Philipps University of Marburg

Department of Physics

Project Lab Report

NN Classification of Parkinson's Disease

Author: Madhuvanthi Venkatesh

Matriculation Number: 3261538

Supervisor: Benjamin Lotter

Submitted on : 23/04/2020

Signature of the Supervisor:

Contents

1 Introduction **4**

2 Neural Networks **4**

2.1 General points 4

2.2 Mathematical Tools 5

2.2.1 Activation function 5

2.2.2 Loss function 5

3 SDCM **7**

4 Dream PD Challenge **8**

4.1 Output Data 8

4.2 Proposed Data Pipeline 8

5 Tensorflow and Keras **9**

5.1 Advantages of using Keras 9

5.2 Keras Sequential Model 10

6 Building and training the NN model **11**

6.1 Combining the data files and interpolate NaNs 11

6.2 Rounding off the values 11

6.3 Training and Testing the Dataset 11

6.4 Define NN-Keras Model 12

7 Outputs and Challenges **12**

8 Appendix **14**

8.1 Interpolate NaNs 14

8.2 NN-Keras Model 17

List of Figures

1 **Sigmoid**[3] 6
2 **Loss and accuracy as a function of number of epochs** 12
3 **Outputs for Auto-prediction** 13

Abstract

We report on our work trying to predict clinical self-assessment scores of Parkinson's disease patients as part of the Dream-PD challenge [6]. We also cover some of the basic ideas about neural networks, Signal Dissection by Correlation Maximization (SDCM) and its algorithm.

1 Introduction

This project is based on the Dream-PD challenge. Recent technological advancements in mobile health have demonstrated great potential to leverage sensor-based technologies for quantitative, remote monitoring of health and disease - particularly for diseases affecting motor function such as Parkinson's disease. Such approaches have been rolled out using research-grade wearable sensors and, increasingly, through the use of smartphones and consumer wearables, such as smart watches and fitness trackers. These devices not only provide the ability to measure much more detailed disease phenotypes, but also provide the ability to follow the patients for a longer span of time with much higher frequency than is possible through clinical exams. However, the conversion of sensor-based data streams into digital biomarkers is complex and no methodological standards have yet evolved to guide this process.[6]

Parkinson's disease (PD) is a neurodegenerative disease that primarily affects the motor system, but also exhibits other symptoms. Typical motor symptoms of the disease include tremors, slowness (dykinesia), posture and walking perturbations, muscle rigidity and speech perturbations. In the clinic, symptoms are evaluated using physician observation and patient reports. Multiple approaches are under investigation for development of digital biomarkers in PD using accelerometer data from mobile sensor devices with the goal of improving monitoring of treatment efficacy and disease progression for use in clinical care and drug development.[6]

The Parkinson's Disease Digital Biomarker DREAM Challenge is designed to benchmark methods for the processing of sensor data for development of digital signatures reflective of Parkinson's Disease. The inputs given were raw sensor (accelerometer, gyroscope, and magnetometer) time series data recorded during the performance of pre-specified motor tasks, and the goal is to extract data features which are predictive of PD pathology.[6]

2 Neural Networks

2.1 General points

The original motivation behind perceptron neural networks was to model animal brain acitivty as a network of many nodes, each of which can receive, modulate and pass on input information. Modern Neural Networks are an abstraction of this idea: A typical feed-forward NN consists of layers of perceptrons, which are sigmoid functions that take as inputs the values of the previous layer, as parameters a set of weights an biases and feed their output towards the next layer. The chaining of such layers allows the creation of arbitrarily complex non-linear functions. Using training data, these networks can be trained to perform tasks such as classification and regression.

Typical applications are classification of images (such as e.g. animal types), recognition of features in images and time-series analysis (e.g. stock market prediction).

A neural network is a network of functions, called perceptrons, which take in input variables and put out output variables by some tunable function. By feeding the NN input variables and comparing the NN output with the known expected output (also called "training" the NN), one can fine-tune the perceptron parameters. After the training process, the NN is able to process previously unknown information.[1]

2.2 Mathematical Tools

2.2.1 Activation function

There are many possible activation functions sigmoid, relu, tanh, step function etc. In our case, only sigmoid function had good performance. Performance was measured by loss function. A sigmoid function is as shown in figure 1 and it can be mathematically defined as,

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}. \tag{1}$$

It revalues the elements in between the range (0, 1). It is also called as logistic function.

