Deep learning project

Haritha Ramachandran

Abstract

The analysis focuses on the prediction of the daily/sports activity recognition from movement sensor data using various model configurations and chose the best model. The dataset if comparatively large 3-dimensional data with dimension 7600x125x45 for the train data and 1520x125x45 for test data. Hence, we incorporate PCA in order to reduce the dimension and fit various models to the new data. First a two-layer keras sequential model is fitted and the performance is evaluated, followed by the fitting of a three-layer sequential model and its evaluation. The models are compared for their performance in the test data.

Then due to overfitting in the data, different methods or regularization are introduced in both the models by tuning so that best combination of the hyperparameters can be chosen simultaneously. Both the tuning models are compared with accuracy vs epochs plots and the best model configurations for both two-layer and three-layer models are evaluated. These models are compared so that a decision can be made on whether the two-layer or three-layer model is the best for prediction.

Introduction

The dataset contains motion sensor measurements of 19 daily and sports activities, each performed by 8 subjects (4 female, 4 males, between the ages 20 and 30) in their own style for 5 minutes. The training data includes 400 signal segments for each activity (total of 7600 signals), while the test data includes 80 signal segments for each activity (total of 1520 signals). The target variable contains the activity labels. The data are organized in the form of 3-dimensional arrays, in which the first dimension denote the signal, the second the sampling instants and the third the sensors, so each signal is described by a vector of 125 × 45 = 5625 features.

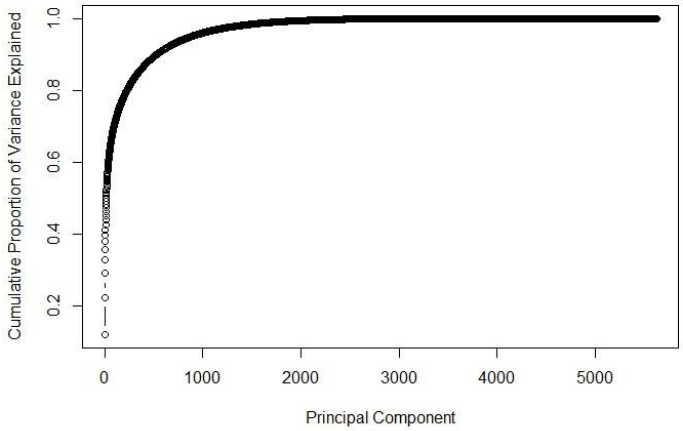
Hence the training data set has dimension 7600x125x45 and the test data has dimension 1520x125x45. The aim of the analysis is to use a predictive model in order to predict the daily/sports activity recognition from movement sensor data using various model configurations. Also, the best model configuration is to be chosen which performs the best.

Methods

Data pre-processing

In order to proceed with the keras sequential model, we first reshape the train and test into twodimensional data so that we can apply PCA to reduce the dimension. After reshaping, the train and test data are normalized i.e., now the values of the data lie between 0 and 1. The test and train data of the target variable contains the daily/sports activities. So, we convert these variables to one-hot coding using the function to_categorical() so that now the train and test data has dimensions 7600x19 and 1520x19 where the column corresponds to the 19 activities.

Next, we use Principal Component Analysis in order to reduce the size of the train and test data using the function pca(). From the cumulative scree plot, we can infer that around 500 features are important in the analysis and thus we include them and discard the rest. Now the dimensions of the train and test input data are 7600x500 and 1520x500.



After extracting the important features, we split the test data into test and validation in order to evaluate the performance of the models.

