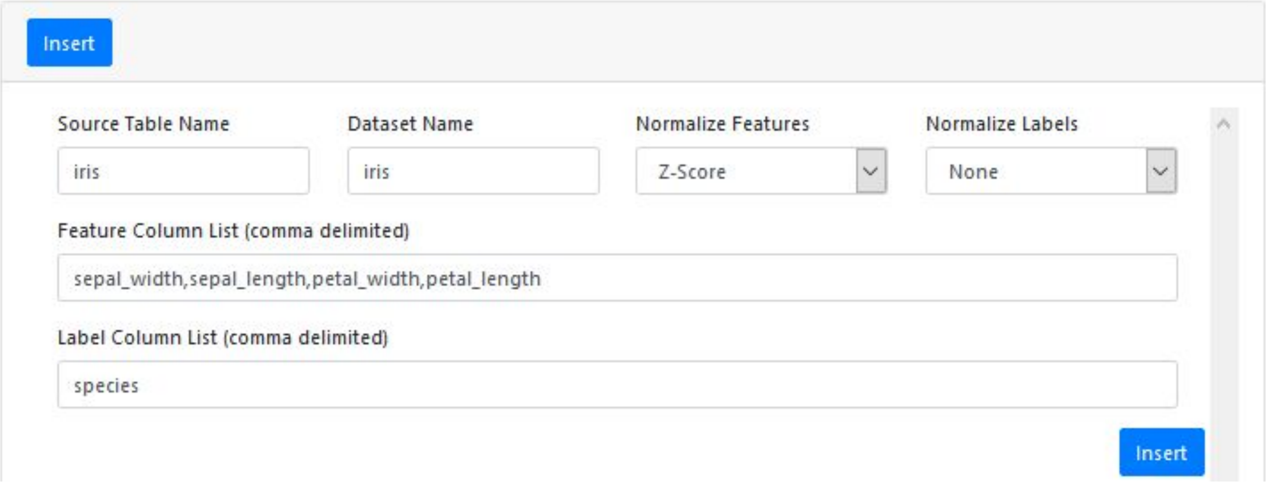
Model Testing

- 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.

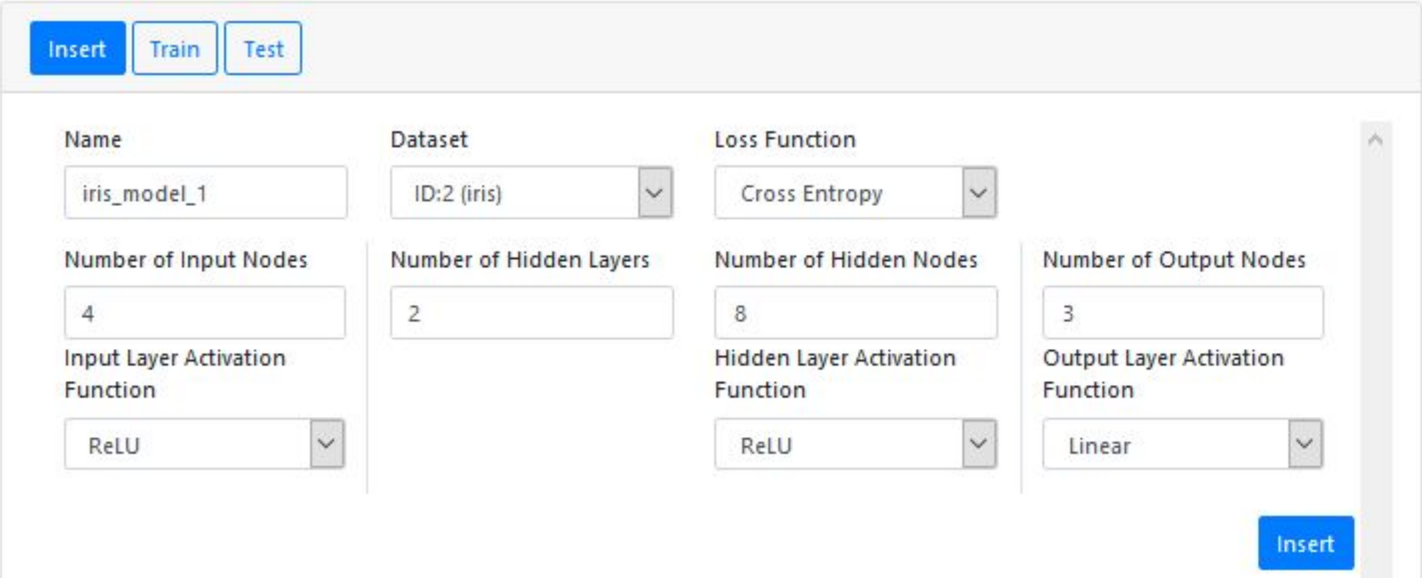


Example Usage

iris	
sepal_width	float
sepal_length	float
petal_width	float
petal_length	float
species	text



