CS74/174 Homework #3

Machine Learning and Statistical Data Analysis: Winter 2016

Due: Feb 27, 2016

The homework should be submitted electronically via Canvas. It should be a ZIP file which includes the main report in a PDF format, and a folder named as "code" that includes all the MATLAB functions and code. Please make sure your code is directly implementable, or you will not receive any credit otherwise. It is important that you include enough detail that we know how you solved the problem, since otherwise we will be unable to grade it. Please include all the figures in the main PDF document with enough comments.

Problem 1: Neural Networks

In this question, we fit a model which can predict what torques a robot needs to apply in order to make its arm reach a desired point in space. The data was collected from a SARCOS robot arm with 7 degrees of freedom. The input vector $x \in \mathbb{R}^{21}$ encodes the desired position, velocity and acceleration of the 7 joints. The output vector $y \in \mathbb{R}^7$ encodes the torques that should be applied to the joints to reach that point. The mapping from x to y is highly nonlinear. The dataset is in `sarcos.mat`,[1] in which you can find N = 44,484 training points and Ntest = 4,449 testing points. For both the training and testing data, the first 21 columns are the input variables, and the remaining 7 columns are the output variables. For simplicity, we following standard practice and focus on just predicting a scalar output, namely the torque for the first joint (that is, the target variable y is taken to be the 22nd column in the data matrix; the first 21 columns are taken to be the feature x).

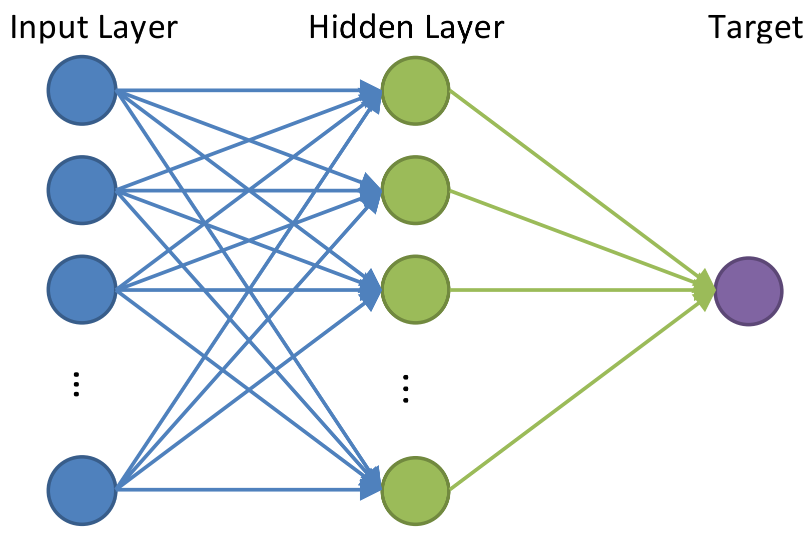

Figure 1: Neural network for regression

A feed-forward neural network is a powerful tool for extracting nonlinear relations in the data. A common neural network model configuration is to place a layer of hidden neurons between the

[1]Data downloaded from http://www.gaussianprocess.org/gpml.

input and output layer, as shown in figure 1. In this example, we use the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$ to do the non-linear transformation. The model above could be represented by

$$\hat{y}(x;\ w, V) = \sum_{i=1}^{h} w_i \sigma(v_i x^\top),$$

or in a compact matrix form:

$$\hat{y}(x;\ w, V) = w\sigma(Vx^\top),$$

where $w = [w_1, \cdots, w_h] \in \mathbb{R}^{1\times h}$ and $V = \begin{bmatrix} v_{1,1} & \cdots & v_{1,d} \\ \cdot & \cdots & \cdot \\ v_{h,1} & \cdots & v_{h,d} \end{bmatrix} \in \mathbb{R}^{h\times d}$, and d is the number of features, and h is the number of hidden units.

