Week 12 - Neural Network

1) **Using the data synthesis R script provided by the instructor as part of the week 11 assignment instructions, produce datasets of the following sizes, and fit deep learning models with the configurations shown below. Associated with each model, record the following performance characteristics: training error, validation (i.e., holdout set) error, time of execution. Use an appropriate activation function.**

Data Size	Configuration	Training error	Validation error	Time of execution	Accuracy
1000	1 Hidden Layer 4 nodes	0.6534	0.4894	10.4880 seconds	0.81
10000	1 Hidden Layer 4 nodes	0.2709	0.2879	33.3940 seconds	0.8825
100000	1 Hidden Layer 4 nodes	0.0796	0.0784	332.8047 seconds	0.734
1000	2 Hidden Layer 4 nodes	0.4671	0.5118	9.3840 seconds	0.85
10000	2 Hidden Layer 4 nodes	0.2228	0.2346	35.0092 seconds	0.903
100000	2 Hidden Layer 4 nodes	0.0436	0.038	383.2921 seconds	0.9851

2) Based on the results, which model do you consider as superior, among the deep learning models fit?

Based on the results, the 2-hidden-layer (4 nodes each) model consistently outperforms the 1-hidden-layer (4 nodes) model across all dataset sizes, making it the superior choice. For smaller datasets (1,000 samples), the 2-layer model achieves higher accuracy (0.85 vs. 0.81) and trains slightly faster (9.3840s vs. 10.4880s), despite a marginally higher validation error. With 10,000 samples, it further improves, delivering better accuracy (0.903 vs. 0.8825) and a lower validation error (0.2346 vs. 0.2879) with comparable training time. Most notably, on the largest dataset (100,000 samples), the 2-layer model excels with near-perfect accuracy (98.51%), the lowest validation error (0.038) and well-balanced training/validation errors, indicating no overfitting. Although training takes longer (~383s), the performance gains justify the computational cost. Thus, the 2-hidden-layer configuration is the best overall, particularly for larger datasets where its robustness and accuracy shine. If further optimization is needed, adjusting node counts or learning rates could be explored, but the current setup already demonstrates strong results.

3) Next, report the results (for the particular numbers of observations) from applying xgboost (week 11 – provide the relevant results here in a table). Comparing the results from XGBoost and deep learning models fit, which model would you say is superior to others? What is the basis for your judgment?

XGBoost in R – direct use of xgboost() with simple cross-validation		
Sample size	Accuracy	Time taken
1000	0.925	0.0855
10000	0.971	0.303622
100000	0.983	2.521756
XGBoost in R – via caret, with 5-fold CV simple cross-validation		
Sample size	Accuracy	Time taken
1000	0.95	9.944
10000	0.98	42.68
100000	0.98	373.66
XGBoost in Python via scikit-learn and 5-fold CV		
Sample size	Accuracy	time taken
1000	0.95	0.2962
10000	0.97	0.4361
100000	0.98	1.0521

When comparing XGBoost and deep learning (DL) models across varying dataset sizes (1,000, 10,000, and 100,000 samples), several critical factors emerge that determine which model is superior for practical applications.

1. Accuracy Performance:
XGBoost consistently achieves higher accuracy than the deep learning model on small and medium-sized datasets. For 1,000 samples, XGBoost (via R/caret or Python) attains 95% accuracy, while the DL model reaches only 85%. This trend continues with 10,000 samples, where XGBoost achieves 97-98% accuracy compared to DL's 90.3%. Even at 100,000 samples, where DL slightly outperforms XGBoost (98.51% vs. 98.3%), the difference is marginal (<0.3%) and likely insignificant for most real-world applications.

2. Computational Efficiency (Training Time):
XGBoost is dramatically faster than deep learning, making it far more practical for deployment. For 1,000 samples, Python's XGBoost trains in just 0.3 seconds, while the DL model takes 9.38 seconds (~30x slower). The gap widens with larger datasets: at 100,000 samples, XGBoost (Python) completes training in 1.05 seconds, whereas the DL model requires 383 seconds (6+ minutes)—a 365x slowdown. Even R's slower XGBoost implementation (with cross-validation) remains competitive, taking 42.68 seconds for 10,000 samples compared to DL's 35.01 seconds, while delivering higher accuracy (98% vs. 90.3%).

3. Data Efficiency and Robustness:
Deep learning models typically require large datasets to perform well, whereas XGBoost excels even with limited data. The DL model struggles on small datasets (e.g., 85% accuracy at 1,000 samples), while XGBoost maintains strong performance (95% accuracy). This makes XGBoost a better choice for scenarios where data collection is expensive or time-consuming.

4. Practical Considerations:
- Ease of Tuning: XGBoost works well with default hyperparameters, while DL models require extensive tuning (layers, nodes, activation functions, regularization).
- Interpretability: XGBoost provides feature importance scores, helping users understand model decisions, whereas DL models act as "black boxes."
- Scalability: XGBoost can handle structured data efficiently, while DL is better suited for unstructured data (images, text).

Final Judgment: XGBoost is the Superior Choice

Basis for Decision:

1. Better Accuracy on small/medium datasets, with near-identical performance on large data.

2. Faster Training (seconds vs. minutes/hours), enabling rapid experimentation and deployment.

3. Greater Flexibility—works well with limited data and requires less tuning.

4. Explainability—provides insights into feature contributions, unlike DL’s opaque nature.