2.2.2 Loss function

The Loss function quantifies the difference between the NN output from the expected training output. Many different loss functions exist, such as Mean Absolute Percentage error, Mean Squared Logarithmic error, Squared Hinge, Hinge, Categorical Hinge, Huber loss, Categorical Crossentropy, Sparse Categorical Crossentropy, kullback leibler divergence, Poisson and Cosine proximity [4]. The loss function we use here is Binary cross entropy. The rea-

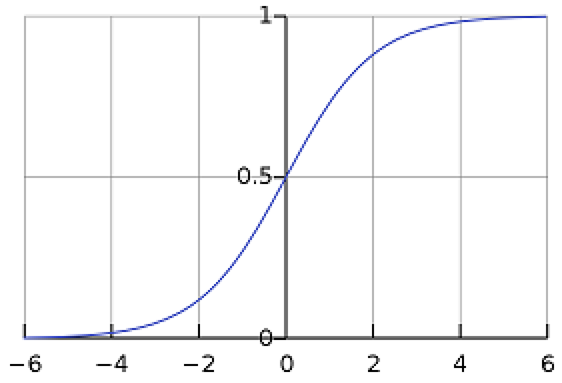

Figure 1: **Sigmoid**[3]

son for using binary cross entropy is because when binary cross entropy is fed with input resulting from sigmoid activation, it will not produce over- or underflow of values. The best example for this is division by 0. Division by zero occurs when the numerator in the sigmoid evaluates to exactly 1 and the denominator then evaluates to zero. This is not the case in sigmoid function as e^x is always positive. Binary cross entropy flattens the sigmoid curve in both the directions.[4]

$$L = -\sum_{i=0}^{N}((ylog(p) + (1-y) \cdot log(1-p) \tag{2}$$

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is an array of probability values between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 will result in a high loss value. A perfect model would have a BCE loss of 0, where, 'p' is the predicted value and y represents categorical data.[3] For example, if [1,0,0] is the output data and the corresponding predictions are [0.1,0.5,0.1], then the loss can be calculated as follows,

L= -(1$\cdot log(0.1) + 0 \cdot log(0.5) + 0 \cdot log(0.1)) = 1$ For multiple categories, the loss function calculates the loss for each label per observation and sums the result.

3 SDCM

Signal dissection by correlation maximization (SDCM) is a clustering technique that dissects high-dimensional datasets into monotonically clustered parts called signatures. Each signature captures a particular signature pattern that is consistently observed from multiple datasets. The key advantage of this method is its precise regression technique. The difference between SDCM and Principal Component Analysis (PCA) is that PCA searches for directions of maximal data variance and ignores perpendicular distances of data points, whereas SDCM functional is maximal for directions and hence, many data points can be aligned as possible. Principle Component (PC) represents a data point distribution as a linear axis, SDCM extends this linear concept to nonlinear monotonic curves along an axis obtained by regression. Points far from the axis have lower influence on this regression than data points. In this way, SDCM extracts data structures locally, whereas PCA globally reduces data dimension by projection along the PC of maximal variance.[5] For multi-classification data the dissection can be conceptualized as,

$$M_o = \sum_k E_k + \eta \tag{3}$$

Here, $\mathrm{M_0}$ represents the input matrix of the signal. This is dissected into a sum of signature matrices $\mathrm{E_k}$ of the same size, along with a noise matrix η. The E_k are assumed to be bimontonic, that is, it is presumed that there exists a sorting of the row and column indices such that the values of each row and column are montonic. Each signature $\mathrm{E_k}$ formally has the same size as the input. In short, SDCM continues to dissect signatures by iterating the following four steps:

1. Search for an initial candidate direction using a signature functional.
2. Optimize this direction by locally maximizing the signature functional.
3. Regress a monotonic curve through data points in the weights-based cone .
4. Subract the found regression curve from the dataset.

The amount a sample partakes in a signature can be quantified by its signature strength. After dissection, a number of K signatures has been found. Each sample can be characterised by a K-tuple of signature strengths.

