Predicting Daily Activities and Sports Using Two Layers Neural Network

Machine Learning & A.I.

Abstract

This project provides a tuning process on the different sets of parameters in predicting 'Daily and Sports Activities', human activities performed while wearing sensor units on the chest, arms, and legs. The dataset comprises motion sensor data of 19 different daily activities and sports each performed by 8 subjects in their own style for 5 minutes. The combination of layers with high densities, dropouts, and optimizers in two layers neural network are considered. The optimal model in this project is the model with 1406 nodes in each layer and dropout of 0.3, with 93% of accuracy.

Introduction

Human activity recognition has received considerable interest recently because of its potential application in a wide range of fields such as medical diagnosis and treatment, computer games, professional simulators, and virtual reality. In this project we are going to train a predictive model to predict which activities users are engaging in based on movement sensor data collected.

The machine will learn from the data by presenting a few sets of scenarios (mostly emphasizing the tuning process and the running of the model). The algorithm that we will use is ***supervised learner,*** which constructs predictive models used for tasks that involve the prediction of a given output (or target) using other variables (or features) in the data set.

Methods

This section covers the entire analysis approach and the discussion of all modelling stages including exploratory data analysis, visualization of data and data preparation.

1. Exploratory Data Analysis

This section covers the detail of data to give us the context needed to develop an appropriate model – and interpret the results correctly. The data consists of motion sensor measurements of 19 daily activities and sports, each performed by 8 subjects (4 females, 4 males, between the ages 20 and 30) in their own style for 5 minutes. For each subject, five sensor units are used to record the movement on the torso, arms, and legs. Each unit is constituted by 9 sensors: x-y-z accelerometers, x-y-z gyroscopes, and x-y-z magnetometers.

The 19 activities are: sitting, standing, lying on their back, lying on their right side, ascending stairs, descending stairs, standing in an elevator still, moving around in an elevator, walking in a parking lot, walking on a treadmill with a speed of 4 km/h (in flat) and 15 degree inclined positions, running on a treadmill with a speed of 8 km/h, exercising on a stepper, exercising on a cross trainer, cycling on an exercise bike in horizontal and in vertical, rowing, jumping and lastly playing basketball.

Sensor units are calibrated to acquire data at 25 Hz sampling frequency. The 5-minute signals are divided into 5-second signal segments, so that a total of 480 signal segments are obtained for each activity, thus 480 × 19 = 9120 signal segments are classified into 19 classes. Each signal segment is divided into 125 sampling instants recorded using 5 x 9 = 45 sensors. Hence, each signal segment is represented as a 125 × 45 matrix, where columns contain the 125 samples of data acquired from one of the sensors of one of the units over a period of 5 seconds, while the rows contain data acquired from all of the 45 sensors at a particular sampling instant.

For each signal matrix:

- columns 1-9 correspond to the sensors in the torso unit,
- columns 10-18 correspond to the sensors in right arm unit,
- columns 19-27 correspond to the sensors in the left arm unit,
- columns 28-36 correspond to the sensors in the right leg unit, and
- columns 37-45 correspond to the sensors in the left leg unit.

For each set of 9 sensors, the first 3 are accelerometers, the second 3 are gyroscopes and the last 3 are magnetometers.

Here is a sample of data visualization from set x_train, time series with a random walk pattern.

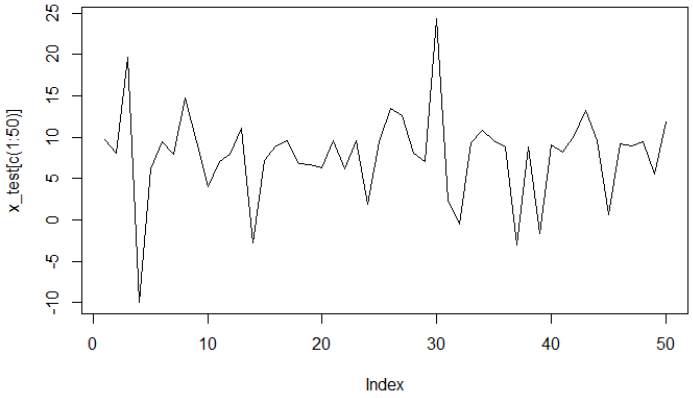


Figure 1. x_train plot

2. Data Analysis

In this section we will extract useful information from the data and make decisions based upon the data analysis.

2.1 Data Preparation

The input data are organized in the form of 3-dimensional arrays, in which the first dimension denotes the signal, the second the sampling instants and the third the sensors, so that each signal is described by a vector of 125 × 45 = 5625 features. In the next section, we are going to aggregate the features and manipulate the data before we do the train and test process.

```
x_train<-data.matrix(array_reshape(x_train,c(nrow(x_train),125*45)))
```

2.2 Data Splitting

We want an algorithm that not only fits well to our existing data, but can also predict future outcomes accurately. How we use our data will help us understand how well our algorithm generalizes to unseen data. The detail of the dataset and splitting details are described as follows.

The data file *data_activity_recognition.RData* contains training and test data. The training data x_train includes 400 signal segments for each activity (total of 7600 signals), while the test data x_test includes 80 signal segments for each activity (total of 1520 signals). The activity labels are in the objects y_train and y_test, respectively. The simplest way to split the data into training and test sets is to take a simple random sample. Where, we are going to use 25% of the data for validation and the rest for training.

2.3 Creating Models

Deep learning provides a multi-layer approach to learning data representations, which are typically performed with a *multi-layer neural network*. It performs learning by mapping the data features to targets through a process of simple data transformations and feedback signals. There are two important activities in this section: tuning parameters as the controller of the model complexity, and model evaluation by assessing the predictive accuracy via loss functions. Loss functions are metrics (metric_val_accuracy) that compare the predicted values to the actual values.

