

A Neural Network Intrusion Detection System

Multi-Layer Perceptron Applied to NSL-KDD Network Traffic

CYSE 420 — Applied Machine Learning in Cybersecurity
Assignment 06 | Carl Lochstampf | Fall 2026

1. Introduction and Dataset

This report documents the design, training, tuning, and evaluation of a multi-layer perceptron (MLP) intrusion detection system implemented in TensorFlow 2.21 with Keras 3.15. The network classifies individual network connection records as normal traffic or intrusion attempts. Because Assignment 04 applied tree-based ensembles to the same dataset and the same test partition, this report also compares the two model families directly, which is where the most interesting result appears.

The dataset, `Intrusion_data-1.csv`, is the NSL-KDD training subset: 25,192 connection records described by 42 columns — 38 numeric features, three categorical features, and a binary class label. The numeric features cover four families: basic connection fields (`duration`, `src_bytes`, `dst_bytes`), content fields (`num_failed_logins`, `root_shell`), time-based traffic statistics (`count`, `error_rate`), and host-based statistics in the `dst_host_*` block. The three categorical features are `protocol_type` (3 levels), `service` (66 levels), and `flag` (11 levels).

Inspection found no missing values and no duplicate records. Two features, `num_outbound_cmds` and `is_host_login`, take a single value across the entire dataset. The class distribution is 13,449 normal to 11,743 anomaly, or 46.6 percent anomaly, which makes the majority-class baseline 53.4 percent accuracy. Unlike the malware dataset used in Assignments 03 and 05, the classes here are close to balanced, so accuracy is a defensible headline metric rather than a misleading one.

Methodological commitment. Every hyperparameter in this report was selected on a validation partition that is disjoint from the test partition, using a rule fixed before any result was examined. The test set was evaluated exactly once, in Section 6. Selecting parameters by test-set performance and then reporting on that same test set would inflate every figure below, so the experiment was structured to make that impossible rather than to avoid it by discipline alone.

2. Data Preprocessing

2.1 Feature and label preparation

The two constant features were removed, leaving 39 predictors. A constant column contributes an identical term to every weighted sum in the network and cannot influence any weight update, so it adds computation without information. Because `nunique() == 1` is a structural property of a column rather than a statistic estimated from data, removing these before partitioning leaks nothing.

The label was mapped explicitly with `(df['class'] == 'anomaly').astype(int)`, making anomaly the positive class. This was deliberate. The label arrives as the strings `normal` and `anomaly`, and an automatic encoder sorts alphabetically, which would assign anomaly to 0 and silently invert every probability and ROC curve downstream without raising an error.

2.2 Partitioning

The data was split three ways rather than two. A 70/30 split produced a 7,558-record test set, matching the partition used in Assignment 04 so the Section 7 comparison is made on identical ground. The remaining 70 percent was split again 80/20, giving 14,107 training and 3,527 validation records. Both splits were stratified, and all three partitions carry an anomaly rate of 0.4661.

The validation partition serves two purposes that cannot be served by the test set. It is the signal early stopping monitors during training, and it is the surface on which the nine tuning configurations in Section 5 are scored. Without it, tuning would either contaminate the test set or be impossible.

2.3 Feature transformation

Three transformations were applied, with all fitted parameters estimated from the training partition alone and then applied unchanged to validation and test.

Logarithmic compression. Three features are unbounded volume counters spanning many orders of magnitude: `dst_bytes` has a skewness of 71.9 and `src_bytes` 25.7 on the training partition, with maxima in the millions against medians near zero, while `duration` reaches 42,862 seconds. A `log1p` transform — $\log(1 + x)$ — was applied to these three. The `1 +` matters: `dst_bytes` has a median of zero, and a plain logarithm would return negative infinity. The transform has no fitted parameters, so applying it per-partition cannot leak information.

