Fundamentals of Machine Learning: Assignment 1 Report

Initial workflow

The first thing I did when reading in the data was to plot the different features and look at their histograms. I wanted to try understand the different features the data had only by looking at their ranges and variances. I calculated the variances of each feature and saw that two of them were really low compared to the others. Since features with low variance do not contribute much to the outputs, I kept in mind that it would be good to try removing them as pre-processing to see if the network would perform better.

Most of my time was spent on what is mentioned in the assignment brief as creative optimisation. I tried training networks with different training functions: 'trainbr', 'trainlm', 'traingdx', 'trainrp'. I looped through different number of layers, e.g. 2,3,5 and 7, and different sizes of the layers, e.g. 5,10,15,20 to see how the network would perform. I also tried playing with the parameters of some of the training methods: for traingdx, there is a learning rate and a momentum parameter which I changed in the same fashion as the loops for the architecture of the network. I couldn't see much difference in the performance, and I found out that was due to the fact that the hidden layers weren't actually connected. Once I was using the correct hidden layers argument, I realised that I didn't need more than two layers for my network to perform okay because otherwise it would be really slow and overfit.

After trying out different training algorithms, I decided to stick with 'trainbr'. It seemed to be the best one, although I was still getting quite a bad MSE: around 30 000. I then decided to work more on pre-processing.

Pre-processing

Pre-processing seemed to be really important when training a good network. I tried normalisation of the data because there is no good reason not to do it, and my network started having slightly better MSE, but it would be still above 28 000. That was when I normalised only the input. Normalising the output would give me worse performance. I also tried doing PCA by calculating the eigen values of the features and then choosing the three features with the largest eigen values. The performance was still the same and I decided to plot the different features to see which ones had the best eigen values. I found out that one of them was the feature that I had calculated the variance at the beginning to be low. I then decided to try removing the features with the low variance manually instead of doing it via their eigen values. When I did that, my performance was slightly better (although again, not that improved).

I also tried using the pre-processing functions that are part of the toolbox. The defaults were mapping matrix row minimum and maximum values to [-1 1] and removing matrix rows with constant values. When training a network without any, it would perform significantly worse. Nevertheless, when trying mapping matrix row means and deviations to standard values or processing rows with principal component analysis, the performance almost did not improve.

Network architecture and free parameters

One change that I did improved massively my performance: it went from almost a constant 30 000 (even with pre-processing) to 20 000 and less. It was changing the 'mu' parameter of the training function. When training my data, I noticed the mu would stay around the same value (e.g. 30) during

the whole training. The default values have a big range so it was possible that the network was overshooting. I made the range smaller (0.4 to 6) and the network performed really well.

Although I tried optimising the learning rate and the momentum of the network, it didn't produce any results: the performance didn't drastically change. Moreover, these parameters are part of the 'traingdx' training function, but not 'trainbr', and the latter was doing better performance overall. That's why I limited myself to only optimising the architecture of the network and the mu parameter. As I've mentioned in the first section, by trial and error, I found out that a network with one layer would perform better than a network with two layers, or three. Changing the number of neurons in the layer wouldn't improve the performance a lot. I was satisfied with a layer of 35 neurons: it seemed it was giving the best performance.

K-fold cross validation

Doing the right validation of the network was something that was vital for coming up with a good model. The default validation that the toolbox is doing is randomly getting training, validation and testing sets of the data which is not an optimal solution. It would perform hold-out cross validation by using the testing set at the end of the training , meaning, it wouldn't use all of the data for testing. K-fold cross validation is a better method because it makes sure that all of the data is used at least once for training, at least once for validation, and at least once for testing. I used a cvpartition function to get indices for a training and a testing set and then arranged my data to include first the former and then the latter. Then, I made the network divide the input by indices and specified the indices for the testing set to be the same as the ones for the partitioned data. I trained my networks by looping through the number of sets, which in my case were five, and then averaging the performance for all of them. Five seems to be a good number since it makes the data to be equally represented and doesn't use much processing power compared to a leave-one-out cross-validation.

An important point to make is that 'trainbr' is not using a validation set and the default maximum failures parameter is set to 0. I trained some networks with 6-10 maximum failures, and a validation set, but the performance didn't get better. That is why I kept the default maximum failures value and divided my data only to training and testing. I set my network to give performance only of the testing set because that is what we are ultimately interested in. Looking at that performance and also the performance over training and testing, the latter would be always slightly better which is normal because training performance means performance on already seen data.

Critical evaluation

I lost valuable time building my network and setting up cross-validation. I should have focused more on pre-processing techniques. What I could have done is reducing the number of rows and thus reducing the noise of the data. The big variance of the different folds of my models is the justification to why I believe noise is the problem. I could have also removed outliers of the features.

Overall, I think my network won't perform very well. Although I worked a lot to try optimising the architecture of the network, it didn't seem to significantly influence the performance.