An Investigation of Neural Network Architecture using Parkinson Speech Data

Jiaxi Zhao - z5305338

***Abstract*—This paper investigates the effect of various hyper-parameter settings on model performance using the Parkinson's Speech training data set. The best model found uses SGD with a learning rate 0.01 and no momentum and a topology of one hidden layer with 25 neurons. Furthermore, the performance of the network was compared to a logistic regression model where it was determined to be much better.**

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.

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 neurons and number of hidden layers. The paper is organised as follows: section II provides a review of the literature, section III outlines the data used, section IV details the methodology, section V presents the results and offers some discussion and finally section VI concludes.

II. Literature Review

Since the introduction of the Adam optimiser in 2014, it has grown in popularity due its computational efficiency and lack of hyper-parameter tuning [1]. The authors, Kingma and Ba showed that the algorithm performed favourably compared to existing optimisers such as SGD Nesterov and AdaGrad, leading to its widespread adoption. Indeed in a review of various optimisers by Ruder (2016) [2], the author wrote:

"*Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. Kingma et al. show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice.*"

However, a more recent study by Wilson et al (2017) [3] has shown that optimisers using an adaptive learning rate such as Adam fail to generalise well to new data, often performing worse than GD and SGD. As a result, Keskar and Socher (2017) [4] have proposed a hybrid optimiser that switches from Adam to SGD when a certain triggering condition is satisfied. This was shown to have comparable test results to SGD whilst still enjoying the faster convergence speed of Adam. However, research on the optimisers is still far from finished. Indeed, it may be that there is no one 'best' optimiser but the choice depends on the problem being solved.

Tran et al (2006) [5] studied the effect of varying the number of hidden neurons and layers using time series weather data. They discovered that increasing the number of parameters did not necessarily improve model performance and often resulted in over-fitting. On the other hand however, having too few meant that the model was unable to learn the data. Thus, a fine balance needs to be struck. Additionally, they found that for a small fixed number of parameters, organising these in a deep network (5 layers in their study) performed better than a single layer using the same number of parameters. However, when the number of parameters was large, the model with a single layer was preferable. Shafi et al (2006) [6] also discovered that more complex models did not necessarily lead to better performance. Their image processing study showed that a single layer of 40 neurons was sufficient for their purposes.

Finally, hyper-parameter tuning has also been a large area of research. Rather than using a single learning rate, Smith (2018) [7] proposed the use of cyclical learning and momentum rates where the rate is varied between some minimum and maximum value throughout training. The maximum could be determined from a pre-training run where the rate was steadily increased until the model no longer converged.

III. Data

The data used in this study is the Parkinson Speech training data set, collected by the Department of Neurology in Cerrahpasa Faculty of Medicine, Istanbul University [8]. This was obtained from the UCI Machine Learning Repository. 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

IV. Methodology

A detailed description of the methodology is provided in this section. All neural networks were implemented in Python using the Tensorflow 2.0 library [9]. Additionally, Geron’s (2019) [10] Hands-On Machine Learning with Scikit-Learn and Tensorflow was consulted to develop the code.

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.

Fig. 1. Feature density plots: 1-9

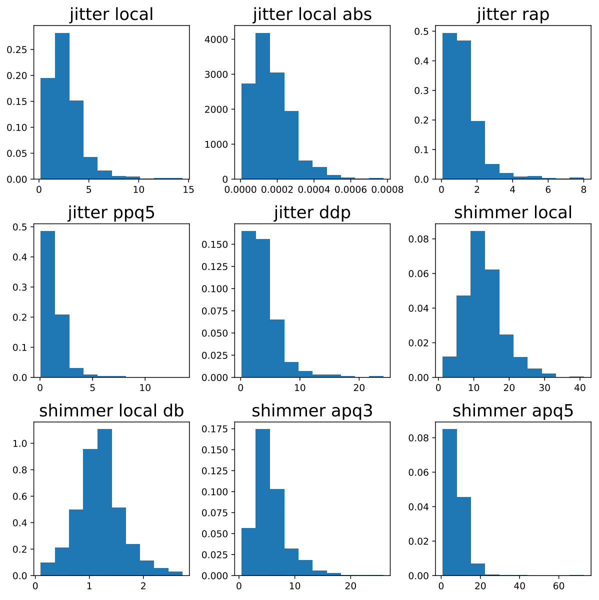
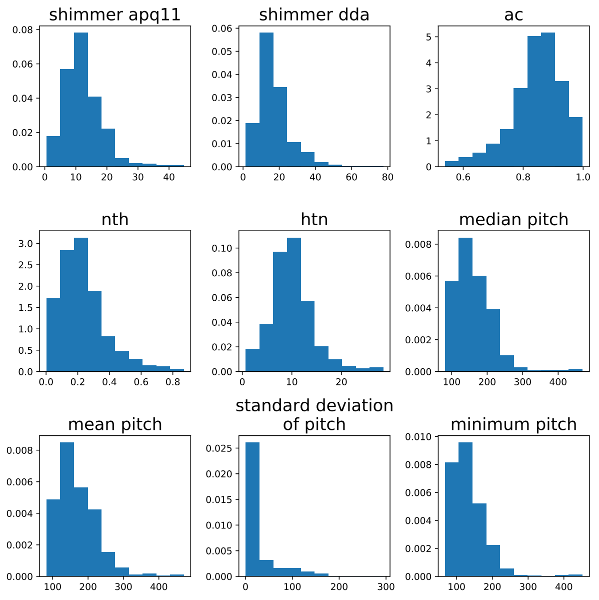


