Inference & Evaluation Engine for Feed-Forward Neural Networks

CS-IS-2010-1 Final Project

Santripta Sharma

May 15, 2024

Introduction

- Through the course of the semester, I have developed a command-line tool to run inference on (or evaluate) an arbitrary feed-forward network classifier.
- The entire process took me from the basics, performing command-line I/O, and using the haskell tool stack, all the way to writing binary data parsers, using monads, and implementing test suites with HSpec.

Introduction

- Through the course of the semester, I have developed a command-line tool to run inference on (or evaluate) an arbitrary feed-forward network classifier.
- The entire process took me from the basics, performing command-line I/O, and using the haskell tool stack, all the way to writing binary data parsers, using monads, and implementing test suites with HSpec.
- Besides these learning outcomes, the tool has been designed & implemented to maximise optimality, generalizability, and usability.

Motivation

- Machine Learning has left no domain/sector untouched.

Motivation

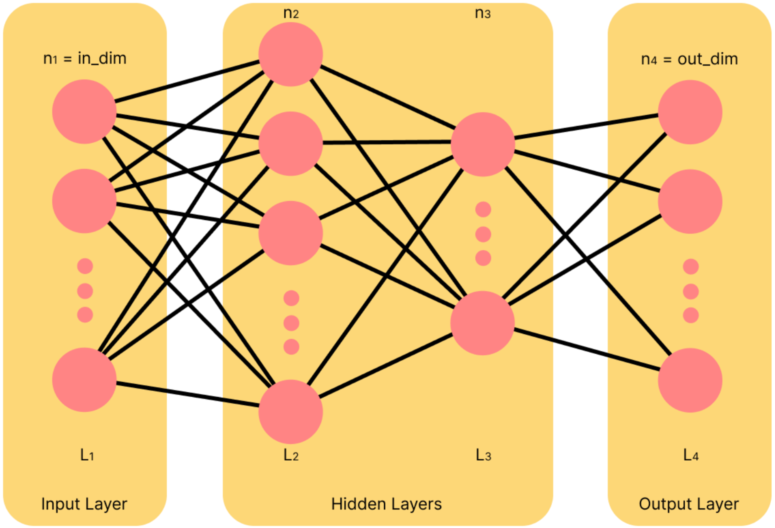
- Machine Learning has left no domain/sector untouched.
- Problem: A large chunk of the users of ML models are not familiar with how they work or how to use them.
- Widespread black-boxing: development/training of the model is outsourced to someone with this technical knowledge
- Besides security risks, also leaves the non-savvy client at the vendor's mercy

Motivation

- Machine Learning has left no domain/sector untouched.
- Problem: A large chunk of the users of ML models are not familiar with how they work or how to use them.
- Widespread black-boxing: development/training of the model is outsourced to someone with this technical knowledge
- Besides security risks, also leaves the non-savvy client at the vendor's mercy
- Requirement of a trusted solution to self-validate vendor's claims about model accuracy

Background & Literature Survey

Feed-Forward Neural Networks



Feed-Forward Neural Networks

Propagation formula (for $2 \leq i \leq n_{layers}$):

$$X_i = \text{Activation}(W_i X_{i-1} + B_i)$$

Lemma

Given the weights matrix $W_i : a \times b$ at layer L_i, we have that $n_{i-1} = b$.

Lemma

Given the weights matrix $W_i : a \times b$ at layer L_i, we have that $n_i = a$.

Lemma

Given the weights matrix $W_i : a \times b$ at layer L_i, the biases vector B_i has the shape $B_i : a \times 1$.

Existing Neural Network Libraries

- We want to avoid reinventing the wheel.
- Several neural network libraries exist for haskell: neural, grenade, hnn, neural-network-hmatrix
- All share the same issue: too general
- Focused on training + inference, trade simplicity for generality

Existing Neural Network Libraries

- We want to avoid reinventing the wheel.
- Several neural network libraries exist for haskell: neural, grenade, hnn, neural-network-hmatrix
- All share the same issue: too general
- Focused on training + inference, trade simplicity for generality
- Time to reinvent the wheel. Where do we begin?

