**Project and Location Directories Overview and Explanations**

Prior to undertaking an in-depth discussion of the individual scripts, their uses, and the various experiment and model prediction sets we undertook as part of this project, it’s worth giving a brief explanation of the two types of directories that we are concerned with for this project. There are two directories that we are concerned with: the project directory (containing the source files, the documentation used and written to, among other things) and the local directory (containing the source ‘.mat’ files that we use as inputs to scripts and some of the outputs of the models, including the models themselves). Below, we cover each of the two in turn, how to access and/or set them up, and so on. The aim is thus to educate any users on how the project is laid out and how each part of the system connects to each other.

**The Project Directory**

The project directory is the directory containing all of the source code, the information required about the NSAA subjects (‘nsaa\_6mw\_info.xlsx’), the documents containing the results of the experiment sets and model prediction sets, and other reports and presentations constructed throughout the duration of the project. Hence, this is where the majority of the deliverables of the project lie, with the exception of constructucted models and the data that is required. The reason we keep these apart (i.e. why project directory and local directory are not synonymous) is as follows:

* The local directory requires >150GB of storage for all the data sets that is required. However, we’ve been making extensive use of storing the project directory within DropBox and as a Git repository and, as it would be not possible and very impractical, respectively, to store the local directory on both DropBox and within GitHub or GitLab, we chose to keep these separate.
* When we run several of the data pipeline scripts (e.g. ‘comp\_stat\_vals.py’ or ‘ext\_raw\_measures.py’) we end up producing a lot of new files within the local directory). Therefore, to avoid constant DropBox synchronizations or many new or deleted files to appear in each Git commit, it was felt that it would be easier to simply keep the data aspects apart from the source code and other documentation.
* Permission may be required to deal with certain data directories contained with the local directory, as this contains data about ongoing subjects that is not freely available. Hence, it was the desire to keep the project directory available to whoever wished to access it, without predicating access on also being able to access the data required to populate the local directory (e.g. if one wished simply to browse or borrow ideas from the scripts within the project directory); this also enables and encourages an easier transition to applying the project to other data sets possibly for other domains.

To access the complete project, the advisable way to obtain it would be to clone it via GitHub. The repo can be accessed <https://github.com/dan-heaton/MSc_indiv_project> and cloned via <https://github.com/dan-heaton/MSc_indiv_project.git>. The current name for the project directory as used in its local form for development has been ‘indiv\_proj; however, one could rename this freely without requiring any other changes to the scripts. However, it’s recommended that the directories without the project directory should not be changed (e.g. ‘source’ or ‘documentation’), as doing so would require rewriting of several of the scripts using hardcoded paths to access things outside of its own directory.

Broadly speaking, the directories can be broken down as the following:

* **background\_research**: this folder contains some preparatory programs that were written (with help from sources cited in the scripts themselves) to familiarize oneself with the building of RNNs with the chosen libraries in order to make using them for the actual project later that much easier; hence, these aren't used by the rest of the project at all and are included for historical documentation reasons only.
* **documentation**: contains a number of files pertaining to the outputs of the data pipeline for the project. This includes the following:
  + 'RNN Results.xlsx', which covers the performances of various model setups on test data (i.e. a large proportion of the experimentation covering different types of raw measurements, sequence lengths, overlaps, etc., sources their results from here). These are what form the basis of our later discussion on the experiment sets.
  + 'model\_predictions.csv', which (unlike 'RNN Results.xlsx') shows the performance of using 'model\_predictor.py' to assess the performance of pretrained models on whole data files. These provide the information needed for the model predictions sets, as discussed later.
  + 'model\_shapes.xlsx', which is just to be used by 'model\_predictor.py' to set the sequence length to the correct value (and is not particularly relevant to the user).
  + 'nsaa\_6mw\_info.xlsx', which contains a table of the subject names and their corresponding single-act and overall NSAA scores (this provides the necessary 'y-labels' for many RNN models).
  + ‘nsaa\_17subtasks\_matfiles.csv’, which is the Google annotations sheet that was collaboratively created by others within the wider research initiative that contains the file names and times within said files where the 17 NSAA activities are performed by each of subjects. This is determined by watching the source videos of the ‘.mov’ files of the subjects performing the activities and recording at what times in the video these activities occur, along with making use of the ‘dis\_3d\_pos.py’ script; this sheet is then used by ‘mat\_act\_div.py’ to create the single-act files used in later model predictions sets.

