Journal of Machine Learning Research 20 (2019) 1-10 Submitted 11/19; Published 12/19

Machine Learning Project 4: Design Document

Brandon Sladek BRANDONSLADEK@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Jared Thompson J.A.THOMPSON22@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Kyle Hagerman HAGERMANKYLE96@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Ryan Hansen RYANHANSEN2222@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

1. Introduction

Our goal for this project is to compare the performance of four neural network training algorithms: backpropagation, a genetic algorithm, differential evolution, and particle swarm optimization. We will compare the performance on two metrics - correctness and convergence time. More specifically, performance will be compared through loss function analysis, and their run times will be compared directly. The following document outlines how we will implement, train, test, tune, and compare the networks using different training techniques.

2. Design Overview

2.1 Class Descriptions

The following subsections provide an overview of the major classes that will be implemented.

- **ExperimentRunner** The core of our design is based upon a main driver class, ExperimentRunner. This class will instantiate multiple neural network instances using the NeuralNetwork class from Project 3. The ExperimentRunner will also use the DataAPI, Preprocessor, and CrossValidator classes to call their respective methods. See Figure 2 at the end of the document for the associated UML diagram.

- **Preprocessor** The Preprocessor class will call a general method that analyzes each raw dataframe and handles replacing or imputing missing values. We also normalize all

data at this stage and handle whether a data set is to be used for discrete classification or continuous regression.

- **CrossValidator** The CrossValidator class will implement a 10-fold cross validation dividing each preprocessed dataframe into ten subsets. In each iteration of a given experiment, nine subsets will make up the training set and the tenth will be reserved as the test set. The test set will rotate each iteration so that each subset of the data will be used as a test set throughout the execution of the full experiment for a given set of parameters. The indices of the entire dataframe are shuffled to randomly sample our data set when partitioning the data into the 10 subsets.

- **ParameterTuner** The ParameterTuner class will simply be used as a single source of parameters for each of the algorithms, fetched by calling the get_parameters() method and specifying the name of the algorithm for which we want the corresponding list of parameters to use in our experiment. It's a one-stop shop for any global definitions we need to make for the entire experiment.

- **GeneticAlgorithm** The GeneticAlgorithm class stores a list of neural networks in a list as its population. It has helper methods that govern updating to future generations, as well as doing mutation and crossover. Its main method, run_genetic_algorithm(), returns the neural network with the best fitness that the algorithm converged to.

- **DifferentialEvolution** The DifferentialEvolution class is similar to the previous algorithms, and stores its population as a list. It has helper methods to control update strategies, mutation, and crossover. Its main method, run_differential_evolution(), returns the neural network with the best fitness that the algorithm converged to (Ilonen et al. (2003)).

- **ParticleSwarmOptimization** Particle Swarm Optimization class is in essence a data structure for particles. It stores particles in a list, implements various helper methods for updating the swarm easily, and finally stores the best known location of the swarm at large. The main method in this algorithm, run_particle_swarm(), will return one neural network that has the best fitness post convergence. The Particle class is the bread and butter of the particle swarm optimization algorithm. Each particle stores its local best known location, its current location, its current velocity, previous velocity, and fitness. Integral update strategies are implemented within the particle class, including velocity update, position update, and fitness update.

- **NeuralNetwork** Finally, the NeuralNetwork class will handle forward propagation and hold any state related variables. Additionally, because backpropagation is already written within the NeuralNetwork class as a method, it will also handle the backpropagation algorithm. There are boolean flags that will be set to build the network specific hidden layers and output functions. From here, each algorithm will have its own class that will implement NeuralNetwork and the training algorithm's specific logic. Most importantly, this class has the critical methods for generating the neural network populations, and deriving the fitness of each given neural network.

- **Results** The Results class will analyze each model's predictions post training using 0/1 Loss. It will print/save the results for later analysis to be presented in the final report.

2.2 Parameter Tuning

- **Number of neurons per layer** We expect more neurons to identify finer details in each pattern. It is possible to go overboard and force the network to try to find patterns that are not there. More neurons increases runtime too.

- **GeneticAlgorithm, DifferentialEvolution** As they are similar algorithms, they both contain the same tuning parameters - *Mutation Rate* and *Crossover Threshold* (Slowik and Bialko (2008)). With both the Genetic Algorithm and Differential Evolution the mutation and crossover rates will be tuned to find optimal values (Gad (2019)). Here, the crossover rate defines the probability that weight values between neural networks will be swapped, while the mutation rate defines the probability that a random weight will be chosen and modified (Ilonen et al. (2003)).