Fig. 2. Feature density plots: 10-18

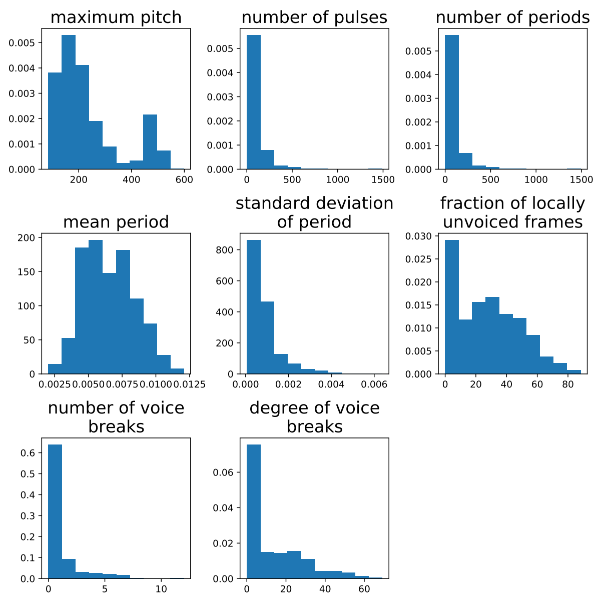


Features were then normalised via standard scaling, that is, by subtracting the sample mean and then dividing by the sample standard deviation:

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

The mean and standard deviation used for standardisation were obtained from the train set only. This ensured that the

Fig. 3. Feature density plots: 19-26



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. Finally, 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 also 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 Adam optimiser with default Tensorflow parameter values.

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 is needed 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.

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. To get a sense of its performance, we also fitted a logistic regression model trained on both the validation and test sets. Both are used for training as the logistic regression model does not use early stopping.

V. Results & Discussion

A. Optimiser

In the comparison between SGD and Adam, 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 and deeper networks, where computational time is 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

In the investigation of 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, 0.6469 and 0.6673 for learning rates of 0.1, 0.5 and 1 respectively (Tables IV - VI). Moreover, the test loss for a learning rate of 1 is outside the 95% confidence interval of the test loss for a learning rate 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 decreased 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.

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

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 (Tables VII - X). 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 use no momentum.

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

Out of the optimiser hyper-parameterisations investigated, 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, SGD with a momentum rate of 0.01) and thus there is no clear best optimiser. The analysis also suggests that any SGD optimiser that uses relatively small learning and momentum rates perform well.

E. Number of hidden neurons

In the investigation of hidden neurons, Using 25 resulted in the lowest test loss of 0.6370 (Table XIV). 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 (Tables III, XI - XIII). Thus, there is practically no difference between using 20 and 25 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 (0.6335 to 0.6404) or close to the boundary. Additionally, we note that the test loss for 10 hidden neurons (base case) does appear to be an outlier. It is substantially lower than 5 and 15. However, as this model is rejected anyway, this is not a major issue for the proceeding analysis. Further examining the train and validation losses, using 25 also outperformed all other parameterisations. Additionally, the differences are significant (ie. outside of the 95% confidence interval) for all except 20 hidden neurons. Thus, 25 was used for the next investigation.

