CSCI4622 Kaggle 03 – German Sign Classification

Steve Su, Kristian Montoya

Team name : OK Computer

4/20/2020

EDA and Preprocessing

Exploratory data analysis was brief and consisted of displaying a random selection of observations so we could see what the images looked like. Some images were almost black and was almost impossible to tell what they were. The raw image data was given to us as a numpy array of size, 32x32x3. The rgb values were between 0 and 255. As such, we normalized the data to be between 0 and 1 by dividing the values by 255. The labels were given as values between 0 and 42. By plotting a histogram of the labels it seemed that there was some disparity in the number of observations for each label. Some had over a thousand examples where others had only a few hundred. This might make some labels more difficult to classify. In order to use a softmax classifier, we hot encoded the labels. Lastly, the given data (n=39209) was shuffled and split into training and validation groups where validation was 0.10 of the total.

Model

Since the problem involves image classification our top candidate was a CNN. CNNs are better at image classification compared to other methods because the convolution layers retain information with adjacent pixels. Other models which flatten the feature array will loose information. The first model contained 4 convolutions with layers 32, 64, 128, 64. The filter was kept constant, 3x3. Padding was set to "same" so that the output size would be the same as input size. The activation was set to sigmoid in each case. Each convolution block was followed by a maxpooling layer with filter, 2x2. The classifier was a three layer dense NN with number of neurons 64, 32, and 43, respectively. The hidden layers used sigmoid activation and the output layer used softmax. The optimizer was set to SGD and the loss function was categorical cross entropy. This was the base model and several variations were created to see how different changes improved the performance. There was no real real reason for choosing this architecture except for seeing similar forms in reading. The keras and tensor flow libraries were used to create these models.

Results

The initial model which consisted of a chain of sigmoid activation functions did poorly. The loss and accuracy was not improving after each epoch, even with changes to the number of layers, learning rate, momentum, and dropouts layer. The reason for this is thought to be vanishing gradients. This article lead us to this theory, https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b. To confirm, the number of layers with sigmoid function was reduced from 5 to 3. By doing this the accuracy increased from 0.1 to 0.97, an amazing improvement. Also by changing the original model from a chain of sigmoids to relus, a similar improvement occurred. Another method of improvement was to add block normalization just after the sigmoid layers. Again the accuracy shot up to 0.98 with this feature. This even helped the model with reclus, but to a much lesser extent, 4% improvement.

These results suggest that when using a series of sigmoid functions block normalization is advised after each layer. Furthermore, these series of experiments show that our larger initial model was not necessary since we can get similar results with the reduced model of only two convolutions and one dense layer followed by the softmax output layer.

The sweet spot for learning rate was near 0.1 for both the relu and sigmoid based models. Both models achieved a validation accuracy of 0.99. At a smaller learning rate of 0.001 there was a slight dip in accuracy but only about a percent difference. At the other end, accuracy plummeted when learning rate was raised to 1 for the sigmoid model and >1 for the relu model. The relu has a slightly larger range where accuracy is stable. This might make relu a better choice compared to sigmoid function.

For the most part the test and validation accuracy were very close to each other. Therefore we chose not to add a dropout layer for regularization. Momentum and Nestrov were set to zero since the loss was gradually approaching zero without it.

Summary

A relatively small model of 2 convolutions, 1 dense and 1 output layer could achieve the same performance as our original larger model which was approximately 2X the size. Our performance accuracy was nearly 0.99 with validation data. Using the relu activation function showed to be more robust versus the sigmoid function. If using long chain of sigmoid functions, block normalization should be used. Relu's performance did not suffer drastically without the use of block normalization, at least with the given data set. Relu also had the advantage of having a wider range of learning rates where accuracy performance was stable.

Contributions

Steve - EDA, design and optimization of full model, report writing

Kristian – design and optimization of reduced model, report writing