An Investigation Neural Network Architecture using Parkinson Speech Data

Jiaxi Zhao - z5305338

***Abstract*—Something about what this is about**

I. Introduction

The popularity of neural networks has exploded with the increase in computing power over the last several years. Their attractiveness stems from the diverse set of problems they can applied to, ranging from image processing to stock market prediction. However, with this flexibility comes the challenging task of determining the many hyper-parameters to be used in any neural network model. This includes the choice of optimiser, network topology, activation functions, loss functions, regularisation and many more. Some of these choices can be guided with common sense: for example, the loss function is often dictated by the whether the problem is a classification or regression problem. Despite this, there is no clear way to determine the vast majority of hyper-parameters.

A. Quick Literature review

Many papers have been dedicated to answering this question. For example, Wilson et al. have proposed

This paper explores the effect of different hyper-parameterisations on model performance using the Parkinson Speech data set. Five hyper-parameters are considered. These are: the choice of optimiser (SGD vs Adam), the optimiser learning rate, momentum rate, number of hidden layers and number of hidden neurons. It is organised as follows: section II outlines the data used, section III details the methodology, section IV presents the results and offers some discussion and finally section V concludes.

II. Data

The data used is the Parkinson Speech training Data Set, collected by the Department of Neurology in Cerrahpasa Faculty of Medicine, Istanbul University. This was obtained from the UCI Machine Learning Repository (add note here). The data was collected from 40 subjects, 20 with Parkinson's Disease (PD) and 20 healthy participants. For each subject, 26 types of sound recordings were taken where the subject was required to say sustained vowels, numbers, words and short sentences. This resulted in a total of 1040 instances. From each recording, 26 features were extracted measuring speech characteristics such as jitter, pitch and voice breaks. The interest of this study is to use these features to predict whether a subject has PD or not. The target variable is 'class' which is 1 if a subject has PD and 0 otherwise. Since the data was collected from 20 healthy and 20 PD subjects, the data is completely balanced and we have 520 instances in each level of 'class.'

Summary statistics and density plots of the features are shown in Table I and Figures 1-3.

TABLE I
Summary statistics

	mean	std	min	25%	75%	max
jitter local	2.68	1.77	0.19	1.51	3.41	14.38
jitter local abs	0.0	0.0	0.0	0.0	0.0	0.0
jitter rap	1.25	0.98	0.06	0.62	1.6	8.02
jitter ppq5	1.35	1.14	0.08	0.67	1.69	13.54
jitter ddp	3.74	2.94	0.18	1.85	4.81	24.05
shimmer local	12.92	5.45	1.19	9.35	15.49	41.14
shimmer local db	1.19	0.42	0.1	0.94	1.41	2.72
shimmer apq3	5.7	3.02	0.5	3.7	6.94	25.82
shimmer apq5	7.98	4.84	0.71	5.16	9.56	72.86
shimmer apq11	12.22	6.02	0.52	8.08	15.31	44.76
shimmer dda	17.1	9.05	1.49	11.11	20.83	77.46
ac	0.85	0.09	0.54	0.8	0.9	1.0
nth	0.23	0.15	0.0	0.13	0.3	0.87
htn	10.0	4.29	0.69	7.51	12.09	28.42
median pitch	163.37	56.02	81.46	124.08	192.48	468.62
mean pitch	168.73	55.97	82.36	126.59	201.22	470.46
standard deviation of pitch	27.55	36.67	0.53	7.3	27.55	293.88
minimum pitch	134.54	47.06	67.96	100.85	159.66	452.08
maximum pitch	234.88	121.54	85.54	143.65	263.8	597.97
number of pulses	109.74	150.03	0	42.75	113	1490
number of periods	105.97	149.42	0	40.75	109	1489
mean period	0.01	0.0	0.0	0.01	0.01	0.01
standard deviation of period	0.0	0.0	0.0	0.0	0.0	0.01
fraction of locally unvoiced frames	27.68	20.98	0	8.15	43.06	88.16
number of voice breaks	1.13	1.61	0	0	1	12
degree of voice breaks	12.37	15.16	0	0	22.26	69.12

