Project 4 Design Document

Carlos Rivas f71j657@msu.montana.edu

Finnegan Davidson Davidsonfinn@gmail.com

Team: 6 Date: 11/15/2024

1 System Requirements

General Requirements

- **Design Document -** Project 4 requires the submission of a design document (the document you are reading) that outlines the requirements, tasks, architecture, and goals to be accomplished for project 4.
- **Programming -** Project 4 requires utilizing 6 datasets from the UCI Machine Learning repository, 3 for classification and 3 for regression. Data processing for project 4 involves normalization, one-hot encoding, 10-fold cross validation with stratification, and may require imputation for some datasets. With the processed datasets, we will implement a neural network using 4 different strategies for updating the weights. The four methods of updating the weights are using backpropagation, using the genetic algorithm (GA), using differential evolution (DE), and using particle swarm optimization (PSO). We will implement all four of these strategies and compare the performance of each strategy for each dataset.
- **Paper -** A paper entailing results from experiments will be produced with a length between 5 and 10 pages. The paper will include a problem statement, experimental approach and design, presentation of results, algorithm behavior discussion, conclusions, references, and an appendix describing workload distribution.
- **Video -** A video of 5 minutes or less will be provided demonstrating program functionality. The video will focus on program behavior and include display of inputs, data structure, and outputs.
- **Submission -** Final project submission will include fully documented code files with outputs after operation, the 5-10 page paper, and the demonstration video. All code files (fully commented) will be compressed into a single zip file and submitted along with a team member contribution report to the code submission dropbox. A PDF version of the paper (JMLR format) will be submitted to the TurnItIn dropbox. The demonstration video will be submitted to its respective dropbox as a text file containing a link to the video.

Dataset As previously mentioned, 6 datasets are utilized for project 4, where 3 of the datasets are meant for classification, and 3 are meant for regression. Pre-processing will

involve 10-fold stratified cross-validation, where 10% of the data will be used as a hyper-parameter tuning set, and 90% will be used as a train-test set. The train-test set will implement 10-fold cross-validation.

Algorithms

- **Neural Network** To create a neural network we first take in a processed dataset and use the ideal values for the number of hidden layers and hidden nodes per layer found in previous experiments to initialize the weights between each node randomly. The examples in the dataset are then fed forward through the network using a logistic activation function. Once an example has been fed through the network, the loss or fitness is computed and the weights can be updated using one of the four strategies being investigated.

- **Backpropagation** To use backpropagation to update the weights we first use the tune set to tune our hyperparameters to find their ideal values. Once the ideal hyperparameters are found, the algorithm trains by propagating from the input layer to the output layer using the logistic equation activation function. The algorithm then begins propagating back to the inputs using the gradient of the loss function to adjust the weights and biases in the neural network. This process is repeated either for a specified number of epochs or until convergence is reached. Once the model has been trained, it can be used to classify or perform regression for examples in the validation set (hold-out fold from training set cross validation).

- **Genetic Algorithm** To use the genetic algorithm to update the weights in a neural network, a population of different trial weight sets is randomly initialized. The algorithm begins training by evaluating the fitness of each member in the population. Based on each chromosome's (weight set's) fitness, the best performing individuals are selected to reproduce. Once the 'parents' are selected, the algorithm will complete crossover to create offspring, mutation to maintain diversity, and replacement to update the population with the new offspring. There are many strategies that can be used to complete crossover, mutation, and replacement that change how quickly the population can evolve. This process is repeated for a specified number of generations (epochs) or until a termination condition is met. Once the training process is completed, the model is ready to perform classification or regression tasks, depending on the dataset.

- **Differential Evolution** Differential evolution works similarly to the genetic algorithm, where a population of arbitrary size is initialized that is comprised of candidates (sets of weights and biases). For each candidate in the population, 3 candidates aside from the candidate being evaluated are selected and used to create a donor vector. The donor vector is used to mutate the original candidate, and the fitness of the original candidate and mutation is compared to determine which progresses to the next generation. This process repeats until a desired number of generations is reached; at the final generation, each candidate's fitness is evaluated and the best candidate is used as the weights and biases for the neural network.

- **Particle Swarm Optimization** To use particle swarm optimization (PSO) to update the weights in a neural network, we first initialize a population. Each particle in the population represents one set of weights for the neural network. Each particle is initialized with a random position (set of weights) and velocity. Each particle computes it's fitness (how well the set of weights performed) based on a fitness metric. Each particle updates its trajectory based on it's own personal best and the best position from the entire population. The velocity of each particle is updated based on its previous velocity, distance from the its own personal best, and distance from the global best. The way that a particle weighs these three factors when updating it's velocity is controlled by the cognitive update rate and social update rate hyperparameters. The position of each particle is then updated using the updated velocity. This process can be repeated for a specified number of updates, or epochs, or until convergence is reached. Once the process has been completed, the network is trained and can be used to perform classification or regression tasks and calculate loss.