- **Particle Swarm** The main tuning parameters for particle swarm are *w*, *c1*, and *c2*. All three of these parameters are involved in the velocity update strategy. There are three components of the velocity update: inertia, which is scaled by w, social velocity which is scaled by c1, and individual velocity which is scaled by c2. The values for these parameters must be tuned to determine which combination gives the best results for each problem.

2.3 Major Decisions

- **Architecture** By writing a general class to house the neural networks, we can reduce the redundancy of our code. Each algorithm specific implementation will utilize the general NeuralNetwork implementation to train a population of NeuralNetwork instances using a particular training technique. In this way, the main differentiating factor is the training technique used in a given experiment, determined by which training class is used, while the underlying population individuals will all be created with the same NeuralNetwork implementation.

- **Generational Inheritance** We chose to use generational inheritance as the method for passing traits across each population generation for both the GeneticAlgorithm and DifferentialEvolution algorithms. Generational inheritance was chosen over the hundred best method because of the possibility that hundred best might lead to over fitting of the data.

- **Fitness Calculation** We calculate fitness in the following manner. Let x be an input vector to our neural network. We define the ideal output vector y and an actual output to be $nn(x)$, where nn represents the neural network. We care mostly about maximal fitness, so we define the fitness to be $fitness(x) = \frac{1}{\Sigma_{x \in X} ||y - nn(x)||}$. In words, this means we take the inverse of the sum of the norm of all error vectors.

3. Experiment

This section discusses the experimental motivation and general approach for accepting or rejecting our hypotheses for the various data sets and parameter values.

3.1 Approach

We want to test our four neural network learning algorithms for correctness and runtime, across 0,1, and 2 hidden layers. For implementation, we will rely heavily on the Pandas library (McKinney (2010)). For the sake of consistency, we plan to use 10-fold cross validation to ensure our results are not particular to a specific training/test set. Of the four algorithms, only backpropagation acts on a single instance of a neural network. Thus, when possible, the parameters within the other three algorithms will be standardized - each will act on the same number of neural network instances, and both the GeneticAlgorithm and DifferentialEvolution will use the same number of generations.

3.2 Hypothesis

We predict that with less hidden layers each algorithm will perform faster but with less accuracy. We also predict that ParticleSwarmOptimization will be the most accurate with 2 hidden layers. Backpropagation will be the slowest training algorithm and ParticleSwarmOptimization will be the fastest. We predict the GeneticAlgorithm will be just slightly faster than DifferentialEvolution but less accurate because it draws from less genetic material. See the following table for the summary of our accuracy and performance hypotheses.

Algorithm	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
Backpropagation	10, 4	6, 8	2, 12
Differential Evolution	11, 3	7, 7	3, 11
Genetic Algorithm	12, 2	8, 6	4, 10
Particle Swarm Optimization	9, 1	5, 5	1, 9

Figure 1: Table depicting our performance hypotheses. The first number of each cell is the ranking of the algorithm and hidden layers on our predicted accuracy with 1 being the most accurate and 12 being the least accurate. The second number in each cell is our prediction for running time, again with 1 being the fastest and 12 being the slowest.

4. Proposed Analysis

As mentioned in previous sections, our performance analysis is twofold. First, we will gauge a form of correctness through 0-1 loss (accuracy) based on results from classification tests. A higher percentage correct indicates a better performance. If we are using a regression data set, we will gauge the correctness by comparing the mean squared error results for each experiment, in which case a lower mean squared error indicated a better performance. Second, we will compare running times to gauge the relative computational costs of each algorithm and parameter combination. Each of the three algorithms will be compared using populations of the same size, and both the GeneticAlgorithm and DifferentialEvolution will be performed using the same number of generations.