Network Architecture

In this project we will design the deep learning with two main building blocks: layers and nodes. After taking into account the simplest model with optimal performance, we will use 2 hidden layers. As the number of nodes in each layer is usually either equal to or less than the number of features, 5625 layers were put into the combination of parameters. It is important to note that the number of hidden layers and nodes in the network can affect its computational complexity (e.g., training time). When dealing with many features and, therefore, many nodes, training deep models with many hidden layers can be computationally more efficient than training a single layer network with the same number of high volume nodes (Goodfellow, Bengio, and Courville 2016).

The models that performed best had 2 to 3 hidden layers with a medium to large number of nodes. All the "small" models underfit and would require more epochs to identify their minimum validation error. The large 3-layer model overfits extremely fast. Preferably, we want a model that overfits more slowly such as the 1 and 2-layer medium and large models (Chollet and Allaire 2018).

Implementation

To control the activation functions used in our layers we specified the activation argument. For the two hidden layers we added the *ReLU* activation function, and for the output layer we used = *softmax*. Dropout (in the context of neural networks) randomly drops out a number of output features in a layer during training. Here we applied dropout by adding *layer_dropout()* in between the layers. In the next section we will cover the two models of the two layers neural network that was used. Firstly, the model with a single dropout, and secondly the model with 2 dropouts will be covered. In both models, the density of layers options are set to approximately the number of observations (please note that we run the models based on sampling).

MODEL 1

```
 # Network architecture
FLAGS = flags(
 flag_numeric("set1",100),
 flag_numeric("set2",100),
 flag_numeric("dropout",0.45))

# model definition
model = keras_model_sequential()%>%
 layer_dense(units =FLAGS$set1,
          activation ="relu",
          input_shape = V,
          name ="layer_1")%>%
 layer_dropout(rate=FLAGS$dropout)%>%
 layer_dense(units =FLAGS$set2,
          activation ="relu",
          name ="layer_2")%>%
 layer_dropout(rate=FLAGS$dropout)%>%
 layer_dense(units =ncol(y_train),activation ="softmax",
          name ="layer_out")%>%

 # Backpropagation need two things: an objective function and an optimizer
 compile(loss ="categorical_crossentropy",
       metrics ="accuracy",
       optimizer =optimizer_sgd())
```

Model training

Now we created a fit function to train the model with some data. The function has arguments: where we use *batch_size* equal to 64, then epoch is used to describe the number of times the algorithm sees the entire data set (would increase it when we expect the model to learn more, adjust the weights, and minimize the loss function). Plotting the output shows how the loss function improves for each epoch.

The function *callback_early_stopping()* will stop training when a monitored quantity has stopped improving. It has a function = patience that will process the training by a 'batch' number of epochs with no improvement, after which training will be stopped. In this fit function, we stop the training process when the *val_accuracy* is not improving in a frame of 25 epochs.

```
# model training
fit = model%>%
 fit(x =x_train,y =y_train,
    validation_data =list(x_val, y_val),
    epochs =100,
    verbose = 1,
    callbacks = callback_early_stopping(monitor = "val_accuracy", patience = 25)  )
```

Below are details of the second model, which has two dropouts.

MODEL 2

```
# model instantiation
FLAGS <- flags(
 flag_numeric("dropout_1", 0.4),
 flag_numeric("dropout_2", 0.4),
 flag_integer("size_1", 10),
 flag_integer("size_2", 10),
 flag_string("optimizer", "sgd"))

# model configuration
model <- keras_model_sequential() %>%
 layer_dense(units = FLAGS$size_1, input_shape = ncol(x_train),
          activation = "relu", name = "layer_1") %>%
 layer_dropout(rate = FLAGS$dropout_1) %>%
 layer_dense(units = FLAGS$size_2, activation = "relu", name = "layer_2") %>%
 layer_dropout(rate = FLAGS$dropout_2) %>%
 layer_dense(units = ncol(y_train), activation = "softmax", name = "layer_out") %>%
 compile(loss = "categorical_crossentropy", metrics = "accuracy",
       optimizer = switch(FLAGS$optimizer,
                  sgd = optimizer_sgd(), rmsprop = optimizer_rmsprop(), adam = optimizer_adam()    ) )
# Train the model
fit <- model %>% fit(
 x = x_train, y = y_train,
 validation_data = list(x_val, y_val),
 epochs = 100,
 verbose = 1,
 callbacks = callback_early_stopping(monitor = "val_accuracy", patience = 10) )
```

Model Tuning

We designed the code in a way that would enable us to tune and change the parameters as needed. The following code is for model 2:

```
# run ------------------------------------------------------------
#hyperparameter tuning values
dropout_set_1 <- c(0.4, 0.3)
dropout_set_2 <- c(0.2, 0.1)
size_set_1 <- c(5625,2813,1406)
size_set_2 <- c(2813,1406,703)
optim_set <- c("sgd", "adam")

runs <- tuning_run("model2.R",
            runs_dir = "result",
            flags = list(
              dropout_1 = dropout_set_1,
              dropout_2 = dropout_set_2,
              size_1 = size_set_1,
              size_2 = size_set_2,
```

optimizer = optim_set), sample = 0.2)

