CS-IS-2010-1: Midsemester Project Report

Santripta Sharma

March 31, 2024

1 Problem

This project is concerned with building a tool for running inference over a given feed-forward neural network. In particular, we are interested in inferencing a network trained on the Iris Dataset, where the goal is to classify Iris flowers into one of three species — setosa, verticolor, or virginica — based on the following four features:

1. Sepal length
2. Sepal width
3. Petal length
4. Petal width

Since we are only concerned with running the inference for a new row for this dataset, we train a classification model separately (for testing purposes) before runtime. At runtime, we restore the structure, weights, and biases of the model, and push the given feature vector through the model in order to arrive at the inferred output.

2 Specification

Formally, we are creating a tool which meets the following specification:

Inputs:

- File containing saved model structure, weights, and biases (in some format)
- Batch of feature vectors to run the inference for (stdin)

Outputs: For every vector in the input batch,

- Most likely class for this observation (string)
- Activation values associated with each class at the output layer (vector)

2.1 Project Architecture

The project can broadly be divided into two modules, the training module & the inference module. These operate independently from each other, with the only interface between them being the inference module taking the training module's output (model structure, weights, biases) as its input.

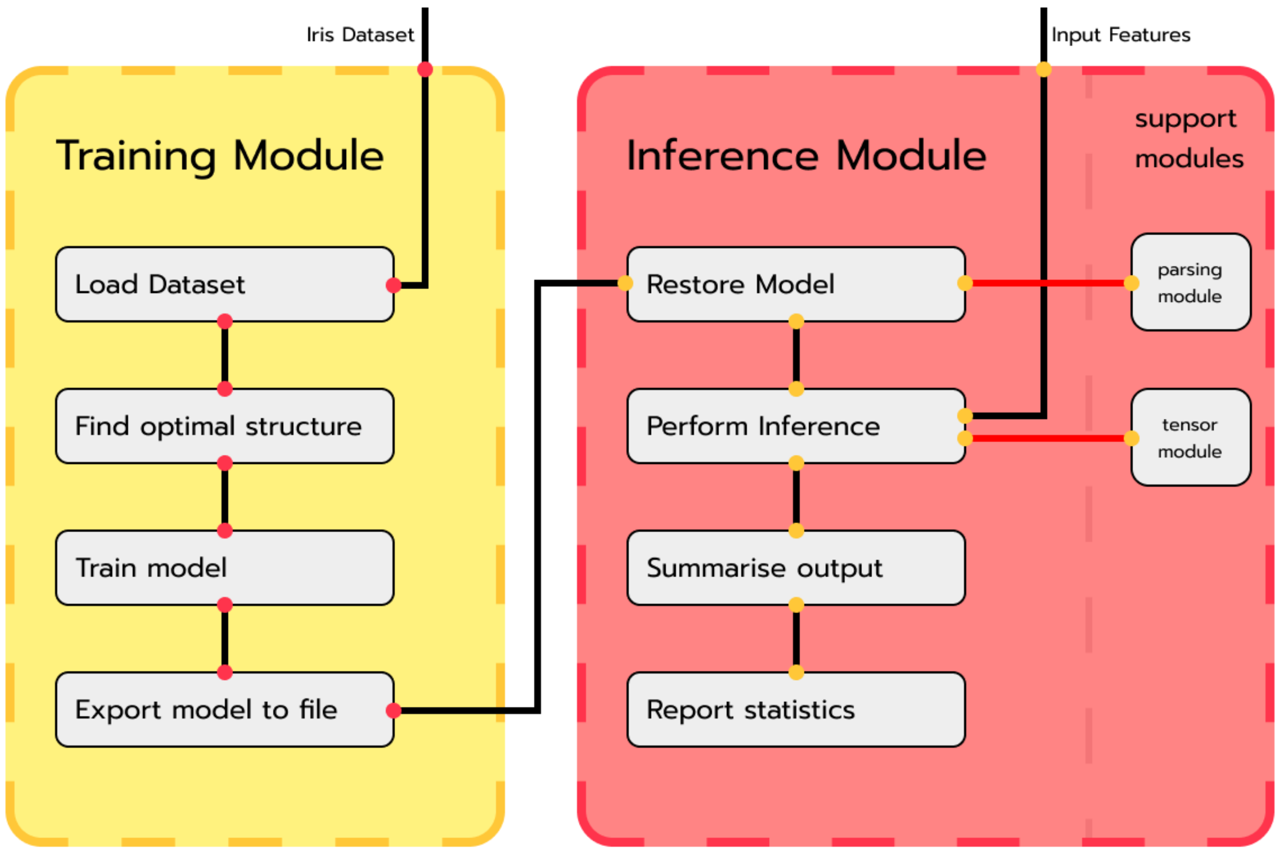

Figure 1: a pictorial representation of the high-level architecture

Then, due to their decoupling, we can individually dissect each of the modules.

2.1.1 Training Module

The purpose of the training module is to provide us with a set of models we can use to test our inference module. As such, we don't strictly require a high performance model. However, we still perform some basic hyperparameter selection after loading the dataset in using a 5-fold cross validation approach to determine the shape of the network (number & size of hidden layers).

The model architecture used is a simple feedforward network. Using a grid search, we find the top 5 performing models, and also one poorly performing model, out of a generated set of possible shapes for the network.

Finally, we write the networks' shapes, weights, and biases into a text file, which can then be used by the inference module to restore the model. In later versions of the project, the intention is to transition this into a binary file format, for higher time & space efficiency at the restoration stage.

2.1.2 Inference Module

The inference module constitutes the bulk of the codebase. Its first task is to restore the model from the file exported by the training module. This utilises the parsing support module, which is simply the inverse operation of the export performed as the training module's final step.

The model shape, and its parameters are loaded into a data structure which takes the form of a linked list of layers, where each layer stores its input shape, neuron count, and the weights & biases matrices.

Based on the user's input feature vectors, the inference is performed using simple matrix operations, which are supported by the tensor module.

2.2 Tooling

2.2.1 Training Module

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

2.2.2 Inference Module

Ok

2.2.3 Testing

Oker

2.3 Test Plan

2.3.1 Unit Tests

Real

2.3.2 Integration Tests

Realer

2.3.3 End-to-End Tests

Realest

3 Prototype Details

Greatest Prototype of all Time

4 Plan for Completion

Greatest Completion of all Time