Standardization. All 36 numeric features were scaled to zero mean and unit variance. This is the single largest preprocessing difference between this assignment and Assignment 04. Decision trees split on thresholds and are invariant to feature scale, so the ensembles needed no scaling at all. A neural network multiplies every input by a weight and sums the results; a feature measured in millions would dominate that sum and produce enormous gradients on the first update, destabilizing training before it began.

Categorical encoding. The three categorical features were one-hot encoded with `handle_unknown='ignore'`, expanding 3 columns into 80. Combined with the 36 scaled numeric features, this gives the network an input dimension of 116. The `ignore` setting emits an all-zero row for an unseen category rather than raising; the training partition happens to contain all 66 `service` levels, but a different seed or a rarer level would otherwise halt the pipeline.

3. Network Architecture

The network is a fully connected feedforward architecture: an input layer of 116 units, two hidden layers of 64 and 32 units with ReLU activation, a dropout layer at rate 0.3 after each hidden layer, and a single output unit with sigmoid activation. It has 9,601 trainable parameters.

Layer	Output shape	Parameters
Input	(None, 116)	0
Dense, ReLU	(None, 64)	7,488
Dropout (0.3)	(None, 64)	0
Dense, ReLU	(None, 32)	2,080
Dropout (0.3)	(None, 32)	0
Dense, sigmoid	(None, 1)	33
Total		9,601

Table 1. Network architecture. All parameters are trainable.

Layer widths were kept deliberately modest. At 9,601 parameters against 14,107 training records, the network has limited capacity to memorize individual examples. A substantially wider network would invite the first cause of overfitting the course module identifies: a model too complex for the available data.

ReLU was chosen for the hidden layers because its derivative is exactly 1 across the entire positive range. Sigmoid and tanh have derivatives well below 1, so gradients shrink multiplicatively as they propagate backward and effectively vanish before reaching the early layers. ReLU is the standard remedy for that vanishing gradient problem.

Dropout at 0.3 randomly disables 30 percent of a layer's neurons on each training step, preventing the network from depending on any single feature pathway. It is active during training only; at prediction time the full network participates. This asymmetry produces a visible artifact in Section 4 worth understanding rather than mistaking for a bug.

The sigmoid output produces a continuous probability in the open interval (0, 1), paired with binary cross-entropy loss. That continuous output is what makes ROC analysis possible natively — a notable contrast with Assignment 05, where K-means and DBSCAN produce no probability and an anomaly score had to be constructed by hand before any ROC curve could be drawn.

4. Training Process

A baseline model was trained first with Adam at its default learning rate of 0.001 and a batch size of 32, establishing a reference against which tuning could be measured. The epoch budget was set to 100 as a ceiling rather than a target, with early stopping monitoring validation loss at a patience of 10 and `restore_best_weights=True`.

Training stopped at epoch 37 and restored the weights from epoch 27, reaching a validation loss of 0.0096, accuracy of 0.9977, and AUC of 0.9997. Each epoch took roughly one second on CPU.

