CS 4793 Programming Assignment 1

Due: September 16, 11:59 p.m.

Version 1.0, last modified: 8/31/2016

Goal: build, train, and test TLUs (threshold logic units, i.e., a single neuron) for classification using the TensorFlow framework.

Description: the assignment consists of implementing two programs, `tluTrain` and `tluEval`, to learn the weights of TLUs from training data and evaluated the performance on test data.

Details:

- Both programs will use command line parameters to obtain the files names:

  `tluTrain` *trainFile validationFile modelFile*

  `tluEval` *testFile modelFile*

  where *trainFile*, *validationFile*, and *testFile* are data files containing the training, validation, and testing partitions of the data. The *modelFile* is created by `tluTrain` and should hold the weights and other information such that `tluEval` can recreate the TLUs from the file.

- **Data file format**: the data files will be line base with one (y, $\boldsymbol{x}$) example per line. The values will be whitespace separated. The components of data point $\boldsymbol{x}$ will be real values. The class label y will be an integer from zero to $k-1$ where k is the number of classes. All lines will have the same number of values. Example line from iris data:

  ```
  2 6.3 2.7 4.9 1.8
  ```

  would mean y is class 2 and $\boldsymbol{x} = [6.3\ 2.7\ 4.9\ 1.8]^T$. Class 2 of iris means *Iris Virginica* and the $\boldsymbol{x}$ is the measurement, in cm, of sepal length, sepal width, petal length, petal width. Different data set will have different dimensions.

- Data sets will be available in the Data folder of the class repository, on departmental servers csx{?}.cs.okstate.edu in `/class/cs4793/data` and by direct links in D2L. You should run your programs on **iris**, **letter**, **2dsynA**, and **2dsynB**.

- The TLUs should use the logistic function for the threshold function. A TLU should be associated with a class and its output should be trained to predict if an input $\boldsymbol{x}$ is a member of that class or not. That is, we can view the output of the ith TLU as $p(c_i|\boldsymbol{x})$, the probability the example is a member of class c_i given knowledge of example location $\boldsymbol{x}$.

- **Training:**
  - Use mini-batch training. That is, modify the weights by gradient descent after each batch of training example is presented to the TLUs (accumulate the change in weights, modify at end of batch). Randomly select samples to create the mini-batch each iteration. Use a mini-batch size of 32 (or the whole training set if smaller than 32). One past through the entire training data is called If a training

set has N examples, then an *epoch* is after we present N example for training (i.e., $N/32$ mini-batches).

  - Use a learning rate (step size) of $\alpha = \dfrac{2}{1+t^2}$ where t is the epoch count.
  - Instead of using 0 and 1 as target value, use 0.1 and 0.9. For example with iris, if a given example has y=2, the TLU$_0$ would have target 0.1, TLU$_1$ would have target 0.1, and TLU$_2$ would use target 0.9.
  - Stop training after the validation error (explained later) does not decrease lower then the minimum seen so far for five epochs. Save the TLUs associated ~~with the minimum validation error~~ with the final epoch.
  - Initial weights of the TLUs should be random values in the range $[-\frac{1}{\sqrt{1+n}}, +\frac{1}{\sqrt{1+n}}]$ where n is the dimensionality of $\boldsymbol{x}$. Your program should set a random number seed so your runs are reproducible. As a safety measure, use 1,000 as a maximum number of epochs. If reached, stop training.

- **Error Measurements**: we will use classification error instead of residual error.
  - We have both the error of an individual TLU and error of the collection of TLUs. For a single example $(y, \boldsymbol{x})$, present $\boldsymbol{x}$ as input to the TLU or TLUs and
    * For an individual TLU, threshold the output at 0.5 to prediction if in that TLU's class or not.
    * For a collection of TLUs, select the TLU with largest output and predict class label as the label associated with that TLU.
  - during training, measure the validation error after each epoch. This is done by presenting each example from the validation set to the TLUs and recording the number of wrong predictions for each individual TLU and the collection of TLUs. The classification error rate is $\dfrac{\text{\# wrong}}{\text{\# validation examples}}$. Use the TLU collection validation error rate when making the decision to stop training. Save the validation error of the individual TLUs and the collection so you make make a graph for your assignment report. **Do not update TLU weights when measuring validation error**.
  - during training, measure the training classification error during each epoch. This is done while presenting the training examples to the TLUs for training (after presenting an example to a TLU, but before updating the weights). The error rate is $\dfrac{\text{\# wrong}}{\text{\# training examples}}$. Save the training error of the individual TLUs and the collection so you make make a graph for your assignment report.
  - during evaluation (`tluEval`), measure the test error by making one past thourgh the test data and recorded the number of wrong predictions for each individual TLU and the collection of TLUs. The error rate is $\dfrac{\text{\# wrong}}{\text{\# test examples}}$. **Do not update TLU weights when measuring test error**.

