CS1675: Homework 7

Due: 11/15/2018, 11:59pm

This assignment is worth 55 points.

Part I: Computing weight updates by hand (15 points)

In recitation, we show how to compute activations in a neural network, and how to perform stochastic gradient descent to train it. We compute activations for two example networks, but only show how to train one of them. Show how to train the second network using just a single example, `x = [1 1], y = [0 0]` (note that in this case, the label is a vector). Initialize all weights to 0.05. Use a learning rate of 0.3. Include your answers in text form in a file `report.pdf/docx`.

Part II: Training a neural network (25 points)

In this exercise, you will write code to train and evaluate a very simple neural network. We will follow the example in Bishop that uses a single hidden layer, a tanh function at the hidden layer and an identity function at the output layer, and a squared error loss. The network will have 30 hidden neurons (i.e. M=30) and 1 output neuron (i.e. K=1). To implement it, follow the equations in the slides and your textbook.

First, write a `function [y_pred, Z] = forward(X, W1, W2)` that computes activations from the front towards the back of the network, using fixed input features and weights. Also use the forward pass function to evaluate your network after training.

Inputs:

- an NxD matrix `X` of features, where N is the number of samples and D is the number of feature dimensions,
- an MxD matrix `W1` of weights between the first and second layer of the network, where M is the number of hidden neurons, and
- an 1xM matrix `W2` of weights between the second and third layer of the network, where there is a single neuron at the output layer

Outputs:

- [5 pts] an Nx1 vector `y_pred` containing the outputs at the last layer for all N samples, and
- [5 pts] an NxM matrix `Z` containing the activations for all M hidden neurons of all N samples.

Second, write a `function [W1, W2, error_over_time] = backward(X, y, M, iters, eta)` that performs training using backpropagation (and calls the activation computation function as it iterates). Construct the network in this function, i.e. create the weight matrices and initialize the weights to small random numbers, then iterate: pick a training sample, compute the error at the output, then backpropagate to the hidden layer, and update the weights with the resulting error.

Inputs:

- an NxD matrix `X` of features, where N is the number of samples and D is the number of feature dimensions,
- an Nx1 vector `y` containing the ground-truth labels for the N samples,
- a scalar `M` containing the number of hidden layers to use,
- a scalar `iters` defining how many iterations to run (one sample used in each), and
- a scalar `eta` defining the learning rate to use.

Outputs:

- [10 pts] `W1` and `W2`, defined as above for `forward`, and
- [5 pts] an `iters`x1 vector `error_over_time` that contains the error on the sample used in each iteration.

Part III: Testing your neural network on wine quality (15 points)

We will use the Wine Quality dataset from HW3 to test the neural network implementation. Write your code in a script `neural_net.m`.

1. [2 pts] Load the wine dataset, define the train/test split as in HW3, and set the number of hidden units, the number of iterations to run, and the learning rate.
2. [3 pts] Call the `backward` function to construct and train the network. Use 1000 iterations and 30 hidden neurons.
3. [5 pts] Then call the `forward` function to make predictions and compute the root mean squared error between predicted and ground-truth labels, `sqrt(mean((y_test_pred - y_test).^2))`. Report this number in a file `report.pdf/docx`
4. [5 pts] Experiment with three different values of the learning rate. For each, plot the error over time (output by `backward` above). Include these plots in your report.

Submission: Please include the following files:

- `report.pdf/docx`
- `forward.m`
- `backward.m`
- `neural_net.m`