Sean Trent

00707820

AI

Project 3

1. The input for this problem is a series of patterns with two characteristics: time and temperature. The output is a second series of time and temperature patterns that have been generated by the decision function. This second series is a *prediction* and this problem is called a time-series problem.
2. I chose the activation function y = mx + b. This gives outputs on the range of -INF to INF, is simple for calculating output using weights, and is differentiable in a way that simplifies Least Square Method learning.
3. The following table shows information from the first iteration and the last, as showing everything would be too much for this report. (Note: at this point, values are still normalized)

iteration= 0, p= 0, output= 6.68, desired output= 0.12, error= -6.560, total error= 43.028

iteration= 0, p= 1, output= 3.77, desired output= 0.33, error= -3.436, total error= 54.835

iteration= 0, p= 2, output= 1.61, desired output= 0.42, error= -1.182, total error= 56.231

iteration= 0, p= 3, output= 0.55, desired output= 0.58, error= 0.030, total error= 56.232

iteration= 0, p= 4, output= 0.65, desired output= 0.68, error= 0.035, total error= 56.233

iteration= 0, p= 5, output= 0.73, desired output= 0.79, error= 0.058, total error= 56.237

iteration= 0, p= 6, output= 0.88, desired output= 0.85, error= -0.032, total error= 56.238

iteration= 0, p= 7, output= 0.82, desired output= 0.94, error= 0.120, total error= 56.252

iteration= 0, p= 8, output= 0.62, desired output= 0.09, error= -0.532, total error= 56.535

iteration= 0, p= 9, output= 0.32, desired output= 0.32, error= 0.005, total error= 56.535

iteration= 0, p= 10, output= 0.44, desired output= 0.43, error= -0.011, total error= 56.535

iteration= 0, p= 11, output= 0.44, desired output= 0.56, error= 0.118, total error= 56.549

iteration= 0, p= 12, output= 0.63, desired output= 0.67, error= 0.036, total error= 56.551

iteration= 0, p= 13, output= 0.72, desired output= 0.78, error= 0.063, total error= 56.555

iteration= 0, p= 14, output= 0.88, desired output= 0.82, error= -0.057, total error= 56.558

iteration= 0, p= 15, output= 0.75, desired output= 1.00, error= 0.246, total error= 56.618

iteration= 0, p= 16, output= 0.70, desired output= 0.13, error= -0.576, total error= 56.950

iteration= 0, p= 17, output= 0.45, desired output= 0.30, error= -0.152, total error= 56.973

iteration= 0, p= 18, output= 0.42, desired output= 0.46, error= 0.038, total error= 56.974

iteration= 0, p= 19, output= 0.52, desired output= 0.61, error= 0.092, total error= 56.983

iteration= 0, p= 20, output= 0.69, desired output= 0.72, error= 0.024, total error= 56.983

iteration= 0, p= 21, output= 0.76, desired output= 0.82, error= 0.061, total error= 56.987

iteration= 0, p= 22, output= 0.92, desired output= 0.88, error= -0.036, total error= 56.988

iteration= 0, p= 23, output= 0.85, desired output= 0.97, error= 0.124, total error= 57.004

iteration= 999, p= 0, output= 0.07, desired output= 0.12, error= 0.048, total error= 0.002

iteration= 999, p= 1, output= 0.23, desired output= 0.33, error= 0.105, total error= 0.013

iteration= 999, p= 2, output= 0.52, desired output= 0.42, error= -0.091, total error= 0.022

iteration= 999, p= 3, output= 0.56, desired output= 0.58, error= 0.013, total error= 0.022

iteration= 999, p= 4, output= 0.75, desired output= 0.68, error= -0.064, total error= 0.026

iteration= 999, p= 5, output= 0.79, desired output= 0.79, error= 0.004, total error= 0.026

iteration= 999, p= 6, output= 0.93, desired output= 0.85, error= -0.080, total error= 0.032

iteration= 999, p= 7, output= 0.86, desired output= 0.94, error= 0.084, total error= 0.039

iteration= 999, p= 8, output= 0.08, desired output= 0.09, error= 0.012, total error= 0.040

iteration= 999, p= 9, output= 0.18, desired output= 0.32, error= 0.140, total error= 0.059

iteration= 999, p= 10, output= 0.51, desired output= 0.43, error= -0.081, total error= 0.066

iteration= 999, p= 11, output= 0.57, desired output= 0.56, error= -0.012, total error= 0.066

iteration= 999, p= 12, output= 0.71, desired output= 0.67, error= -0.047, total error= 0.068