Table 1. The sets of parameters for tuning

Scenario	Layer1	Layer2	Dropout1	Dropout2	Optimizer Type	Epoch	Batch Size	Patience	Sample	#run
First	5625, 2813, 1406	2813,1406,703	0.4, 0.3	0.2, 0.1	sgd, adam	20	-	10	0.2	14
Second	5625,2813,1407	2813,1406,704	0,0.25,0.45	-	sgd	100	-	25	0.1	1
Third	5625,2813,1408	2813,1406,705	0, 0.3	-	sgd	100	-	25	0.1	2
Fourth	5625,2813,1409	2813,1406,706	0, 0.2	-	sgd	20	64	10	0.2	3
Fifth	5625,1406	1406,703	0.3	-	sgd	100	64	10	1.0	4

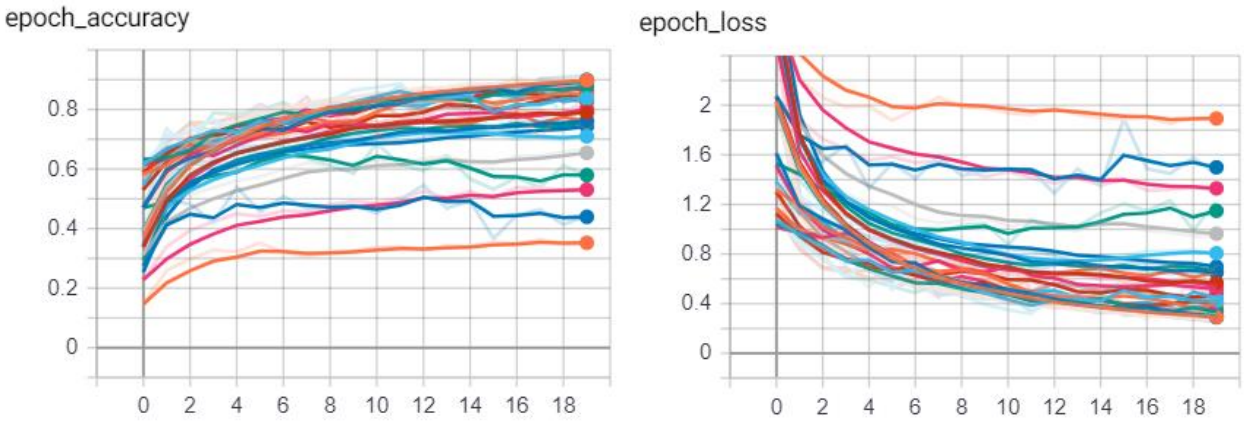
After tuning the models, the results were plotted with brief elaboration about the optimal model and why the parameters were changed in each scenario and will be presented in the next section.

Results and Discussion

In this section there are three points to be observed. The result of tuning based on plot, accuracy and loss, the evaluation and next step of tuning.

First

Since our grid search assesses 72 combinations (3x3x2x2x2), we performed a random grid search and assessed only 14 models (20% of the total models), taking approximately 25 minutes for each model.



The model gives a good visualization of accuracy and loss, however when the highest accuracy and loss is inspected further, the validation accuracy seems to be even higher than the training accuracy. This might be the effect of the two dropouts and low epochs in the model.

Accuracy

Name	Smoothed	Value	Step	Time	Relative
2020-04-26T20-33-39Z\logs\validation	0.8992	0.914	19	Sun Apr 26, 21:42:33	8m 26s
2020-04-26T20-33-39Z\logs\train	0.8977	0.9068	19	Sun Apr 26, 21:42:33	8m 26s

Loss

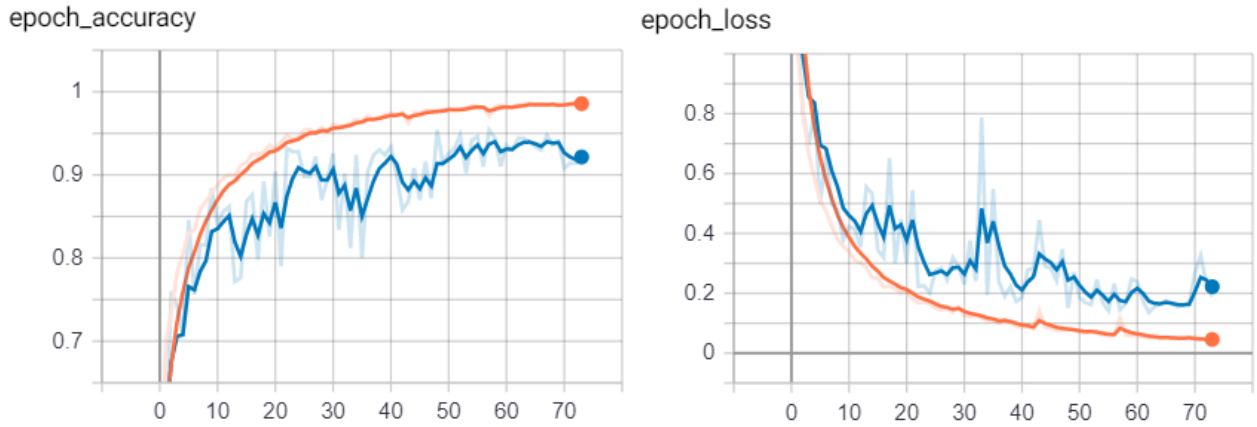
	Name	Smoothed	Value	Step	Time	Relative
	2020-04-26T20-33-39Z\logs\validation	0.2898	0.2535	19	Sun Apr 26, 21:42:33	8m 26s
	2020-04-26T20-33-39Z\logs\train	0.2908	0.2693	19	Sun Apr 26, 21:42:33	8m 26s

The combination of 2813 and 703 nodes in layers (respectively), plus the dropouts (0.3 and 0.1) made overfit symptoms. In order to improve the model, we need to increase the number of epochs and use one dropout, while adding more options of dropouts.