Model configuration

Model 1

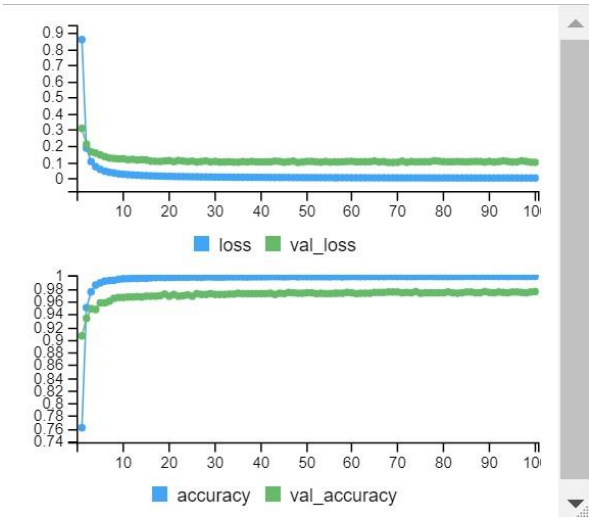
First, we define a keras sequential model with two hidden layers and evaluate the performance of the model on the test data. The model has two layers with ReLU activation and one output layer with softmax output activation. We define the model with 300 units on the first layer, 150 units on the second layer and 19 units in the output layer. In the training configuration we consider standard stochastic gradient descent for optimization, and cross-entropy as error function, with accuracy as performance measure.

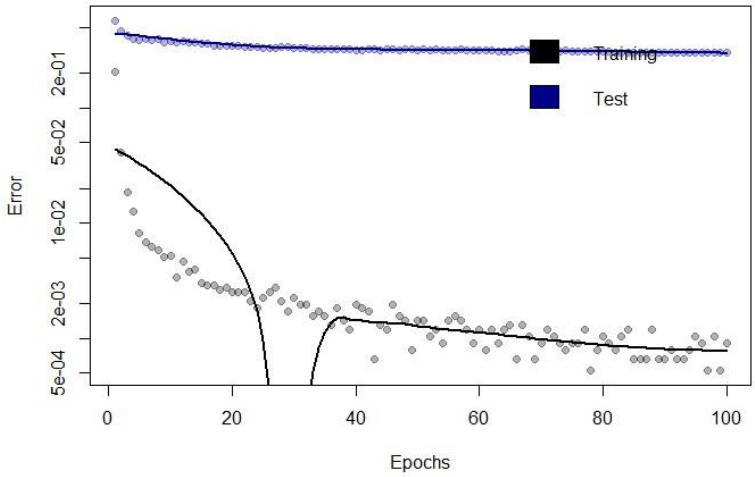
After defining the model, we use the function fit() to train the model. The function allows to specify the number of epochs of the training procedure which is the number of full scans of the data points in the training set. Here we consider 100 epochs. The function allows also to evaluate the

performance of the model on a separate validation set at each epoch, to monitor the progress during the training process. The validation data can be provided by inputting a list to the argument validation_data. But here we use validation_split which splits the training data in the given proportion. Argument verbose = 1 produces an interactive visualization of the training procedure and batch_size is number of observations used for each update of the parameters.

The performance of the model can be evaluated on the test data using the function evaluate() and the output of this will gives us the performance metric i.e., the loss and accuracy and also using a plot of training and test error of the model.

The test accuracy of the model is 0.5723 .And the minimum training and test errors of the model is 0.000526309 and 0.297368407 respectively.

