PROJECT

Due December 17, 2019 (before start of exam)

Almost any process one can imagine can be thought of as computing a certain function, where $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ are the inputs and $\mathbf{y} = \{y_1, y_2, \ldots, y_m\}$ are the outputs. Note that in general there are n inputs and m outputs.

The goal is then to either find that exact function, if possible, or otherwise to find ways to approximate it as closely as possible.

For example, to approximate a nonsingular scalar function $f(x)$ for x within a given range to within some desired accuracy $\epsilon > 0$ means that a neural network can be found such that its output $g(x)$ satisfies

$$\|g(x) - f(x)\| < \epsilon, \ \forall x \text{ in the range.}$$

There is a universality theorem that states that no matter what the function is, as long as it is continuous, one can always find a neural network that approximate it as closely as one wants. All that is needed is a neural network with only a single hidden layer sitting between an input layer and an output layer. The accuracy increases with increasing number of neurons in the hidden layer.

In this project, create this universal computing machine by using a three layer feed-forward neural network with N_0 neurons in the input layer, N_1 neurons in the hidden layer, and N_2 neurons in the output layer. Use the backpropagation to solve a classification problem. Your program must be able to accept rather general values for N_0, N_1 and N_2.

It is very important to make sure that your program is functioning correctly before moving on to explore various modifications to it. First write the basic program to implement the above. Check you program by testing it on the `XOR` logic function:

x_1	x_2	y
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

using $N_1 = 4$ neurons in the hidden layer. Train it using the training set with the four training vector selected sequentially (in the same order as defined above). Use the following initial weights and biases.

The initial weight to the hidden layers are:

−0.3378	0.2771	0.2859	−0.3329	biases to the 4 hidden neurons
0.1970	0.3191	−0.1448	0.3594	weights from the first input neuron
0.3099	0.1904	−0.0347	−0.4861	weights from the second input neuron

The initial weights from the hidden neurons to the output neuron are:

−0.1401	bias on the output neuron
0.4919	weight from the first hidden neuron
−0.2913	weight from the second hidden neuron
−0.3979	weight from the third hidden neuron
0.3581	weight from the fourth hidden neuron

Report the weights and biases after 1 epoch of training starting with the above initial weights and biases, and an α of 0.2. (Note that our definition of the error function is one half of the square of the Euclidean norm of the difference between the network output and the target vector.)

Send your results as soon as possible to our GA to verify that they are indeed correct. **Only** after verifying that your results are correct, you can continue to work on the remaining part of the project.

The project is based on the use of the conventional backpropagation to solve a classification problem similar to what you did in HW05. Input neurons are bipolar, and the bipolar Sigmoid transfer function is used:

$$f_{\text{tansig}}(x) = \frac{1 - e^{-x/x_0}}{1 + e^{-x/x_0}} = \tanh\left(\frac{x}{2x_0}\right)$$

Its total derivative is given by

$$\dot{f}_{\text{tansig}}(x) = \frac{1}{2x_0}\left[1 + f_{\text{tansig}}(x)\right]\left[1 - f_{\text{tansig}}(x)\right]$$

We have introduced a positive parameter x_0, which sets the scale for the argument of the transfer function. Note that the smaller x_0 is, the more abrupt the transfer function switches from -1 to $+1$.

However, for your project, you will need to generalize it to a $N_0 - N_1 - N_2$ network (N_0 input neurons, N_1 hidden neurons, and N_2 output neurons). You can assume here that there is only one hidden layer, and only one output neurons, but N_0 and N_1 are quite arbitrary.

For this project use the following termination condition. During training, if $\mathbf{s}^{(q)}$ is presented to the neural network, and if the network output is y, then the error is given

by $e(q) = y - t^{(q)}$. The square errors are summed at the end of each epoch, and if this sum is smaller than a given tolerance T, then training is terminated. Try using $T = 0.05$.

This termination condition is used for a given T, whether or not we assume a quadratic cost function is used in the BP algorithm. In fact we will use exactly the same termination criterion even when the cross entropy cost function is used instead of the quadratic cost function. The choice of the termination condition is completely independent of the form of the cost function used in the BP algorithm.

Use initial weights and biases that are created randomly and uniformly with values between $-\zeta$ and ζ. Before we chose $\zeta = 1$. Now we need to explore other choice of values as well. Run your program many times for $N_1 = 4$ using a training rate of $\alpha = 0.2$ and a tolerance $T = 0.05$, and you should notice that there are times when the program appears not to converge at all.

To protect your program from getting into an infinite loop, set a value for the maximum number of epochs, I. Your program must never execute more than I epochs even though the above termination condition has not been met. Experiment with choosing an appropriate value for I. Of course if you set too low a value for I, then the program may not have a chance to converge yet before stopping. If I is set to high, then you are just wasting too much computational time. When your program runs exactly T epochs, then it is safe to say your program has not yet converged.

Once you get through to this point, you are ready to actually start exploring the following 4 hyper-parameters of the problem: N_1, α, ζ, and x_0. Perform calculations using the following choices of parameter values: $\alpha = 0.1, 0.2, 0.3$, $\zeta = 0.5, 1.0, 1.5$, and $x_0 = 0.5, 1.0, 1.5$.

Separately for $N_1 = 2, 4, 6, 8, 10$, run your program 100 times, each time using randomly created weights and biases. Use the same training rate and T as before. For a given N_1, report the number of times there is no convergence. For the remaining cases where you have convergent results, report the number of epochs needed, the average and median number of epochs. as well as the minimum and maximum number of epochs. Comment on your findings.

Consider next the case where $N_1 = 1$. What do you find? Can you explain it?

Repeat the above computation by replacing the use of the quadratic cost function with the bipolar cross-entropy cost function (still using the same termination conditions). I have updated the lecture notes on backpropagation by including the bipolar cross entropy function (slide 17). Report your findings. What comments can you make?

For runs that do not converge (for either choice of the cost functions), what was the main problem?

One way for me to check the correctness of your results for the cross-entropy cost function, report the results for the weights and biases after just one epoch for the case where $N_1 = 4$, $\alpha = 0.2$, $\zeta = 1.0$, and $x_0 = 1.0$.