Grid Search Across Neural Net Machine Learning Models for Classification

Marc Ishak
Student
UNSW
Sydney, Australia
m.ishak@student.unsw.edu.au

***Abstract*—A Grid Search is used to evaluate Neural Net Models across multiple parameters and how they evaluate a test set in multiple metrics. Layer count, layer size, loss function and optimisation algorithms are all considered.**

***Index Terms*—Grid Search, Epoch, Neural Net, Tensorflow, Keras**

I. Introduction

Since the emergence of the backpropgation algorithm [1] finding the optimal parameters for neural network based models has been a fraught task, subject to much research [2] and debate. Over time, many different optimisation algorithms have appeared, including stochastic gradient descent [3], Adam [4], RMSProp, and others [5]. This combined with the increase in the number of available loss functions [8] [9] (such as Kullback–Leibler divergence [6], binary cross entropy [7] and hinge among others) and the increase in computing resources available (allowing for deeper nets with more neurons per layer) have made it difficult to identify the optimal parameters. In this paper I utilise a grid search method across loss function, optimiser, hidden layer size and depth to identify the optimal parameters for the 'Parkinson Speech Dataset with Multiple Types of Sound Recordings' dataset [10].

II. Methodology

A. Reproducability

A `conda` [11] environment was created to handle package management and compatibility. An `environment.yml` file was created to help ensure that the results and methodology explored is reproducible.

B. Data Preparation

After initial data cleaning (as guided [12]), exploratory data analysis (EDA) [13] utilising the python package `pandas-profiling` [14]. Utilising `pandas-profiling`, it became clear that a majority of the variable in the dataset were highly correlated.

As such `scikit-learn` [15] was utilised to carry out min-max scaling and principle component analysis (PCA) [16] was used to reduce the number of input variables down from twenty six to six (as per the elbow method from the scree plot). Approximately 87.86% of the variability of the data was accounted for in theses 6 variables. Fundamentally, reducing the number of input variables helps reduce the complexity of the system, increasing its computational and (minimally) memory efficiency.

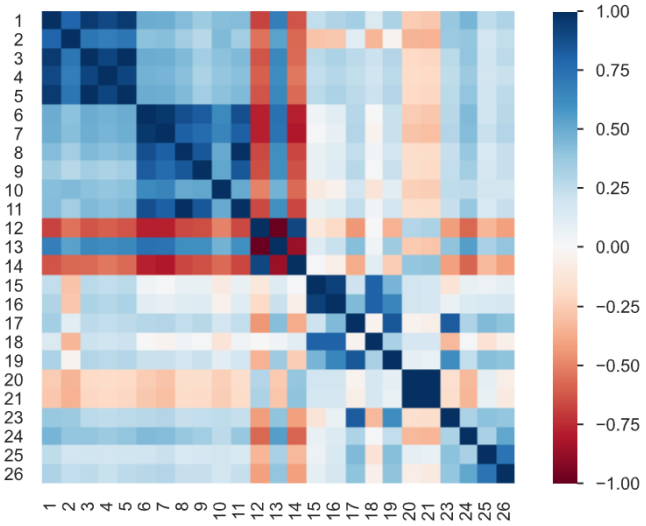

Fig. 1. Correlation Plot for vars in dataset

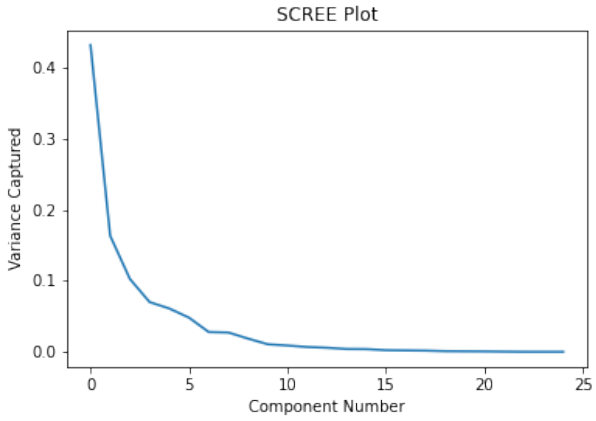

Fig. 2. Scree Plot showing proportion of variance explained per component

After the PCA transformation, a train-test-split was carried out (with a fixed seed) with a test size of 20% of the size of

the data. Once completed, the split data was written to disk in the comma separated values (.csv) format. Furthermore, the training data was then split into training and validation sets at a 50/50 ratio for the model fitting process.