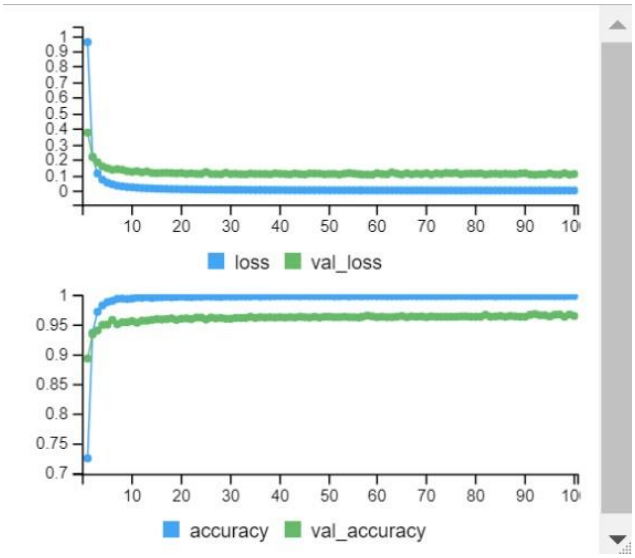


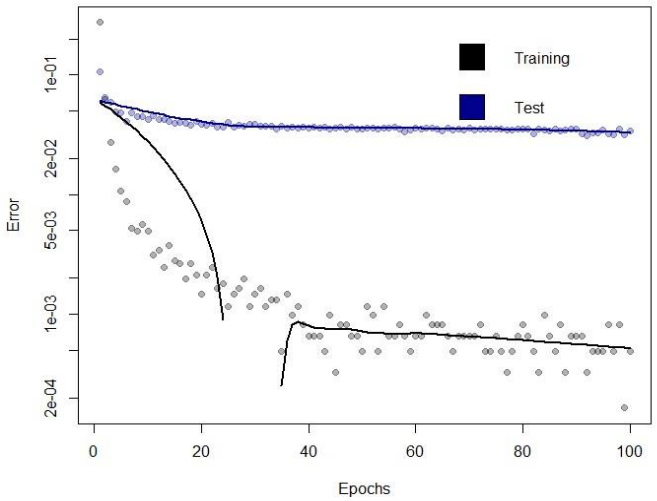


Model 2

Next, we define a keras sequential model for three hidden layers to compare the performance between two layers and three layers models. Similar to the two-layer model, ReLU activation is use in the three hidden layers and softmax output activation in the output layer. The number of units considered in the three layers are 300,150 and 70 respectively and 19 units in the output layer which is the number of columns in y_train. .Similar to previous model, standard stochastic gradient descent for optimization, and cross-entropy as error function, with accuracy as performance measure are used.

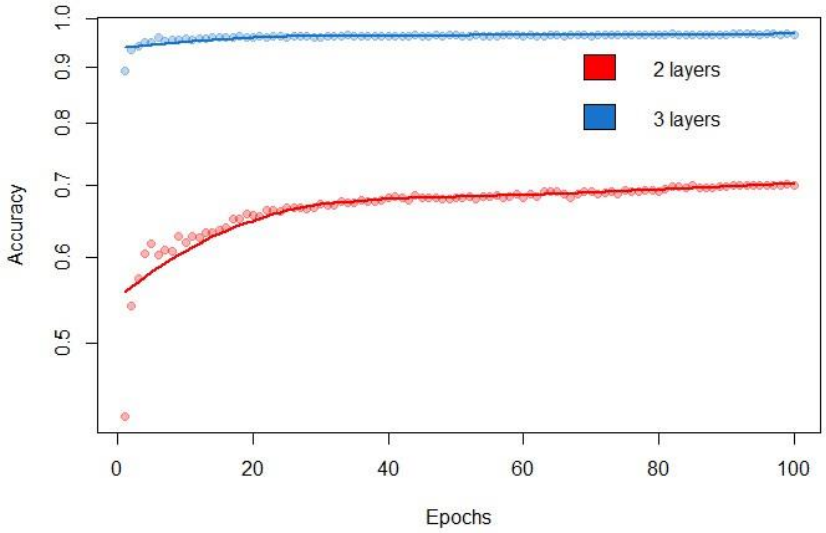
After model definition, it is fitted on the train data with number of epochs as 100, validation_split 0.2 and the performance of the model is evaluated on the test data. The accuracy of the model is 0.746 and the minimum training and test errors of the model is 0.0001644492 and 0.0309210420 respectively . A plot of the test and training error is also plotted.





Model comparison

We can compare the two models using a plot of their test accuracies with the number of epochs.



Model tuning

From the plots of train and test error for the two models, there were signs of overfitting which was indicated by the wide gap between the errors. This is due to the better performance of the model on the training data compared to the new unseen test data and it indicates that the model performs poorly when fitted on an unseen data. In order to compensate for the poor generalization capacity of the models, we introduce various regularization methods such as penalty regularization ,

dropout, early stopping etc. In other words, regularization is the process of adding external information to the machine learning algorithm in order to reduce/prevent overfitting and reduce the generalization error. These methods help to make the training error small and reduce the gap between train and test errors.

In order to select the optimal values of these hyperparameters, we deploy the method of model tuning. The tuning is done on the validation set since for any hyperparameter to have an impact on the effective capacity of a learning system and on its performance. Tuning is basically the decision steps that we take during the deployment of the deep neural network.

The model is tuning is deployed using the function tuning_run from the package tfruns. Here we input sets of possible values of hyperparameters and attach the file which contains the model definition and fitting. This package allows to run models with different configurations which is used to tune the model setting and hyperparameters given in the function. The package also uses “flags” to denote hyperparameters and settings which vary from one run to other. Flags are defined for key hyperparameters, then training and evaluation is performed over the combination of those flags to determine which combination yields the best model.