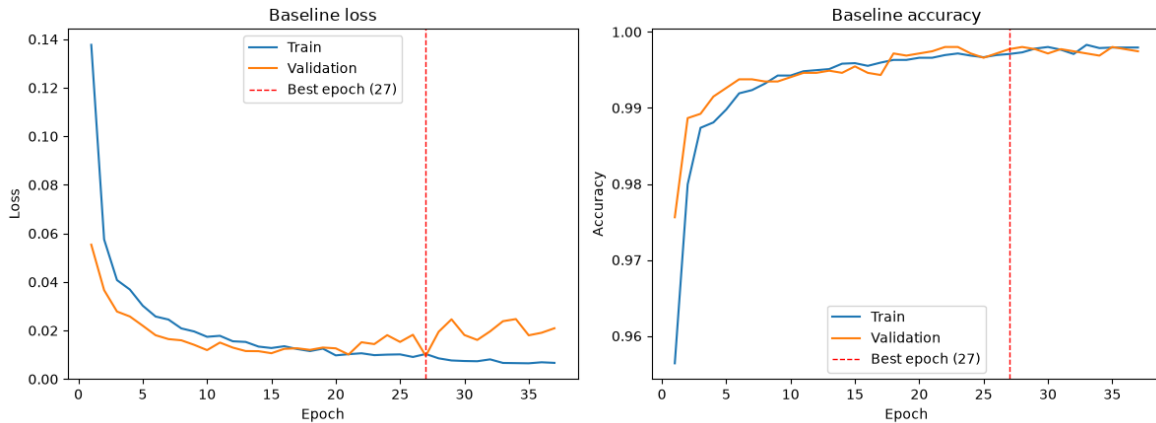


Figure 1. Baseline training and validation curves. The red line marks epoch 27, where validation loss reached its minimum and early stopping later restored the weights.

Three observations from Figure 1 shaped the rest of the experiment.

First, the left panel shows textbook overfitting. Training loss declines monotonically from 0.1377 to 0.0067 and never stops improving, while validation loss bottoms at 0.0096 on epoch 27 and then climbs to 0.0209 by epoch 37. Without `restore_best_weights`, training would have concluded holding the epoch-37 weights — more than twice as bad on validation loss as the weights actually kept.

Second, validation loss sits *below* training loss for the first eighteen epochs, which appears backwards. This is the dropout asymmetry described in Section 3: during training 30 percent of neurons are disabled and the handicapped network reports inflated loss, while validation runs the complete network. The curves cross into the expected arrangement only once genuine overfitting begins.

Third, and most useful for interpreting Section 6, loss and accuracy disagree after epoch 27. Validation accuracy holds between 0.997 and 0.998 even as validation loss doubles. The network is not getting more answers wrong; it is becoming more confident about answers it already had wrong. Cross-entropy penalizes confident errors heavily while accuracy cannot see them at all. This is precisely why early stopping monitors loss rather than accuracy.

5. Hyperparameter Tuning

The assignment names three hyperparameters: learning rate, batch size, and epochs. The first two were searched over a 3×3 grid, while the third was determined automatically by early stopping within each configuration — which turns out to be the more informative treatment, since the optimal epoch count is not independent of the learning rate.

Selection rule, fixed before any run: the winning configuration is the one with the lowest validation loss. Learning rates spanned two orders of magnitude (0.01, 0.001, 0.0001) and batch sizes were powers of two (32, 64, 128). Critically, the random seed was reset before every run, so all nine models begin from identical weight initialization, shuffling, and dropout masks. Without that reset, part of the difference between configurations would reflect random starting conditions rather than the hyperparameters under test.

Learning rate	Batch size	Epochs run	Best epoch	Val. loss	Val. accuracy	Val. AUC
0.01	128	22	12	0.01296	0.9952	0.99965
0.01	32	29	19	0.01325	0.9977	0.99965
0.01	64	34	24	0.01374	0.9972	0.99966
0.001	32	30	20	0.01392	0.9974	0.99962
0.001	64	28	18	0.01624	0.9974	0.99961
0.001	128	48	38	0.01630	0.9974	0.99911
0.0001	32	91	81	0.01798	0.9938	0.99908
0.0001	64	100	100	0.01890	0.9938	0.99906
0.0001	128	100	100	0.02243	0.9935	0.99943

Table 2. Tuning grid sorted by validation loss. The selected configuration is the highlighted first row.