Fast Linear Algebra in Haskell

Library	$n = 10$	$n = 50$	$n = 100$
DLA	2.65us	289.0us	2.24ms
Hmatrix	1.32us	55.8us	292.0us
NumHask	714.0us	63.5ms	593.0ms
Massiv	12.0us	205.0us	1.52ms
Massiv (Parallel)	76.1us	220.0us	866.0us
Matrix	12.6us	1.1ms	8.44ms
Naive C Implementation	51us	323us	4.78ms

- To do fast neural network inference, do fast matrix multiplication and addition
- Many linear algebra libraries: matrix, DLA, massiv, hmatrix, numhask
- The table above shows a benchmark from 2015, for matrix multiplication of size $n \times n \times n$

Fast Linear Algebra in Haskell

Library	$n = 10$	$n = 50$	$n = 100$
DLA	2.65us	289.0us	2.24ms
Hmatrix	1.32us	55.8us	292.0us
NumHask	714.0us	63.5ms	593.0ms
Massiv	12.0us	205.0us	1.52ms
Massiv (Parallel)	76.1us	220.0us	866.0us
Matrix	12.6us	1.1ms	8.44ms
Naive C Implementation	51us	323us	4.78ms

- To do fast neural network inference, do fast matrix multiplication and addition
- Many linear algebra libraries: matrix, DLA, massiv, hmatrix, numhask
- The table above shows a benchmark from 2015, for matrix multiplication of size $n \times n \times n$
- Conclusion: use hmatrix

Problem Definition & Objectives

Problem Statement

Problem Statement

Develop a classification model over the Iris Dataset and store the model. Then write a Haskell code to restore the model, input new data (based on the four features of the iris) and generate a prediction in real time.

Problem Statement

Problem Statement

Develop a classification model over the Iris Dataset and store the model. Then write a Haskell code to restore the model, input new data (based on the four features of the iris) and generate a prediction in real time.

1. Input: The Iris Dataset.
2. Output: Real-time prediction for the class of the flower of a new data row containing the sepal width, petal width, sepal length and petal length.

Problem Statement

Problem Statement

Develop a classification model over the Iris Dataset and store the model. Then write a Haskell code to restore the model, input new data (based on the four features of the iris) and generate a prediction in real time.

1. Input: The Iris Dataset.
2. Output: Real-time prediction for the class of the flower of a new data row containing the sepal width, petal width, sepal length and petal length.
3. Method: You can use Python to train and save a classification model (SVM or NN). However, restoring the model and the real-time prediction of a new data row has to be written only in Haskell.

Objectives

Based on this statement & the motivation, we set our objectives:

Objectives

Develop a system which:

- Can load a trained FFN-based classification model.
- Can make predictions on an unseen/new datapoint belonging to the same dataset.
- Is as performant as possible, since the problem description suggests that the application domain is a real-time system.
- Is simple to use for a layperson.
- Generalises beyond the iris dataset.
- Can evaluate the given model in some capacity

Scope

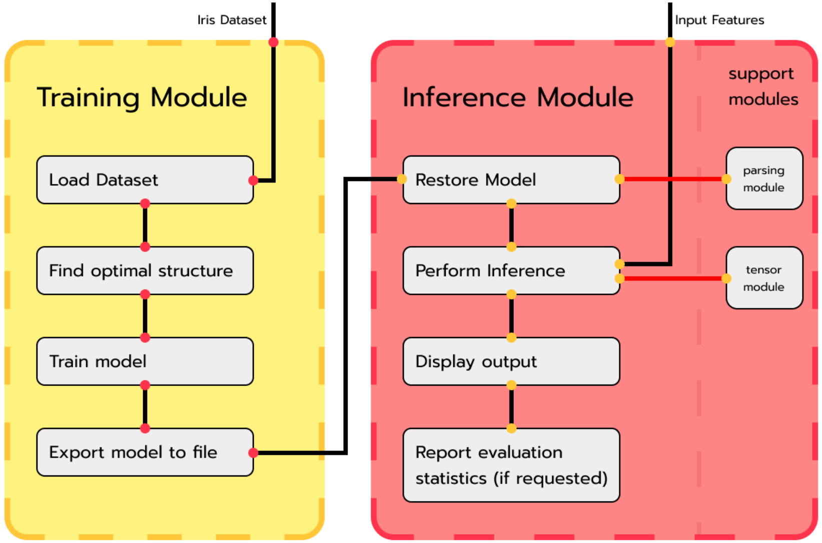
We have developed a system with the following featureset:

1. Parsing capabilities to restore the shape, weights, and biases of **an arbitrary** FFN classifier from a file on disk.
2. Interactive & batched modes of performing inference.
   - **Interactive**: The user enters a single new datapoint and gets predictions for it in real-time. Structured as a Read-Eval-Print-Loop (REPL).
   - **Batched**: The user provides many datapoints in a csv file, and receives predictions for each point.
3. An evaluation mode: an extension of batched inference mode which also takes in expected outputs, and determines model accuracy.

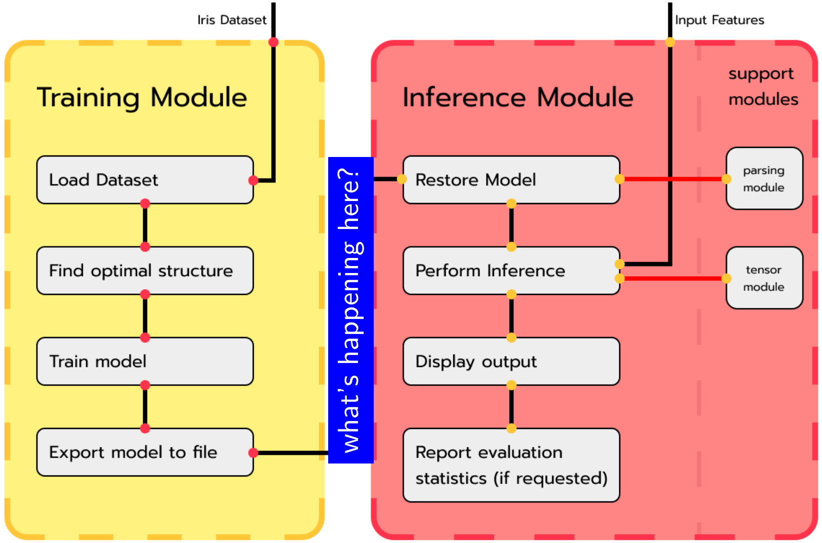
We limit the class of models we allow to using ReLU (at the hidden layers) & Softmax (at the output layer) activations.

System Design & Methodology

High-level view



High-level view



Trained Model Format

We use a custom binary format to store our trained models.
A naive implementation of such a format goes as follows:

```
number of layers
for each non-input layer in the network:
  weights matrix rows
  weights matrix columns
  weights matrix
  biases vector length
  biases vector
```

Since each integer stored is 4-bytes, this requires $4(3l + 1) + D$ bytes to store, where D is the constant size of the data (weights & biases), and l is the number of layers.

Trained Model Format

Using our lemma, we can infer the biases vector length from the weights matrix rows:

```
number of layers
for each non-input layer in the network:
  weights matrix rows
  weights matrix columns
  weights matrix
  biases vector
```

This requires $4(2l + 1) + D$ bytes to store

Trained Model Format

Using our other lemma, we can infer the weights matrix columns from the previous weights matrix rows (i.e. number of neurons in previous layer):

```
number of layers
number of neurons in input layer
for each non-input layer in the network:
  weights matrix rows
  weights matrix
  biases vector
```

This requires $4(l+2)+D$ bytes to store, which scales much better with number of layers, and since we iteratively parse layer after layer, we can memoise the previous layer's neuron count to determine the number of columns in any layer's weights matrix.
We store the number of neurons in the input layer for initialisation.