Hyperparameter Tuning Considering all networks will use the same architecture tuned from project 3, this means we do not need to tune the number of hidden nodes in networks with hidden layers.

- **Backpropagation** This weight update method requires the same hyperparameters described in project 3 which includes batch size, learning rate, and momentum values. Batch size determines the number of examples to take into consideration when computing mini-batch gradient descent. The gradient is computed for each example in the batch, and the gradients are averaged to update all of the weights in the batch. The learning rate controls how big of changes we make to the weights during back propagation. Increasing this number increases the size of the step that the algorithm takes when following the gradient. It is important to tune the learning rate because if the learning rate is too large, the algorithm will take too big of a step, and can miss convergence at a local minimum. If the learning rate is too low, the algorithm will take very small steps toward the optimal solution, and will increase computation time. The momentum coefficient hyperparameter determines how strongly we want to consider the last update to the weights when calculating the current change in the weights. This allows the gradient descent process to approach a minimum faster because if the previous change in weights was large, it will increase the size of the current change in weights. Once the changes start to get smaller the effect of the momentum coefficient will get proportionally smaller. This allows gradient descent to converge quicker by moving the weights a further distance when the gradient has a larger magnitude and a smaller distance as the gradient smooths out and a minima is approached.

All three of the following algorithms utilize an initial population size and a generation/epoch limit. Population size determines the number of initial participants in the tests, where a large population size means more weights and biases are initialized. The number of generations/epochs is a limiter that stops the algorithm after an arbitrary number of generations/iterations.

- **Genetic Algorithm** Specific genetic algorithm hyperparameters include crossover rate and mutation rate. Crossover rate determines the likelihood that genetic infor-

mation is exchanged between parents. For example, a crossover rate of 50% would result in offspring that contains roughly half of the genes from one parent and half of the genes from another parent. Mutation rate is similar, except this probability controls the likelihood of a gene randomly being changed (mutated) in an offspring. Mutations allows the algorithm to escape local optimums in performance.

- **Differential Evolution** For differential evolution, unique hyperparameters include scaling factor and crossover probability. The scaling factor determines the weight of the difference between two candidate vectors when creating donor vectors. Crossover probability for differential evolution is essentially the same as in the genetic algorithm, where this parameter controls the likelihood of an offspring containing components from the parents.

- **Particle Swarm Optimization** Finally, particle swarm optimization has three unique hyperparameters: social update rate, cognitive update rate, and inertia. Social update rate (SUR) and cognitive update rate (CUR) balance how heavily an individual considers the global best position (SUR) and the prior positions the individual has occupied (CUR) when updating its velocity. Inertia controls how impactful the individual's previous velocity is when updating its velocity.

Evaluation Metrics With 6 datasets implemented for project 4 (3 for classification, 3 for regression), this means 72 models will need to be evaluated (4 different weight adjustment algorithms, each at 0, 1, and 2 hidden layers, i.e. 4 algorithms $\cdot$ 3 layer configurations $\cdot$ 6 datasets = 72 models). For classification our error metric will be 0/1 loss. For regression we will use mean squared error (MSE). We will also use a fitness metric for the population based algorithms.

2 System Architecture

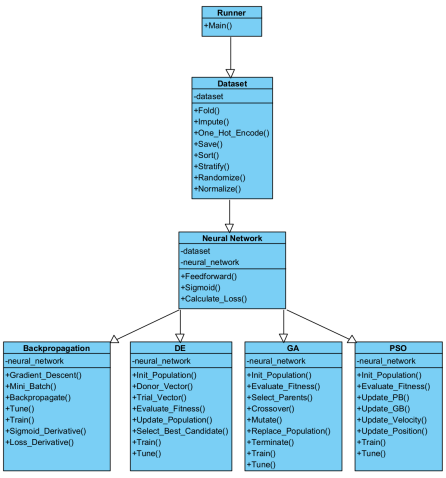

Figure 1: UML Class Diagram

Runner Class The runner class controls the flow of the dataset class and various algorithm classes. It will first create the dataset object from the raw dataset, then it will call the necessary methods to process the data. Once the data is processed, the runner class will use the processed dataset object to create an algorithm object and call the necessary methods to tune and run the algorithm. This process will be completed for each algorithm and each dataset.

