Project 3 Design Document

Carlos Rivas f71j657@msu.montana.edu

Finnegan Davidson Davidsonfinn@gmail.com

Team: 6 Date: 10/21/2024

1 System Requirements

General Requirements

- **Design Document -** Project 3 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 3.
- **Programming -** Project 3 requires utilizing 6 datasets from the UCI Machine Learning repository, 3 for classification and 3 for regression. Data processing for project 3 involves normalization, one-hot encoding, 10-fold cross validation with stratification, and may require imputation for some datasets. We are implementing a multi-layer feedforward network with backpropagation learning capable of training a network with an arbitrary number of inputs, an arbitrary number of hidden layers, an arbitrary number of hidden nodes by layer, and an arbitrary number of outputs. This algorithm will be used for classification and regression tasks and needs to be tested with 0, 1, and 2 hidden layers.
- **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 3, 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.

Algorithm The algorithm we are implementing in project 3 is a feed forward, multi-layer, perceptron network. To do this 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 a sigmoid (either logistic or hyperbolic tangent) 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). The algorithm will output the predicted class during classification tasks or the predicted response value during regression tasks.

Hyperparameter Tuning Four hyperparameters need to be tuned for the implementation of our feed-forward neural network algorithm. These parameters include the number of hidden nodes when hidden layers are incorporated, along with batch size, learning rate and a momentum coefficient for gradient descent. Hidden node count only applies when a hidden layer is implemented; the number of nodes affects the network's learning capacity, where too few nodes means the network may not be able to learn meaningful relationships from the data, whereas a hidden node count that is too high means the network may overfit the data.

The last three hyperparameters relate to gradient descent. 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. This results in drastically increased computing time.The momentum coefficient hyper parameter 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.

Evaluation Metrics With 6 datasets implemented for project 3 (3 for classification, 3 for regression), this means 18 models will need to be evaluated (3 different networks on each dataset). Classification evaluation metrics will include F1 score and 0/1 loss. For regression we will use mean squared error (MSE).

2 System Architecture

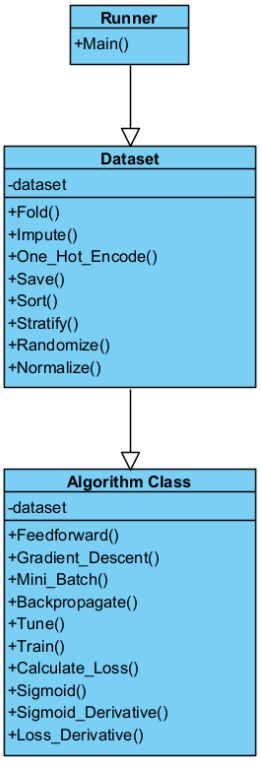

Figure 1: UML Class Diagram

Runner Class The runner class controls the flow of the dataset class and algorithm class. 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.

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.

Algorithm Class The algorithm class will take in the dataset object created by the dataset 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 algorithm class will need methods that provide the functionality to tune the hyperparameters, feed data forward through the network, 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.

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.

Algorithm Once the data is processed, the tune set will be used to tune the 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. The algorithm will be run in one of 3 configurations: 0 hidden layers, 1 hidden layer, and 2 hidden layers. Each example will first be evaluated at the input layer, where the number of input nodes will equal the number of features for a given example. In the 0 hidden layer case, the features are calculated as a linear combination with the initialized weights, and sent directly to the output layer. When hidden layers are active, the input features are sent to the hidden layer(s) with sigmoid activation, until the output layer is reached. 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.

4 Test Strategy

Pre-Processing Verification Pre-processing steps for project 3 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.

Algorithm Verification To verify the algorithm we will use print statements to show that the process of the algorithm is completed in the correct order and also manually verify that each network has the expected number of layers and nodes per layer, based on what implementation we are testing. We will also create test cases that allow us to verify that backpropagation is correctly adjusting the weights and gradient descent correctly finds a local minimum.

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 3, separate evaluation metrics need to be used for each learning type. Classification metrics will include F1 score and 0/1 loss. F1 score provides a holistic evaluation to model performance, where it accounts for precision and recall, and 0/1 loss indicates the percentage of examples that were classified incorrectly. 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	10/24/2024
Algorithm Class	Carlos	10/24/2024
Loss Calculations and Data Presentation	Finn	10/25/2024
Process Verification	Carlos	10/27/2024
Project Report - Introduction	Finn/Carlos	10/28/2024
Project Report - System Architecture	Finn/Carlos	11/1/2024
Project Report - Results	Finn/Carlos	11/3/2024
Project Report - Conclusions	Finn/Carlos	11/3/2024
Project Video - Code	Finn/Carlos	11/2/2024
Project Video - Recording	Carlos	11/3/2024

Table 1: Task Allocation and Dates