Training Module

- The training module provides us with a set of models we can use to test our inference module.

Training Module

- The training module provides us with a set of models we can use to test our inference module.
- Using a 5-fold cross-validation, we perform a grid search to find the best 5 network shapes for the architecture mentioned above, as well as the worst shape.

Training Module

- The training module provides us with a set of models we can use to test our inference module.
- Using a 5-fold cross-validation, we perform a grid search to find the best 5 network shapes for the architecture mentioned above, as well as the worst shape.
- We then train these 6 models on a split of the dataset, and save them to disk in the format discussed above. Additionally, we train a large 12-layer model to be used for benchmarking.

Inference Module

- The inference module produces an executable with the following signature:
  `inference-exe <path_to_model_file> [<path_to_batch_csv>] [path_to_ground_truth]`
  If no optional arguments are given, it starts in interactive mode, if the batch csv is provided it starts in batch mode, and if the ground truth csv is provided, it starts in evaluation mode.

Inference Module

- The inference module produces an executable with the following signature:
  `inference-exe <path_to_model_file> [<path_to_batch_csv>] [path_to_ground_truth]`
  If no optional arguments are given, it starts in interactive mode, if the batch csv is provided it starts in batch mode, and if the ground truth csv is provided, it starts in evaluation mode.
- It first restores the model from the given model file, utilising the parsing support module.

Inference Module

- Then, based on which mode it was started in, it collects user input accordingly, and uses the tensor support module to perform the inference. If it was started in interactive or batched mode, it reports the output activations and class labels, otherwise it reports % accuracy and the number of correct predictions.

Work Done

Tooling

Training Module
The training module is written in **python**, using **numpy**, **pandas**, and **sklearn** to perform the data processing & model training/selection.

Inference Module
The inference module is written in **Haskell**. We use the **Haskell Tool Stack** (or just Stack) as our build tool. The following libraries are used:

- **hmatrix**
- **cereal**
- **cassava**

Tooling

Misc/Testing
A mix of bash & python scripts is used to implement the end-to-end tests and certain convenience utilities. Additionally, we use HSpec for unit testing & timeit for benchmarking performance.

Testing Strategy

Due to the large number of disjoint moving parts in the project, the plan is to implement a comprehensive test suite consisting of unit & end-to-end tests. Additionally, we will be profiling our system to ensure it meets the real-time requirements.

Unit tests are implemented using **HSpec**, the testing framework provided by **Stack**.

Unit Testing

First, we note that the training module does not require testing, due to its simplicity and it almost exclusively using sklearn library functions.

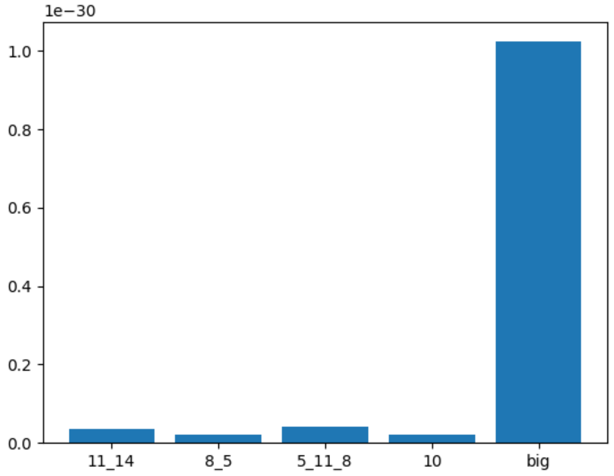
For the inference module, the inference functions are complex enough to warrant unit testing.
As it stands, the batch inference function is tested by the end-to-end test. Therefore, we focus our attention on the single-row inference function.