4 Dream PD Challenge

4.1 Output Data

The output data is a 6-month long clinical study of Parkinson's Disease (PD) patients consisting of clinic visits and at-home monitoring with Apple watch devices, and was generated by researchers at Northwestern University, University of Rochester, University of Alabama, and University of Cincinnati. It consists of self-assessment scores for disease severity for each patient and several sets of measurement. These three symptoms-on/off medication state,dyskinesia, and tremor were rated from 0 to 4, where 0 to 4 increases the severity the symptoms "dyskinesia" and "tremor", while in the case of "medication state", 0 means full medication, and 4 "no medication". [6]

Severity	**On/off**	**Dyskinesia or Tremor**
0	Fully on	None
1	Partially on	Mild
...	...	...
4	Fully off	Severe

Table 1: **Output Data**

4.2 Proposed Data Pipeline

The training data are 20 minute recordings of smartwatch accelerometer data. The data was recorded for 30 minutes, and then the first and last 5 minutes were cropped from the measurement. At the end of each measurement, the patient had to give self assessment scores in the categories medication state "on_off", "dyskinesia" and "tremor".

The pre-processing of data was as follows, the vector norm of the x,y and z axis of measurement was calculated, thereby reducing the data load. The measured data was then divided by its median value, thereby setting the constant gravitational offset to g=1. This comes at a cost of potentially deleting relevant information between the x, y and z axis. Subtracting by 1 eliminates the gravitational offset without the need of applying filters.

The measured data was then segmented into 10s intervals. Each interval was processed with an Auto Regression algorithm [5], reducing the input data to 250 coefficients. AR is a linear times-series training model that tries to model each point in the timeseries as linearly dependent on the previous p points:

$$t_q = A_(q-1)t_(q-1) + A_(q-2)t_(q-2)... + A(q-p)t_(q-p),$$

where $\mathrm{A_i}$ are scalar coefficients. Each set of coefficients for every segment of each measurement was then fed into SDCM as the input matrix. SDCM thus looks for common (sub)sets of coefficients that were consistently observed throughout multiple different time-series segments.

After SDCM, the signature strengths for each segment was obtained, but data labels only for each time series. Hence, the signature strengths of each segment in a time series was merged back together with the median.

Thus one obtains a different signatures vector for each time series, which was 23 in our case, that can be used as training input for the data labels.

5 Tensorflow and Keras

Tensorflow is an open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources which helps developers to easily build and deploy machine learning powered applications. Tensorflow has a high-level neural network API package called as Keras which has many in-built functions along with a fully-implemented back-propagation algorithm. [2]

5.1 Advantages of using Keras

Since Keras has many inbuilt functions, the number of lines in code decreases thereby increasing the efficiency and decreasing the execution time. Keras can be used as a deep learning library that supports Convolutional and Recurrent Neural Networks.It runs seamlessly on both CPU and GPU.

5.2 Keras Sequential Model

In the following we briefly the describe the choices and options we made to construct our NN learning model.The Sequential model is a linear stack of layers in which the output of one layer is fed as the input to the next. Sequential models can be created by passing a list of layer parameters to the constructor.

Input shape: The neural network needs to know the input dimension fed into the model and this is done by passing an argument input_shape to the NN model. The value of inputshape is chosen according to the number of signatures SDCM, which is 23 in this case.

Batch size: A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.In recurrent networks, the batch size are defined for the inputs through the batch size argument. In our case, batch_size = 500 and input_shape = (23,) to a layer. Hence, it will expect every batch of inputs to have the batch shape (500, 23,).[7]

Compilation: Before training a model, the learning process is configured via the compile method. It receives three arguments:

- **An optimizer**. This could be the string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the optimizer class. The optimizer we use is Adam.
- **A loss function**. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as categorical_crossentropy or mse), or it can be an objective function. The loss function we use here is binary cross entropy.
- **Metrics**. A metric could be the string identifier of an existing metric or a custom metric function. The metric we use here is accuracy. [7]

Training: Keras models are trained on Numpy arrays of input data and labels. For training a model, we use the fit function.[7]

6 Building and training the NN model

6.1 Combining the data files and interpolate NaNs

Here, the output files and input files are combined together along a common index. The output data base has many unknown values. These missing values will reduce the accuracy or efficiency of prediction. Hence, these values are filled with the ,"most suitable value". This is done by calculating the mean value of the rows that does not contain NaN values. By this, we can actually guess the missing values and improve the accuracy of the program.

The input data contains a lot of missing values. When such values are imputed into the neural network, they might wrongly train the model and give inaccurate prediction. Hence, we need to interpolate NaNs by replacing them with the nearest value. If we skip the columns or rows with missing data, the training data would not be sufficient enough to give accurate prediction. The NaNs are interpolated by taking the rows which have matching entries in the non-NaN cells, and calculating the mean of the missing column.