Fig. 1. Feature density plots: 1-9

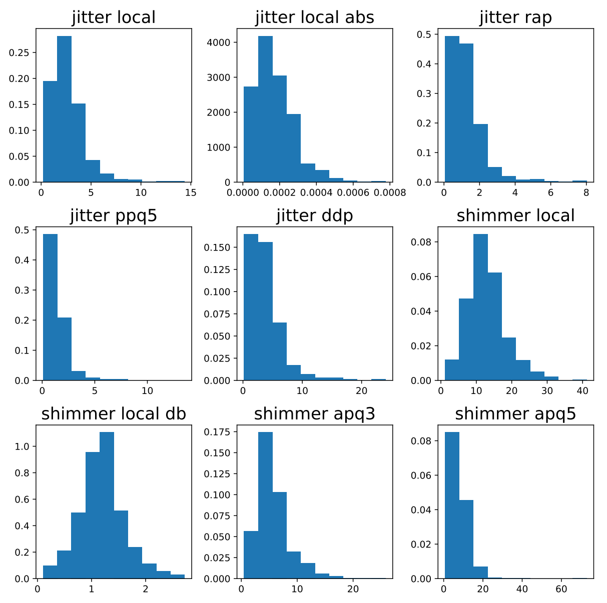


Fig. 2. Feature density plots: 10-18

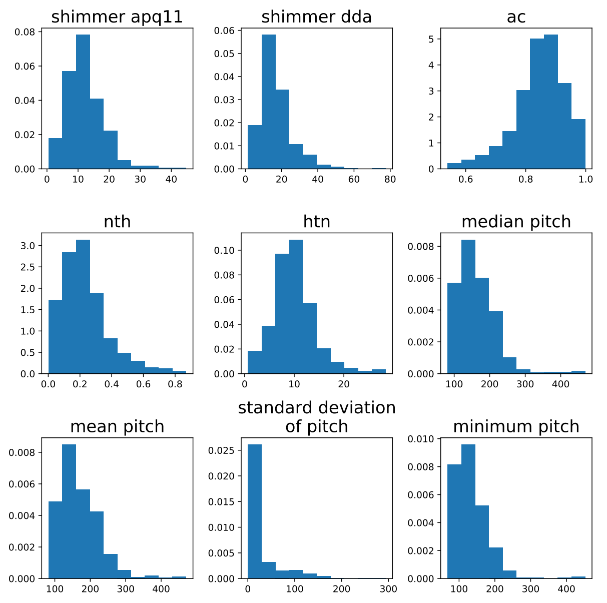
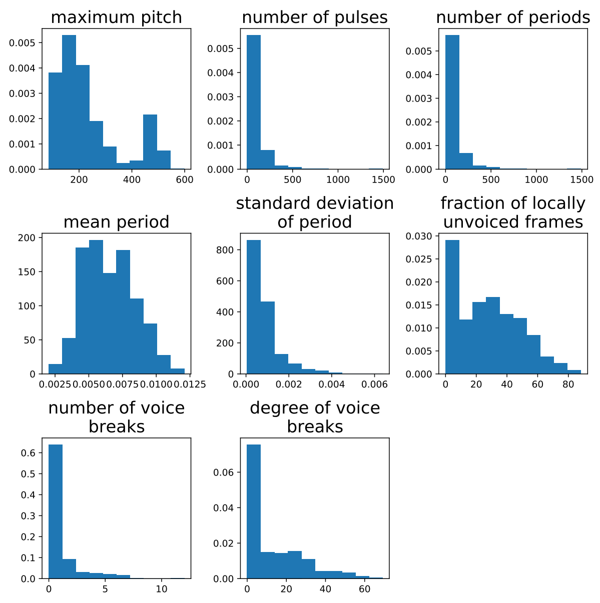


Fig. 3. Feature density plots: 19-26



III. Methodology

The aim of this paper is to investigate how different neural network hyper-parameters affect model performance. A detailed description of the methodology is provided in this section. All neural networks were implemented in Python using the Tensorflow 2.0 library.

A. *Data Preparation*

Firstly, the data was split randomly into train, validation and test sets. The data was first split 80-20 into a train-validation set and a test set. Then the train-validation set was split 80-20 again into train and validation sets. Overall, 64% (665 instances) were allocated to the train set, 16% (167) to the validation set and 20% (208) to the test set.

The data was then normalised via standard scaling, that is, by subtracting the sample mean and then dividing by the sample standard deviation:

$$x_{\text{scaled}} = \frac{x - \bar{x}}{\text{sd}(x)}$$

The mean and standard deviation used for standardisation were obtained from the train set only. This ensured that the validation and test sets were completely unseen by the model when learning the weights.