metric_val_accuracy <dbl>	eval_accuracy <dbl>	flag_dropout_1 <dbl>	flag_dropout_2 <dbl>	flag_size_1 <int>	flag_size_2 <int>
0.914	0.9353	0.3	0.1	2813	703
0.898	0.9098	0.3	0.1	2813	703
0.886	0.8892	0.4	0.1	5625	2813
0.870	0.8686	0.3	0.2	5625	1406
0.858	0.8598	0.3	0.2	2813	2813
0.852	0.8608	0.4	0.1	5625	1406
0.838	0.8314	0.3	0.1	5625	1406
0.834	0.8333	0.3	0.1	2813	1406
0.820	0.8255	0.3	0.2	2813	703
0.816	0.8147	0.4	0.1	5625	703

Second

In this scenario we only used one type of optimizer and stopped it before the end of 100 epochs, thus there is no improvement in the last 25 epochs of training.



As can be seen above, the plot of accuracy and loss are good enough, the validation accuracy is lower than the training accuracy, while the validation loss is slightly higher than the training accuracy. still shows that the data is a bit volatile. Moreover, there is no dropout in the model, and the first layer seems to be less dense than the second layer.

Accuracy

	Name	Smoothed	Value	Step	Time	Relative
	2020-04-27T06-49-18Z\logs\train	0.9857	0.9855	73	Mon Apr 27, 08:56:07	1h 5m 53s
	2020-04-27T06-49-18Z\logs\validation	0.9215	0.926	73	Mon Apr 27, 08:56:07	1h 5m 53s

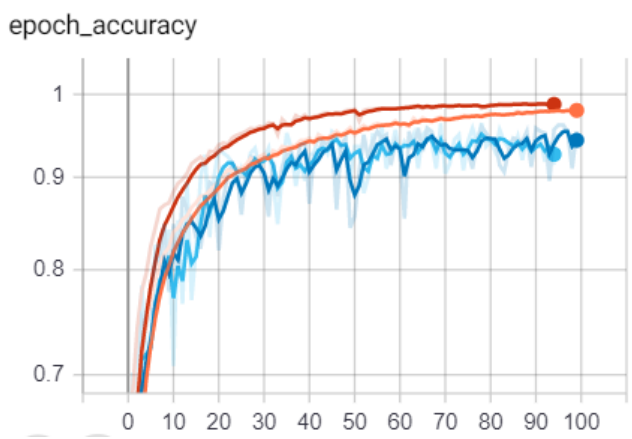
Loss

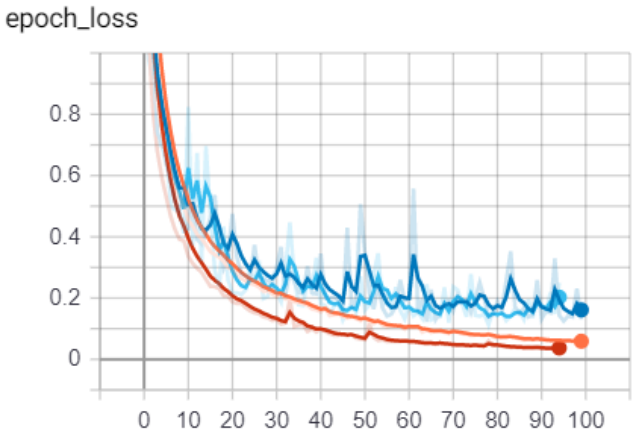
	Name	Smoothed	Value	Step	Time	Relative
	2020-04-27T06-49-18Z\logs\train	0.04565	0.04466	73	Mon Apr 27, 08:56:07	1h 5m 53s
	2020-04-27T06-49-18Z\logs\validation	0.2219	0.1886	73	Mon Apr 27, 08:56:07	1h 5m 53s

metric_val_accuracy <dbl>	eval_accuracy <dbl>	flag_set1 <int>	flag_set2 <int>	flag_dropout <int>	samples <int>
0.926	0.9402	2813	5625	0	7600

Third

For the third scenario, the highest accuracy was the one with 1406 and 5625 nodes in each layer respectively, which also had a dropout equal to 0.3. Even though it is better than the previous model, we can still see the volatile pattern. There is an improvement in both the accuracy and loss values.





Accuracy

Name	Smoothed	Value	Step	Time	Relative
2020-04-27T08-08-59Z\logs\train	0.9799	0.9811	99	Mon Apr 27, 09:59:01	49m 28s
2020-04-27T08-08-59Z\logs\validation	0.9434	0.954	99	Mon Apr 27, 09:59:01	49m 28s

