

ASSIGNMENT 2, due Thurs., Feb. 15

Discussions are encouraged but no written records can be taken away from a discussion. Books and notes can be consulted but not copied from. Homeworks are due in class or in my mail box by 4pm on the due date. For experimental problems, debugging help may be obtained but all code should be your own and all results reported should be from running your own code. In order to verify “curious” results, we may request you submit your code so DO NOT delete your code after you hand in a problem set.

1 (400pts) Neural Networks and Backpropagation

Write a program in your favorite language (even matlab code would be fine) to implement gradient descent (not normalized gradient descent) for a 2 input ($d^0 = 2$), N_H -hidden unit ($d^1 = N_H$), 1 output multilayer perceptron ($L=2$). For each hidden layer node activation function, use the $\tanh(\cdot)$ function. For the output node activation function, allow for both the *identity* function (regression problems) and the $\tanh(\cdot)$ function (binary classification problems). The error measure that we will use is the squared error. Thus for a data set $D = \{x_i, y_i\}_{i=1}^N$,

$$E(\mathbf{w}) = R_{emp}(\mathbf{w}) = \frac{1}{4N} \sum_{i=1}^N (g(x_i, \mathbf{w}) - y_i)^2$$

(Understand how you could easily change your code in order to use different error measures.) Your code should be able to accept a data set and the perform gradient descent on $E(\mathbf{w})$ obtained from that data set. It should also be able to keep track of the error measure on various other sets (for example the training set and a test set). Perform the following steps.

- (a) Write the code to do the above.
- (b) How many adjustable parameters are there for a single hidden layer network with N_H hidden units ($d^1 = N_H$), $d^0 = d$ inputs. Write down the functional form for a function in this learning model.
- (c) Check your gradient calculations as follows.
 - (i) In this part use a network with 2 hidden units ($N_H = 2$). Set all the weights in your network to 0.25. Set the two inputs to 1 ($x_1 = x_2 = 1$) and the desired output to 1 ($y = 1$). For both the *identity* and $\tanh(\cdot)$ output node activation functions, obtain the gradient of the squared error measure ($E(\mathbf{w})$) on this one data point using the backpropagation algorithm. Report this result - the result should be as many numbers as parameters in this network.
 - (ii) Now, perturb each weight in turn by 0.0001 and numerically obtain the gradient. Report this result. (Hint: If this result is not similar to your previous result then there is something wrong with your backpropagation gradient calculation.)

2 (150pts) Training on some Data

We will now see our method working on a “real” data set.

- (a) Two data sets, a training set and a test set, can be downloaded from
http://www.cs.rpi.edu/magdon/courses/csci6100_spring2001/assign.html

- (b) The training set has size 100, the test set has size 1000. Each row is a data point. The first two columns are the two inputs and the last is the output. From the training set, select a set of size 20 randomly. (It would be a good plan to have in your code some method for selecting randomly a given number of data points from the training set.)
- (c) Initialize all the weights randomly in the range $[-0.25, 0.25]$. Run your code (with a $\tanh(\cdot)$ output unit) using a step size (η) of 0.001 and 2000 gradient descent iterations on this set of size 20 using a network with 1 hidden unit. Plot the training and test errors (on the same plot) as a function of gradient descent iteration number.
- (d) Repeat part (b) for 5, 10, 15 and 50 hidden units.
- (e) BRIEFLY summarize your observations and give plausible explanations for the behavior you observe.

3 (150pts) Expected Test Error

- (a) Now select, randomly from the training set, 20 sets of 20 data points each. As in problem 2 (c), perform gradient descent on each training set using 1 hidden unit and obtain the test errors at the end of training. Report the average of these test errors. Repeat this for 2, 3, 5, 10, 15 and 50 hidden units. Provide a plot of this average test error as a function of number of hidden units. (Feel free to explore in more detail the range 1-50 h.u.)
- (b) Articulate exactly what the numbers reported in part (a) signify.

4 (150pts) Regression Problems

So far we have discussed in depth the 2 class decision problem. What about many actions? In fact, what about uncountably many actions indexed by a continuous variable (say) y ? These problems are generally called regression problems. You observe a feature vector x and you predict an action (state) y . Thus the decision function is a function $y(x)$.

- (a) Suppose that the risk associated with predicting state y when the true state is z is now given by a risk function $\lambda(y, z)$. (Note that the risk function is equivalent to what we called the the loss matrix). Argue that the conditional risk is given by

$$R(y(x)|x) = \int dz \lambda(y(x), z)p(z|x)$$

- (b) For the case of squared error ($\lambda(y, z) = (y - z)^2$), show that the Bayes optimal decision function $y(x)$ is given by

$$y(x) = E[z|x] \stackrel{def}{=} \int dz zp(z|x)$$

This function is sometimes called the regression function.

5 (150pts) Boolean Univerasal Approximation

In this problem we will prove a universal approximation theorem for Boolean functions of Boolean input variables, i.e., the only values that the function and input variables can take are ± 1 . The AND, OR and

NOT function are defined as follows:

$$\begin{aligned} AND(x_1, x_2, \dots, x_d) &= \begin{cases} 1 & x_1 = x_2 = \dots = x_d = 1 \\ -1 & \text{otherwise} \end{cases} \\ OR(x_1, x_2, \dots, x_d) &= \begin{cases} -1 & x_1 = x_2 = \dots = x_d = -1 \\ 1 & \text{otherwise} \end{cases} \\ NOT(x_1) &= -x_1 \end{aligned}$$

- (a) Provide an implementation of AND, OR and NOT using a simple perceptron (no hidden layers, with the output unit having a $sign(\cdot)$ activation function).
- (b) Prove the following universal approximation theorem.

Theorem *Any Boolean function can be implemented using a multilayer perceptron with all the activation functions being $sign(\cdot)$*

[Hint: The following theorems may be of use (and can be used without proof)]

Theorem (DNF) *Any Boolean function can be represented as a disjunction of conjunctions (OR of ANDS), where each element in the conjunction can be either x_i or $NOT(x_i)$.*

Theorem (CNF) *Any Boolean function can be represented as a conjunction of disjunctions (AND of ORS), where each element in the disjunction can be either x_i or $NOT(x_i)$.*

]