B. *Model*

Different hyper-parameterisations were investigated using a sequential procedure. That is, starting from a base model, each hyper-parameter was investigated one at a time. After the best setting for one hyper-parameter was found, it was used in the following rounds of investigation. For example, suppose that the optimal number of hidden neurons and hidden layers were to be investigated, in that given order. After obtaining the optimal number of hidden neurons, that setting would be then used in determining the optimal number of layers.

The base model consisted of an input layer of 26 neurons (equal to the number of features), a single hidden layer with 10 hidden neurons using the ReLU activation function and an output layer with a single hidden neuron using the sigmoid activation function. Thus, the output of the model is the probability that the given person has PD. The default optimiser used was SGD with a learning rate of 0.01 and no momentum, which is the default setting in Tensorflow.

The binary cross entropy loss function, also known as log-loss, was used. This is generally the appropriate choice for binary classification problems.

The hyper-parameters and the values used are listed below. The order presented represents the order of investigation.

- Optimisers: SGD, Adam
- Learning rate (SGD only): 0.01, 0.1, 0.5, 1.0
- Momentum rate (SGD only): 0, 0.01, 0.1, 0.5, 1.0
- Number of hidden neurons: 5, 10, 15, 20, 25
- Number of hidden layers: 1, 2, 3, 4

Learning and momentum rates were only investigated for SGD and not Adam. That is, in the evaluation of optimisers, the SGD optimiser with the best learning and momentum rates was compared to the default Adam optimiser (ie. the default Tensorflow parameterisation).

C. Fitting

Each model was trained on the train set for a maximum of 300 epochs. At the end of each epoch, the model was evaluated on the validation set. If the validation loss failed to improve for 10 epochs, training was stopped and the model weights were rolled back to the best model. That is, early stopping with a patience of 10 epochs was implemented. The motivation for doing this is to allow the model to determine when to stop training and thus prevent over-fitting. Splitting out an additional validation set as this allows the model to still be evaluated on an unseen test set.

D. Evaluation

Each model was fitted 10 times using randomised starting weights. At the end of each fit, the loss and prediction accuracy of the model evaluated on the train, validation and tests sets were recorded. Additionally, the number of epochs and training time in seconds were also recorded. After the 10 fits, the mean, standard deviation and 95% confidence interval of each of the recorded metrics were computed. The confidence interval formula used was:

$$\bar{x} \pm t_{0.025,9} \times \frac{\mathrm{sd}(x)}{\sqrt{10}}$$

The best setting for each hyper-parameter was deemed to be the one that resulted in the lowest mean test loss. Other metrics are also discussed in the results but generally, the test loss was the determining metric for which setting was to be used in the following rounds of investigation.

At the end of all rounds of investigation, the best model was determined. Using this hyper-parameter set, one final model was fit and the performance of this model was evaluated using diagnostic tools such as the confusion matrix, ROC curve and precision-recall curve. This was compared to the performance of a logistic regression model trained on the train and validation sets (note: since the logistic regression model does not need to decide when to stop training).

IV. Results & Discussion

A. Optimiser

SGD resulted in a lower test loss of 0.6401 (Table II) which compared to 0.6425 (Table III) for Adam. Taking into account the standard deviation and confidence intervals, this difference is small and may be mostly attributed to randomness. SGD also displayed a lower validation loss (0.5787) compared to Adam (0.5909) whilst Adam displayed a lower train loss (0.5536) compared to SGD (0.5607). Thus, neither is clearly superior to the other.

The Adam optimiser however, converged much faster beating out SGD both in terms of number of epochs (72.7 vs 194.7) and time (4.43s vs 10.93s). This is unsurprising since Adam uses an adaptive learning rate, which often results in larger weight updates near the start of training. SGD, on the other hand, only uses a constant learning rate. Thus, for more complex problems involving larger data sets where computational time may be an issue, Adam may be preferred to SGD. However, for the purposes of this study, SGD was deemed to be slightly better given the test loss.