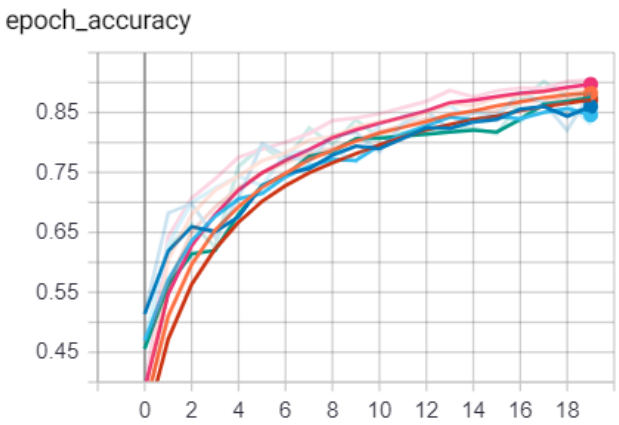
Loss

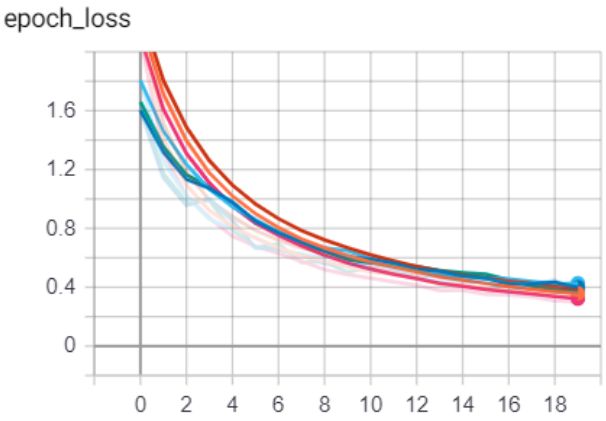
Name	Smoothed	Value	Step	Time	Relative
2020-04-27T08-08-59Z\logs\train	0.06034	0.06125	98	Mon Apr 27, 09:58:30	48m 58s
2020-04-27T08-08-59Z\logs\validation	0.1818	0.2308	98	Mon Apr 27, 09:58:30	48m 58s

metric_val_accuracy <dbl>	eval_accuracy <dbl>	flag_set1 <int>	flag_set2 <int>	flag_dropout <dbl>	samples <int>
0.954	0.9618	1406	5625	0.3	7600
0.914	0.9314	2813	5625	0.0	7600

Fourth

In this set, in order to improve the model, the dropouts are set to 0 and 0.2 rather than 0.3. Due to the lower number of epochs, we have lower accuracy and higher loss than the previous model. The next scenario will have more epochs and more dropout options.





Accuracy

Name	Smoothed	Value	Step	Time	Relative
2020-04-27T11-03-25Z\logs\train	0.8825	0.8868	19	Mon Apr 27, 12:25:13	20m 38s
2020-04-27T11-03-25Z\logs\validation	0.8599	0.884	19	Mon Apr 27, 12:25:14	20m 38s
2020-04-27T11-25-17Z\logs\train	0.871	0.8788	19	Mon Apr 27, 12:31:46	6m 7s
2020-04-27T11-25-17Z\logs\validation	0.8459	0.83	19	Mon Apr 27, 12:31:46	6m 7s
2020-04-27T11-31-49Z\logs\train	0.897	0.9043	19	Mon Apr 27, 12:46:45	14m 12s
2020-04-27T11-31-49Z\logs\validation	0.8749	0.884	19	Mon Apr 27, 12:46:45	14m 12s

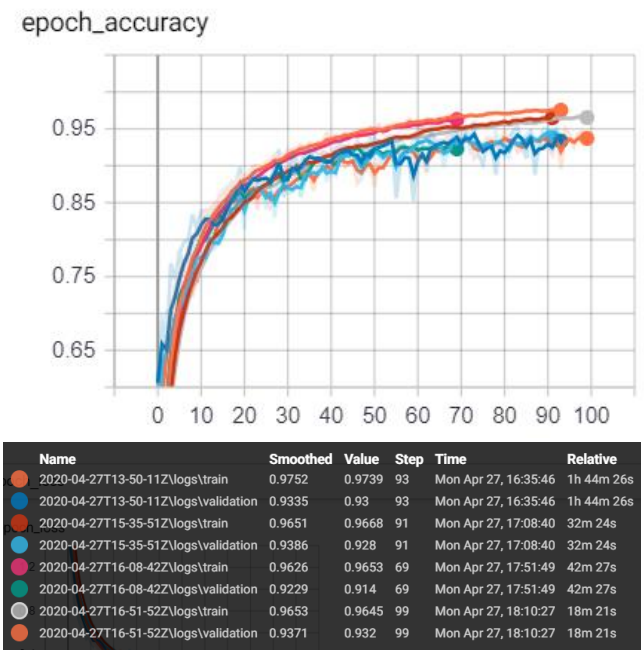
Loss

Name	Smoothed	Value	Step	Time	Relative
2020-04-27T11-03-25Z\logs\train	0.3582	0.3394	19	Mon Apr 27, 12:25:13	20m 38s
2020-04-27T11-03-25Z\logs\validation	0.4006	0.3519	19	Mon Apr 27, 12:25:14	20m 38s
2020-04-27T11-25-17Z\logs\train	0.3902	0.3671	19	Mon Apr 27, 12:31:46	6m 7s
2020-04-27T11-25-17Z\logs\validation	0.4272	0.4352	19	Mon Apr 27, 12:31:46	6m 7s
2020-04-27T11-31-49Z\logs\train	0.3222	0.2998	19	Mon Apr 27, 12:46:45	14m 12s
2020-04-27T11-31-49Z\logs\validation	0.374	0.3556	19	Mon Apr 27, 12:46:45	14m 12s