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 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 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 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 (the best model from the previous investigation) performed the best with a test loss of 0.6370 (Table XIV). The results for two, three and four hidden layers were 0.6374, 0.6440 and 0.6485 respectively (Tables XV - XVI). Thus, there is practically 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 a single 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 may have also performed well, we opted for the less complex model.

Examining training speed, using more hidden layers improves training speed. Using a single hidden layer resulted in 192.5 epochs and 12.30s on average, steadily decreasing 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 the trade-off that complex models are able to learn patterns faster evidenced by the lower number of epochs. It appears that this latter effect 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 thus far, the best model produced uses SGD with a learning rate of 0.01 and no momentum. It consists of a single hidden layer with 25 hidden neurons. We did a single fit of this model and compared its performance 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. Since our model outputs a probability rather than a level (ie. PD or healthy), the prediction is 1 (PD) if the probability exceeds 0.5 and 0 (healthy) otherwise. In other words, we use a threshold of 0.5. The confusion matrices for the neural network and logistic regression model are shown. For ease of reading, row and column totals have been added.

Though the confusion matrix provides useful information regarding the performance of the model, it is often easier to analyse the evaluation metrics computed from the confusion matrix. These are discussed in 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

2) *ROC and Precision-Recall Curves:* The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) 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 model are shown in figures 4 and 5. The neural network predicts better on the train and test sets where as the logistic regression predicts better on the validation set. This is to be expected however, as the logistic regression uses the validation set for training whilst the network only uses this data to determine the stopping time.

Fig. 4. Neural Network: ROC curve

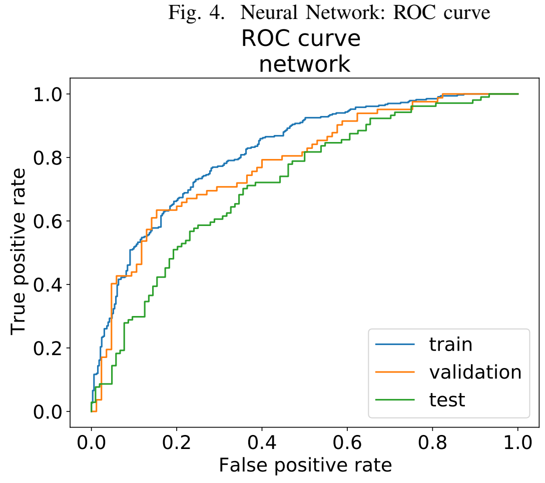
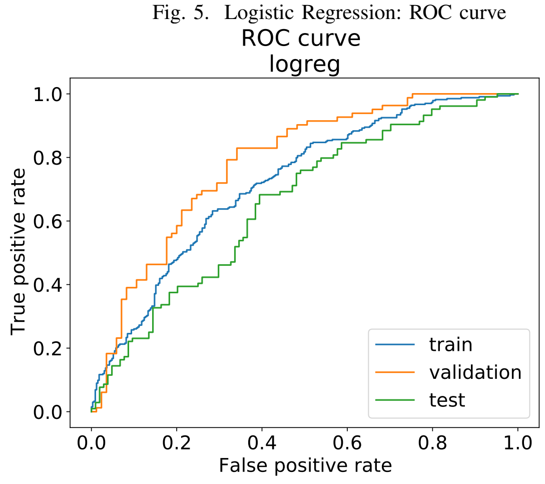


Fig. 5. Logistic Regression: ROC curve



The precision recall curve plots the precision against recall (which is another name for TPR) for different thresholds. The further the curve is towards the upper right corner, the better the model. The precision recall curves for the neural network and logistic regression model are shown in figures 6 and 7. Again, we can see that the network performs better on the train and test sets whilst logistic regression performs better on the validation set.