iteration= 999, p= 13, output= 0.78, desired output= 0.78, error= 0.003, total error= 0.068

iteration= 999, p= 14, output= 0.92, desired output= 0.82, error= -0.098, total error= 0.078

iteration= 999, p= 15, output= 0.80, desired output= 1.00, error= 0.200, total error= 0.117

iteration= 999, p= 16, output= 0.16, desired output= 0.13, error= -0.034, total error= 0.119

iteration= 999, p= 17, output= 0.31, desired output= 0.30, error= -0.003, total error= 0.119

iteration= 999, p= 18, output= 0.50, desired output= 0.46, error= -0.036, total error= 0.120

iteration= 999, p= 19, output= 0.64, desired output= 0.61, error= -0.029, total error= 0.121

iteration= 999, p= 20, output= 0.77, desired output= 0.72, error= -0.058, total error= 0.124

iteration= 999, p= 21, output= 0.82, desired output= 0.82, error= -0.000, total error= 0.124

iteration= 999, p= 22, output= 0.96, desired output= 0.88, error= -0.078, total error= 0.130

iteration= 999, p= 23, output= 0.89, desired output= 0.97, error= 0.080, total error= 0.137

1. The following is the results of testing my prediction on the last day of data

predicted output= 58.13, actual output= 59.50, error= -1.37, percent error= 2.30%

predicted output= 62.02, actual output= 64.00, error= -1.98, percent error= 3.10%

predicted output= 65.90, actual output= 68.70, error= -2.80, percent error= 4.08%

predicted output= 69.78, actual output= 73.65, error= -3.87, percent error= 5.26%

predicted output= 73.66, actual output= 78.43, error= -4.77, percent error= 6.08%

predicted output= 77.54, actual output= 82.00, error= -4.46, percent error= 5.44%

predicted output= 81.42, actual output= 85.20, error= -3.78, percent error= 4.43%

predicted output= 85.31, actual output= 87.00, error= -1.69, percent error= 1.95%

predicted output= 89.19, actual output= 90.67, error= -1.48, percent error= 1.64%

* The learning algorithm iterated 1000 times. This was the iteration max established in project 2, and it seemed to work just as well as much larger numbers, so I kept it.
* The learning constant was set to 0.007. This number was found after some testing; anything larger than 0.009 was too large, and anything smaller than 0.001 was too small. My normalized data was very sensitive to this learning constant.
* The data for this project went through considerable pre-processing before it was usable in the intended way. Since there was no established “correct” output while training, but the time-series problem requires supervised learning, I had to somewhat create a dataset of correct outputs. This was done by setting the temperature recorded at each hour as the input for the next temperature in the set. Thus, the input was made up of time and temperature pairs being the input for the decision function, while the successive temperature would be the desired output. With this method, only the first temperature recorded at 5 AM each day was not a desired output, but this did not matter once the unit was properly trained.

I also normalized the temperatures between 0 and 1 as not to dwarf the time inputs, as they were very important.

1. Technically, extra iterations help this decision unit make predictions. Yet, on such a small dataset, you start to see diminishing returns in the learning of the agent at relatively low iteration numbers. I saw no noticeable benefit of 10000 iterations over 1000 iterations.
2. Yes, the error could be reduced by using a neural network with added layers; however, these hidden layers *must* use activation functions that are non-linear in order truly increase the effectiveness of the prediction model. The final output function is linear (y=mx+b), so adding more linear functions would be the same as just making the existing linear function longer and needlessly complex.

The remnants of a second hidden decision function are in my code. I originally had my data normalized between 0 and 1. This data would then pass through a hyperbolic tangent (tanh) decision function. Using the delta learning rule, this non-linear decision function allowed for back propagation in order to adjust weights appropriately. Once fully iterated through the tanh function, the linear function y=mx+b would use the weights to predict temperature values at a given time (stored in the x value).

CODE

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.io.PrintWriter;  
**import** java.util.Random;  
**import** java.util.Scanner;  
  
**public class** Thermostat {  
  
 **private static final int *MAX\_ITERATIONS*** = 1000;  
 **private static final double *ALPHA*** = 0.007;  
 **private static final double *DERR*** = 0.001;  
 **private static final double *K*** = 0.2;  
  
 **private static double** *tempMax*;  
 **private static double** *tempMin*;  
 **private static double** *tempRange*;  
 **private double**[] **temps1**;  
 **private double**[] **temps2**;  
 **private double**[] **temps3**;  
 **private double**[][] **temps0**;  
 **private double**[][] **tempsNorm**;  
 **private double**[] **temps4**;  
 **private double**[][] **denormed**;  
  