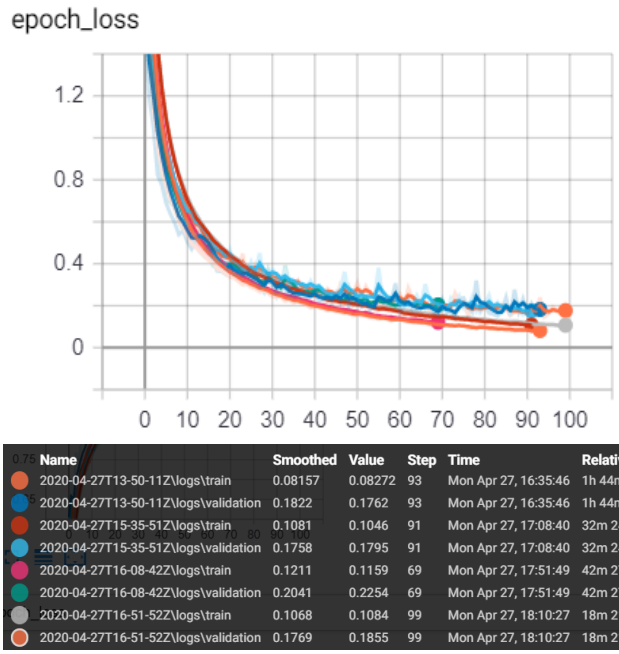
metric_val_accuracy <dbl>	eval_accuracy <dbl>	flag_set1 <int>	flag_set2 <int>	flag_dropout <dbl>	samples <int>
0.884	0.8961	5625	2813	0.0	7600
0.884	0.8980	5625	5625	0.2	7600
0.830	0.8559	2813	1406	0.2	7600

Fifth

In this scenario, we have more epochs and more dropout options.

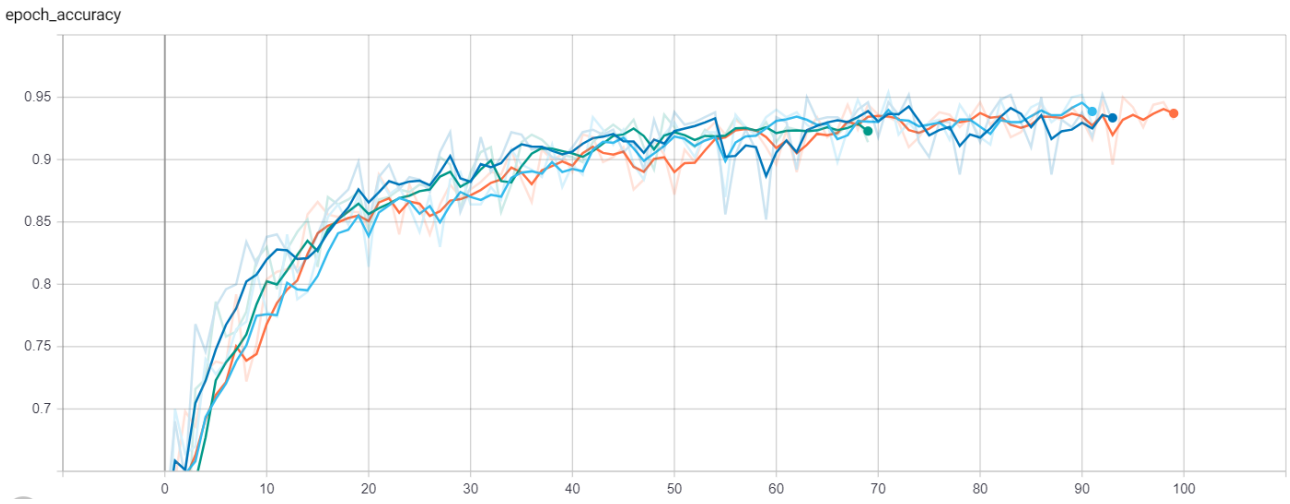


Name	Smoothed	Value	Step	Time	Relative
2020-04-27T13-50-11Z\logs\train	0.9752	0.9739	93	Mon Apr 27, 16:35:46	1h 44m 26s
2020-04-27T13-50-11Z\logs\validation	0.9335	0.93	93	Mon Apr 27, 16:35:46	1h 44m 26s
2020-04-27T15-35-51Z\logs\train	0.9651	0.9668	91	Mon Apr 27, 17:08:40	32m 24s
2020-04-27T15-35-51Z\logs\validation	0.9386	0.928	91	Mon Apr 27, 17:08:40	32m 24s
2020-04-27T16-08-42Z\logs\train	0.9626	0.9653	69	Mon Apr 27, 17:51:49	42m 27s
2020-04-27T16-08-42Z\logs\validation	0.9229	0.914	69	Mon Apr 27, 17:51:49	42m 27s
2020-04-27T16-51-52Z\logs\train	0.9653	0.9645	99	Mon Apr 27, 18:10:27	18m 21s
2020-04-27T16-51-52Z\logs\validation	0.9371	0.932	99	Mon Apr 27, 18:10:27	18m 21s