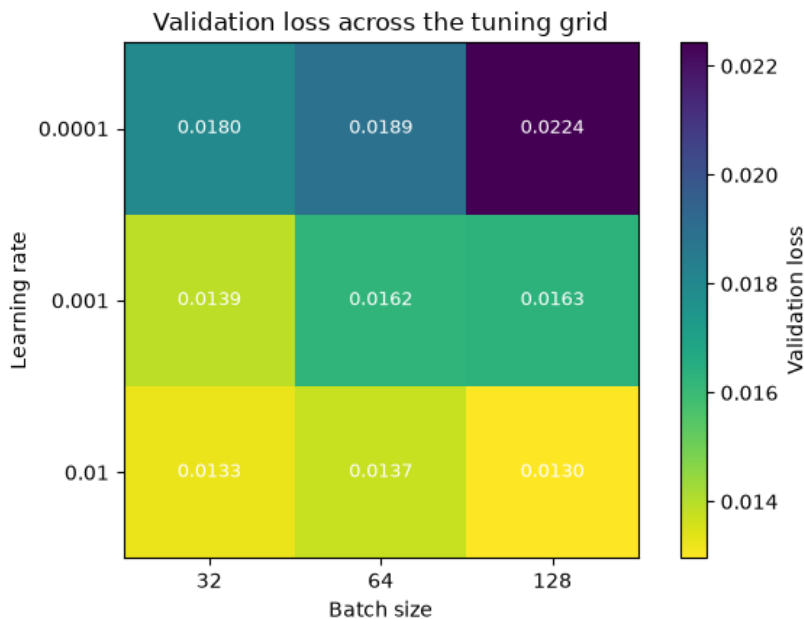


Figure 2. Validation loss across the grid. Darker indicates higher loss; the bottom row, at learning rate 0.01, is uniformly best.

5.1 What the grid establishes

Learning rate has a real and monotonic effect. Mean validation loss falls from 0.0198 at a learning rate of 0.0001, to 0.0155 at 0.001, to 0.0133 at 0.01. That is a 50 percent spread across the range, visible as the horizontal banding in Figure 2, and it is the one claim this grid supports unambiguously.

Epoch count is a consequence of learning rate, not an independent choice. The configurations at learning rate 0.01 converged in 22 to 34 epochs; those at 0.0001 needed 91 to 100. Fixing a single epoch budget across all nine would have handicapped one end of the range or wasted computation at the other. Early stopping resolves this automatically, and the `best_epoch` column documents the outcome per configuration.

5.2 What the grid does not establish

The top four configurations cannot be meaningfully ranked. They span validation losses of 0.01296 to 0.01392 — a range of 0.00096. To calibrate that figure against run-to-run variance, note that the Section 4 baseline used learning rate 0.001 with batch size 32 and reached a validation loss of 0.0096, while the grid's cell for those same hyperparameters reached 0.01392. The two runs differ only in random seed state, and the gap between them is 0.0043 — more than four times the spread separating the top four configurations. The ordering within that group is noise.

This has a direct consequence for how the result should be stated: **tuning did not produce a measurably better model than the default configuration.** It established that learning rate matters, that batch size does not within this range, and that the default was already near the achievable ceiling. Claiming an improvement would not survive scrutiny.

Validation AUC is saturated and was useless for selection. All nine configurations fall between 0.99906 and 0.99966, a total range of 0.0006. Had AUC been the selection metric, the choice would have been arbitrary.

Two configurations were truncated rather than converged. Learning rate 0.0001 at batch sizes 64 and 128 both recorded their best epoch as epoch 100, meaning validation loss was still falling when the budget expired. They are underfit by the epoch ceiling, not by the learning rate, and should not be read as converged comparisons. This is the concrete form of the standard warning that too small a learning rate makes training prohibitively slow.

The selected configuration has the lowest accuracy of the nine at 0.9952, against 0.9977 for learning rate 0.01 with batch size 32. The rule was fixed on validation loss in advance and therefore stands, but a rule written on accuracy would have selected a different model. Neither difference exceeds the measured noise. This is the same loss-versus-accuracy divergence seen in Figure 1, now influencing model selection.

The final model — learning rate 0.01, batch size 128 — was retrained under the same seed and reproduced the grid result exactly at 0.01296 validation loss, converging at epoch 12 and stopping at epoch 22.

6. Performance Evaluation

The test partition was evaluated once, with the final model and a decision threshold of 0.5. No threshold tuning was performed.