TABLE II
SGD

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5607	0.0201	0.5463	0.5751
Loss: val	0.5787	0.0216	0.5632	0.5941
Loss: test	0.6401	0.0121	0.6314	0.6488
Acc: train	0.7075	0.017	0.6954	0.7197
Acc: val	0.7108	0.0187	0.6974	0.7242
Acc: test	0.6428	0.0199	0.6286	0.657
Num epochs	194.6	67.8269	146.0796	243.1204
Time (s)	10.9267	3.9055	8.1329	13.7205

TABLE III
ADAM

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5536	0.0157	0.5424	0.5648
Loss: val	0.5909	0.0242	0.5735	0.6082
Loss: test	0.6425	0.0097	0.6355	0.6494
Acc: train	0.7159	0.0164	0.7042	0.7277
Acc: val	0.7072	0.0267	0.6881	0.7263
Acc: test	0.6423	0.0176	0.6297	0.6549
Num epochs	72.7	35.0905	47.5978	97.8022
Time (s)	4.4372	1.9637	3.0324	5.842

B. Learning rate

The base hyper-parameterisation 0.01 outperformed all other learning rates in test loss. Using 0.01 resulted in a loss of 0.6401 (Table III) which compared to losses of 0.6475 (Table IV), 0.6469 (Table V) and 0.6673 (Table VI) for learning rates of 0.1, 0.5 and 1 respectively. The loss for 1 is outside the 95% confidence interval of 0.01 (0.6314 to 0.6488) indicating that it performed significantly worse. Though the test losses for 0.1 and 0.5 are both near the upper end of this interval, they both displayed better train and validation losses. Thus, the difference between 0.01, 0.1 and 0.5 is far from clear.

The finding that using a learning rate of 1 resulted in poorer performance is not unsurprising. With a higher learning rate, the model makes larger weight updates and so fails to explore the parameter space as finely. This may result in the model actually missing good parameter combinations.

Also as expected, convergence was reached much faster with higher learning rates. A large improvement was seen when increasing from 0.01 to 0.1 whether the number of epochs (194.6 to 34.5) and time (10.93s to 2.48s) both decrease substantially. Smaller drop offs were seen for further increases. Thus, should computational time have been a problem, a learning rate of 0.10 or 0.50 would have been preferred. However, for this study, a learning rate of 0.01 was used for further investigations due to this setting showing the best test loss performance.

C. Momentum rate

In the comparison of momentum rate, it was again the base optimiser, which uses no momentum, that performed the best with a test loss of 0.6401 (Table III). Using momentum rates of 0.01, 0.1 0.5 and 1.0 resulted in test losses of 0.6407, 0.6503, 0.6495 and 0.7185 respectively. Thus, there is almost no difference between using no momentum and a momentum rate of 0.01. However, the test losses for 0.1, 0.5 and 1.0 are outside of the 95% confidence interval of no momentum (0.6314 to 0.6488) indicating that no momentum is significantly better than these higher rates. This is likely due to a similar effect as in the case of a higher learning rate.

As expected, higher momentum leads to shorter training times. There is practically no difference between using no momentum (194.6 epochs, 10.93s) and 0.01 (193.6 epochs, 11.56s) but we see a slight decrease when using 0.1 (172.1 epochs, 10.13s) and a much larger decrease when using 0.5 (116.8 epochs, 7.25s).

Thus, our conclusion is that using no momentum and 0.01 momentum both work equally well. However, for the next section we continued to us no momentum.

TABLE IV
LEARNING RATE = 0.1

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5506	0.0283	0.5304	0.5709
Loss: val	0.575	0.0246	0.5574	0.5927
Loss: test	0.6475	0.0189	0.634	0.661
Acc: train	0.7131	0.02	0.6987	0.7274
Acc: val	0.7084	0.0192	0.6947	0.7221
Acc: test	0.6312	0.0205	0.6166	0.6459
Num epochs	34.5	13.0491	25.1653	43.8347
Time (s)	2.4823	0.7823	1.9227	3.042

TABLE V
LEARNING RATE = 0.5

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5525	0.0215	0.5371	0.5679
Loss: val	0.563	0.0218	0.5474	0.5786
Loss: test	0.6469	0.0252	0.6289	0.6649
Acc: train	0.7021	0.0184	0.6889	0.7153
Acc: val	0.7042	0.028	0.6842	0.7242
Acc: test	0.6293	0.0296	0.6082	0.6505
Num epochs	19.7	3.7431	17.0223	22.3777
Time (s)	1.6195	0.3319	1.3821	1.857

TABLE VI
LEARNING RATE = 1.0

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5708	0.0383	0.5434	0.5982
Loss: val	0.5766	0.0226	0.5605	0.5927
Loss: test	0.6673	0.0282	0.6471	0.6874
Acc: train	0.6839	0.028	0.6639	0.704
Acc: val	0.7138	0.0178	0.701	0.7265
Acc: test	0.6245	0.0278	0.6046	0.6444
Num epochs	17.3	4.0838	14.3786	20.2214
Time (s)	1.6332	0.4208	1.3322	1.9342