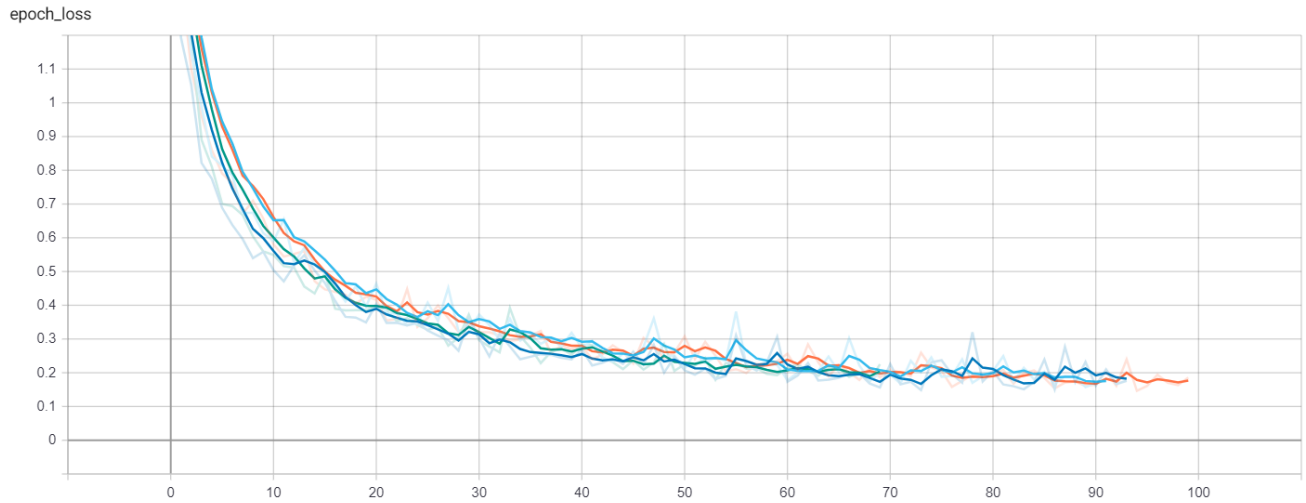
Name	Smoothed	Value	Step	Time	Relative
2020-04-27T13-50-11Z\logs\train	0.08157	0.08272	93	Mon Apr 27, 16:35:46	1h 44m 26s
2020-04-27T13-50-11Z\logs\validation	0.1822	0.1762	93	Mon Apr 27, 16:35:46	1h 44m 26s
2020-04-27T15-35-51Z\logs\train	0.1081	0.1046	91	Mon Apr 27, 17:08:40	32m 24s
2020-04-27T15-35-51Z\logs\validation	0.1758	0.1795	91	Mon Apr 27, 17:08:40	32m 24s
2020-04-27T16-08-42Z\logs\train	0.1211	0.1159	69	Mon Apr 27, 17:51:49	42m 27s
2020-04-27T16-08-42Z\logs\validation	0.2041	0.2254	69	Mon Apr 27, 17:51:49	42m 27s
2020-04-27T16-51-52Z\logs\train	0.1068	0.1084	99	Mon Apr 27, 18:10:27	18m 21s
2020-04-27T16-51-52Z\logs\validation	0.1769	0.1855	99	Mon Apr 27, 18:10:27	18m 21s

The accuracy and loss values of the models in this scenario is good, with an accuracy of 90%, and approximately a 2.2% loss. The lines of validation are still volatile, but smoother than the previous models. In the next plot, we zoom the plot of validation to see it looks like.



Accuracy

Name	Smoothed	Value	Step	Time	Relative
2020-04-27T13-50-11Z\logs\validation	0.9335	0.93	93	Mon Apr 27, 16:35:46	1h 44m 26s
2020-04-27T15-35-51Z\logs\validation	0.9386	0.928	91	Mon Apr 27, 17:08:40	32m 24s
2020-04-27T16-08-42Z\logs\validation	0.9229	0.914	69	Mon Apr 27, 17:51:49	42m 27s
2020-04-27T16-51-52Z\logs\validation	0.9371	0.932	99	Mon Apr 27, 18:10:27	18m 21s



Loss

Name	Smoothed	Value	Step	Time	Relative
2020-04-27T13-50-11Z\logs\validation	0.1822	0.1762	93	Mon Apr 27, 16:35:46	1h 44m 26s
2020-04-27T15-35-51Z\logs\validation	0.1758	0.1795	91	Mon Apr 27, 17:08:40	32m 24s
2020-04-27T16-08-42Z\logs\validation	0.2041	0.2254	69	Mon Apr 27, 17:51:49	42m 27s
2020-04-27T16-51-52Z\logs\validation	0.1769	0.1855	99	Mon Apr 27, 18:10:27	18m 21s

The best model has an accuracy of 93%, with the same number of nodes in every layer, and a dropout of 0.3

metric_val_accuracy <dbl>	eval_accuracy <dbl>	flag_set1 <int>	flag_set2 <int>	flag_dropout <dbl>	samples <int>
0.932	0.9441	1406	1406	0.3	7600
0.930	0.9441	5625	5625	0.3	7600
0.928	0.9490	1406	5625	0.3	7600
0.914	0.9225	5625	1406	0.3	7600

Conclusion

In machine learning, more time is required when experimenting with deep learning, in order to tune the parameter. The optimal model for one dataset will not necessarily fit another dataset, as every dataset is unique and cannot be treated the same.

When predicting the daily activities and sports from the dataset, we used the two layers neural network and had trial-error in parameter tuning. The strategy involved small sampling and started with a small epoch in order to get the best value before jumping to the final code and running all the combinations of parameters and dropouts.

To conclude, the best parameter has accuracy of approximately 93%, with the same hidden layers (1406) in each layer and a 0.3 dropout. Two layers model is preferable as we want a model that overfits more slowly with more nodes to fit the number of observations. The model took a while to fitting but the accuracy is worth it. The usual dropout is 0.2-0.5, we decided to make it 0.3 helped the model to avoid the overfitting. We used Stochastic gradient descent (often abbreviated **SGD**) as it is known for optimizing an objective function with suitable smoothness properties and suitable large enough number of epochs to see the learning curve. This is the ideal model configuration for predicting daily and sport activities data in this project. For further research, more tuning and evaluation may improve the suggested model with a higher accuracy.

References

Chollet, François, and Joseph J Allaire. 2018. *Deep Learning with R*. Manning Publications Company.
Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. Vol. 1. MIT Press Cambridge.
Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. Vol. 1. MIT Press Cambridge.

https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities
https://towardsdatascience.com/data-science-case-study-classification-in-iot-474726c7559
https://bradleyboehmke.github.io/HOML

APPENDICES

The code is adapted from:
Machine Learning and AI– Practical Lab and homework solution (with additional modification).
I did not attach all the code for each scenario because this code is reproducible.