Fig. 6. Neural Network: Precision-recall curve

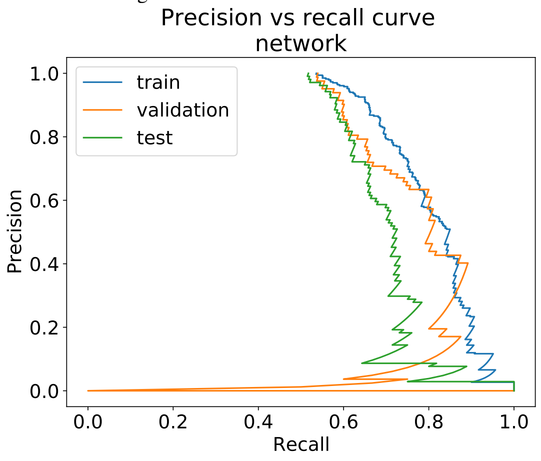
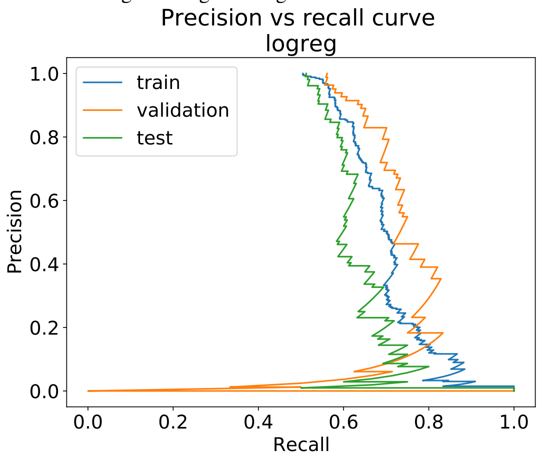


Fig. 7. Logistic Regression: Precision-recall curve



3) Evaluation metrics: Finally, we compare evaluation metrics between the neural network and logistic regression model. The TPR, FPR, precision recall and f1 score have been computed using a threshold of 0.5. These statistics are shown in Table XXIV and XXV.

Examining primarily performance on the test set, the neural network performs better in all metrics. It has a lower loss: 0.6261 vs 0.6595 and a higher accuracy: of 0.6587 vs 0.6202. The TPR (recall) is higher (0.6538 vs 0.6058) and the FPR is lower (0.3365 vs 0.3654) indicating that it has predicted a higher proportion of people with PD correctly and has less incorrect predictions of people without. The neural network also shows a higher precision score (0.6602 vs 0.6238) meaning that a higher proportion of its PD predictions were actually correct. Additionally, the f1 score, which is designed to balance recall and precision is also higher for the network (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. Thus, it can be concluded that the neural network model is superior to logistic regression.

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

VI. CONCLUSION

To conclude, selecting the best hyper-parameters for neural networks has been a challenging task for practitioners and no doubt will continue to be in the future. The sequential investigation procedure used in this model provides a method for doing so but is unlikely to scale well to more complex models or larger data sets.

The analysis concluded that the choice of optimiser between SGD and Adam is not crucial for the Parkinson data set. Both had similar performances on the test set over 10 model fits. Additionally, the learning and momentum rates used should be 'small' as using large rates results in poorer performance. This can be attributed to larger weight updates causing the model to miss optimal parameterisations. However, the definition of 'small' changes problem to problem so simply using the best learning and momentum rates from this analysis in another may not yield good results. In terms of training speed, using Adam performs faster than SGD as does higher learning and momentum rates. One should be careful not to select too high a learning rate or this may lead to poor results.

The investigation of the topology led to one hidden layer with 25 neurons performing the best. Using too few neurons led to poor performance as the model was unable to learn the data and using too many hidden layers resulted in over-fitting.

Finally, the comparison between the best neural network model and a logistic regression model showed that the network was far superior, beating out logistic regression in all metrics examined. This exemplifies the predictive power of networks over classical statistical models and demonstrates the importance of future research in selecting optimal hyper-parameters.

References

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[2] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[3] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in neural information processing systems*, 2017, pp. 4148–4158.

[4] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," *arXiv preprint arXiv:1712.07628*, 2017.

[5] T. T. K. Tran, T. Lee, and J.-S. Kim, "Increasing neurons or deepening layers in forecasting maximum temperature time series?" *Atmosphere*, vol. 11, no. 10, p. 1072, 2020.

[6] I. Shafi, J. Ahmad, S. I. Shah, and F. M. Kashif, "Impact of varying neurons and hidden layers in neural network architecture for a time frequency application," in *2006 IEEE International Multitopic Conference*. IEEE, 2006, pp. 188–193.

[7] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[8] B. E. Sakar, M. E. Isenkul, C. O. Sakar, A. Sertbas, F. Gurgen, S. Delil, H. Apaydin, and O. Kursun, "Collection and analysis of a parkinson speech dataset with multiple types of sound recordings," *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 4, pp. 828–834, 2013.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

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