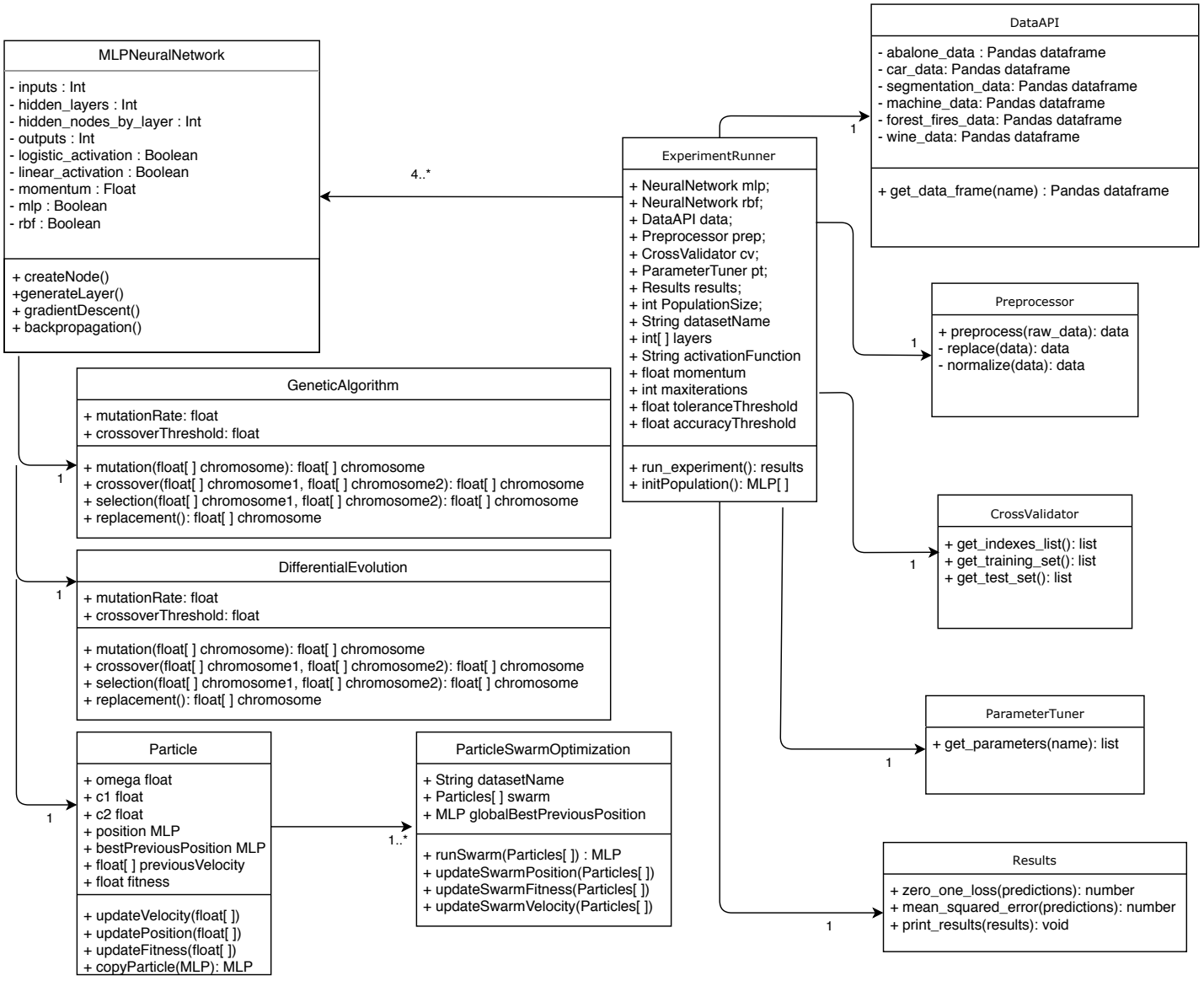

Figure 2: UML implementation of MLP feedforward network equipped with either backpropagation, genetic algorithm, differential evolution, or particle swarm optimization for training. Also included is the Experiment Runner class equipped with the testing suite featured in our previous programs.

References

Ahmed Gad. Artificial neural networks optimization using genetic algorithm with python, Aug 2019. URL https://towardsdatascience.com/artificial-neural-networks-optimization-using-genetic-algorithm-with-python-1fe8ed17733e.

Jarmo Ilonen, Joni-Kristian Kamarainen, and Jouni Lampinen. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17(1): 93–105, 2003. doi: 10.1023/a:1022995128597. URL http://vision.cs.tut.fi/data/publications/NPL2003.pdf.

Wes McKinney. Data structures for statistical computing in python, 2010. URL https://pandas.pydata.org/.

Adam Slowik and Michal Bialko. Training of artificial neural networks using differential evolution algorithm. *2008 Conference on Human System Interactions*, May 2008. doi: 10.1109/hsi.2008.4581409. URL https://ieeexplore-ieee-org.proxybz.lib.montana.edu:3443/stamp/stamp.jsp?tp=&arnumber=4581409&tag=1.