6.2 Rounding off the values

After interpolating the NaN values, the dataset now contains lot of decimal values. Since the output must be of integer type to be parsed as categorical data, the output dataLabel is rounded off. The reason for doing this is because at the end of the model, we use regression analysis for prediction. In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates', or 'features'). This analysis requires numerical variables. In order to include a categorical variable (variables of different categories) in a regression model, the categorical variables are recoded into a set of non-decimal integers. This is done by two ways. At first, the decimal values are rounded off into integers. Then the integers are reshaped into binary arrays.Hence, we round off the values of variables to 0's and 1's.

6.3 Training and Testing the Dataset

The testSize is defined as 0.5 which means 50% of the data is used for training and the remaining 50% is used for validation. This value is kept in order to prevent over-training of

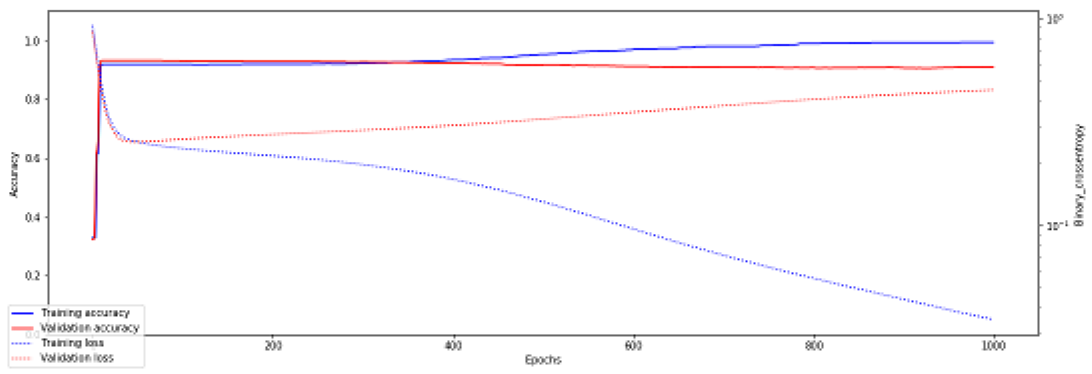

Figure 2: **Loss and accuracy as a function of number of epochs**

the model. The loss function is Binary Crossentropy.

6.4 Define NN-Keras Model

The input shape are defined through numSignatures and it is given as 23 . The output activation function used here is sigmoid. The optimizer used here is Adam with the learning rate 0.01. Number of epochs (The number of times the training data is exposed to the network) is chosen as 1000. The batch_size of each epoch is chosen as 500.

7 Outputs and Challenges

Figure 3 represents Training accuracy, Validation accuracy, Training loss and Validation loss. From the graph, we can clearly see that as the number of epochs (the number of times the data gets exposed to the model) increases, the training loss becomes less and validation loss becomes more. This indicates over-fitting of the data, i.e. the network is trained too well for a specific data set, but performs worse on others.

The .hist() function in python plots histogram for the given data. The histogram contains counts vs computation time plot for the data. From the Figure 4, we can clearly see that dyskinesia and tremor has more counts than on_off. The further ways to improve the data accuracy are,

- Taking aside a part of the known data to test the trained model and thereby, the over-training of the model can be avoided.