MODEL 1

```
# model instantiation ----------------------------------------------
# set defaul flags
FLAGS = flags(
  flag_numeric("set1",100),
  flag_numeric("set2",100),
  flag_numeric("dropout",0.45)
)

# model definition
model = keras_model_sequential()%>%
  layer_dense(units =FLAGS$set1,
              activation ="relu",
              input_shape = V,
              name ="layer_1")%>%
  layer_dropout(rate=FLAGS$dropout)%>%
  layer_dense(units =FLAGS$set2,
              activation ="relu",
              name ="layer_2")%>%
  layer_dropout(rate=FLAGS$dropout)%>%
  layer_dense(units =ncol(y_train),activation ="softmax",
              name ="layer_out")%>%
  compile(loss ="categorical_crossentropy",
          metrics ="accuracy",
          optimizer =optimizer_sgd())

# model training
fit = model%>%
  fit(x =x_train,y =y_train,
      validation_data =list(x_val, y_val),
      epochs =100,
      batch_size = 64,
      verbose = 1,
      callbacks = callback_early_stopping(monitor = "val_accuracy", patience = 20)
  )

# store accuracy on test set for each run
score <- model %>% evaluate(
  x_test, y_test,
  verbose = 0
)
```

MODEL 2

```
# model instantiation ------------------------------------------------
# set defaul flags
FLAGS <- flags(
  flag_numeric("dropout_1", 0.4),
  flag_numeric("dropout_2", 0.4),
  flag_integer("size_1", 10),
  flag_integer("size_2", 10),
  flag_string("optimizer", "sgd")
)

# model configuration
model <- keras_model_sequential() %>%
  layer_dense(units = FLAGS$size_1, input_shape = ncol(x_train),
              activation = "relu", name = "layer_1") %>%
  layer_dropout(rate = FLAGS$dropout_1) %>%
  layer_dense(units = FLAGS$size_2, activation = "relu", name = "layer_2") %>%
  layer_dropout(rate = FLAGS$dropout_2) %>%
  layer_dense(units = ncol(y_train), activation = "softmax", name = "layer_out") %>%
  compile(loss = "categorical_crossentropy", metrics = "accuracy",
          optimizer = switch(FLAGS$optimizer,
                             sgd = optimizer_sgd(),
                             rmsprop = optimizer_rmsprop(),
                             adam = optimizer_adam()
          )
  )

fit <- model %>% fit(
  x = x_train, y = y_train,
  validation_data = list(x_val, y_val),
  epochs = 100,
  batch_size = bs,
  verbose = 1,
  callbacks = callback_early_stopping(monitor = "val_accuracy", patience = 10)
)

# store accuracy on test set for each run
score <- model %>% evaluate(
  x_test, y_test,
  verbose = 0
)
```

```
load("D:/2020 - Spring/STAT40970  Machine Learning & AI
(online)/Assignment/PROJECT/data_activity_recognition.RData")

library(keras)  # for fitting
library(tfruns) # for additional grid search & model training functions
library(tensorflow)

# prepare data
dim(x_train)
x_train<-data.matrix(array_reshape(x_train,c(nrow(x_train),125*45)))
x_test=data.matrix(array_reshape(x_test,c(nrow(x_test),125*45)))
dim(x_train)
dim(x_test)
dim(y_train)
dim(y_test)

y_train=data.matrix(data.frame(y_train))
y_test=data.matrix(data.frame(y_test))
dim(y_train)
dim(y_test)

y_train=to_categorical(y_train-1)
y_test=to_categorical(y_test-1)

dim(y_train)
dim(y_test)

# normalize input data to 0 - 1
range_norm <- function(x, a = 0, b = 1) {
  ( (x - min(x)) / (max(x) - min(x)) )*(b - a) + a }

x_train=apply(x_train,2,range_norm)
x_test=apply(x_test,2,range_norm)
dim(x_train)
dim(x_test)

# split the test data in two - validation and actual testing
val <- sample(1:nrow(x_test), 500)
test <- setdiff(1:nrow(x_test), val)
x_val <- x_test[val,]
y_val <- y_test[val,]
x_test <- x_test[test,]
y_test <- y_test[test,]

#checking the data
range(x_train)
dim(x_test)
plot(x_test[c(1:50)], type = 'l')
```

```
#the dimensions of the training set
N =nrow(x_train)
V =ncol(x_train)

library(tfruns)
# hyperparameter tuning values
# run ------------------------------------------------------------
set1_set=c(5625,1406)
set2_set=c(5625,1406)
dropout_set=c(0,0.3)

# tuning procedure
runs <- tuning_run("model1.R",
                   runs_dir = "final",
                   flags = list(
                     set1 = set1_set,
                     set2 = set1_set,
                   dropout = dropout_set))



# run ------------------------------------------------------------
#hyperparameter tuning values
dropout_set_1 <- c(0.4, 0.3)
dropout_set_2 <- c(0.2, 0.1)
size_set_1 <- c(5625,2813,1406)
size_set_2 <- c(2813,1406,703)
optim_set <- c("sgd", "adam")

runs <- tuning_run("model2.R",
                   runs_dir = "result",
                   flags = list(
                     dropout_1 = dropout_set_1,
                     dropout_2 = dropout_set_2,
                     size_1 = size_set_1,
                     size_2 = size_set_2,
                     optimizer = optim_set), sample = 0.01)
# or run all combinations


res = ls_runs(metric_val_accuracy > 0.0,
              runs_dir = "resulttest4", order = metric_val_accuracy)

res = res[,c(2,4,8:11)]
res[1:15,] # top 10 models
tensorboard("resulttest3")
```

==========