Project 3 Design Document

CSCI 447
10/25/2019

Andrew Kirby
Kevin Browder
Nathan Stouffer
Eric Kempf

Class Description

DataReader: This class takes in a file name and reads in the data to instances of Set classes. The data is assumed to be in a standard format that a DataReader object can process.
Set: A Set is composed of a group of examples (data points). This will be used to store examples so that our learning algorithm can be trained and tested.
IDataReducer: This interface characterizes the methods that will be used to reduce the data points used to represent the number of nodes in a hidden layer of a Radial Basis Function Network. The *Edited, Condensed, CMeans,* and *CMedoids* classes will implement this interface through their respective algorithms to reduce the representative set of data points.
INeuralNet: This interface defines the methods required to implement a Neural Network. There are two types of networks that will be implemented: *Radial Basis Function (RBF)* and *Multi-Layer Perceptron (MLP)*. Each have their own class that will handle the specifics of each type of network.
Layer: The Layer class composes networks. Each layer consists of a weight matrix and an activation function. An input vector can be passed into a layer to produce the output to the next layer.
IActFunct: This interface defines the methods used in an activation function. There are two types: *Sigmoidal* and *Linear*. Each take in a Vector and compute the activation of each component in the vector.
Backpropagator: The Backpropagator class is the weight learning algorithm. A network is passed in along with a batch to train on. The Backpropagator class then applies the backpropagation algorithm to compute the gradient of each weight in the network.

Algorithm Design

Both networks (and most neural nets) have the basic structure of an input layer, hidden layers, and an output layer. Thus, the functionality of layers—nodes, edge weights, and backpropagation—are generic. Each layer may utilize linear or sigmoidal activation functions. The sigmoidal activation function of choice will be the logistic function. The two neural networks are implemented using the Strategy Pattern (since they implement the same interface).

RBF networks consist of an input layer, a single hidden layer, and an output layer. The RBF implemented will use a Gaussian basis function. Upon instantiation, each RBF network will require an input of an arbitrary number of “clusters” that each consist of a representative example and variance. These clusters will be used to construct each Gaussian basis function for the hidden layer (Wu 2012). Training will utilize an arbitrarily sized set of examples and apply gradient descent (implemented via the backpropagation class) to the output layer in batches.

Weights on the output layer will be initialized to a random value between -0.01 and 0.01. These weights are chosen to be small to avoid node saturation, which allows the network to converge faster (Scalero 1992).

MLP networks may have an arbitrary number of hidden layers with an arbitrary number of hidden nodes. Upon instantiation, each MLP will require the number of hidden layers, number of hidden nodes, and activation function to be used in each layer. Weights on all layers will be initialized to a random value between -0.01 and 0.01. Tests will prefer use of sigmoidal activation functions in the hidden layers in order to ensure the network functions as a universal approximator. Training with an arbitrarily sized set of examples is done in batches using backpropagation from the output through the hidden layers (Svozil 1997). Each MLP may have a tunable momentum. The previous gradient vector returned from backpropagation is stored for handling upon the next return from backpropagation.

In both the RBF and MLP networks, a classification or regression dataset is detected and the output layer chosen upon training. A classification network will have an output node corresponding to each class in the dataset, each with a sigmoidal activation function. Predicted class will be chosen based on node with the highest activation value. A regression network will have a single output node with a linear activation function. Predicted values will be the activation value of this node given an input.

The training of each network relies on gradient descent, which is implemented in the Backpropagator class. This class takes in the current weights as well as a training batch. For each example in the training batch, the algorithm will compute a prediction using the current weights. The gradient with respect to an individual weight is computed as $\frac{\partial Err}{\partial w_{ji}} = \delta_j * x_{ji}$ where x_{ji} is the i^{th} input to the j^{th} node and δ_j is the error term that is propagated backwards from the error at the output layer. The error function used at the output layer is Mean Squared Error. For hidden layers, δ_j is a function of δ_k (where δ_k represents the δ values in the next layer). This sets up a recursive structure for backpropagation that can be applied to any number of hidden layers. The gradient for each example in the batch is then averaged. This is returned to the learner for updates to the network.