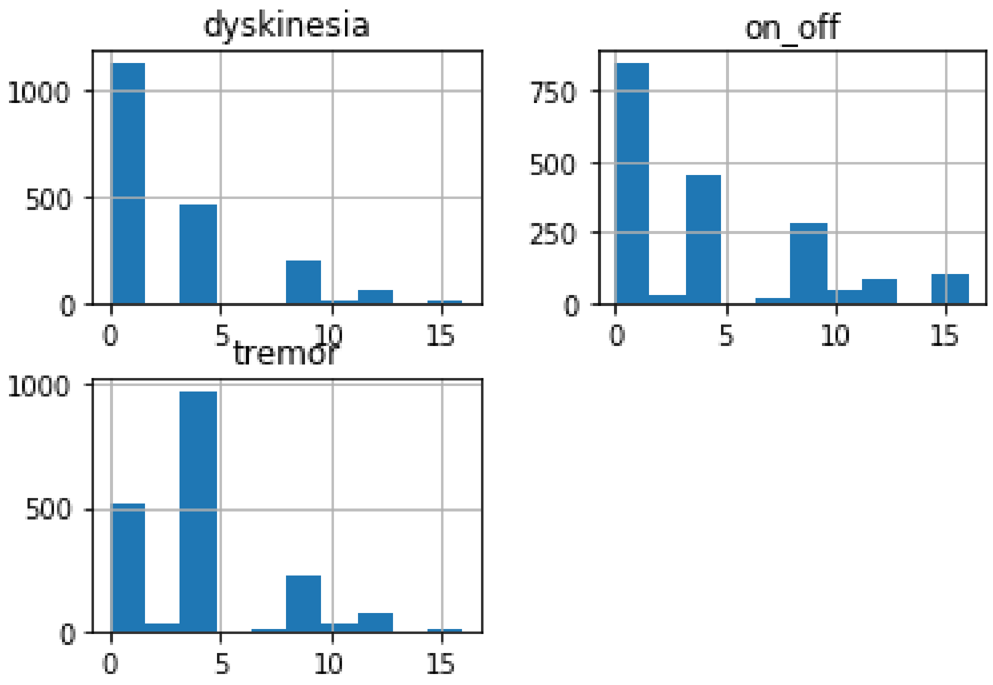

Figure 3: **Outputs for Auto-prediction**

- Performing a hyper-parameter search, i.e. training many different models with different parameters and evaluate their corresponding accuracies. Finally, the results can be compared and the best model can be chosen.
- Column-wise prediction, i.e. individual prediction of each column might improve results.

Some of the problems in the pipeline are,

- SDCM values are obtained from the mean of the signatures of a whole pipeline, thus making the input too coarse to train.
- NaN interpolation might be dangerous to perform as it might introduce wrong training labels or the predicted values might differ from the real values and eventually, this would introduce errors and inaccuracies in the training data.

8 Appendix

8.1 Interpolate NaNs

```
import pandas as pd
def interpolateNaNs(inputDf,
                    allowNaNRows=True,
                    allowNaNCols=False,
                    dropEmptyEntries=False,
                    handleNoMatch='cleanMean'):


    if inputDf.isna().all(axis=0).any() and not allowNaNCols:
        raise ValueError("At least one entire column is NaNs. \
                         Set allowNaNCols=True to ignore.")
    if (inputDf.isna().all(axis=1).any()
            and not allowNaNRows
            and not dropEmptyEntries):
        raise ValueError("At least one entire row is NaNs. \
                         Set allowNaNRows=True to ignore or \
                    dropEmptyEntries=True to remove empty
                    rows.")

# copy the input df (probably not necessary but safe)
interpolateDf = inputDf.copy()
# drop empty rows if present
if dropEmptyEntries:
    interpolateDf =
    interpolateDf[~interpolateDf.isna().all(axis=1)]
# mask all NaN row indices
nanmask = ~inputDf.isna().any(axis=1)
# separate input df into a "clean" df and a df with NaNs
cleanDf = inputDf[nanmask]
nanDf = inputDf[~nanmask]
# iterate through all NaN entries
for idx, row in nanDf.iterrows():

    # find columns that don't contain NaNs
    naNColumns = nanDf.columns[row.isna()]
    nonNaNColumns = nanDf.columns[~row.isna()]
    # take part of row that does not contain NaNs
    nonNaNPart = row[nonNaNColumns]
    match = (cleanDf[nonNaNColumns] == nonNaNPart)

    # decide how to handle rows without match in cleanDf or
    all-NaN rows
    if match.empty or row.isna().all():
        newRow = row.copy()
        if handleNoMatch == 'pass':
            # just ignore entry keeping it NaN
            continue
        elif handleNoMatch == 'totalMean':
                # calculate mean of all entries
                newRow[naNColumns] =
                inputDf[naNColumns].mean(axis=0,

                                                    skipna=True)
            elif handleNoMatch == 'cleanMean':
                # calculate mean of all clean entries
                newRow[naNColumns] =
                cleanDf[naNColumns].mean(axis=0)
            else:
                raise ValueError(f"Unknown mode {handleNoMatch}")
        else:
            # find all entries in the clean df that match the
            non-NaN entries
            # of the row
            matchInCleanDf = cleanDf[match.all(axis=1)]
            newRow = matchInCleanDf.mean(axis=0, skipna=True)
        # set the interpolated df entry to the mean of of the
        previously
        # defined entries
        interpolateDf.loc[idx] = newRow
    return interpolateDf
if __name__ == "__main__":
    # for testing, replace this line with the path to the
    datalabels file
    labelPath = "/mnt/gastonSda/dreamchallenge/data/raw/cis-pd/cis-
    pd.data_labels/data_labels/CIS-PD_Training_Data_IDs_Labels.csv"
    dataLabels = pd.read_csv(labelPath).set_index(['subject_id',

                                                        measurement_id'])
    interpolateDf = interpolateNaNs(dataLabels)
```

