# Iris Challenge

Kyle A Williams

January 2023

## 1 Solution Overview

The entry point to the solution to the Iris Challenge is Iris_Numpy.py. The file contains numpy implementations of a neural-net(NN), autograd, and gradient descent, all of which roughly follow the pyTorch api/style. Iris_Numpy.py contains the following classes:

- NumpyModel: From-scratch, linear feed-forward artificial neural network. 1 hidden layer 256 units wide, 1 output layer. ReLu activation. 3 outputs for multi-class classification of iris type.
- Graph: A crude approximation to pytorch/tensorflow computation graphs required for autograd. Stores NN parameters, layer activations, and NN inputs during the forward-pass, stores parameter gradients during the backward pass.
- CrossEntropyLoss: Computes the cross entropy loss between prediction logits and target classes. Prediction logits are passed through softmax to calculate class probabilities. The loss is the mean negative log of the probability assigned to the target class.

  This class contains a backward() method, which populates the Graph with parameter gradients. First, the partial derivative of the loss wrt the outputs is calculated, then the partial of outputs wrt the params in the output layer, and the partial of outputs wrt inputs from the previous layer. Likewise for the ReLu activation and the first layer in the network. The chain rule is used to get the partial of loss wrt each paramater in the network.
- SGD: Stochastic gradient descent with momentum. Gradients are over the entire (small) training set so not very stochastic, technically. The class also implements weight decay.
- IrisChallenge: Main class which loads data, initializes network, loss, optimizer and tensorboard writer. Calls an infinite training loop of forward pass, backward pass, optimizer step, with periodic logging of train/test loss and accuracy to tensorboard.

# 2 Results

## 2.1 Tensorboard

Tensorboard will push live training stats to http://localhost:6006/ if the command 'tensorboard –logir=runs' is called from the console (assuming packages are installed and you're in the project directory). Figures 1 and 2 show tensorboard accuracy and loss plots respectively.


Figure 1: Accuracy over training epochs for training and test set.

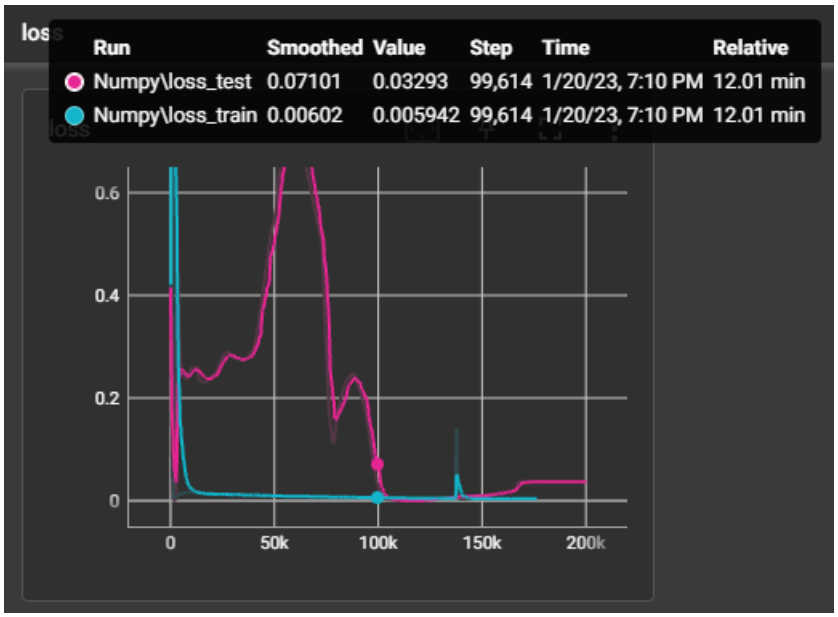


Figure 2: Loss over training epochs for training and test set.

### 2.2 Analysis

The loss plots do not show the text book trends where training loss tends to zero, while test loss initially decreases, flat lines, and then increases as over fitting begins to occur. The accuracy plots are not exceptionally clean either. This may be due to the relatively small data set of 150 samples. A larger data set may produce more repeatable text book results with less variation between runs.

Loss however does decrease in general while accuracy increases, signifying that learning is occurring, with final accuracies significantly better than random probability at .30. Near 100k epochs may be a good time to halt training, as test loss is minimal and test accuracy is locally maximal.

## 3 Next Steps

- K-fold Cross Validation: This may create smoother more interpretable loss and accuracy curves.
- Adam Optimization: Implementing this more advanced form of gradient decent would likely improve performance.
- Hyper Parameter Tuning: tune for optimal hyper params using simple grid search or Bayesian optimization ala optuna.

- Data: collect more data if possible, visualize the data, consider data augmentation, try to understand the failure cases.

# 4 Other Files

Iris_pyTorch.py contains a nearly identical pyTorch based solution to the Iris Challenge. It was useful for debugging. It contains a semi from scratch torch nn Parameter() based feed forward neural net and a traditional pyTorch neural net. The parameter based approach can be useful for meta learning, reinforcement learning, or custom network implementation.

Utils.py contains static methods useful to both the numpy and pyTorch implementations. It contains the following methods:

- load_data(): Pulls data from the .txt file, normalizes it, and converts the strings labels to one-hot encodings.
- split_data(): Shuffles the data and splits it into training and test sets.
- accuracy(): Computes how often the highest probability class prediction matches the target. Expressed from 0 to 1.
- log_metrics(): Computes the train/test set loss and accuracy and writes them to tensorboard for live viewing during training.

# 5 Conda Instructions

These are not rigourous set up instructions, but the following three commands should be sufficient to run the code in a fresh conda environment on a windows machine:

- conda install pytorch cpuonly -c pytorch
- conda install numpy
- conda install -c conda-forge tensorboard