COMP7703 – Demo 3 (Practical 9)
Joel Thomas 44793203

Model Setup:

- Number of hidden layers = 2
- Number of neurons in each layer = 500
- Activation function = reLU (rectified linear unit)
    - $f(x) = \begin{cases} 0, x \leq 0 \\ x, x > 0 \end{cases} = \max(0, x)$
    - Non-linear and differentiable almost everywhere except at $x = 0$ (left-derivative of 0, right-derivative of 1 → derivative undefined)
    - Lower bound at 0 but no upper bound
    - Advantage = enables network to run much faster during both training (several derivatives during backprop will simply be 0) as well as during prediction (faster matrix multiplications) → computationally efficient.
        - Because the sigmoid and hyperbolic tangent functions have almost every input yielding an output > 0, for a large neural network, almost all neurons will be activated in some way, so the activation is intense (Kızrak, 2019).
    - Disadvantage = whenever the reLU outputs 0 (i.e. $x \leq 0$), no learning is done on that neuron (Kızrak, 2019)
- Optimiser = Adam optimiser with parameters $\{\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999\}$
    - Essentially Momentum and RMSProp optimisers combined together
    - Notion of momentum as well as an adaptive learning rate similar to RMSProp (intuition – heavy ball with friction) (Ruder, 2016)
    - "Gold standard" of optimisers since it attempts to cover the weaknesses of each optimiser before it
- Number of training iterations (epochs) = 100

1.

- This methodology for assessing a model is known more formally as the Train-and-Test method according to Michie et al. (Michie, 1994)
- Hardest aspect for Q1-Q4 was enabling reproducible results working with Python
    - Had to set seeds for not just TensorFlow but Numpy as well
    - After hours of checking code, commenting out $tf.reset\_default\_graph()$ worked
    - Also came across an issue when setting Numpy seed within $MLPModel()$ function would completely reset random number generation (RNG) → needed to bring this out
    - Finally, always need to restart kernel in notebook and run code again for reproducibility due to above issue
        - i.e. simply running code block again won't reset the RNG
- Experiment closest to mean result (given by seed value):
    - For Sonar dataset → (TensorFlow) seed = 2
    - For Diabetes dataset → seed = 8

Results from training network 10 times with random weight initialisations (Test set accuracies)		
DATASET	**Sonar**	**Diabetes**
Minimum	0.8286	0.5352
Maximum	0.9286	0.7031
Mean	0.8914	0.6394
Standard deviation (sample/unbiased)	0.0321	0.0568

Table 1: Results for Q1.

2.

- Using an alternative method known as k-fold Cross Validation to assess performance of the model (same terminology used by Michie et al.)
    - Preferred in **moderate/large-sized datasets** i.e. ≥ 1000 samples
        - See explanation in **Q4.** below, confidence intervals on the mean estimated classification rate using this method tend to be much larger
    - Expect higher statistical variability i.e. standard deviation relative to the Bootstrap method
    - Expect average of estimated classification rates to be unbiased
- $k = 10$:
    - $\frac{1}{10} \times 100 = 10\%$ validation/test and $\frac{9}{10} \times 100 = 90\%$ training sets for each split