TABLE VII
MOMENTUM = 0.01

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5668	0.0165	0.555	0.5786
Loss: val	0.5772	0.0194	0.5634	0.5911
Loss: test	0.6407	0.0097	0.6337	0.6476
Acc: train	0.6976	0.0173	0.6852	0.71
Acc: val	0.7126	0.0231	0.696	0.7291
Acc: test	0.6365	0.0213	0.6213	0.6518
Num epochs	193.6	76.8392	138.6325	248.5675
Time (s)	11.5644	4.4266	8.3978	14.7311

TABLE VIII
MOMENTUM = 0.1

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5599	0.0206	0.5451	0.5747
Loss: val	0.5828	0.0217	0.5673	0.5984
Loss: test	0.6503	0.0119	0.6418	0.6588
Acc: train	0.7128	0.0181	0.6998	0.7258
Acc: val	0.7102	0.0228	0.6939	0.7265
Acc: test	0.6409	0.015	0.6301	0.6516
Num epochs	172.1	56.1851	131.9076	212.2924
Time (s)	10.125	3.2596	7.7932	12.4568

TABLE IX
MOMENTUM = 0.5

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5594	0.0257	0.541	0.5777
Loss: val	0.5749	0.0295	0.5538	0.596
Loss: test	0.6495	0.0182	0.6365	0.6626
Acc: train	0.7086	0.014	0.6985	0.7186
Acc: val	0.7132	0.0284	0.6928	0.7335
Acc: test	0.6389	0.0242	0.6216	0.6563
Num epochs	116.8	57.4027	75.7366	157.8634
Time (s)	7.2452	3.1715	4.9765	9.514

TABLE X
MOMENTUM = 1.0

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.6376	0.0383	0.6102	0.6651
Loss: val	0.5855	0.0392	0.5574	0.6135
Loss: test	0.7185	0.0567	0.678	0.7591
Acc: train	0.6466	0.0268	0.6275	0.6658
Acc: val	0.6814	0.0366	0.6553	0.7076
Acc: test	0.5755	0.0396	0.5472	0.6038
Num epochs	17.8	4.1042	14.864	20.736
Time (s)	1.5416	0.3323	1.3039	1.7793

D. Summary of optimisers

Of the optimisers, the base optimiser which uses SGD, a learning rate of 0.01 and no momentum performed the best. We note that there were a few parameterisations that resulted in very similar test losses (ie. Adam, momentum rate of 0.01) and thus there is no clear best optimiser. The analysis also suggests that any SGD that uses relatively small learning and momentum rates are adequate as this allows the model to explore a sufficient number of weight parameterisations.

E. Number of hidden neurons

Using 25 hidden neurons resulted in the lowest test loss of 0.6370. Using hidden neurons of 5, 10 (base case), 15 and 20 resulted in test losses of 0.6522, 0.6401, 0.6508 and 0.6378 respectively. Thus, there is basically no difference between using 20 and 25 hidden neurons but using 25 does substantially outperform 5, 10 and 15. The test losses for these paramaterisations are either outside of the 95% confidence interval for 25 or very close to the boundary (0.6335 to 0.6404). Additionally, we note that the test loss for 10 hidden neurons appears to be an outlier as it is substantially lower than 5 and 15. However, given that this model is eventually rejected anyway, this is not a major issue. Further examining the train and validation loss, using 25 hidden neurons outperformed other parameterisations. All of these differences were significant (ie. outside of the 95% confidence interval) except for using 20 hidden neurons. Thus, though there appears to be not much difference between using 20 and 25, we opted for 25 for the next section given it did have the best test loss.

Somewhat surprisingly, training speed was fairly even across all parameterisations, both in terms of epochs and time. Though we may have expected longer training times for the more complex models, the number of hidden neurons is not substantially higher, thus leading to practically no changes in training time.