 */\*\*  
 \* Constructor. Combines data from 3 train days into a single 2D array temps0. Creates a normalized version of the 2D array tempsNorm.  
 \*  
 \** ***@param train1*** *explicit path of txt files as strings ("src/train\_data\_1.txt")  
 \** ***@param train2*** *see above  
 \** ***@param train3*** *see above  
 \** ***@param test*** *see above  
 \*/* **public** Thermostat(String train1, String train2, String train3, String test) {  
 **try** {  
 **this**.**temps1** = readIn(train1);  
 **this**.**temps2** = readIn(train2);  
 **this**.**temps3** = readIn(train3);  
 **this**.**temps4** = readIn(test);  
 } **catch** (FileNotFoundException e) {  
 System.***err***.println(**"FileNotFoundException: "** + e.getMessage());  
 System.*exit*(-1);  
 }  
  
 **this**.*tempMax* = Double.***MIN\_VALUE***;  
 **this**.*tempMin* = Double.***MAX\_VALUE***;  
 **this**.*tempRange* = 0;  
  
 *//Make the datasets usable in supervised learning by shifting, then adding it all to one dataset* **double**[][] day1 = shift(**this**.**temps1**);  
 **double**[][] day2 = shift(**this**.**temps2**);  
 **double**[][] day3 = shift(**this**.**temps3**);  
 **this**.**temps0** = append(day1, day2, day3);  
  
 *//Normalize the data, sets the tempMin, tempMax, and tempRange fields* **this**.**tempsNorm** = normalize(**this**.**temps0**);  
  
 *//****todo change me*** *//this.denormed = denormalize(this.tempsNorm);* }  
  
 */\*\*  
 \* Normalizes the temperature values in between 0 and 1  
 \*  
 \** ***@param temps*** *the 2D array of values from text files  
 \** ***@return*** *a version of temps0 where temperature values have been normalized  
 \*/* **public double**[][] normalize(**double**[][] temps) {  
 **double** max = Double.***MIN\_VALUE***;  
 **double** min = Double.***MAX\_VALUE***;  
 **double** range = 0;  
 **int** i, j;  
 **double**[][] result = temps.clone();  
 **for** (i = 0; i < temps.**length**; i++) {  
 **for** (j = 0; j < temps[i].**length**; j++) {  
 **if** (temps[i][j] > max) {  
 max = temps[i][j];  
 }  
 **if** (temps[i][j] < min) {  
 min = temps[i][j];  
 }  
 }  
 }  
 range = max - min;  
  
 *//Sets fields, this method is only called in constructor so it is fine* **this**.*tempMax* = max;  
 **this**.*tempMin* = min;  
 **this**.*tempRange* = range;  
  
 **for** (i = 0; i < result.**length**; i++) {  
 **for** (j = 0; j < result[i].**length**; j++) {  
 result[i][j] = (result[i][j] - min) / range;  
 System.***out***.println(**"I AM HERE "** + result[i][j]);  
 }  
 }  
  
 **return** result;  
 }  
  
 */\*\*  
 \* Denormalizes the 2d array of data  
 \*  
 \** ***@param norms*** *the normalized data  
 \** ***@return*** *\*/* **public static double**[][] denormalize(**double**[][] norms) {  
 **int** i, j;  
 **double**[][] result = norms.clone();  
 **for** (i = 0; i < result.**length**; i++) {  
 result[i][1] = (result[i][1] \* 33) + 59;  
 }  
 **return** result;  
 }  
  
 */\*\*  
 \* reads in the temperature values from text files to be stored in arrays temps1, temps2, and temps3  
 \*  
 \** ***@param txt*** *the file name as a string  
 \* e.g."src/train\_data\_1.txt"  
 \** ***@return*** *array of just temperature values  
 \** ***@throws*** *FileNotFoundException  
 \*/* **public double**[] readIn(String txt) **throws** FileNotFoundException {  
 File file = **new** File(txt);  
 Scanner scan = **new** Scanner(file);  
 String buffer;  
 String[] timeTemp;  
 **double**[] data = **new double**[9];  
 **for** (**int** i = 0; i < 9; i++) {  
 buffer = scan.nextLine();  
 timeTemp = buffer.split(**","**);  
 data[i] = Double.*parseDouble*(timeTemp[1]);  
 }  
 **return** data;  
 }  
  