  - `tluEval` should output the test classification error of each TLU and the collection.
  - `tluTrain` should output the training and validation classification error of each TLU and the collection for the saved model (the minimum validation error TLUs). In addition, it should output the epoch iteration count of the saved model.

- Make a graph or graphs of classification error vs epoch iteration for each data set. You may save the training and validation errors into a separate log file for later processing my another program to create the graphs. Combine the graphs into a single pdf document (one or two graphs per page).

- Use are expected to use the TensorFlow framework, but you may also implement your own TLUs. If so, you may use supporting matrix frameworks such as numpy, eigen, JAMA, etc. See the extra credit section if you wish to try to make both a TensorFlow and your own implementation.

Submit your files using subversion, D2L or handin. The files should include the error graph documment, a readme.txt file, all model files, and all source code need to build the two programs. You should verify that you have submitted all required files by testing in a different build directory, preferably on different machine like csx. If submitting on D2L, zip or tar archive must be used for turning in the source code and model files. If you are using an external framework, say a matrix library or boost, check with the instructor and TA on whether you need to include framework in your submission. Include the model files saved by `tluTrain` for the data sets **iris**, **letter**, **2dsynA**, and **2dsynB**. The model files need to be platform independent. A readme.txt file is required. The readme.txt should give a description of how to compile and execute the code. It should also state the version number of the compiler, language, and architecture used. Add comments about bugs remaining in your code and bugs/problems that your encountered and were able to fix. Give an estimate on the amount of time you spent on different components of the assignment. Be sure to list in the readme file and in the comments in your code all resources that you used in developing your program. That is, if you visited a website to try to figure out how to implement a certain technique then you need to credit that web site for that technique. Good programming style should be followed. Poor programming techniques will incur a penalty.

Optional Extra Credit

- (15%) Create both a Tensorflow implementation and your own implementation. Compare training and test times between the two implementations.

- (10%) repeat the experiments, but first standardize the data into having zero mean and unit variance. That is, determine the mean $\boldsymbol{\mu}$ and variances σ_i^2 of the training and validation data and then transform each example $\boldsymbol{x}$ to $\boldsymbol{x}'$ by

$$\boldsymbol{x}_i' = \frac{\boldsymbol{x}_i - \boldsymbol{u}_i}{\sigma_i^2}$$

for each attribute dimension i. Note that you will need to store $\boldsymbol{\mu}$ and $\boldsymbol{sigma}$ in the model file so it is available to transform the test data in `tluEval`.

- (10%) repeat the experiments, but replace the logistic function $f(a) = \dfrac{1}{1+e^{-a}}$ with the hyperbolic tangent function:

  $$f(a) = b_0 \tanh(b_1 a) = \frac{2b_0}{1+e^{-b_1 a}} - b_0$$

  with constants $b_0 = 1.716$ and $b_1 = \frac{2}{3}$. Use targets for training of -1 or +1 instead of 0.1 and 0.9. Note that the weight update rule changes to use the gradient of the new $f(a)$ function.

- (10%) Repeat the experiment using a 10-fold cross validation. Combine the three data partition (training, validation, and test) into one and then randomly partition into ten set. For each of the ten set in turn, use it as a test set, randomly select three other sets as the validation set, and use the six remaining as the training set. Use the average over the ten runs when making the graphs of the errors.

If the extra credit is implement, use the driver program names `tluTrainEC1`, `tluTrainEC2`, etc. For the cross validation extra credit, use the name `tluCV` for a driver program that takes the three data sets as command line parameters.