Unit Testing

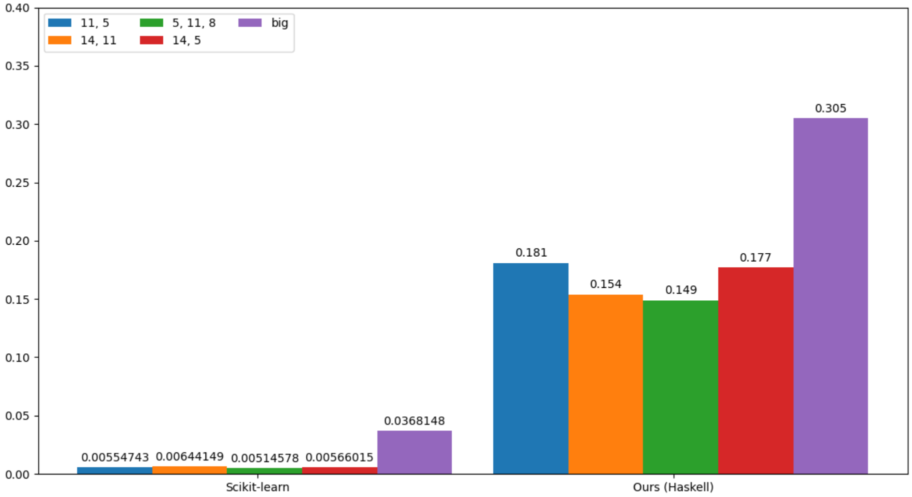
Using HSpec, the default testing framework included with the Haskell stack, we test this function for four cases:

- Empty Network: Should throw an error
- Invalid Input Dimensions: Should throw an error
- Invalid Network Configuration: Should throw an error
- Correct Functioning: Use randomly generated valid inputs to check our function output against the explicit formula.

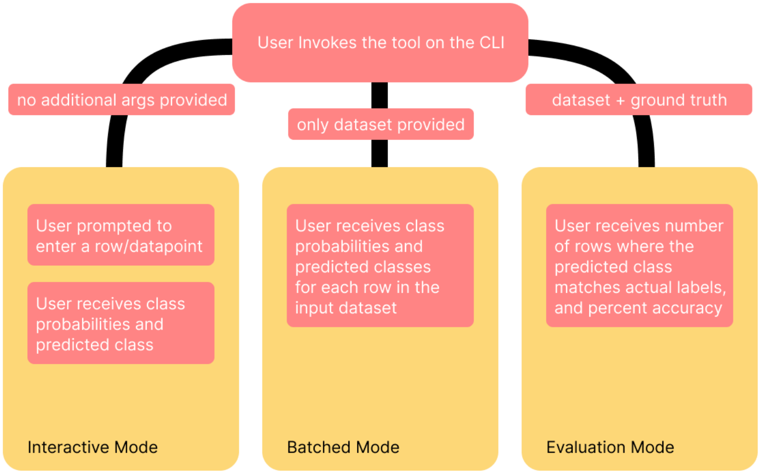
E2E Testing



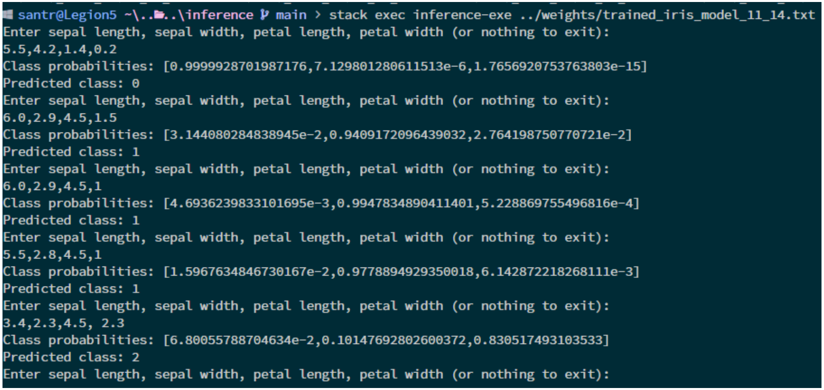
Profiling


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
11, 5
14, 11
5, 11, 8
14, 5
big
0.00554743
0.00644149
0.00514578
0.00566015
0.0368148
0.181
0.154
0.149
0.177
0.305
Scikit-learn
Ours (Haskell)