Dataset Class Within the dataset class, the data needs to be processed so it is ready to be used by the algorithm. To accomplish this, we will create methods to provide the functionality to ensure the data is in the correct format. This functionality includes imputation of missing features, one-hot encoding for categorical features, normalization for numerical features, and removing an attribute from each example in the entire dataset. This functionality allows us to ensure every example has the attributes properly formatted. The dataset class will also have methods to randomize the dataset, sort the dataset based on class or response value, split the dataset into the tuning and training subsets, fold the data so that cross validation can be utilized within the algorithm, as well as methods to save and extract the data to/from a CSV file. With this functionality, the dataset class is able to properly process the raw dataset and prepare it to be used by the algorithm.

Algorithms

- **Neural Network Class** The neural network class will take in a dataset object from the dataset class. Using this dataset and previously determined values for the number of hidden layers and the number of hidden nodes per layer, the neural network is created and the weights between each node will be randomly initialized. The neural network class will also include the functionality to feed an example forward through the network. To do this we will need methods to determine how many weights are needed, initially set the weights and biases in the network, complete forward propagation through the network, and compute loss or fitness for an example.

- **Backpropagation Class** The backpropagation class will take in a neural network object created by the neural network class. The tuning set will be used by the tune method to determine the best hyperparameters to use for classification and regression tasks. Once this has been done, these hyperparameters will be used to train a model with the training set and complete classification or regression for examples in the validation set (hold-out fold from cross validation). To complete these tasks, the backpropagation class will need methods that provide the functionality to tune the necessary hyperparameters, use mini-batches and gradient descent to determine how to adjust the weights, complete backpropagation to adjust the weights, calculate the result of a sigmoid function and it's derivative, and determine the loss of the algorithm. There will also be a train method that calls the necessary methods to create the model.

- **Genetic Algorithm Class** The genetic algorithm class will take in a neural network object from the neural network class. The tuning set will be used by the tune method to find the ideal set of hyperparameters to train a model with. Once these values have been determined, the train method can be used to call all the necessary functions to train a model. In order to train a model, the algorithm needs the functionality to

initialize a population, evaluate the fitness of an individual in the population, select the best performing individuals to reproduce, complete crossover and mutation to create offspring, update the population to include the offspring and remove individuals that performed poorly, and terminate the evolution of the population.

- **Differential Evolution Class** Differential evolution utlizes a neural network object as input. The tuning set will be used to determine optimal hyperparameters P_n, (population size), F, (scaling factor), and CR, (binomial crossover probability). Once ideal hyperaparameters are determined, they will be utilized to train and test on a given dataset via 10-fold cross validation. The DE class will require methods to intialize a population of candidates, create donor and trial vectors, evaluate fitness of candidates, and update populations based on trial candidate fitness. These will all be wrapped together with a train method that will implement these steps to create a robust set of weights/biases for a neural network.

- **Particle Swarm Optimization Class** The particle swarm optimization class will take in a neural network object from the neural network class. The tuning set will be used by the tune method to determine the ideal hyperparameter values to use for classification and regression tasks. Once the ideal values have been determined, they will be used to train a model with the training set so that it can be used to classify examples in the validation set. To complete these tasks, the PSO class will need methods that provide the functionality to initialize the particles and their velocities, evaluate the fitness of a particle, update the personal and global bests for a particle, update the velocity and position of a particle, and a train method that calls the necessary methods to create a model.

3 System Flow

Processing The processing flow differs slightly between datasets, but overall is very similar across datasets. The system flow begins by passing the raw dataset file into the dataset class. From there, any categorical features are removed using one-hot encoding and any numerical features are normalized using min/max normalization. After the data has been continuized and normalized, it is shuffled to remove any patterns from the original database. After being shuffled, the data will have any non-relevant attributes removed, such as an ID or case number, and will be imputed if there are any missing attribute values in the data. Once this has been completed, the examples in the dataset are properly formatted. To finish processing, the dataset is sorted by class label or response value, split into the tuning and training subsets, and the training subset is folded so that 10-fold cross validation can be used within the algorithm. Once the training set has been folded, processing is complete and the dataset object can be passed to the algorithm class.

Algorithms From previous experiments, the number of input nodes, hidden layers, hidden nodes per layer, and output nodes have already been determined. Once the data is processed, the tune set will be used to tune the other hyperparameters necessary for the neural network. After ideal hyperparameter values are found, the 10 folds in the testing set will be used for classification or regression.

- **Neural Network** The neural network class will randomly initialize the weights and then will be used by the other classes to feed examples forward through the network and compute fitness or loss, depending on the algorithm. This will then be used to update the weights.

- **Backpropagation** To complete backpropagation to update the weights in the network, the backpropagation class uses the neural network class to feed examples forward through the network using a logistic activation. The output layer uses its respective activation function (softmax for classification, or linear combination for regression).