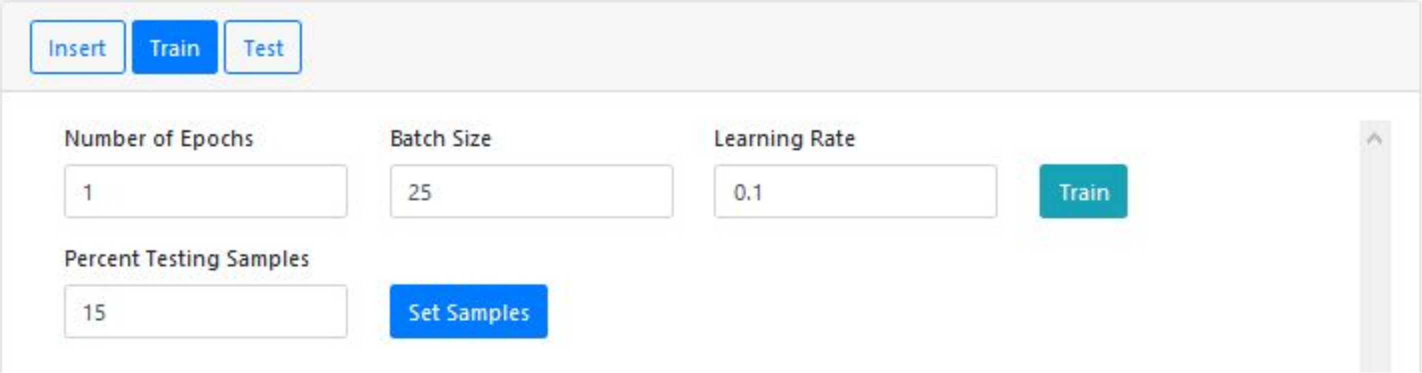
Example Usage (cont.)

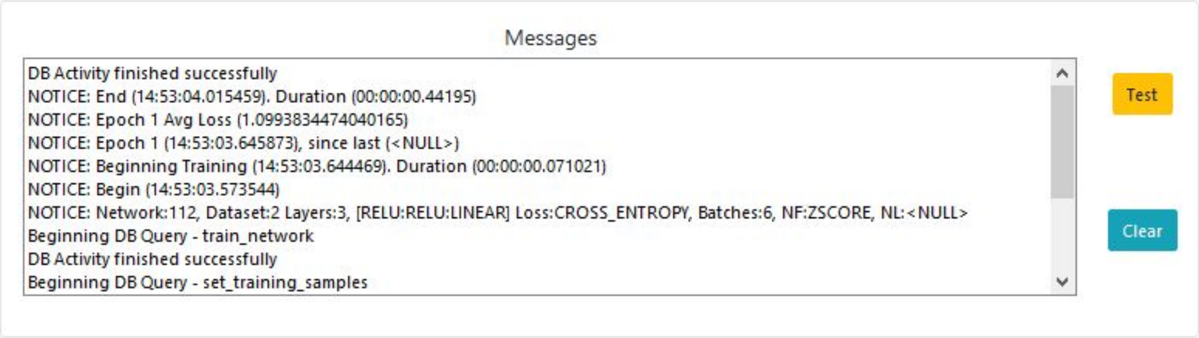


Networks

	ID	Name	Dataset	Layers	Activation Fns	Loss Fn
◉	112	iris_model_1	iris	IN:4, HDN:2x8, OUT:3	RELU,RELU,LINEAR	CROSS_ENTROPY

Example Usage (cont.)





Example Usage (cont.)