TABLE XI
HIDDEN NEURONS = 5

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5919	0.0338	0.5678	0.6161
Loss: val	0.6083	0.0511	0.5717	0.6449
Loss: test	0.6522	0.0176	0.6396	0.6648
Acc: train	0.6711	0.0442	0.6395	0.7027
Acc: val	0.679	0.0331	0.6554	0.7027
Acc: test	0.6163	0.0509	0.58	0.6527
Num epochs	193.9	93.6191	126.9289	260.8711
Time (s)	12.2476	5.9057	8.0229	16.4722

TABLE XII
HIDDEN NEURONS = 15

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5476	0.0218	0.532	0.5633
Loss: val	0.5848	0.0263	0.5659	0.6036
Loss: test	0.6508	0.0181	0.6379	0.6638
Acc: train	0.7183	0.0122	0.7096	0.7271
Acc: val	0.706	0.0188	0.6925	0.7195
Acc: test	0.6375	0.0149	0.6268	0.6482
Num epochs	185.3	53.9116	146.734	223.866
Time (s)	11.6631	3.1072	9.4403	13.8858

TABLE XIII
HIDDEN NEURONS = 20

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5414	0.016	0.53	0.5529
Loss: val	0.5682	0.0135	0.5586	0.5779
Loss: test	0.6378	0.0096	0.6309	0.6447
Acc: train	0.7179	0.0112	0.7099	0.7259
Acc: val	0.7228	0.0157	0.7115	0.734
Acc: test	0.6457	0.0199	0.6314	0.6599
Num epochs	187.4	59.3824	144.9204	229.8796
Time (s)	12.5842	3.8264	9.8469	15.3214

TABLE XIV
HIDDEN NEURONS = 25

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5328	0.018	0.5199	0.5457
Loss: val	0.5652	0.0218	0.5496	0.5808
Loss: test	0.637	0.0049	0.6335	0.6404
Acc: train	0.7289	0.0121	0.7202	0.7376
Acc: val	0.7144	0.0196	0.7004	0.7284
Acc: test	0.6558	0.0122	0.647	0.6645
Num epochs	192.5	51.4636	155.6852	229.3148
Time (s)	12.301	3.4937	9.8018	14.8003

F. Number of hidden layers

The final comparison was between differing number of hidden layers. The model using a single hidden layer, which is the one carried over from the previous section, performed the best with test loss of 0.6370 (see Table XV). The results for two, three and four hidden layers were 0.6374, 0.6440 and 0.6485 respectively. Thus, there is basically no difference between using one and two hidden layers, but using one does significantly outperform three and four. The test losses for the latter two are outside of the 95% confidence interval for one hidden layer (0.6335 to 0.6404). The significantly worse test losses indicate that using three or four layers may have resulted in over-fitting. Overall, we concluded that using one layer appears to be the best. Though using two may have been adequate, given that it increases model complexity and objectively had a worse test loss, one hidden layer is the preferred choice.

Looking at training speed, using more hidden layers increases training speed. Using a single hidden layer results in a 192.5 epochs and 12.30s on average, which steadily decreases to 119.4 epochs and 7.43s when using four hidden layers. Though we may have expected training time to increase with model complexity, there is also that trade-off that complex models are able to learn patterns faster. This is evidence by the lower number of training epochs and it appears that this latter effect is the one that dominates.

TABLE XV
HIDDEN LAYERS = 2

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5237	0.0118	0.5153	0.5322
Loss: val	0.575	0.0169	0.5629	0.5871
Loss: test	0.6374	0.0121	0.6287	0.6461
Acc: train	0.7371	0.0131	0.7278	0.7465
Acc: val	0.7102	0.0142	0.7	0.7203
Acc: test	0.6476	0.0223	0.6316	0.6636
Num epochs	139.7	24.1295	122.4388	156.9612
Time (s)	9.2494	1.9511	7.8537	10.6452

TABLE XVI
HIDDEN LAYERS = 3

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5366	0.0162	0.525	0.5482
Loss: val	0.5795	0.0186	0.5662	0.5928
Loss: test	0.644	0.0155	0.6329	0.6551
Acc: train	0.7229	0.0152	0.712	0.7338
Acc: val	0.697	0.011	0.6891	0.7049
Acc: test	0.6433	0.0229	0.6269	0.6596
Num epochs	117.8	30.1139	96.2578	139.3422
Time (s)	7.1709	1.7643	5.9088	8.433

TABLE XVII
HIDDEN LAYERS = 4

	Mean	Std. dev	CI: lower	CI: upper
Loss: train	0.5278	0.0149	0.5171	0.5385
Loss: val	0.5861	0.0333	0.5622	0.6099
Loss: test	0.6485	0.0163	0.6368	0.6601
Acc: train	0.7364	0.0156	0.7252	0.7476
Acc: val	0.7006	0.0268	0.6814	0.7198
Acc: test	0.6399	0.0142	0.6297	0.6501
Num epochs	119.4	24.4549	101.906	136.894
Time (s)	7.4321	1.2553	6.5341	8.3301

G. Best model

To summarise the analysis so far, the best model produced uses SGD with a learning rate of 0.01 and no momentum. It also consists of a single hidden layer with 25 hidden neurons. We did a single fit of this model and evaluated its performance. Additionally, to get a sense of its performance, we also compared it to a logistic regression model.