Functionality/User Flow


User Invokes the tool on the CLI
no additional args provided
only dataset provided
dataset + ground truth
User prompted to enter a row/datapoint
User receives class probabilities and predicted class
Interactive Mode
User receives class probabilities and predicted classes for each row in the input dataset
Batched Mode
User receives number of rows where the predicted class matches actual labels, and percent accuracy
Evaluation Mode

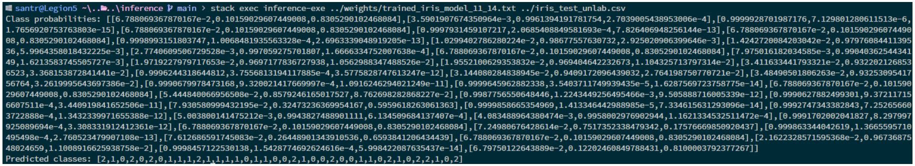
Examples



```
santr@Legion5 ~\..\..\inference  main > stack exec inference-exe ../weights/trained_iris_model_11_14.txt
Enter sepal length, sepal width, petal length, petal width (or nothing to exit):
5.5,4.2,1.4,0.2
Class probabilities: [0.9999928701987176,7.129801280611513e-6,1.7656920753763803e-15]
Predicted class: 0
Enter sepal length, sepal width, petal length, petal width (or nothing to exit):
6.0,2.9,4.5,1.5
Class probabilities: [3.144080284838945e-2,0.9409172096439032,2.764198750770721e-2]
Predicted class: 1
Enter sepal length, sepal width, petal length, petal width (or nothing to exit):
6.0,2.9,4.5,1
Class probabilities: [4.6936239833101695e-3,0.9947834890411401,5.228869755496816e-4]
Predicted class: 1
Enter sepal length, sepal width, petal length, petal width (or nothing to exit):
5.5,2.8,4.5,1
Class probabilities: [1.5967634846730167e-2,0.9778894929350018,6.142872218268111e-3]
Predicted class: 1
Enter sepal length, sepal width, petal length, petal width (or nothing to exit):
3.4,2.3,4.5, 2.3
Class probabilities: [6.800557887046340e-2,0.10147692802600372,0.830517493103533]
Predicted class: 2
Enter sepal length, sepal width, petal length, petal width (or nothing to exit):
```