In this case we are going to tune the depth (number of layers), number of nodes (hidden units in each layer), batch size, dropout rate and learning rate. The search strategy used is “grid search” where a grid of possible hyperparameter configurations are given after plausible ranges have been set. An argument sample is used to perform a random search over the combinations of the grid, thus running a random sub-sample of the possible configurations and reducing the number of models. Here we use the sample size as 10%. The function also contains the argument that specifies the path to the training script containing the baseline model configuration settings. We also use two model configurations, one for two-layer model and the other for three-layer model. Thus, we will have two tuning functions.

After defining this function and running the model configurations, we can find the results and information about the runs in the local directory. It can used to plot the learning scores and produce the validation accuracy learning curve.

Model configuration

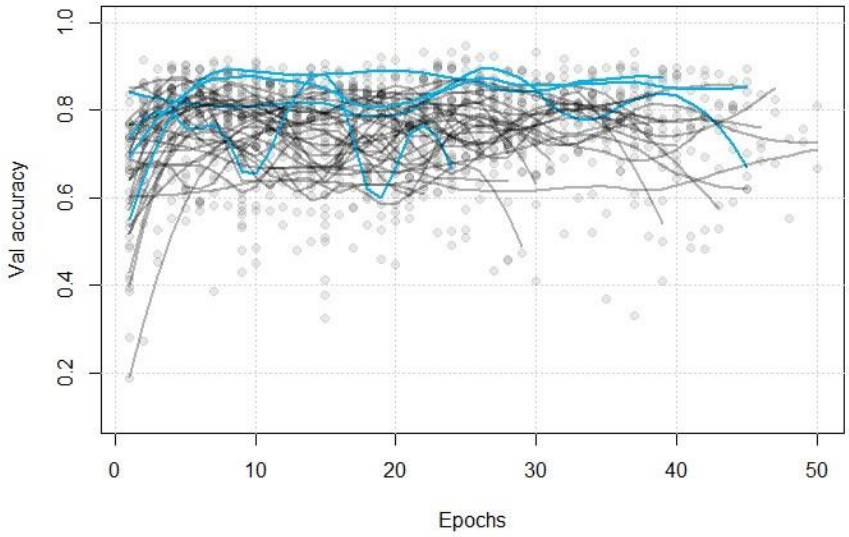
Model 1

The model configuration file contains the default flags to be used to tuning, model definition, fitting of the model and the plotting of learning curve. This model contains two layers with ReLU activation and regularization parameters dropout, penalty-based regularization (L2 regularization) and early stopping. L2 regularization is an approach based on the inclusion of penalty terms which act directly on the weights of the network. Early stopping is the method which stops the training procedure when the validation error start increasing (despite the training error keeps decreasing), i.e., when the network starts overfitting. The function implementing early stopping during training is callback_early_stopping. Among the arguments of this function, patience sets the number of epochs with no improvement after which training will be stopped. And we consider as a criterion to be monitored the value of the accuracy on the test data, by specifying monitor = "val_accuracy".

Regularization via dropout can be easily implemented in Keras by adding additional layers in the model configuration. This is different from the penalty-based regularization because dropout acts directly on the layout of the network, and not on the parameters, like weight decay. The argument

rate is used to set the fraction of the input units to the layer to be dropped. Because of the additional regularization due to dropout, we may want to increase the patience, as the training process can potentially run longer without incurring in overfitting.

