Machine Learning - Homework 4

Tim Kmecl, 25.4.2024

Implementation

I first implemented ANNRegression as a standalone class, with all the requirements from instructions. I made sure not to regularize the biases. It specifically includes a function for the forward pass after the last layer that is just the identity, a function for the cost function without regularization (combined cost function with regularization is computed in a separate function using this one), and a function for preprocessing the target vector before training (it just reshapes it). Loss over all the dataset is mean of losses of individual samples, and sum of squared weights id divided by their number to make regularization's contribution to loss independent from the size of the network.

Most of the code would be the same for classification – backward pass is the same, forward pass only differs at the last layer (softmax has to be applied), and the cost is different.

ANNClassification is therefore implemented as a subclass of ANNRegression. It inherits all its functions, only changes required are to the three functions mentioned above. Last layer has to compute softmax, loss is now logloss, and input vector of target variables is preprocessed to make it one-hot encoded, so its dimension matches the dimension of the last layer. All in all, the code for ANNClassification (which is all that is required to change a regression model to a classifer) is about 10 lines long.

Numerical verification

I wrote a function that computes the gradient of a given model for given data numerically. It compares it to analytic gradient and computes the relative size of their differences to their average sizes. If those differences are small enough, we can know that the cost and the gradient are compatible.

I computed this for both classifier and regression both before and after fitting, with epsilon 10^{-6}

When computing gradients before fitting (with random weights), the average relative difference of a partial derivative was below 10^{-8} and the maximum difference below 10^{-7}. When evaluated after fitting, both averages and maximums of relative differences were below 10^{-4} (here, they may be larger due to smaller absolute values). The small difference between both methods shows that the cost and the gradient are compatible, for both models.

Housing data

Regression

I compared ANNRegression with Linear Regression from sklearn, since we haven't implemented any regression models on our own. I used 4-fold cross validation, loss used for evaluating was MSE.

I compared logistic regression, NN without hidden layers and NN with 1 hidden layer of 5 units and regularization factor 0.1.

Both LR and NN without hidden layers have mean loss of 42.0 with SE of 1.46. This is expected, since the approaches are equivalent – this speaks in favor of correctness of my implementation. Adding one hidden layers improves the loss to 29.1 (SE of 0.83).

Classification

I compared ANNClassification to my implementation of MultinomialLogReg. I again used 4-fold cross validation, this time with log loss.

MultinomialLogReg achieves loss of 0.321 (SE 0.033), whereas ANNClassification with 1 hidden layer of size 3 and regularization factor 0.5 improves it to 0.297 (SE 0.022). Compared to the previous example, the improvement here is very small, with the difference being similar in size to one SE. I tried to improve it by trying several different combinations of hyperparameters, however with no success.

Final predictions

I unzipped and loaded the provided data. Data was shuffled since the classes appear one after another in the file. For evaluation, 4-fold CV was used. The data was standardized inside of each fold.

First I ran Multinomial Regression on it to use the result as a baseline. It reaches mean log loss of 0.644 (SE of 0.005) and accuracy of 76% (SE of 0.2%).

I then tried several different hyperparameters for NN. I didn't try to use more than one hidden layer because of the already long runtime with just one, so the hyperparameters left were regularization rate and number of hidden neurons. I used just one 75-25 train-test split for quick evaluation of different combinations (first split of 5-fold CV from above), again with standardized data. The goal was to improve the log loss. I was constrained due to long runtime caused by running the code on a slow laptop.

I decided to use a hidden layer with 20 neurons and learning rate 0.3. This is an approximate solution – using more precise changes, they could certainly be improved. I also didn't use grid search, which could help find more exact optimum, because of long runtime of fitting the models.

Evaluating this model using 4-fold CV with same splits as Regression gives mean log loss of 0.559 (SE of 0.004) and accuracy of 79% (SE of 0.2%). Since the model is trained on less data than the final model this may be a little pessimistic, however the dataset is quiet large. Moreover, hyperparameters could be overfitted to this data which would make the evaluation optimistic, however those two parameters I was changing should not contribute to overfitting too much especially without exact search for optimum.

New model was then trained on the whole standardized training set. The test set was then standardized using the same parameters, and predictions given by the model saved to the file final.txt in CSV format using pandas.