We also have several other subdirectories in ‘documentation’:

* + **Graphs**: all graphs created by 'graph\_creator.py' are placed in here. These source from 'RNN Results.xlsx' and 'model\_predictions.csv' to create graphs that are easier to display the results of groups of experiments done. We see many of these graphs later on within the discussion of the experiment sets.
  + **Script Explanations**: collection of 'READMEs' for each of the scripts within 'project\_files\source'. The idea is that, if one wishes to find out what each script does, why it was written, etc., then reading its relevant 'README' should provide sufficient detail. Much of these READMEs form the basis of our script overview later on.
* **paper\_reviews**: contains paper reviews done of research papers that are believed to be relevant to the project. These predominantly focus on the use of RNNs when applied to real-world human movement data, and each paper consists of a slightly-shortened bullet pointed version of the paper and then a section of the most significant points from these bullet points. Hence, these papers are useful in justifying decisions taken with respect to model choices, experimentation directions, etc., and will also be heavily used in construction of the final project report.
* **presentations**: contains a collection of presentations that have been created to display to group members about the project's progress thus far (which are kept in order to be used in final report writings at a later date).
* **report\_stuff**: contains several initial reports and other documentations of project progress thus far, and also 'MSc Project Plan.ods', which is where the already-completed and upcoming task lists are stored; this is particularly useful if one wishes to see what is currently being worked on or has recently been completed. The vast majority of the contents of this directory, however, is contained within this report.
* **source**: contains all of the scripts that are needed by the project pipeline to run. This includes the core Python scripts (such as 'comp\_stat\_vals.py' and 'rnn.py'), along with some 'supporting' scripts, such as 'settings.py' (to contain global variables that are used across several scripts). For information about how to run each of these scripts, run the script of interest through the command line/terminal with the ‘--help' optional argument set (e.g. ‘python comp\_stat\_vals.py --help'). This will display each of the arguments that are available to be set, the significance of each, how they interact with other arguments (if relevant), etc.
  + **Batch files**: Within this, we also contain the batch scripts that are used to automate some of the running of the scripts. For further info about the significance of any or all of the scripts, consult the 'README(s)' for the relevant scripts in 'Script Explanations', the script ecosystem overview in 'plans\_and\_presentations', or the diagram of the scripts and their connections to each other found in 'Source'. Along with some of the simpler automation of the tasks, we also run each of the model predictions sets from their respective batch files, as many of them require building many models and testing many different combinations of files, which require many runs of ‘rnn.py’ and ‘model\_predictor.py’; hence, the automation of this makes the process of replication hopefully much easier for the user.

**The Local Directory**

The local directory is considered to have two purposes: to store the data sets that we make use of in this project, and to store the direct outputs of the ‘rnn.py’ scripts that include the models themselves and the ‘.csv’ output predictions that are written directly on a sequence by sequence basis (e.g. for a model created with ‘rnn.py’ using a test set of 1000 sequences, there will exist within this ‘.csv’ each 1000 predicted value and true value, dependent on the output type set by the user). There are three reasons why we include this ‘rnn.py’ output within the local directory:

* As we create many models as part of this project, the size of this directory has become an issue and thus we would prefer to keep it separate from the project directory for space reasons. We therefore want to keep a lot of this data ‘clutter’ apart from the what is considered the ‘core’ of the project with the project directory.
* We also what to keep a consistent philosophy with we consider to be ‘intermediate data’, which is data that exists as a product of one script and that is used by other scripts: in this case, the models produced are intermediate data in that they are created by ‘rnn.py’ and used by ‘model\_predictor.py’. This holds for other forms of intermediate data such as data produced by ‘comp\_stat\_vals.py’ and ‘ext\_raw\_measures.py’, and so we wish to do the same for the models and ‘rnn.py’ predictions output.
* The majority of these models are only used once as part of one particular experiment set or model predictions set, and therefore they don’t form a part of the ‘complete’ system. Thus, we keep these models separate from the ones constituting the completed system at the end of all experimentation; in other words, the models that are intended to be used by a user to do assessments with specific subjects are contained within the project directory, while the models created during all experimentation are contained within the local directory.