1) Confusion matrices: The confusion matrix is a contingency table of predictions and actual values. In our Parkinson's example, our target only has two levels which results in a 2×2 matrix. Given that the model outputs a probability rather than a class, the predicted class is 1 if the outputted probability is greater than 0.5 and 0 otherwise. In other words, we use a threshold of 0.5 for making predictions. The confusion matrices for the neural network model as well as the logistic regression are shown. For ease of reading, we have also added the row and column totals.

Though the confusion matrix provides useful information regarding the performance of the model, it is often easier to analyse the evaluation metrics that can be computed from the confusion matrix. This is discussed in the latter sections.

TABLE XVIII
NEURAL NETWORK
CONFUSION MATRIX: TRAIN

	Pred negative	Pred positive	Total actual
Actual negative	236	95	331
Actual positive	77	257	334
Total pred	313	352	665

TABLE XIX
LOGISTIC REGRESSION
CONFUSION MATRIX: TRAIN

	Pred negative	Pred positive	Total actual
Actual negative	225	106	331
Actual positive	121	213	334
Total pred	346	319	665

TABLE XX
NEURAL NETWORK
CONFUSION MATRIX: VAL

	Pred negative	Pred positive	Total actual
Actual negative	59	26	85
Actual positive	24	58	82
Total pred	83	84	167

TABLE XXI
LOGISTIC REGRESSION
CONFUSION MATRIX: VAL

	Pred negative	Pred positive	Total actual
Actual negative	63	22	85
Actual positive	25	57	82
Total pred	88	79	167

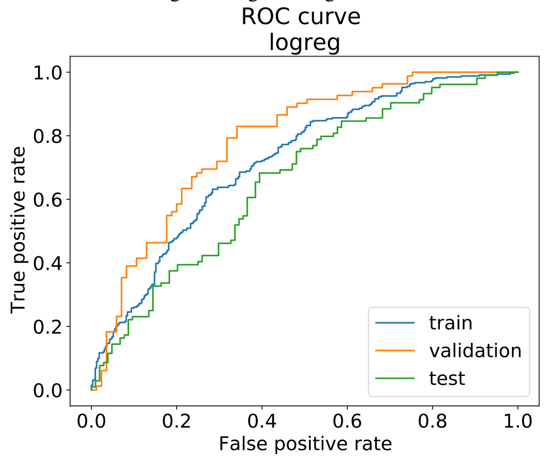
TABLE XXII
NEURAL NETWORK
CONFUSION MATRIX: TEST

	Pred negative	Pred positive	Total actual
Actual negative	69	35	104
Actual positive	36	68	104
Total pred	105	103	208

TABLE XXIII
LOGISTIC REGRESSION
CONFUSION MATRIX: TEST

	Pred negative	Pred positive	Total actual
Actual negative	66	38	104
Actual positive	41	63	104
Total pred	107	101	208

Fig. 5. Logistic Regression: ROC curve



2) ROC and Precision-Recall Curves: The ROC curve plots the true positive rate (tpr) against the false positive rate (tpr) for different threshold settings. The ROC curve of a model that predicts completely randomly is a 45°line where as a model that predicts perfectly passes through the point $(0, 1)$. Thus, the further the curve is towards the upper left corner, the better the model. The ROC curves for the neural network and logistic regression models are shown. It appears that the neural network predicts better on the train and test sets where as the logistic regression performs better on the validation set. This is expected as the logistic regression uses the validation set for training whilst the network only uses this data to determine the stopping time.

The precision recall curve plots the precision against recall (equal to tpr) for different threshold settings. The further the curve is towards the upper left corner, the better the model. The precision recall curves for the neural network and logistic regression models are shown. Again, we can see that the network model performs better on the train and test sets whilst the logistic regression performs better on the validation set.