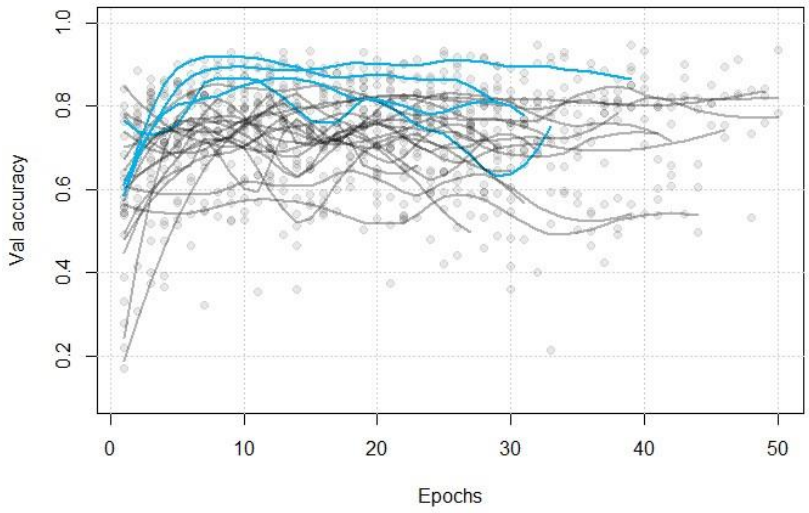
We use softmax activation for the output layer and the optimizer used is Adam. In general, Adam tends to perform well in many applications and convergence and performance are less sensitive to tuning of the learning rate. After defining the model, we fit the model to the training data also with the validation data for evaluation of the performance. After fitting the model with all possible model configurations, a plot of the validation accuracies vs the number of epochs is plotted and with the model outputs, we can identify the best model and its configurations.



Model 2

This model contains three hidden layers with ReLU activation and regularization parameters dropout, penalty-based regularization (L2 regularization) and early stopping. The model is similar to the previous except that it has an additional hidden layer. Similar to the previous case, we define the flags, the model with all the hyperparameters and the output layer with softmax activation. Then we fit the model on the validation data and then evaluate the performance on the test data. This model will have more combinations than the previous one because of the additional hidden layer. All the values of other hyperparameters are unchanged. After fitting the model, we evaluate the performance with the plot of the validation accuracy vs the number of epochs.

From the plot we can see the best models have accuracies higher that 0.85 and we extract them from the metrics.



Results and discussion

First, we used PCA in order to reduce the dimension of the dataset. Then we defined two keras sequential models of two hidden layers and three hidden layers, respectively. From the minimum train and test errors of the models, we can see that the errors are high for the model with 2 layers.

Next, we compare both the models with a plot of their accuracies vs the number of the epochs. In the plot, the blue line indicated the accuracy of the two-layer model and red line indicated that of the three-layer model. It is clear from the plot that the accuracy of the three-layer model is higher than the accuracy of the two-layer model. Hence, we can conclude that the three-layer model performs better even though there are signs of overfitting in both the models.

To compensate the overfitting, we then introduced several regularization techniques and then fitted all the combinations of the hyperparameters through tuning for both two-layer and three-layer model. The best model configurations of both the models are extracted.

The best model configurations for the two-layer model are given below whose accuracies are above 85%.

	metric_val_accuracy	eval_accuracy	flag_dropout	flag_lr	flag_bs	flag_dense_units1	flag_dense_units2
1	0.8934	0.8921	0	0.001	76	150	70
2	0.8895	0.8763	0	0.001	38	300	100
3	0.8829	0.8895	0	0.001	228	300	70
4	0.8658	0.8697	0	0.001	76	300	150

For this model, the best model has 89.2% accuracy and the configurations are given as drouput rate 0, learning rate 0.001, batch size 76, 150 nodes on the first layer and 70 nodes on the second layer.

The best model configurations for the three-layer model are given below whose accuracies are above 85%.

	metric_val_accuracy	eval_accuracy	flag_dropout	flag_lr	flag_bs	flag_dense_units1	flag_dense_units2	flag_dense_units3
1	0.9342	0.9092	0	0.005	228	150	100	100
2	0.9250	0.9316	0	0.005	228	150	150	50
3	0.9053	0.8947	0	0.005	228	150	100	50
4	0.8855	0.8829	0	0.001	228	150	100	100

For the three-layer model, the best model has 90% accuracy and the configurations are given as drouput rate 0, learning rate 0.005, batch size 228, 150 nodes on the first layer, 100 nodes on the second layer and 100 nodes on the third layer.

Conclusion

To conclude, we can say that after running different model configurations on two-layer and threelayer models, the highest accuracy was for the three-layer model. Hence the best model for predicting the daily/sports activity recognition from movement sensor data is a deep neural network model with three hidden layers, drouput rate 0, learning rate 0.005, batch size 228, 150 nodes on the first layer, 100 nodes on the second layer and 100 nodes on the third layer.