 */\*\*  
 \* Transforms the data of one day into something that can be used in supervised learning,  
 \* preceding temperatures are the input and successive temperatures are desired output.  
 \*  
 \** ***@param data*** *the temperatures from a single day  
 \** ***@return*** *2D array of preceding temperatures in column 0 and successive temperatures in column 1  
 \*/* **public double**[][] shift(**double**[] data) {  
 **double**[][] result = **new double**[8][2];  
 result[0][0] = data[0];  
 result[0][1] = data[1];  
 **for** (**int** i = 0; i < data.**length** - 1; i++) {  
 result[i][0] = data[i];  
 result[i][1] = data[i + 1];  
 }  
 **return** result;  
 }  
  
 */\*\*  
 \* Helper method to append all shifted 2d arrays from the 3 days into one 2d array  
 \*  
 \** ***@param day1*** *\** ***@param day2*** *\** ***@param day3*** *\** ***@return*** *one complete 2d array of all shifted data  
 \*/* **public double**[][] append(**double**[][] day1, **double**[][] day2, **double**[][] day3) {  
 **double**[][] result = **new double**[day1.**length** + day2.**length** + day3.**length**][];  
 System.*arraycopy*(day1, 0, result, 0, day1.**length**);  
 System.*arraycopy*(day2, 0, result, day1.**length**, day2.**length**);  
 System.*arraycopy*(day3, 0, result, day1.**length** + day2.**length**, day3.**length**);  
 **return** result;  
 }  
  
 */\*\*  
 \* Implementation of the delta learning algorithm. Adjusts learning by the value of delta, which is returned by fbips  
 \*  
 \** ***@param temps*** *normalized values for the first 3 days of temperature  
 \** ***@return*** *the weights corresponding the the separation line, also printed to sep\_line.txt  
 \** ***@throws*** *FileNotFoundException thrown because of file writing  
 \*/* **public static double**[] deltaLearning(**double**[][] temps) **throws** FileNotFoundException {  
 **int** i, j;  
 **double**[][] pats = **new double**[temps.**length**][3];  
 **double**[] dOut = **new double**[temps.**length**];  
 *//set up pats (patterns) and dOut (desired out) in the following format [hour][last\_temp][bias], [target\_temp]* **for** (i = 0; i < pats.**length**; i++) {  
 **if** (i < 8) {  
 pats[i][0] = i + 6;  
 pats[i][1] = temps[i][0];  
 } **else if** (i < 16) {  
 pats[i][0] = i - 2;  
 pats[i][1] = temps[i][0];  
 } **else** {  
 pats[i][0] = i - 10;  
 pats[i][1] = temps[i][0];  
 }  
 pats[i][2] = 1;  
 dOut[i] = temps[i][1];  
 }  
  
 **for** (i = 0; i < pats.**length**; i++) {  
 **for** (j = 0; j < pats[i].**length**; j++) {  
 System.***out***.print(pats[i][j] + **" "**);  
 }  
 System.***out***.println();  
 }  
  
 *//set up out (actual output) and initialize weights to random values between 0 and 1* **double**[] out = **new double**[temps.**length**];  
 Random random = **new** Random();  
 **double**[] weights = **new double**[3];  
 weights[0] = random.nextDouble();  
 weights[1] = random.nextDouble();  
 weights[2] = random.nextDouble();  
 *//set up delo to hold derivation of activation function* **double**[] delo = **new double**[temps.**length**];  
 *//array to hold return values of activation function* **double**[] act = **new double**[2];  
  
 *//Begin iterating through deltaLearning algorithm, calculating error and adjusting weights* **int** iteration;  
 **int** p;  
 **double** net, err, te, learn, gradient;  
 net = err = te = learn = gradient = Double.***NaN***;  
 *//Initialize output string and printwriter for output table and separation line* String text = **"Hodor"**;  
 PrintWriter writer = **new** PrintWriter(**"src/outputs.txt"**);  
 **for** (iteration = 0; iteration < ***MAX\_ITERATIONS***; iteration++) {  
 *//Initialize te (total error)* te = 0;  
 gradient = 0;  
 **for** (p = 0; p < pats.**length**; p++) {  
 *//Reset net to 0 before iterating through each pattern to get net* net = 0;  
 **for** (i = 0; i < 3; i++) {  
 net = net + (weights[i] \* pats[p][i]);  
 }  
 *//Result of activation function and its derivation stored in out[p] and delo[p] NO ACTIVATION FUNCTION, JUST NET  
 //act = fbip(net);* out[p] = net;  
 *//Calculate err (error)* err = dOut[p] - net;  
 *//Caclulate te (total error)* te = te + (err \* err);  
 *//gradient = -1 \* gradient+(2\*err\*  
 //Adjust learn (learning signal)* learn = -***ALPHA*** \* err;  
 *//Adjust weights for number of augmented inputs (pats[p].length) should be 3* **for** (i = 0; i < 3; i++) {  
 weights[i] = weights[i] + (2 \* ***ALPHA*** \* err \* pats[p][i]);  
 }  
 **if** (iteration == 0 || iteration == 999) {  
 text = String.*format*(**"iteration= %d, p= %d, output= %5.2f, desired output= %5.2f, delta= %5.2f, error= %6.3f, total error= %6.3f"**,  
 iteration, p, out[p], dOut[p], delo[p], err, te);  
 writer.println(text);  
 }  
 }  
 **if** (te < ***DERR***) {  
 **break**;  
 }  
 }  
 writer.close();  
  