Fig. 4. Neural Network: ROC curve

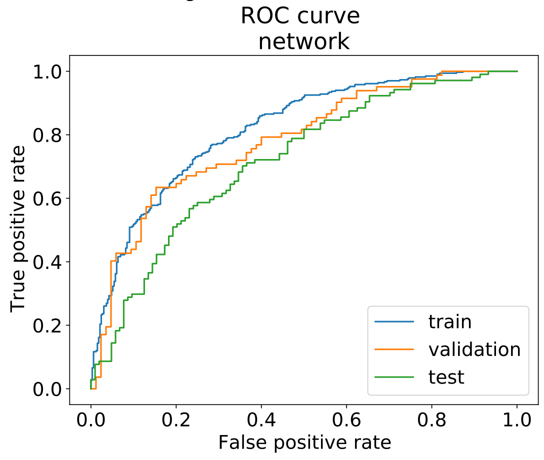


Fig. 6. Neural Network: Precision-recall curve

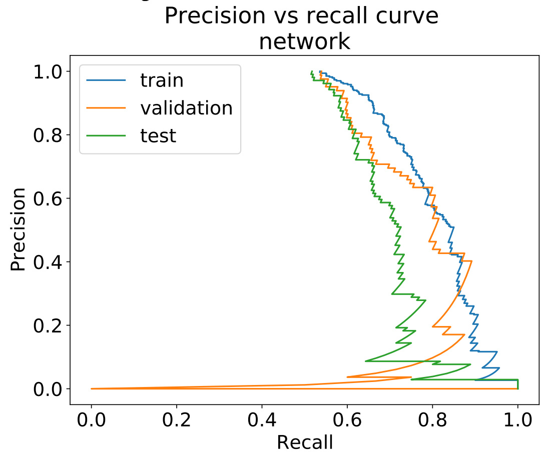
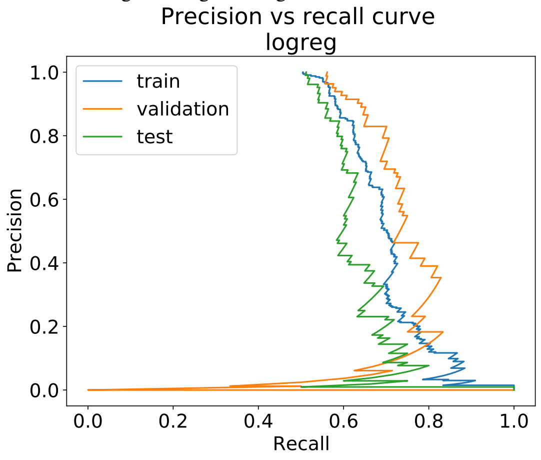


Fig. 7. Logistic Regression: Precision-recall curve



3) Evaluation metrics: Finally, we compare the evaluation metrics between the neural network and logistic regression models. The tpr, fpr, precision recall and f1 score have been computed using a threshold of 0.5.

Examining primarily performance on the test set, we can see that the neural network performs better in all metrics. It has a lower loss (log-loss) of 0.6261 compared to 0.6595 and a higher accuracy of 0.6587 compared to 0.6202. The tpr (or recall) is higher (0.6538 vs 0.6058) and the fpr is lower (0.3365 vs 0.3654) indicating it has predicted a higher proportion of people with PD correctly and has less incorrect predictions of people without PD. The neural network shows a higher precision score (0.6602 vs 0.6238) meaning that a higher proportion of its positive predictions were correct. The f1 score, which is designed to balance recall and precision is also higher (0.6570 vs 0.6146). Finally, the auc, which measures the area under the ROC curve, is higher for the network (0.7186 vs 0.6586). This indicates, as we have already seen in the graph previously, that the ROC curve for the neural network model is more towards the upper left corner and is therefore making better predictions.

TABLE XXIV
NEURAL NETWORK
EVALUATION METRICS

	train	val	test
loss	0.5259	0.5829	0.6261
accuracy	0.7414	0.7006	0.6587
tpr/ recall	0.7695	0.7073	0.6538
fpr	0.287	0.3059	0.3365
precision	0.7301	0.6905	0.6602
f1 score	0.7493	0.6988	0.657
auc	0.8176	0.7816	0.7186

TABLE XXV
LOGISTIC REGRESSION
EVALUATION METRICS

	train	val	test
loss	0.6133	0.5908	0.6594
accuracy	0.6586	0.7186	0.6202
tpr/ recall	0.6377	0.6951	0.6058
fpr	0.3202	0.2588	0.3654
precision	0.6677	0.7215	0.6238
f1 score	0.6524	0.7081	0.6146
auc	0.7209	0.7835	0.6586

V. CONCLUSION

Here is some text. [1]

REFERENCES

[1] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.