Experimental Design

Validation

Both networks will be tested on every data set. Regression and Classification will be used as needed. Each test will involve ten-fold cross validation, which requires separate training and test sets that will be allocated during preprocessing.

Tuning

Starting with the feed forward network, the most important attribute to tune will be the number of nodes in each hidden layer. The optimum number of nodes in a hidden layer is somewhere between the number of nodes in the input and output layers (Heaton 2008). Tuning will start with the mean of the number of nodes in the input and output layers for classification problems and half the number of nodes in the input layer for regression. The momentum will also be tuned. Tuning of momentum will start at .5 as it is a common value to start tuning with (Goodfellow 2016). From the starting value we will tune downwards to find a more optimal local minimum. Lastly the learning rate will be tuned. Since the input data is normalized between 0 and 1, learning rates should be between 10^-6 and 1 and the starting value will be .1 because it is a standard starting value (Bengio 2012). We will tune in both directions from this starting value to find an optimal resolution and speed. This will be the same for both the feed forward network and the RBF network.

The variance of the clusters in the RBF networks is the last attribute that will be tuned. This will be tuned by modifying *k* in K Nearest Neighbors (K-NN) which is how variance will be calculated. The number of neighbors *k* for the K-NN algorithm will be chosen initially to be the square root of the size of the data set, a common value for *k* (Shichao 2017). From this initial value *k* will be tuned both up and down to determine the optimal number of neighbors.

Evaluation Metrics

Mean Squared Error (MSE), and accuracy will be used to evaluate the classification problems. Classes do not have real value associated with them, so MSE is calculated as the square root of the sum of squared differences between total actual examples in a class and total predicted examples in a given class. In this case, MSE is a good indicator of how far off the predicted distribution is from the actual distribution. Accuracy indicates how well the algorithm is classifying examples on an individual basis. These two metrics supplement each other because it is possible to get a MSE of 0 and classify nothing right. Using both metrics give a better overall evaluation of an algorithms performance.

Mean Absolute Error (MAE) and MSE will be used to evaluate the regression problems. MSE takes the distance between real and predicted values and squares it. MAE is similar but takes the absolute value of the distance between real and predicted values. One issue with MSE is that if the error is less than 1 the performance is underestimated and if the error is greater than 1 the performance is overestimated. This can be addressed by using MAE as well because it is linear and will show if the MSE is over or underestimating the performance of the algorithm. An issue encountered in previous projects is that both MAE and MSE are proportional to the real-valued outputs. This means evaluating these metrics requires that the context (the scale of the outputs) is known. This will be accounted for in this project by computing the MAE and MSE of the z-scores. The mean and standard deviations used to compute the z-score will be computed based on the testing set.

References

Bengio, Y., 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade* (pp. 437-478). Springer, Berlin, Heidelberg.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT press.

Jeff Heaton. 2008. Introduction to Neural Networks for Java, 2nd Edition (2nd ed.). Heaton Research, Inc..

Scalero, R. S. and Tepedelenlioglum, N. "A fast new algorithm for training feedforward neural networks," in *IEEE Transactions on Signal Processing*, vol. 40, no. 1, pp. 202-210, Jan. 1992. doi: 10.1109/78.157194

Shichao Zhang, Xuelong Li, Ming Zong, Xiaofeng Zhu, and Debo Cheng. 2017. "Learning *k* for kNN Classification." ACM Trans. Intell. Syst. Technol. 8, 3, Article 43 (January 2017), 19 pages.

Svozil, D., Kvasnicka, V., Pospichal, J. "Introduction to multi-layer feed-forward neural networks." Chemometrics and Intelligent Laboratory Systems, Volume 39, Issue 1, 1997, Pages 43-62.

Wu, Y., Wang, H., Zhang, B. and Du, K.L., 2012. Using radial basis function networks for function approximation and classification. *ISRN Applied Mathematics*, 2012.