Machine Learning for Embedded Systems

Home assignment 1 – Neural Network in C

Name: Ondřej Schejbal

UNI-ID: onsche **School code:** 214308IV

GitLab repository: gitlab.cs.ttu.ee/onsche/ias0360

Description of how the task was solved

The neural network consists of 4 layers. Input layer, two hidden layers and one output layer. Input layer consists of 3 nodes for each feature that the input data point can have (temperature, humidity and air quality). The first hidden layer consists of 5 neurons. The second hidden layer consists of 4 neurons. All neurons in hidden layers use ReLu as their non-linear activation function. The output layer has only one neuron which uses the Sigmoid function and outputs either 1 or 0. The outputs expresses if the provided input values are suitable for human life or not.

During the first 4 labs I developed the core code base for this homework. The code consists of functions focused on performing necessary mathematical computations in most cases working only with matrices.

My code can be divided into 4 parts:

1. **Forward propagation**

   This part takes the current values of the weights and biases and computes neurons potential values and the output of neurons activation function. This is applied layer wise. Forward propagation is done for each input separately, so my implementation does the forward propagation for each input data point before progressing to the next phase. At the end of this step the cost value is computed (using the formula for logistic regression cost function), which represents the size of an error over all input data.

2. **Backward propagation**

   In this part all the collected outputs from the forward propagation of all the inputs are taken and using the formulas from the lectures I compute the values of dW, dZ and dA which are the derivatives propagated back through the whole network.

3. **Weights and bias update**

   The propagated values from previous step are used to update the weights and biases in the whole network. The update is dependent on the learning rate value.

4. **Steps 1-3 are repeated until the cost function's value (computed at the end of step 1) is low enough**

I have then put all the above steps into a for-cycle, where each cycle represents an epoch in the training process.

Results

I was able to confirm that all the computations in my implementations are correct. I have computed 2 whole epochs by hand and confirmed that all the computed matrices are correct and with correct dimensions.

I have experimented with the maximum number of epochs in the training process, the threshold of accepted cost value and also the learning rate.

The main meaning of the **maximum number of epochs** is to limit the maximum number of iterations, so the program always ends, but also let the process converge to the minimal value.

I have also set the **minimal number of epochs** in order to more explore problems state space.

To avoid passing and missing the local minima I have used multiple values for the **learning rate** parameter – from 0.001 to 0.01. From those I was able to find that the lowest cost value and the fastest learning time were for the learning rate of value 0.01.

I was looking at the evolution of cost value in time and determined that the threshold equal to 0.0002 is the optimal value to use, because then the value often starts rising again and the NN would miss the local minima.

For my testing I have prepared dataset with 11 data points. I have tried to use values which would make sense with respect to the meaning of the features. I have then prepared 1 input, that the neural network has never seen before and tried predicting the outcome using the trained NN. The output was predicted correctly.

Issues

After more experiments I was able to find some input data points which would be predicted incorrectly. The main cause is probably small amount of data in the dataset. With a proper (much bigger) size of the dataset the NN weights could be more optimized and therefore the predictions would perform better on new data.