C. Testing Process - Fixed and Explored Parameters

The key parameters explored are:

1) The number of hidden layers
   - 1,2,5 layers
2) The size of all hidden layers (ie: neurons per hidden layer)
   - 1,5,10,50 neurons
3) The loss function
   - Kullback–Leibler divergence
   - Hinge
   - Binary cross entropy
4) The optimisation function
   - Stochastic gradient descent
   - Adam
   - RMSProp

- Note: the optimisation and loss functions utilised their default arguments.

Each model then utilised the same metrics of Receiver Operating Characteristic Area Under the Curve (ROC-AUC) and accuracy. Each model also used a single sigmoidal activation neuron for output and was trained over 300 epochs with a batch size of 300. Overall there were 108 unique models trained at 10 repetitions each, leading to 1080 models trained over the grid search.

D. Testing Process - `ModelWrapper.py`

The `ModelWrapper.py` file consists of the `ModelWrapper` class, which acts as a simple wrapper to the larger Keras [18] modelling package. It contains methods that store the model parameters, build, compile plot and summarise the model, as well as providing further reporting and prediction features.

E. Testing Process - `ModelInstance.py`

The `ModelInstance.py` file consists of the `ModelInstance` class and the `summarise_model_instances` function. The `ModelInstance` class acts as a wrapper for the instance (i.e: the "run") of the model, storing the model, dataset, layers and names. Due to this structure the actions of fitting and predicting are merged into a single function, and the model is built and compiled during the initiation step. The class also produces a comprehensive report of that specific model instance, including plots showing the loss and ROC-AUC over the course of training, a classification report, the specific ROC-AUC score and the structure of the model itself. The `summarise_model_instances` function helps summarise these model instances by taking in a list of model instances, retrieving their properties and performance metrics and exporting them as a dictionary for ease of use and compatibility with the pandas [19] library.

F. Testing Process - Reporting and Results Processing

By utilising `ModelInstance.py` and `ModelWrapper.py` the reporting and results processing is pretty simple. Each model generated generates a new folder with a plots of ROC-AUC and loss over the epoch training process, a model outline plot, an ROC-AUC curve on the test set and an overall model report txt (consisting of a model outline, classification report and ROC-AUC metric). Statistics are collected into a pandas dataframe where they are then written to a .csv. These results are then imported and aggregated within a jupyter notebook [20] using pandas and analysed with matplotlib [21].

III. Results

A. ROC

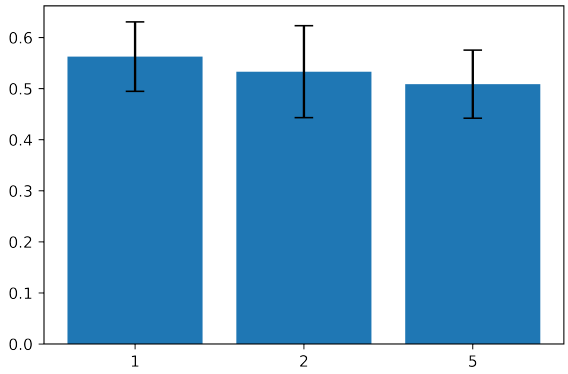

Fig. 3. Bar plot showing mean AUC-ROC (with standard error) for various neural network depths

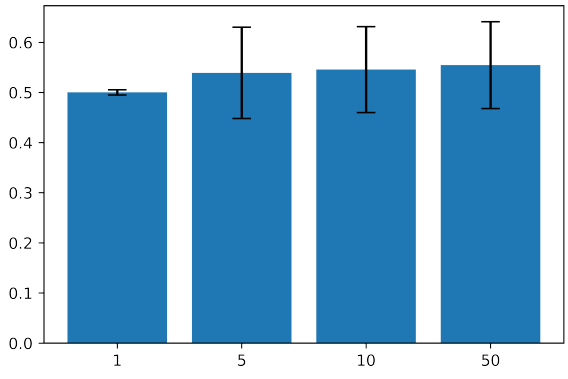

Fig. 4. Bar plot showing mean AUC-ROC (with standard error) for various hidden layer sizes

Initial inspection of the average results shows that single layer networks tend to preform best and that as layer size increases, more performance is achieved. Adam and RMSPRop

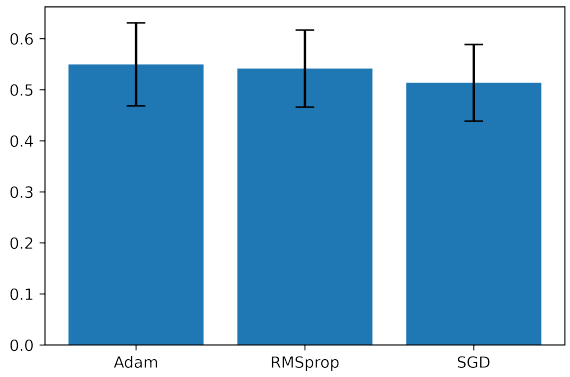

Fig. 5. Bar plot showing mean AUC-ROC (with standard error) for various optimisation functions