(a) **[8 points]** Given data $\{x^j, y^j\}_{j=1}^n$, we train the network by minimizing the mean square error:

$$E(w, V) = \frac{1}{n}\sum_{j=1}^{n}(y^j - \hat{y}(x^j;\ w, V))^2,$$

I tried to implement `nn_MSE.m` to calculate $E(w, v)$ and its gradient w.r.t. w and v, but got stuck with several bugs and did not manage to get it work. Please help debug `nn_MSE.m` to make it work! Like what we did in HW2, you can check the correctness of your gradients by running *gradient_ descent_ checker.m.*

(b) **[8 points]** After you complete `nn_MSE.m`, please complete `nn_train.m` to fit a feedforward neural network using mini-batch gradient descent. Mini-batch gradient descent splits the data into m mini-batches $\{I_k\}$, each of size $b = n/m$, and at each iteration t, it selects a mini-batch I_t and update the parameter via

$$w_{t+1} \leftarrow w_t - \mu_t \frac{1}{|I_t|}\sum_{j\in I_t}\frac{\partial E_j(w, V)}{\partial w}, \qquad V_{t+1} \leftarrow V_t - \mu_t \frac{1}{|I_t|}\sum_{j\in I_t}\frac{\partial E_j(w, V)}{\partial V},$$

where $\partial E_j(w, V)$ is the derivative of $(y^j - \hat{y}(x^j;\ w, V))^2$, corresponding to error on the j-th data point. Run `test_sarcos.m` to test your code; report your training and testing errors as well as your training time (it should be several minutes with an efficient implementation). Use cross validation to pick the number of hidden units.

(c) **[5 points]** Use cross validation to pick the number of hidden units (5 points) in $h = [10, 50, 100, 200]$, and report your training and testing errors.

(d) **[Optional; upto 8 bonus points]** Play with more parameter settings to see what is the best performance you can get. You can play with the hyperparameters in your gradient descent (step size, batch size, etc), or improve your code by adding a L_2 regularization, or use the Matlab neural network toolbox (see `test_sarcos_matlabtoolbox.m` for some example code). You will get 3 bonus points if you achieve a MSE < 9 in the testing data, 5 bonus points if you achieve MSE < 8, and 8 bonus points if you can get MSE < 7! Please describe how you made it in your report, and include a runnable code so that the graders can repeat your result (because neural network training uses random initialization, you may want to fixed the random seed). For this problem, you can also use other neural network toolbox as you prefer.

(e) **[Optional; 15 bonus points for undergrads and 10 for graduate students]** Write a general software that implements general multi-layer neural networks, and test it on this dataset. Your software should work for any number of layers and units (e.g., it should take a vector $[h_1, h_2, \ldots, h_L]$ as input, where h_ℓ is the number of hidden units in the ℓ-th layer). My handout `backpropagation.pdf` on Canvas can help you work out the derivatives.

Problem 2: Decision Trees

We'll use the same data as in our earlier homework: In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

x_1	x_2	x_3	x_4	x_5	y
know author?	is long?	has 'research'	has 'grade'	has 'lottery'	$\Rightarrow$ read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

In the case of any ties, we will prefer to predict class +1.

(a) **[5 points]** Calculate the entropy of the class variable y

(b) **[8 points]** Calculate the information gain for each feature x_i. Which feature should I split on first?

(c) **[8 points]** Draw the complete decision tree that will be learned from these data.

Problem 3: Start Your Project

I want to encourage you to start your project early, so that you will have more time to improve and get better results. For this problem, you will get **10 points** by completing all of the following action items before the deadline of this homework:

(a) Form your team and sign it on Canvas.

(b) If you participate a Kaggle competition, register a Kaggle account and submit an initial testing result to the leaderboard (so you will see your team name on the leaderboard); it does not matter how good your result is, I just want you to get started and warm up!

If you are working on your own project idea, please give a brief description on your problem and the dataset that you will use, and show an initial result (again, it can be the simplest method you can think of). If a large portion of your project is data collection and your data is not available yet, explain it in the homework report.

(c) Write a few sentences on what you will do to contribute to your team; you may end up with doing something different, but I want you to have a clear picture about it.