Metric	Value
Accuracy	0.9938
ROC-AUC	0.9989
Majority-class baseline accuracy	0.5339
False positive rate	0.0067
False negative rate	0.0057

Table 3. Test-set performance on 7,558 held-out records.

Class	Precision	Recall	F1-score	Support
Normal	0.9950	0.9933	0.9942	4,035
Anomaly	0.9924	0.9943	0.9933	3,523

Table 4. Per-class test performance.

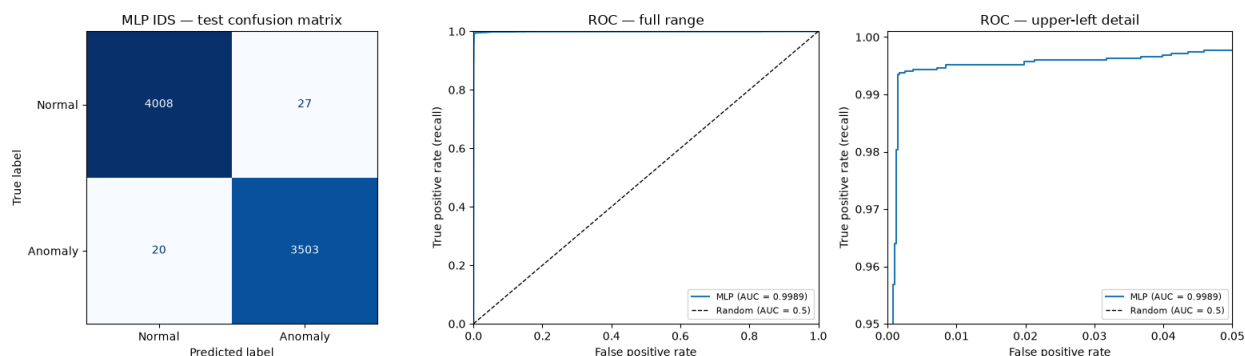


Figure 3. Test-set confusion matrix and ROC curve. The third panel zooms to false positive rates below 0.05, the region in which an IDS actually operates.

The model correctly classified 4,008 normal and 3,503 anomalous records, producing 27 false positives and 20 false negatives. Accuracy of 0.9938 stands well above the 0.5339 baseline, and with near-balanced classes that margin is meaningful rather than an artifact of class imbalance.

The zoomed ROC panel is the informative one. At full range the curve is pinned into the upper-left corner and conveys only that performance is high. Zoomed, it shows the curve rising almost vertically to approximately 0.994 recall at a false positive rate near zero, then flattening into a slow climb. Operationally this means the model separates the overwhelming majority of traffic cleanly, and that the final half-percent of intrusions can only be recovered by accepting a rapidly increasing volume of false alarms.

The two error types carry different operational costs. A false negative is an intrusion delivered to the endpoint undetected. A false positive is legitimate traffic raised for analyst review. Which matters more depends on deployment: an alerting system with human triage can absorb false positives to maximize recall, while an automated blocking system must weigh precision far more heavily because each false positive interrupts legitimate activity.

7. Comparison with Ensemble Methods

Assignment 04 applied three tree-based ensembles to this same dataset and the same 7,558 record test partition, which permits a direct comparison rather than an approximate one.

Model	Accuracy	ROC-AUC	FP	FN	Total errors
MLP (Assignment 06)	0.9938	0.9989	27	20	47
Gradient Boosting, tuned (Asgn 04)	0.9968	0.9998	5	19	24
Random Forest, default (Asgn 04)	0.9971	0.9999	4	18	22
AdaBoost, default (Asgn 04)	0.9774	0.9976	61	110	171

Table 5. Test-set comparison across model families on identical data.

The MLP matches the ensembles on detection and loses decisively on false alarms. Its 20 false negatives are statistically indistinguishable from Gradient Boosting's 19 and Random Forest's 18 across 3,523 anomalies. Its 27 false positives are roughly six times their 4 and 5. Expressed operationally: all three models catch essentially the same intrusions, and the MLP hands analysts six times the alert volume to triage for that identical detection rate. In a production IDS that ratio is the difference between a tool security teams trust and one they learn to ignore.