Insert Train Test

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

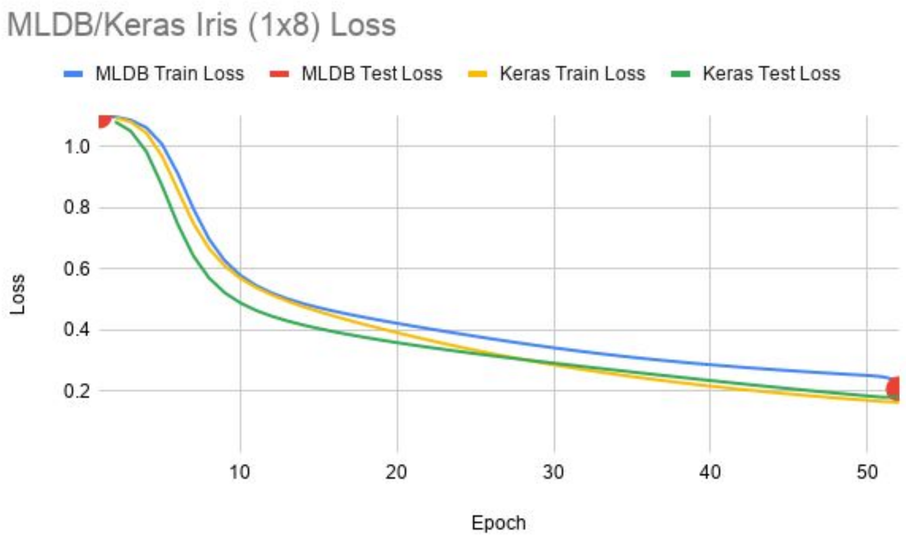
- Tested different models using the datasets Iris, Wine Cultivar, and Airfoil.
- Recorded accuracy before and after training, loss during training.
- Compared to a Keras models with equivalent architecture and initialization.
- Results averaged across 5 runs.

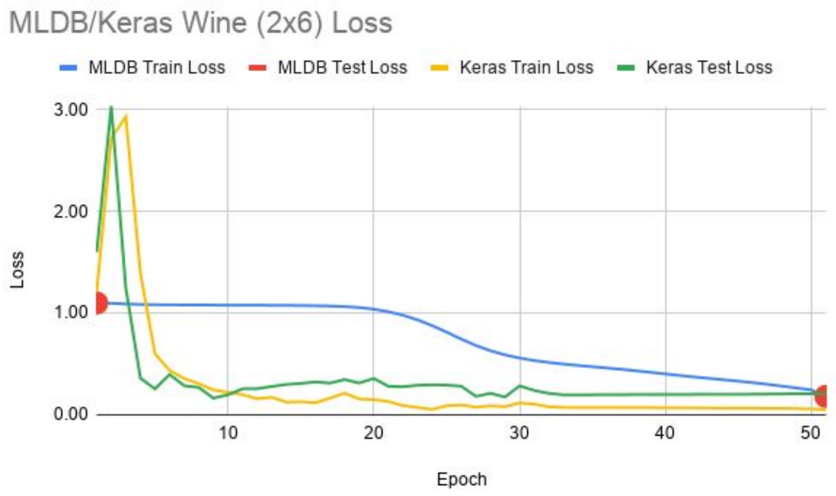
Dataset	Features	Samples	Type	Tested Models
Iris	4	150	Classification	(1x8 tanh), (2x6 tanh), (2x12 linear)
Wine Cultivar	13	178	Classification	(1x8 tanh), (2x6 tanh), (2x12 linear)
Airfoil	5	1503	Regression	(1x8 tanh), (1x16 tanh), (2x6 tanh)

* 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.





Results (cont.)

- Final accuracy comparable to Keras.

Iris (1x8)

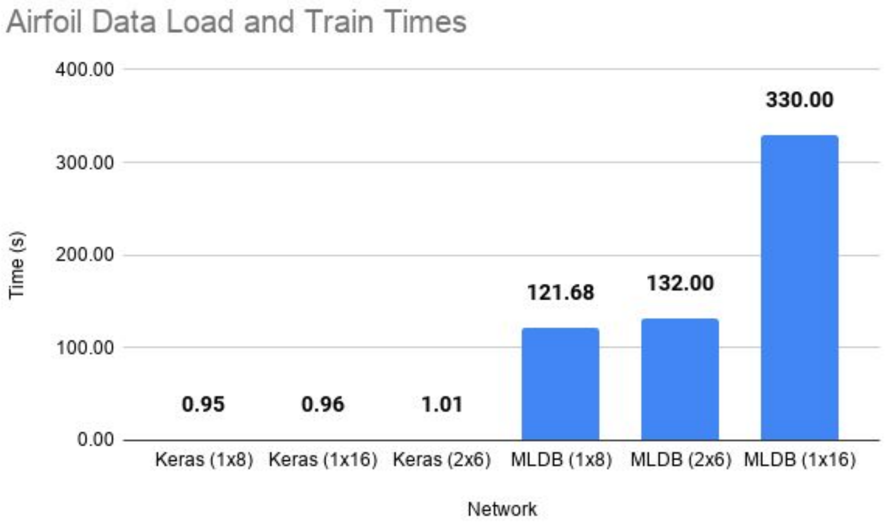
Epoch	MLDB Train Acc.	Keras Train Acc.	MLDB Test Acc.	Keras Test Acc.
0	22.34	8.66	22.73	8.69
25	88.91	95.28	99.09	93.91

Airfoil (1x16)

Epoch	MLDB Train Acc.	Keras Train Acc.	MLDB Test Acc.	Keras Test Acc.
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.



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 and 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/mldb_report.pdf