Homework 1: a fully connected feed-forward network

Abstract

In this homework, the focus was on finding the set of hyperparameters for a simple 2-hidden layer, fully connected feed-forward network which lead to the best performance on a given set of data. Such hyperparameters are the activation function, the loss function (and so, the employed regularization), the number of hidden units in the two hidden layers. The evaluation of the network's performance is done by computing the mean squared error on a test set.

In addition to this, I also extended the given model by allowing an arbitrary number of hidden layers, and by implementing a series of utilities (grid search, random search, k-fold cross validation) which make the hyperparameters tuning process less tedious.

Code development

Instead of simply using the provided Python script, I wrote a brand new class, called `My_net`, which allows the user to specify the architecture of the network by providing a list of integers as input (so that the i-th entry specifies the number of hidden units in the i-th layer), the activation function, the regularization (either L2, L1 or none at all) and the regularization coefficient.

In order to probe the hyperparameter space, I also wrote a grid search function, which trains a network for each combination of the provided hyperparameters, and a random search function, which instead draws the hyperparameters from a compact set provided by the user. The random search is more effective than the grid search, as it is able to explore more of the hyperparameter space; furthermore, as the number of parameters increases, a grid search becomes less and less feasible, as it results in too many training routines.

The random search was performed by using a k-fold cross validation; this, together with an early stopping clause, allowed to avoid overfitting during the choice of the parameters.

Once the best parameters were chosen, they were collected in a dictionary and were fed to a specific function, which trains the network specified by that dictionary. After the training (which, by default, has a learning rate decay), the weights are saved by a dedicated function.

Another Python script, `trained_model.py`, then initializes the network, loads the weights and calculates the mean squared error on a file named `test_set.txt`.

Results

After some trials, the first result was that, for the task at hand, the L1 regularization is not particularly suitable; the main purpose of L1 regularization is to remove unnecessary features from a model, but in this case it only results in models not being powerful enough.

In the end, the best results are achieved by either using L2 regularization with a very small (~ 0.01) coefficient, or no regularization at all. As far as learning rates go, values larger than ~ 0.01 tend to result in numerical overflows; this is even more stark if one employs L2 regularization. The best activation functions seem to be ELU, ReLU and sigmoid, with ReLU being the slight favorite.

The following plots and results are referred to the set of parameters:

- hidden layers: $h_1 = 119$ and $h_2 = 192$;
- activation function: ReLU;
- regularization: no regularization;
- number of epochs: 3000.

The learning rate was set to 0.0091, with decay at each epoch.

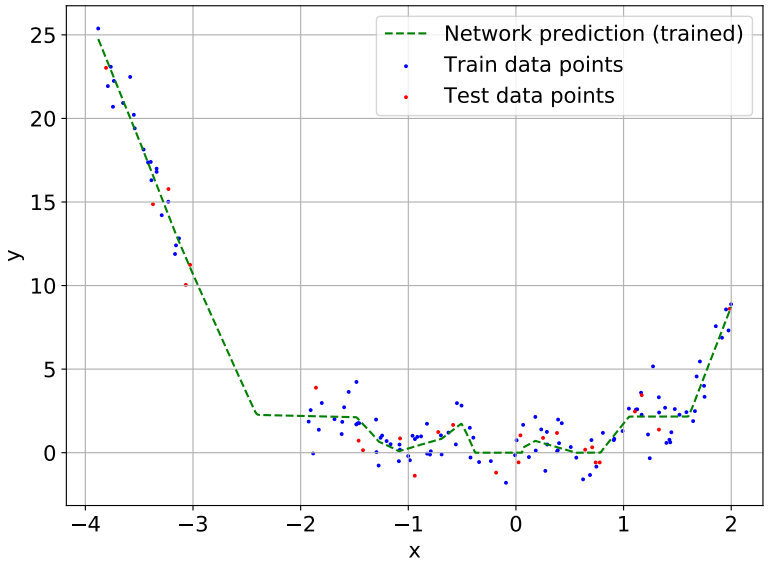


Figure 1: The predictions of the trained network and the provided points.