Figure 1: examples of inference in interactive mode

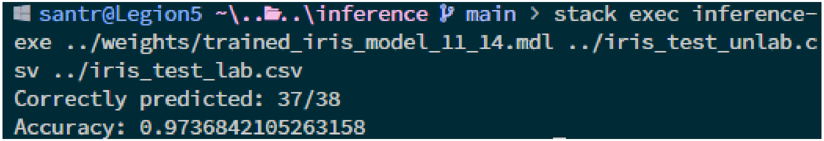
Examples



```
santr@Legion5 ~\.. ..\inference main › stack exec inference-exe ../weights/trained_iris_model_11_14.txt ../iris_test_unlab.csv
Class probabilities: [[6.788069367870167e-2,0.10159029607449008,0.8305290102468084],[3.5901907674350964e-3,0.9961394191781754,2.7039005438953006e-4],[0.9999928701987176,7.129801280611513e-6,
1.7656920753763803e-15],[6.788069367870167e-2,0.10159029607449008,0.8305290102468084],[0.9997931459107217,2.0685408849581693e-4,7.826406948256144e-13],[6.788069367870167e-2,0.101590296074490
08,0.8305290102468084],[0.9998993151803747,1.0068481935563328e-4,2.6963339048919205e-13],[1.0299402786280224e-2,0.98677557630732,2.925020906399646e-3],[1.4242720084203042e-2,0.97976084411395
36,5.9964358018432225e-3],[2.7740609506729528e-3,0.997059275701807,1.6666334752007638e-4],[6.788069367870167e-2,0.10159029607449008,0.8305290102468084],[7.975016182034585e-3,0.99040362544341
49,1.6213583745505727e-3],[1.9719227979717653e-2,0.9697177836727938,1.0562988347488526e-2],[1.9552100629353832e-2,0.969404642232673,1.104325713797314e-2],[3.411633441793321e-2,0.932202126853
6523,3.368153872841441e-2],[0.9996244318644812,3.7556813194117885e-4,3.5775828747613247e-12],[3.144080284838945e-2,0.9409172096439032,2.764198750770721e-2],[3.48490501806263e-2,0.93253095417
56764,3.2619995643697386e-2],[0.9990679978473168,9.320021417669997e-4,1.0916246294021249e-11],[0.9999645962882338,3.5403711749939435e-5,1.6287569723758775e-14],[6.788069367870167e-2,0.101590
29607449008,0.8305290102468084],[5.444840066956508e-2,0.8579246165017527,8.762698282868227e-2],[0.9987756550648446,1.2243449256495466e-3,9.505888716005339e-12],[0.9990627882499301,9.37211715
6607511e-4,3.440919841652506e-11],[7.930580999432195e-2,0.32473236369954167,0.5959618263061363],[0.9999858665354969,1.413346442988985e-5,7.334615631293096e-14],[0.9992747343382843,7.25265660
3722888e-4,1.3432339971655388e-12],[5.003800141475212e-3,0.9943827488901111,6.134509684137407e-4],[4.083488964380474e-3,0.9958002976902944,1.1621334532511472e-4],[0.9991702002041827,8.297997
925089694e-4,3.308331912412361e-12],[6.788069367870167e-2,0.10159029607449008,0.8305290102468084],[7.24980676428614e-2,0.7517352338479342,0.17576669850920437],[0.999863344042619,1.3665595710
495498e-4,2.760523479907108e-13],[7.612686591745083e-2,0.26448901343910536,0.6593841206434439],[6.788069367870167e-2,0.10159029607449008,0.8305290102468084],[2.1622328571595368e-2,0.96736875
48024659,1.1008916625938758e-2],[0.9998457122530138,1.5428774692624616e-4,5.998422087635437e-14],[6.79750122643889e-2,0.12202460849788431,0.8100003792377267]]
Predicted classes: [2,1,0,2,0,2,0,1,1,1,2,1,1,1,1,0,1,1,0,0,2,1,0,0,2,0,0,1,1,0,2,1,0,2,2,1,0,2]
```

Figure 2: example of inference in batched mode

Examples



```
santr@Legion5 ~\..\..\inference  main > stack exec inference-exe ../weights/trained_iris_model_11_14.mdl ../iris_test_unlab.csv ../iris_test_lab.csv
Correctly predicted: 37/38
Accuracy: 0.9736842105263158
```

Figure 3: example of evaluation mode

Challenges & Pitfalls

While designing and implementing the project, I ran into several pitfalls, including but not limited to:

- **Migrating Libraries from Midterm**
- **Configuring LAPack & Blas**
- **Initial Benchmarks**

Migrating Libraries from Midterm

- During the final stage of the project, we made several migrations to different libraries. Some planned, some unplanned.
- Even the planned migration from parsec to attoparsec did not go to plan, attoparsec doesn't support parsing IEEE754 numbers correctly
- Later, we found cereal, and despite migration to it being smooth, a lot of time was lost in the process.

Migrating Libraries from Midterm

- During the final stage of the project, we made several migrations to different libraries. Some planned, some unplanned.
- Even the planned migration from parsec to attoparsec did not go to plan, attoparsec doesn't support parsing IEEE754 numbers correctly
- Later, we found cereal, and despite migration to it being smooth, a lot of time was lost in the process.
- A less smooth transition was seen after completing our literature survey and finding hmatrix.
- Required a rewrite of our entire inference codebase, to fit hmatrix's interface.

Configuring LAPack & Blas

- hmatrix owes most of its speed gains to it essentially being a wrapper for the industry standard numerical linear algebra packages, LAPack & Blas.
- Once the inference module had been rewritten for hmatrix, it no longer compiled, leading us down a deep rabbithole of dependency management.
- Only old and unreliable answers available online.
- Eventually, the Haskell Tool Stack documentation revealed that on windows, it maintains its own MSYS2 environment for this purpose.
- Big time sink.

