Haritha Ramachandran

Handwriting Recognition

Introduction

The data for the analysis consists of handwritten digits, i.e., grayscale (16x16) grid representations of image scans of the digits “0” through “9” (10 digits) collected from the United States Postal Service (USPS). The data is divided into train and test data, containing respectively 7291 and 2007 observations. The task is to predict the digit type, using 265 input features corresponding to the 16x16 grid representation of the images and to tune the model to increase the performance.

Data analysis

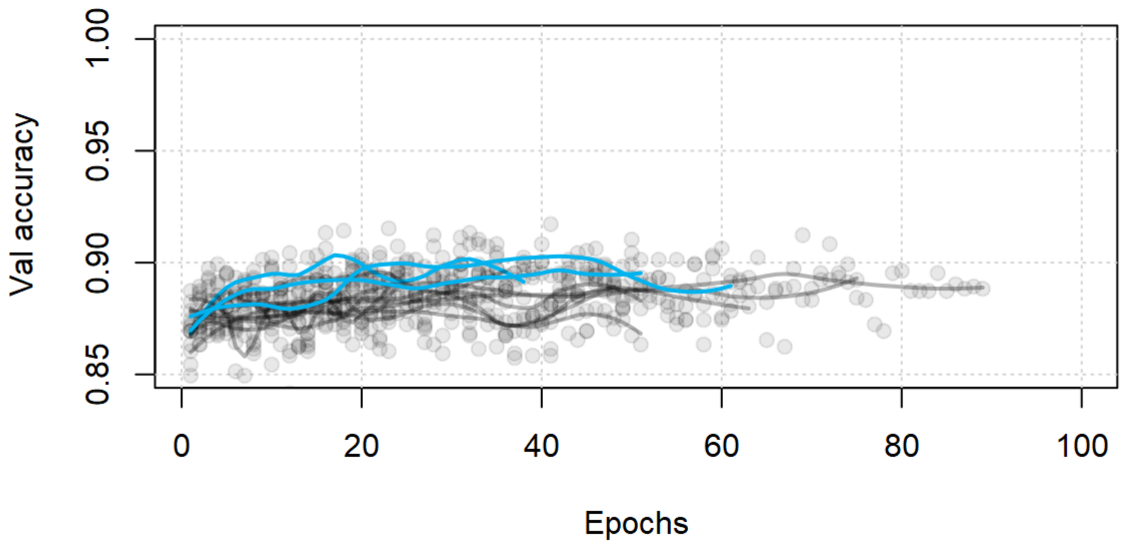
Data preparation

- First step is to load the necessary packages and the data. Since we use Keras platform to implement deep learning methods, we need to load the Keras package.
- We convert the target variable to one-hot coding (both the test and train data).
- We range normalize the input variables, i.e., x_train and x_test.
- Then split the test data into test and validation data.

Fitting and tuning of model

- Load the tfruns package in order to tune the model configuration effectively.
- Package tfruns helps us to run multiple models with different configurations for tuning the model.
- Next, we define flags to define key hyperparameters and settings which vary for each run of the model and then the training and evaluation is performed over the combinations of those flags to determine which one yields the best model.
- Here we need to tune the number of nodes and the dropout rate. Hence ,we declare these in the flags.
- Next, we define the model and we use the functionalities in Keras in order to define multilayer neural network with 2 layers. We use the ReLU activation for the 2 hidden layers and softmax output activation for the output layers. The Adam optimizer is used, and dropout regularization is added to both the layers.
- The model uses the default dropout rate and the number of units from the flags.
- Fit the model using the fit function to the training datasets and number of epochs of the training procedure as 100. We also evaluate the performance of the model using the validation datasets.

- Also store the accuracy of the model on the test dataset for each run and plot the learning curve.
- Now, to tune the model, we run a function tuning_run in another file which runs all combinations of the hyperparameters that we define.
- The function performs a grid search and we specify an argument sample (0.2 or 20%) which is used to perform a random search over the combinations of the grid and thus helping to reduce the number of models.
- We define 4 values for dropout and 2 values each for number of nodes in the grid.
- Also define the name of the folder to which all the runs will be stored using the argument runs_dir in the running_run function.
- Once the function is run, the model in run for different random combinations of the number of nodes and dropout rate and determine optimal configuration. The learning curve plotted is given below.



- The plot shows the validation accuracy against the number of epochs. The blue lines show the runs with the highest validation accuracy at the convergence.
- We can get an output of the runs with accuracy greater than 87% select the best run from that using ls_runs function.
- From the output of this function, we can see that the model with the highest validation accuracy (which is 90.1 %) has 256 nodes in the first layer, 64 nodes in the second layer, dropout value of 0.0 in the first layer and 0.5 in the second layer. Hence this is the best model and has 91.83% test accuracy.

Conclusion

We defined a two-layer neural network and deployed tuning of the number of nodes and the dropout rate in order to find the best model. The best model is the one with 256 nodes on the first layer, 64 on the second, 0.0 dropout rate on the first layer and 0.5 on the second layer.