8.2 NN-Keras Model

```
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gs
from interpolateNaNs import interpolateNaNs
#from scipy.special import softmax
#input
signatureFile = "signatureStrengths_psdWelch.csv"
dataLabelFile = "CIS-PD_Training_Data_IDs_Labels.csv"

signatureDf = pd.read_csv(signatureFile).set_index("Row")
signatureDf.index.name = "measurement_id"
dataLabelDf = pd.read_csv(dataLabelFile).set_index(['subject_id','m
easurement_id'])
interpolateDf = interpolateNaNs(dataLabelDf)
dataLabelDf=interpolateDf.round()

combinedDf = dataLabelDf.join(signatureDf)
#combinedDf = combinedDf.loc[1004,:]

signatures = list(signatureDf.columns)
numSignatures = len(signatureDf.columns)
observations = list(dataLabelDf.columns)
inData = combinedDf[signatures]
outData = combinedDf[observations]
testSize = 0.5
lossfn = 'binary_crossentropy'
```

```
yNp=outData.to_numpy()
y_normed = yNp / yNp.max(axis=0)
y_normed=y_normed.round()



#training
inTrain, inTest, outTrain, outTest = train_test_split(inData,
y_normed, test_size=testSize, random_state=0)

model = keras.Sequential([
    keras.layers.Dense((numSignatures+3)/2,input_shape=(numSignatu
    res,),activation=tf.nn.sigmoid),
    keras.layers.Dense(3,activation=tf.nn.sigmoid),
    ])

optRMSprop=keras.optimizers.RMSprop(learning_rate = 0.01)
optAdam=keras.optimizers.Adam(learning_rate = 0.01)
model.compile(optimizer=optAdam, loss=lossfn,metrics=['accuracy'])
history = model.fit(inTrain, outTrain, epochs=1000, batch_size=500,
validation_split=0.5)
# Plotting block
fig = plt.figure(figsize = (15,5), constrained_layout=True)
spec = gs.GridSpec(ncols=1, nrows=1, figure=fig)
axAcc = fig.add_subplot(spec[0,0])
axLoss= axAcc.twinx()
axAcc.plot(history.history['accuracy'], c='b', label="Training
accuracy")
axAcc.plot(history.history['val_accuracy'], c='r',
label="Validation accuracy")
#axAcc.axhline(1,color='k', ls='dashed')
axAcc.set_ylim(0,1.1)
axAcc.set_ylabel('Accuracy')
axAcc.set_xlabel('Epochs')
#axAcc.legend()
axLoss.semilogy(history.history['loss'], c='b',ls='dotted',
label="Training loss")
axLoss.semilogy(history.history['val_loss'], c='r', ls='dotted',
label="Validation loss")
axLoss.set_ylabel(lossfn.capitalize())
#axLoss.legend()
fig.legend(loc='lower left')
plt.show()


autoPredictions = model.predict(inData)
autoPredictionDf = pd.DataFrame(autoPredictions,
index=outData.index, columns=outData.columns).round()
labelDf= (combinedDf[observations]*4).round()
difDf = (labelDf-autoPredictionDf).abs()
difDf.hist()
print(difDf.median(axis=0))
```

References

[1] Neural Networks, https://web.archive.org/web/20091216110504/http://www.doc.ic.ac.uk/ nd/surprise_96/journal/vol4/c11/report.html

[2] Tensorflow, https://www.tensorflow.org/

[3] https://gombru.github.io/2018/05/23/cross_entropy_loss/

[4] https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[5] Dissection of gene expression datasets into clinically relevant interaction signatures via high-dimensional correlation maximization, *Nat Comm 10, 5417 (2019)*, Grau, M., Lenz, G.,Lenz, P.

[6] CIS-PD Data, https://www.synapse.org/

[7] Keras Sequential Model, https://keras.io/models/sequential/