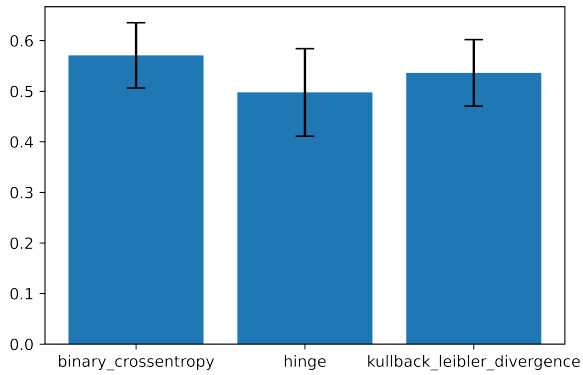

Fig. 6. Bar plot showing mean AUC-ROC (with standard error) for various loss functions

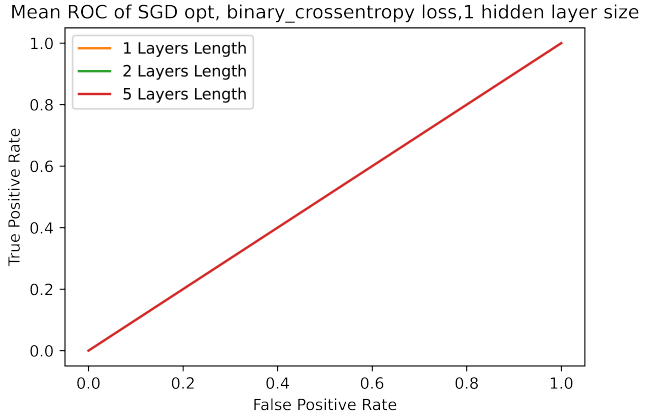

Fig. 7. Mean ROC plot for networks with binary crossentropy loss, 1 hidden layer size and SGD optimisation

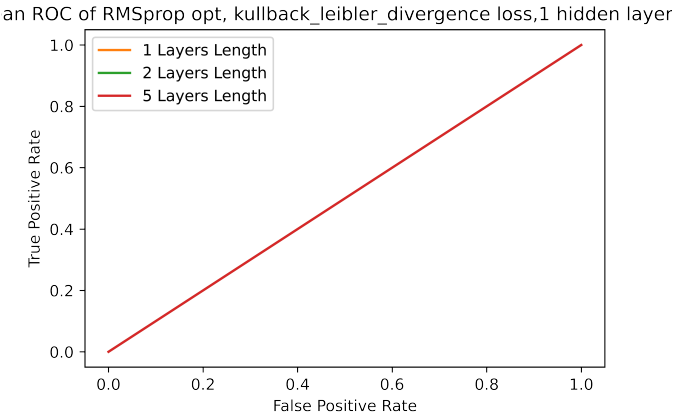

Fig. 8. Mean ROC plot for networks with kl loss, 1 hidden layer size and RMSProp optimisation

loss functions preformed comparably, leading stochastic gradient descent and binary cross entropy as the best loss function. However, as everything lies within our standard error (and hence a 95% confidence interval) of each other, none of these results are significant. A high variance between individual instances of the same time suggest a highly complex error surface for all loss functions.

Further inspection reveals particular characteristics that tend to preform poorly. Layer sizes of 1 yield zero improvement over random chance in ROC-AUC, no matter the depth of the network, the loss function or the optimiser. This is ultimately unsurprising as effectively the error boundary fails to be defined in the model, leading to a distorted error surface.

By running a regression model on the ROC-AUC scores we see that increasing the overall depth, changing the loss function from binary cross entropy to hinge or kl-divergence or changing the optimisation function from Adam to stochastic gradient descent of the model reduces performance. RMSprop did decrease performance as well compared to Adam but failed to do so at a level that's significant at an $\alpha = 0.05$ level. Contrary to the other variables, increasing the hidden layer size had a positive (and significant) affect on the ROC-AUC score. However, increasing in the hidden layer size does not scale proportionally to the ROC-AUC score and significantly increases the computational complexity (especially as more layers are added). Also worth noting is the poor fit, an R^2 value of 0.331 suggests that this model doesn't capture the entire relationship of the model. This could potentially be remedied by increasing the number of runs per model, fitting 10 points lends itself to a increased likelihood of high variance

For the most part these result are the same when looking at f1 scores instead of the ROC-AUC scores. However, in this case kl-divergence actually improved accuracy over binary cross entropy. This is something which could potentially explored further, given the relationship between the metrics.