  After activation is applied on the output node for each example in the dataset, the examples are separated into mini-batches. For a given mini batch, loss is calculated for the data after output activation and the true labels. Next, the derivative of the respective activation function is used to back propagate through the function, and the gradient with respect to the weights is calculated. The weights and biases for each layer are then updated using mini-batch gradient descent. After the entire network has been backpropagated, forward propagation can occur again. This process of moving back and forth through the network to update the weights can be repeated until the desired number of epochs (forward-back propagation cycles) have occurred.

- **Genetic Algorithm** To train a model using the genetic algorithm, the GA class first initializes a random population of trial weights. For each generation (epoch), the fitness of each individual in the population is calculated. From these values, the best performing individuals are selected to reproduce. Crossover is applied to the selected parents with a chance of mutation to maintain diversity. Once the offspring have been created, they are added into the population while removing the individuals that performed the worst. After the offspring have been added to the population, the termination condition is checked. If the condition has been met, training is complete and the algorithm will return the best set of weights. If the condition hasn't been met, the process is repeated for another generation of offspring. This can be repeated either a set number of generations or until a termination conditions such as maximum error is met.

- **Differential Evolution** Differential Evolution requires an initialized neural network architecture (neural network object) as input. DE works to find an ideal set of weights and biases by generating a population of candidates (in our case each candidate contains a set of weights and biases). From these initial candidates, new candidates are created over an arbitrary number of generations by mutating the pre-existing candidates. Mutation is performed by selecting 3 candidates (X_a, X_b, X_c) from the population and creating a donor vector V_i via the formula $V_i = X_a + F \cdot (X_b - X_c)$, where F is a tuned scaling factor which determines the amplification of the difference between two of the candidate vectors. Crossover is then performed between a candidate and the donor vector to create a trial vector (X_i). Binomial crossover will be used to determine how much of the donor vector combines with the candidate to create the trial vector. Finally, selection can be performed where fitness is compared between the candidate and trial vector, and whoever has better fitness moves on to the next generation. After an arbitrary number of generations, candidate fitness can

be evaluated for the population, and the candidate with the best fitness will be used as the final weights and biases for the neural network.

- **Particle Swarm Optimization** To complete particle swarm optimization, the PSO class initializes the population with random positions. For each epoch, the fitness of each particle is calculated and the personal and global bests are updated. Each particles velocity is updated using the previous velocity, personal, and global bests. The positions of each particle are then updated using the new velocity. This process is repeated for a specified number of epochs or until convergence is reached.

4 Test Strategy

Pre-Processing Verification Pre-processing steps for project 4 will be verified using both visual inspection in a save file and Jupyter Notebook print statements to determine if the processing steps are occurring correctly. We will specifically be looking for a proper split between the tuning and training data, stratification and division into 10 partitions for the training data, proper one-hot encoding of categorical attribute values, proper normalization of numerical feature values, and the imputation of examples where necessary.

Verification of Algorithm Operation To verify that the algorithms are operating as expected we will use print statements to show that the process of each algorithm is completed in the correct order. To verify the operation of the backpropagation algorithm, we will verify that the gradient is computed correctly and that convergence is determined properly. To verify the operation of the population based algorithms, we will verify that fitness is being properly calculated and utilize print statements to verify a randomly distributed initial population. We will also verify that the population gradually improves during the training phase of the algorithm's operation.

Tuning Verification Parameter tuning will be verified by observing program behavior via print statements. We will be looking for final chosen hyperparameters to have the best performance metrics out of all that were tested. This will involve observing the performance of a range of hyperparameter values, and ensuring the highest performing value is the one picked.

Loss Calculations Since both classification and regression tasks need to be evaluated for project 4, separate evaluation metrics need to be used for each learning type. For classification, we will use 0/1 loss due to it providing information about the overall performance of the algorithm and it's simplicity to calculate. Regression performance will be evaluated using Mean Squared Error (MSE). This will give us the ability to compare the performance of the algorithm across datasets since we normalized our data.

5 Task Assignments and Schedule

Task	Student	Date
Dataset Class	Finn	11/15/2024
Neural Network Class	Carlos	11/15/2024
Genetic Algorithm Class	Finn	11/20/2024
Differential Evolution Class	Carlos	11/20/2024
Particle Swarm Optimization Class	Finn	11/20/2024
Loss Calculations and Data Presentation	Finn	11/30/2024
Process Verification	Carlos	11/30/2024
Project Report - Introduction	Finn/Carlos	12/2/2024
Project Report - System Architecture	Finn/Carlos	12/3/2024
Project Report - Results	Finn/Carlos	12/4/2024
Project Report - Conclusions	Finn/Carlos	12/4/2024
Project Video - Code	Finn/Carlos	12/5/2024
Project Video - Recording	Carlos	12/5/2024

Table 1: Task Allocation and Dates