**The Local Directory: ‘rnn.py’ Outputs**

We first look at the two sub-directories within the local directory containing the outputs of ‘rnn.py’:

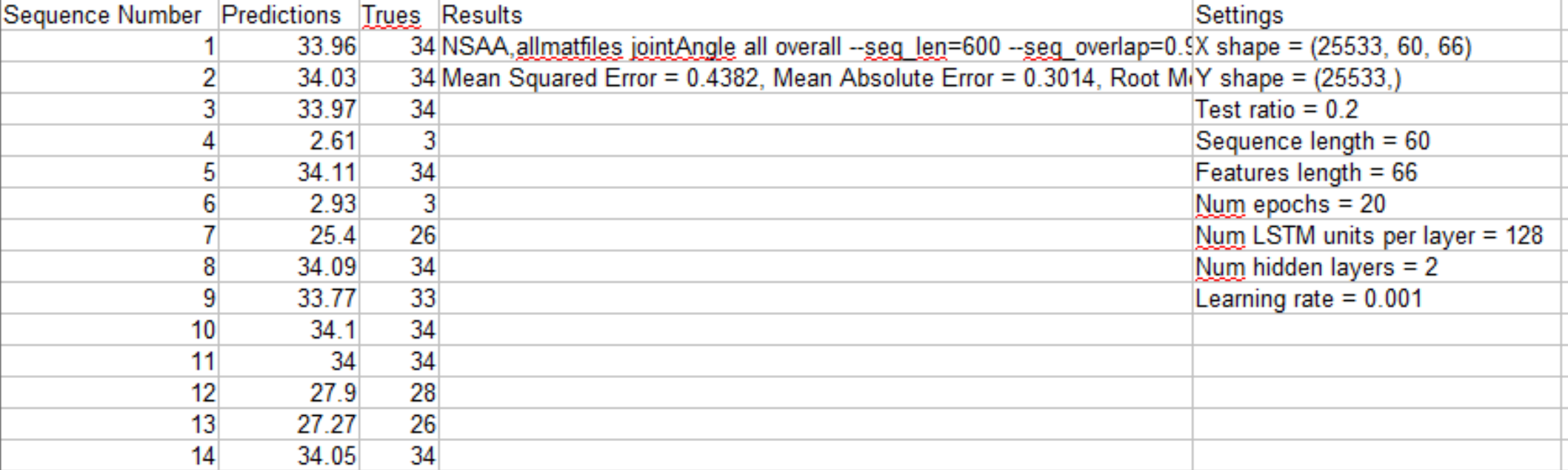
* **output\_files\rnn\_models**: this contains all models that has been produced by ‘rnn.py’ throughout the course of the project, excluding the final models used that is contained within the project directory. Each model’s contents are the product of the TensorFlow library working through ‘rnn.py’ and each model consists of a directory that looks something like this:



These files constitute a single model in the eyes of ‘model\_predictor.py’, which is the only script that makes used of these models. For instance, within these files contains the model input and output shapes, the numbers of hidden layers, other hyperparameter settings, and the weights of each of the neuron connections. In other words, it’s a fully trained model that is ready to be used by ‘model\_predictor.py’ to be used on a whole subject’s data file.

The names of the models may seem unnecessarily long and complete, but they are in fact simply the exact arguments used to invoke the instance of ‘rnn.py’ that creates this specific model, excluding the always-necessary ‘python rnn.py’ parts of the argument. There are two reasons why we do this:

* + The names of the directories are created automatically, which takes much of the human-element of labelling each directory out of the process based on what experiment set or model prediction set it is used for.
  + In having them written by name based on the ‘rnn.py’ arguments, we can ensure that ‘model\_predictor.py’ will always use the correct models during experimentation by writing a simple set of rules within ‘model\_predictor.py’. For instance, if we want ‘model\_predictor.py’ to use models that have been created with added Gaussian noise, we can ensure that it uses only model directories that contain ‘--noise' within their names. Hence, it’s an easier way to determine the correct models to use rather than analysing the contents of the model directory (e.g. the ‘model.ckpt.meta’ file) to determine if it’s a model we need to use.
* **output\_files\RNN\_outputs**: each model that is built by ‘rnn.py’ that is also tested on a test partition of data (i.e. if the ‘--no\_testset’ argument is not set) writes a separate ‘.csv’ file to this directory. Each file contains, for a given created model, the arguments used to invoke it (in the form of the ‘.csv’ file name), the ‘overall’ results of the test set on the model (e.g. MAE of data for the ‘overall’ output type, accuracy of data for the ‘dhc’ output type, etc.), the individual predictions made for each test sequence versus their true values, and the model hyperparameter settings. An example of this can be seen below:



The contents of this file are entirely optional to use, however, as all the requisite information about the test set performance (that is to be inputted into ‘RNN Results.xlsx’ and then used as part of experiment sets) are also produced as console output, which is easier to copy over for the user. However, this files serve as a log of model performance through time as we continue to create more if we wish to use them as a reference at any point.

**The Local Directory: Data Sets**

With the directories containing model outputs having been covered, we shall now look at the directories containing the raw data sets, what is contained within each directory, and what ‘type’ of data these directories contains. It should be noted that the ‘source’ scripts assume that each of these directories are a constant (i.e. that any other users don’t modify the names of the directories); any changes made to the names here require modifications to the necessary variables within ‘settings.py’.

Prior to looking at each data set in turn, it’s preferable to briefly discuss the general locations of raw source data (e.g. as source ‘.mat’ files) and intermediate data. This becomes particularly relevant when we shall shortly be discussing the contents of each data directory, and knowing what produced intermediate data and where is conducive towards understanding the data pipeline. Below, we can see how each script within the data pipeline produces data and where the data is stored:

1. **‘comp\_stat\_vals.py’**: This script takes the data stored as source ‘.mat’ files from within the data directories within the local directory and outputs data to the corresponding path within **output**’. For example, if ‘comp\_stat\_vals.py’ intends to operate on ‘NSAA’ (based on the ‘dir’ argument passed to it), it sources its data from **<local directory>\NSAA\matfiles** as ‘.mat’ files and produce the computed statistical values in **<local directory>\output\_files\NSAA\AD** as ‘.csv’ files. This functions in the same way for other source data sets (e.g. **6minwalk-matfiles**) where the path to the computed statistical value files is more-or-less the same as the source directory path with **output\_files** appended after the path to the local directory.
2. **‘ext\_raw\_measures.py’**: Unlike computed statistical values, the produced raw measurement files are instead stored within subdirectories of the source data set directory that they are sourced from. For example, if ‘ext\_raw\_measures.py’ intends to extract the raw measurements from **6minwalk-matfiles** directory, it retrieves files from **<local directory>\6minwalk-matfiles\all\_data\_mat\_files** and writes each measurement extracted per file to a sub-directory with a name matching the measurement name (e.g. ‘D4’s joint angle data is written to **<local directory>\6minwalk-matfiles\all\_data\_mat\_files\jointAngle** as ‘D4-6MinWalk-jointAngle.csv’ while its position data is written to to **<local directory>\6minwalk-matfiles\all\_data\_mat\_files\position** as ‘D4-6MinWalk-position.csv’, and so on.
3. **‘mat\_act\_div.py’**: It should be noted first that, as this script extracts single-act files from complete-act files, it therefore only expects to be used on the **NSAA** directory, as only files from **NSAA** contain NSAA activities. When activities are divided, they are placed either within **act\_files** or **act\_files\_concat** (depending on whether ‘--single\_act\_concat’ was set for the script) within the **NSAA** directory itself, much like ‘ext\_raw\_measures.py’. For example, ‘mat\_act\_div.py’ would pull source ‘.mat’ files from **<local directory>\NSAA\matfiles** and write single act files (non-concatenated) to **<local directory>\NSAA\matfiles\act\_files** as source ‘.mat’ files but only containing single-activities within each. Note that if ‘ext\_raw\_measures.py’ then operates on these single-act files, they get written to a subdirectory within here, much like how it would operate on complete-act files; hence, drawing the joint angle files from the above case would write the files to **<local directory>\NSAA\matfiles\act\_files\jointAngle**. Similarly, when we compute the statistical values of single-act files, they get written to the **output\_files** directory; in the above case, the computed statistical values of the single-act NSAA files would be written to **<local dir>\output\_files\NSAA\AD\act\_files**.
4. **‘ft\_sel\_red.py’**: To simplify the process of storing the feature-reduced variants of the ‘.csv’ outputs of ‘comp\_stat\_vals.py’, we decided to store the files in the exact same directory as the files they operate on. For example, we wish to reduce the dimension of the computed statistical value files for the **NMB** source data directory, we would write the feature-reduced files to **<local directory>\output\_files\NMB\AD**, where the computed statistical values are already stored. The difference here, however, is that the feature-reduced equivalents will have ‘FR\_’ appended to the front of each of the files. For example, the computed statistical values for the ‘D4’ subject whose data is from **NMB** will be stored within the above path as ‘AD\_D4\_stats\_features.csv’, and when ‘ft\_sel\_red.py’ operates on this subject, it will take the data from this file and write to a file stored in the above path as ‘FR\_AD\_D4\_stats\_features.csv’ (alternatively, if the feature-reduced-concatenation option is taken within ‘ft\_sel\_red.py’ it will instead be stored as ‘FRC\_AD\_D4\_stats\_features.csv’). This naming convention ensures that ‘rnn.py’ fetches the feature reduced variants of files to ensure that models of input nodes size of >4000 isn’t required.
5. **‘rnn.py’**: Although this is considered part of the data pipeline, this simply fetches the data from all of the above locations dependent on the arguments given, while we have already discussed the output locations of ‘rnn.py’ in the previous section.