 writer = **new** PrintWriter(**"src/sep\_line.txt"**);  
 writer.printf(**"%f%n%f%n%f%n"**, weights[0], weights[1], weights[2]);  
 writer.close();  
  
 **return** weights;  
 }  
  
 */\*\*  
 \* Activation function used in delta learning. This is the only function in the hidden layer of this network. Represents tanh (hyperbolic tangent) and its derivative.  
 \*  
 \** ***@param net*** *\** ***@return*** *\*/* **public static double**[] fbip(**double** net) {  
 **double**[] result = **new double**[2];  
 result[0] = (2 / (1 + Math.*exp*(-2 \* ***K*** \* net))) - 1;  
 result[1] = ***K*** \* (1 - (result[0] \* result[0]));  
 **return** result;  
 }  
  
 */\*\*  
 \* Linear activation function f(a) = a. Written in this format as to be easily switched with fbip(net) in the deltaLearning() function  
 \*  
 \** ***@param net*** *\** ***@return*** *just gives back net and a slope of 1  
 \*/* **public static double**[] linearActivation(**double** xVal, **double**[] weights) {  
 **double** m, x, b;  
 x = xVal;  
 **double**[] result = **new double**[2];  
 m = -(weights[0]) / weights[1];  
 b = -(weights[2]) / weights[1];  
 result[0] = (m \* x) + b;  
 result[1] = m;  
 **return** result;  
 }  
  
 **public static double**[][] predict(**double**[] weights) {  
 **double** m, x, b;  
 **double**[][] result = **new double**[9][2];  
 m = (weights[0] / weights[1]);  
 b = (weights[2] / weights[1]);  
 **for** (**int** i = 0; i < 9; i++) {  
 x = i + 5;  
 result[i][0] = x;  
 result[i][1] = (m \* x) + b;  
 }  
 **return** result;  
 }  
  
 **public static void** analyze(**double**[][] predictions, **double**[] actual) **throws** FileNotFoundException{  
 PrintWriter writer = **new** PrintWriter(**"src/testingResults.txt"**);  
 String text = **"Hodor"**;  
 **for** (**int** i=0;i<predictions.**length**;i++){  
 **double** predicted = predictions[i][1];  
 **double** actualVal = actual[i];  
 **double** error = predicted - actualVal;  
 **double** percentError = Math.*abs*(error/actualVal)\*100;  
 text = String.*format*(**"predicted output= %5.2f, actual output= %5.2f, error= %5.2f, percent error= %5.2f"**,  
 predicted, actualVal, error, percentError);  
 writer.println(text);  
 }  
 writer.close();  
 }  
  
 **public static void** main(String[] args) {  
 Thermostat test = **new** Thermostat(**"src/train\_data\_1.txt"**, **"src/train\_data\_2.txt"**, **"src/train\_data\_3.txt"**, **"src/test\_data\_4.txt"**);  
  
 **try** {  
 *//double[] weights = deltaLearning(test.tempsNorm);* **double**[] weights = {0.071014, 0.603732, -0.370911};  
 **double**[][] prediction = *predict*(weights);  
 **for** (**int** i = 0; i < prediction.**length**; i++) {  
 **for** (**int** j = 0; j < prediction[i].**length**; j++) {  
 System.***out***.print(prediction[i][j] + **" "**);  
 }  
 System.***out***.println();  
 }  
 **double**[][] denormed = *denormalize*(prediction);  
 **for** (**int** i = 0; i < denormed.**length**; i++) {  
 **for** (**int** j = 0; j < denormed[i].**length**; j++) {  
 System.***out***.print(denormed[i][j] + **" "**);  
 }  
 System.***out***.println();  
 }  
 *analyze*(denormed, test.**temps4**);  
 } **catch** (FileNotFoundException e) {  
 System.***err***.println(**"FileNotFoundException: "** + e.getMessage());  
 System.*exit*(-1);  
 }  
 }  
  
  
}