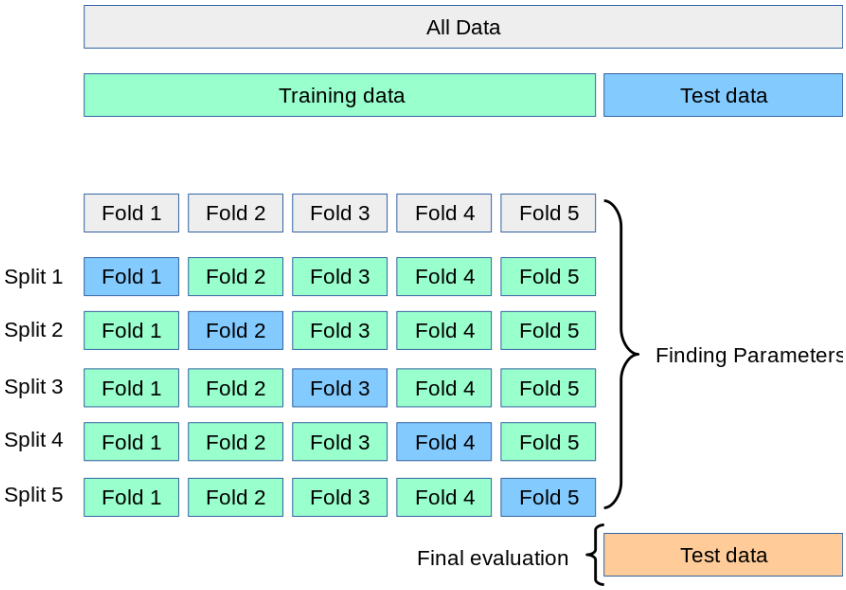

Figure 1: How k-Fold Cross Validation works.

Results from 10-fold cross validation		
DATASET	**Sonar**	**Diabetes**
Minimum	0.7143	0.5658
Maximum	0.9524	0.7532
Mean	0.8324	0.647
Standard deviation (sample/unbiased)	0.0736	0.072

Table 2: Results for Q2.

3.

- Using Train-and-Test method again here
    - Difference to Q1 – using same dataset but generating 10 random partitions (fixed 67%/33% splits) to assess model performance
- To generate 10 random partitions, first randomly shuffle data then split into 67% training and 33% test sets
    - Used $np.random.permutation()$ function here

Results from retraining network 10 times based on 10 random partitions of the dataset		
DATASET	**Sonar**	**Diabetes**
Minimum	0.7	0.4102
Maximum	0.8571	0.7109
Mean	0.7971	0.6281
Standard deviation (sample/unbiased)	0.0446	0.0832

Table 3: Results for Q3.

4.

- Using an alternative method known as the Bootstrap to assess performance of the model (same terminology used by Michie et al.)
    - Preferred in **small-sized datasets** i.e. < 1000 samples
        - Preferable for both Sonar (208 samples) and Diabetes (768 samples) datasets
        - **Why?** Gives much smaller confidence intervals for mean of estimated classification rates but the downside is that these estimated classification rates are optimistic (i.e. biased upwards)
    - Expect lower statistical variability i.e. standard deviation relative to the k-fold CV method
    - Expect average of estimated classification rates to be biased upwards
- Aside: different to Bagging because we are training the same model each time
    - Have controlled stochasticity of the neural network by ensuring we use the same seed value every time when training model repeatedly within a loop
    - Also not changing model parameters → same model being retrained on different data (applicable to not just Q4, but Q2-Q4)

Results from retraining network 10 times based on 10 bootstrapped datasets		
DATASET	**Sonar**	**Diabetes**
Minimum	0.75	0.4615
Maximum	0.8514	0.6949
Mean	0.8098	0.624
Standard deviation (sample/unbiased)	0.0307	0.065

Table 4: Results for Q3.

Comparison between Q2 and Q4 hypotheses:

- The Bootstrap appears to have lower statistical variability than k-fold CV:
    - **True** for both datasets, analysed by verifying standard deviations and comparing Q2 and Q4 graphs
- Average of estimated classification rates for the Bootstrap tends to be upward biased:
    - **True** for both datasets, analysed by comparing mean result to both minimum and maximum results
        - "On average, results are seemingly closer to the maximum for the Bootstrap compared to k-fold CV" (?)
- Important – these claims need to be verified formally using statistical hypothesis testing

References:

- Ruder, S., 2016. *An overview of gradient descent optimization algorithms*. [online] Available at: <https://ruder.io/optimizing-gradient-descent/> [Accessed 22 May 2021].
- Kızrak, A., 2019. *Comparison of Activation Functions for Deep Neural Networks*. [online] Medium. Available at: <https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a> [Accessed 22 May 2021].
- Michie, D., Spiegelhalter, D. and Taylor, C., 1994. *Machine Learning, Neural and Statistical Classification*. pp.108-109.