With the relationship between the data pipeline scripts and the data set directories having been established, we now move on to examining the data that’s contained within each of these directories. The data directories are as follows:

* **6minwalk-matfiles**: This contains the 6-minute walk data of many (but not all) of the subjects within two sub-directories within this directory: **all\_data\_mat\_files** and **joint\_angles\_only\_matfiles**. The former contains the source ‘.mat’ files for all available measurements for the subjects’ walking assessments, while the latter contains source ‘.mat’ files with only joint angles. Therefore, what we consider ‘JA’ and ‘DC’ files (‘Joint Angle’ and ‘Data Cube’, respectively) that are reference in experiment set 1 come from **jint\_angles\_only\_matfiles**, while in most other cases when we use the data of 6-minute walk assessments, we use the **all\_data\_mat\_files** directory.
* **6MW-matFiles**: This directory also contains some of the 6-minute walk assessment files for the subjects and, while some of the files overlap with **6minwalk-matfiles**, it also contains assessment data that is not found in the other directory. Additionally, we don’t see any ‘joint angle only’ data within this data set, and so all source ‘.mat’ files are stored directly within this directory.
* **allmatfiles**: This contains the natural movement behaviour as source ‘.mat’ files that contains only the joint angle data (much like **6minwalk-matfiles\joint\_angles\_only\_matfiles** files). As we only had the natural movement in ‘joint angle only’ form, we had to make use of this for several of the later model predictions sets, though we later received the natural movement behaviour in true ‘AD’ form (i.e. containing all the measurements’ data for each subject).
* **left-out**: This is a small sample of data files that have been excluded from the main data sets that can act as data that is left-out of any of the data sets used to train models. It should be noted the difference between assessing using ‘model\_predictor.py’ on files from **left-out** as opposed to left-out subjects (as used in many model predictions sets): assessing a file from the **left-out** directory will assess a model that is familiar with the subject (through being trained on other files of the same subject), while assessing a complete left-out subject will assess a model that is not familiar with any files for the specific subject.
* **NMB**: This is the complete ‘AD’ data for the natural movement behaviour, as opposed to **allmatfiles** which only contains the joint angle data in source ‘.mat’ file form. As a result of receiving this data late in the project lifecycle, we are only able to use this data set in later model predictions sets. It should be noted the sheer number of files per subject: many of the subjects have up to 30 files captured from each of them that will have captured a variety of ‘natural’ activities, such as sitting, playing, eating, and so on. The data contained within these files, therefore, are much more ‘unstructured’ than the 6-minute walk or NSAA assessment data.
* **NSAA**: This contains the data for each of the subjects’ full NSAA assessments, and the source ‘.mat’ files specifically are contained within the **NSAA\matfiles** subdirectory. It should be noted that, for some subjects, their assessments are split over several files. We account for this when extracting raw measurements and computing statistical values by simply concatenating these source files with respect to time, while we select the correct file to use to get the single-act files via ‘mat\_act\_div.py’ by referencing the file name columns within the Google annotations sheet.