Initial Benchmarks

- In our initial benchmarks, even after the migration to hmatrix, we observed less than ideal results.
- Problem narrowed down to one line of code:

```
col = (toColumns mat) !! colIdx
```

Initial Benchmarks

- In our initial benchmarks, even after the migration to hmatrix, we observed less than ideal results.
- Problem narrowed down to one line of code:
  `col = (toColumns mat) !! colIdx`
- Replacing this with `col = flatten $ mat ? [colIdx - 1]` immediately led to a 4x speedup.
- The pitfalls of interoperation.

Discussion & Conclusion

Limitations

- The command-line interface, while designed for usability, still may serve as a barrier for the most technically-challenged individuals
- Requires the weights to be in a particular format
- During interactive inference, invalid inputs are not handled gracefully
- Class labels aren't currently considered, which hinders readability
- The system's performance, while optimised, fails to compete with sklearn's.

Conclusions

We have built a successful baseline inference package using haskell, though it is clear that it is far from complete. Besides lacking many configuration options and flexibility, it also fails to achieve the performance one would expect from a more standard package implemented in python with something like sklearn. Nonetheless, we have developed an accurate and acceptably performant system for inferencing feed-forward network classifiers.

Conclusions

Further, we have expanded the scope of the project from the intial problem statement to provide a more general solution than was expected. In doing so, we have also provided a partial solution to the black-boxing problem discussed in our motivation.

Finally, we were able to propose a specialised format for serializing trained feed-forward classifiers, which is space effective and utilises our understanding of the network architecture to achieve this optimisation.

Extensions & Future Work

- Infer the format of the weights file based on a few well-known formats (.pkl, .pth, .safetensors), instead of only working for our custom format. This requires more than just looking at the extension, as we have to further infer where the weights and biases of each layer lie based on the keys in the file.

Extensions & Future Work

- Infer the format of the weights file based on a few well-known formats (.pkl, .pth, .safetensors), instead of only working for our custom format. This requires more than just looking at the extension, as we have to further infer where the weights and biases of each layer lie based on the keys in the file.
- The inference performance should be inspected more deeply and optimised accordingly.

Extensions & Future Work

- Infer the format of the weights file based on a few well-known formats (.pkl, .pth, .safetensors), instead of only working for our custom format. This requires more than just looking at the extension, as we have to further infer where the weights and biases of each layer lie based on the keys in the file.
- The inference performance should be inspected more deeply and optimised accordingly.
- Currently, we have not performed any profiling on the parser, which is a core component of the entire system.

Extensions & Future Work

- Infer the format of the weights file based on a few well-known formats (.pkl, .pth, .safetensors), instead of only working for our custom format. This requires more than just looking at the extension, as we have to further infer where the weights and biases of each layer lie based on the keys in the file.
- The inference performance should be inspected more deeply and optimised accordingly.
- Currently, we have not performed any profiling on the parser, which is a core component of the entire system.
- A graphical user interface could greatly enhance the program's usability.

Extensions & Future Work

- Infer the format of the weights file based on a few well-known formats (.pkl, .pth, .safetensors), instead of only working for our custom format. This requires more than just looking at the extension, as we have to further infer where the weights and biases of each layer lie based on the keys in the file.
- The inference performance should be inspected more deeply and optimised accordingly.
- Currently, we have not performed any profiling on the parser, which is a core component of the entire system.
- A graphical user interface could greatly enhance the program's usability.
- Model input space could be extended to accept more architectures, such as support-vector machines or decision trees.

References

1. Haskell Linear Algebra Libraries Benchmark
2. Hoogle: Haskell Documentation Search Engine
3. Haskell Tool Stack Documentation
4. Hackage: Haskell Package Registry
5. Haskell Wiki
6. Real World Haskell: An Introductory Book on Haskell