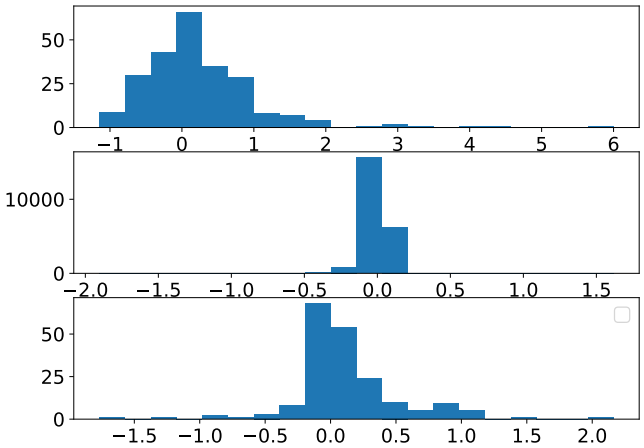
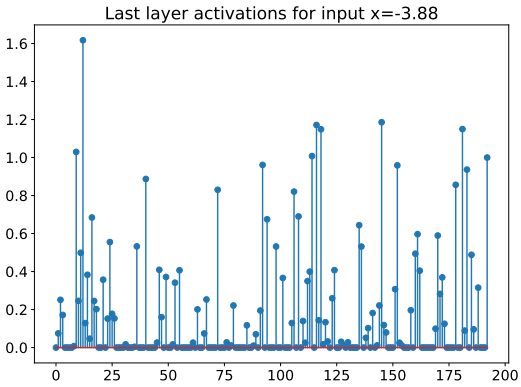
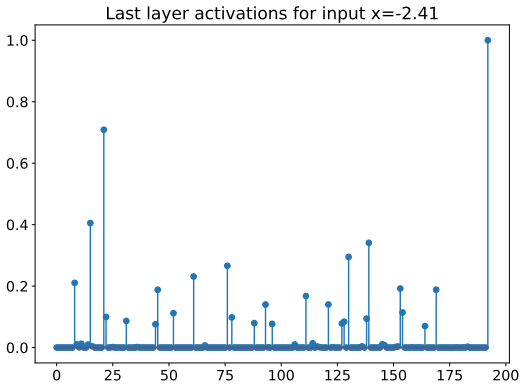


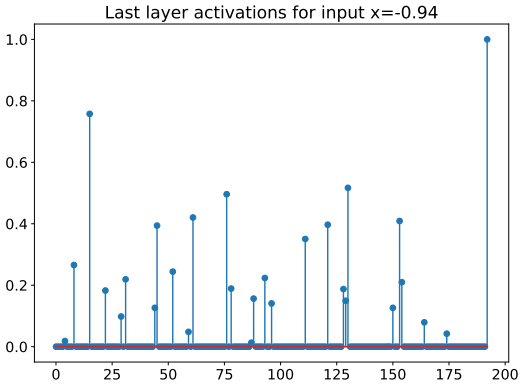
Figure 2: The histogram of the weights of the trained network.



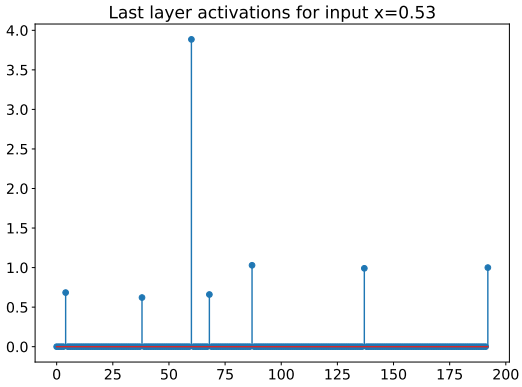
(a) $x = -3.88$



(b) $x = -2.41$



(c) $x = -0.94$



(d) $x = 0.53$

Comments

The network seems to be capable of grasping the almost 'sinusoidal' behavior of the training data. Where there are no training points, however, the network does not seem to 'waste' any parameter to avoid the sharp kink. This can be ascribed to the lack of regularization, which, in turn, resulted in higher losses when compared to this unregularized model.

Indeed, testing this model on the missing data will probably result in poor performance; using some degree of regularization may improve this, but I preferred not to make any assumption about the missing data, and I simply kept the model which best performs on the given test data.