Another finding worth noting is that more complex models tended to "overfit" the data, particularly models with 5 layers at 50 neurons per layer. Regularisation techniques [22] [23] like dropout [24] may help boost reduce overfitting and boost prediction accuracy.

Dep. Variable:	roc_auc_scores	**R-squared:**	0.331
Model:	OLS	**Adj. R-squared:**	0.325
Method:	Least Squares	**F-statistic:**	58.80
Date:	Wed, 28 Oct 2020	**Prob (F-statistic):**	2.77e-87
Time:	18:04:11	**Log-Likelihood:**	1429.3
No. Observations:	1080	**AIC:**	-2839.
Df Residuals:	1070	**BIC:**	-2789.
Df Model:	9		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	0.5789	0.006	92.955	0.000	0.567	0.591
C(layers_length)[T.2]	-0.0295	0.005	-6.121	0.000	-0.039	-0.020
C(layers_length)[T.5]	-0.0540	0.005	-11.198	0.000	-0.063	-0.045
C(hidden_layer_size)[T.5]	0.0389	0.006	6.977	0.000	0.028	0.050
C(hidden_layer_size)[T.10]	0.0454	0.006	8.155	0.000	0.034	0.056
C(hidden_layer_size)[T.50]	0.0543	0.006	9.756	0.000	0.043	0.065
loss_func[T.hinge]	-0.0732	0.005	-15.184	0.000	-0.083	-0.064
loss_func[T.kullback_leibler_divergence]	-0.0346	0.005	-7.183	0.000	-0.044	-0.025
opt_func[T.RMSprop]	-0.0083	0.005	-1.716	0.087	-0.018	0.001
opt_func[T.SGD]	-0.0362	0.005	-7.495	0.000	-0.046	-0.027

Omnibus:	42.413	**Durbin-Watson:**	1.183
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	46.843
Skew:	-0.510	**Prob(JB):**	6.73e-11
Kurtosis:	2.980	**Cond. No.**	6.48

TABLE I
OLS REGRESSION RESULTS AND DIAGNOSTICS

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Dep. Variable:	f1_scores	**R-squared:**	0.466
Model:	OLS	**Adj. R-squared:**	0.462
Method:	Least Squares	**F-statistic:**	103.9
Date:	Fri, 30 Oct 2020	**Prob (F-statistic):**	2.51e-139
Time:	20:39:24	**Log-Likelihood:**	95.570
No. Observations:	1080	**AIC:**	-171.1
Df Residuals:	1070	**BIC:**	-121.3
Df Model:	9		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	0.2564	0.021	11.974	0.000	0.214	0.298
C(layers_length)[T.2]	-0.0850	0.017	-5.126	0.000	-0.118	-0.052
C(layers_length)[T.5]	-0.1026	0.017	-6.184	0.000	-0.135	-0.070
C(hidden_layer_size)[T.5]	0.2910	0.019	15.196	0.000	0.253	0.329
C(hidden_layer_size)[T.10]	0.2866	0.019	14.965	0.000	0.249	0.324
C(hidden_layer_size)[T.50]	0.3190	0.019	16.657	0.000	0.281	0.357
loss_func[T.hinge]	-0.1099	0.017	-6.625	0.000	-0.142	-0.077
loss_func[T.kullback_leibler_divergence]	0.2564	0.017	15.462	0.000	0.224	0.289
opt_func[T.RMSprop]	-0.0070	0.017	-0.421	0.674	-0.040	0.026
opt_func[T.SGD]	-0.0459	0.017	-2.770	0.006	-0.078	-0.013

Omnibus:	31.678	**Durbin-Watson:**	1.408
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	32.974
Skew:	-0.409	**Prob(JB):**	6.91e-08
Kurtosis:	2.750	**Cond. No.**	6.48

TABLE II
OLS REGRESSION RESULTS AND DIAGNOSTICS

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

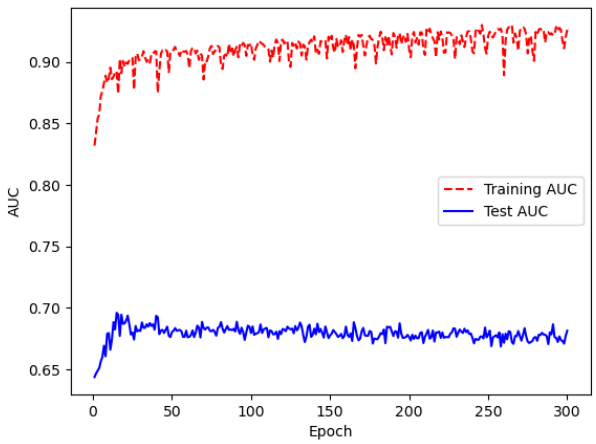

Fig. 9. Sample Epoch Plot showing overfitting, model has 5 hidden layers of 50 neurons each

