Mohammed J. Zaki

Dmcourse /

Assign5

Assignment 5 (Neural Networks): Due Date: Fri 18th Nov, Before Midnight

In this assignment you will implement the backpropagation algorithm to train a three-layer neural network.

Given a comma-seprated datafile, whose last column denotes the class, you have to train a neural network. First, the size of the input layer N_i is determined by the number of attributes (other than the class), and the size of the output layer N_o is determined by the number of classes. You should input the size of the hidden layer N_h from the command line. After training, you should compute the accuracy of the classifier on the testing set. Note that if there are k classes, they should be coded as 1-of-k vectors, e.g., if there are three classes you should code them as $\{1, 0, 0\}, \{0, 1, 0\}, \{0, 0, 1\}$. Thus the classes are converted into binary vectors, i.e., a string/scalar value y_i is converted into a vector $\mathbf{y}_i$.

The pseudo-code for feedforward and backpropagation based training is given below .

Input: Dataset: $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}, N_h, \eta$, epochs
Output: weights (inner to hidden, hidden to output), accuracy

for $e = 0, \ldots, epochs - 1$
    for each point $\mathbf{x}_i$ in random order **do**
        $\hat{\mathbf{y}}_i$ = feedforward $(\mathbf{x}_i)$
        $E = \frac{1}{2}\|\hat{\mathbf{y}}_i - \mathbf{y}_i\|^2$
        if $E > 0$ **do**
            backpropagation $(\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}}_i)$

print the weight matrices (input-to-hidden and hidden-to-output)
compute accuracy on testing set
print accuracy

You must implement the feedforward and backpropagation methods. For initia lizing the synapse weights, set them randomly between -0.1 and +0.1.

Use the feedforward step to compute the net_j value at each unit, then use the logistic sigmoid function to compute the output value o_j at each hidden and output layer . Obviously the output of the input layer neurons is the value itself x_i. Make sure that you also add an extra input neuron for the bias terms for the hidden layer, and an extra hidden neur on for the biases at the output layer .

For the back propagation, compute the δ_j values at each output and hidden layer neuron. This way the weight w_{ij} between neuron i and neuron j (either for input-hidden or hidden-output layers) can be updated as follows:

$$w_{ij}^{new} = w_{ij} + \Delta w_{ij}$$

where

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot o_i$$

Once new weights have been computed you go back to the feedforward step as shown in the main algorithm.

For the back propagation step, first compute the δ_j values, and then update the weights w_{ij}. For the output layer, we have:

$$\delta_j = o_j \cdot (1 - o_j) \cdot (t_j - o_j)$$

Here t_j is the true output, and o_j is the output of the j-th output neuron. For the hidden layer , we have:

$$\delta_j = o_j \cdot (1 - o_j) \cdot \sum_{k=1}^{N_o} \delta_k w_{jk}$$

Here o_j is the output of the j-th hidden neuron, and the δ_k are the values computed above for the N_o output neurons.

CSCI6390 Only: Linear Neural Network for Dominant Principal Component

You will implement a hebbian learning rule to extract the dominant eigenvecto r for a given dataset. This network has the same N_i dimensional input vector $\mathbf{x} = (x_1, x_2, \ldots, x_{N_i})^T$, i.e., all attributes except the class, which you should ignore. On the other hand, there is only one output neuron y. Let $\mathbf{w} = (w_1, w_2, \ldots, w_{N_i})^T$ be the set of weights connecting the input value x_i with weight w_i to the output y. The output neuron is linear, i.e., $y = \sum_{i=1}^{N_i} w_i x_i = \mathbf{w}^T \mathbf{x}_i$. Starting with a random set of weights between $[-0.1, 0.1]$, we update the weight vector as follows for each input vector:

$$\Delta \mathbf{w} = \eta \cdot y \cdot (\mathbf{x}_i - y \cdot \mathbf{w})$$

Thus, the algorithm to extract the first principal component (or the dominant eigenvector) is given as:

Input: Dataset: $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, η, epochs
center $\mathbf{X}$
for $e = 0, \ldots, epochs - 1$
    for each point $\mathbf{x}_i$ in random order **do**
        $y = \mathbf{w}^T \mathbf{x}_i$
        $\Delta \mathbf{w} = \eta \cdot y \cdot (\mathbf{x}_i - y \cdot \mathbf{w})$
        $\mathbf{w} = \mathbf{w} + \Delta \mathbf{w}$

print the weight vector $\mathbf{w}$
compute the dominant eigenvector $\mathbf{u}_1$ of the matrix $\mathbf{X}^T\mathbf{X}$
verify that $\mathbf{w}$ is close to $\mathbf{u}_1$, i.e., print $\|abs(\mathbf{w}) - abs(\mathbf{u})\|$

Note that we are doing $\|abs(\mathbf{w}) - abs(\mathbf{u})\|$, since entries in $\mathbf{u}$ can have opposite sign to entries in $\mathbf{w}$.

What to turn in

Write a script named **assign5-LAST-FIRST.py** where **LAST** and **FIRST** are your last and first names, respectively. The script will be run as follows:
assign5.py TRAIN TEST N_h η epochs
where TRAIN/TEST are the training and testing files, which will contain each point on a line, with comma separated attributes, and with the last attribute denoting the class (do NOT assume that there are only 2 classes); N_h is the size of the hidden layer , η is the learning step size, and epochs is the number of runs through the entire dataset. Note that N_h does not include the bias; you should add an extra neuron for that.

Save your output to a text file **assign5-LAST-FIRST.txt**. It should contain the output of the weights (and biases), and the accuracy value. Try different N_h and η values, and report the best accuracy obtained on the Concrete Dataset: Attach:Concrete_Data_RNorm_Class_train.txt for training set, and Attach:Concrete_Data_RNorm_Class_test.txt for testing set. Note that it can take a while to converge, so be patient. Also, start early with the assignment. Also, try to use matrix operations to speed up the computation. You can test your code on the smaller Attach:iris-numeric.txt dataset.

In addition, the CSCI6390 students should submit a separate script **assign5-PCA.py** that will be run as follows:
assign5-PCA.py DATA η epochs
where DATA is a data file (ignore the last column for the PCA). You can test the code on the smaller Attach:iris-numeric.txt data from above, and then submit the eigenvector for Attach:Concrete_Data_RNorm_Class_train.txt.

Email your script and output file to datamining.rpi@gmail.com . The subject of the email should be **assign5-LAST-FIRST**.

Page last modified on November 15, 2016, at 01:47 PM