IV. Conclusion

This paper has taken a tour of the performance of various neural networks on the 'Parkinson Speech Dataset with Multiple Types of Sound Recordings' dataset. Given the limited timeframe, there is still a wide berth to explore and analyse. I have built a struture which aims to reduce the complexity of this analysis and provide a streamlined way to develop and design such techniques.

References

[1] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[2] S. Trenn, "Multilayer Perceptrons: Approximation Order and Necessary Number of Hidden Units," in IEEE Transactions on Neural Networks, vol. 19, no. 5, pp. 836-844, May 2008, doi: 10.1109/TNN.2007.912306.

[3] Bottou, L., 2010. Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

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

[5] Xu, Jianzhong & Yan, Fu 2018, 'Hybrid Nelder–Mead Algorithm and Dragonfly Algorithm for Function Optimization and the Training of a Multilayer Perceptron', Arabian journal for science and engineering (2011), vol. 44, no. 4, pp. 3473–3487.

[6] Kullback, S.; Leibler, R. A. On Information and Sufficiency. Ann. Math. Statist. 22 (1951), no. 1, 79–86. doi:10.1214/aoms/1177729694. https://projecteuclid.org/euclid.aoms/1177729694

[7] De Boer, P.T., Kroese, D.P., Mannor, S. and Rubinstein, R.Y., 2005. A tutorial on the cross-entropy method. Annals of operations research, 134(1), pp.19-67.

[8] Rosasco, L., Vito, E.D., Caponnetto, A., Piana, M. and Verri, A., 2004. Are loss functions all the same?. Neural Computation, 16(5), pp.1063-1076.

[9] Janocha, K. and Czarnecki, W.M., 2017. On loss functions for deep neural networks in classification. arXiv preprint arXiv:1702.05659.

[10] Erdogdu Sakar, B., Isenkul, M., Sakar, C.O., Sertbas, A., Gurgen, F., Delil, S., Apaydin, H., Kursun, O., 'Collection and Analysis of a Parkinson Speech Dataset with Multiple Types of Sound Recordings', IEEE Journal of Biomedical and Health Informatics, vol. 17(4), pp. 828-834, 2013.

[11] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Web. ¡https://anaconda.com¿.

[12] Chandra, R., 2020. Ed — Digital Learning Platform. [online] Edstem.org. Available at: ¡https://edstem.org/courses/4751/lessons/4575/slides/33747¿ [Accessed 29 October 2020].

[13] Tukey, J.W., 1977. Exploratory data analysis (Vol. 2, pp. 131-160).

[14] Martin Sotir, Joseph Yuen, Brian Lee, Stephanie Rivera, nscsekhar, abdulAziz, Pandas-profiling.github.io. 2020. Introduction — Pandas-Profiling 2.9.0 Documentation. [online] Available at: ¡https://pandas-profiling.github.io/pandas-profiling/docs/master/rtd/¿ [Accessed 29 October 2020].

[15] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, D., Brucher, M., Perrot, M., and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, p.2825–2830.

[16] Wold, S., Esbensen, K. and Geladi, P., 1987. Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3), pp.37-52.

[17] Abdi, H. and Williams, L.J., 2010. Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4), pp.433-459.

[18] Chollet, F., and others. (2015). Keras. https://keras.io.

[19] Wes McKinney 2010 . Data Structures for Statistical Computing in Python . In Proceedings of the 9th Python in Science Conference (pp. 56 - 61).

[20] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing 2016. . Jupyter Notebooks – a publishing format for reproducible computational workflows

[21] Hunter, J. 2007. Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), p.90–95.

[22] Kakade, S.M., Shalev-Shwartz, S. and Tewari, A., 2012. Regularization techniques for learning with matrices. The Journal of Machine Learning Research, 13(1), pp.1865-1890.

[23] Kukačka, J., Golkov, V. and Cremers, D., 2017. Regularization for deep learning: A taxonomy. arXiv preprint arXiv:1710.10686.

[24] Gal, Y. and Ghahramani, Z., 2016. A theoretically grounded application of dropout in recurrent neural networks. In Advances in neural information processing systems (pp. 1019-1027).