The MLP is nonetheless a capable model. It outperforms AdaBoost by a wide margin — 47 total errors against 171 — placing it between the shallow boosting ensemble and the deeper tree ensembles rather than at the bottom of the field.

The broader observation concerns problem structure. On tabular data with 116 engineered features, gradient-boosted trees and random forests remained competitive with, and here superior to, a neural network — while requiring no feature scaling, no logarithmic compression, and no learning-rate search. The MLP required all three. Neural networks earn their advantage on data with spatial or sequential structure, where convolutional and recurrent architectures exploit relationships between neighboring inputs. Network connection records, represented as independent tabular features, offer no such structure for the architecture to exploit.

This comparison has a real limit. It measures one modest architecture against tuned ensembles. A wider or deeper network, different dropout rates, or class-weighted loss might narrow the false-positive gap. The result characterizes this model, not the ceiling for neural networks on this problem.

8. Error Analysis

The 47 misclassified records were examined against their original, untransformed feature values to determine whether the errors are diffuse or concentrated.

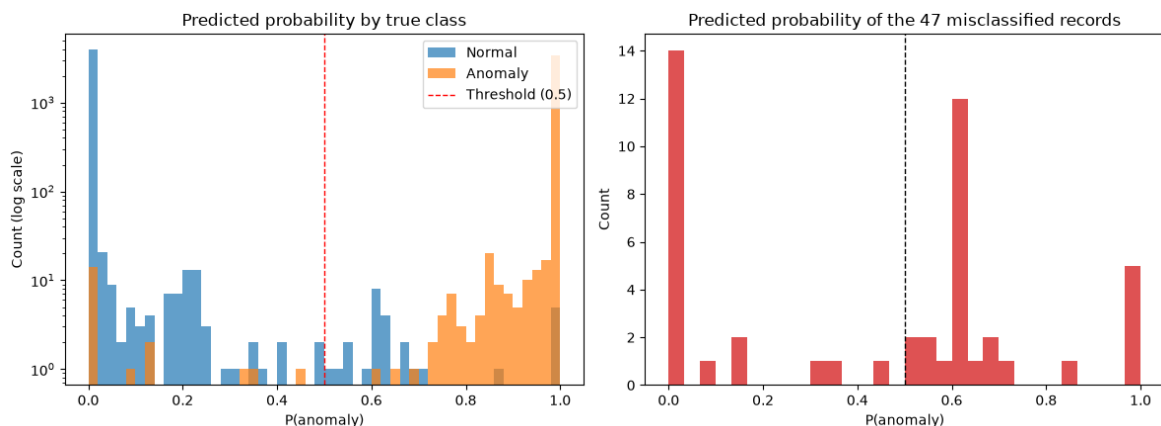


Figure 4. Predicted probability by true class on a logarithmic count scale (left) and the probability distribution of the 47 errors (right).

8.1 Errors are concentrated in ICMP echo traffic

Of the 27 false positives, 16 are ICMP — and every single ICMP error is a false positive, with none in the opposite direction. The service breakdown localizes them precisely: `eco_i` (echo reply) accounts for 12 and `ecr_i` (echo request) for 4. The model is flagging ordinary ping traffic as intrusion.

This is explicable rather than arbitrary. In NSL-KDD, ICMP echo serves both as routine network diagnostics and as the delivery vehicle for smurf and ping-of-death attacks. The available header features evidently do not separate benign pings from malicious ones. It is also the most actionable finding in this report: addressing ICMP handling alone would eliminate more than half of all false alarms.

8.2 Missed intrusions concentrate in session protocols

The false negatives cluster in `ftp_data` (6), `telnet` (4), `http` (3), `ftp` (2), and `smtp` (2) — 17 of 20. These are attacks conducted through protocols whose legitimate use looks structurally similar, which is the difficult case for any feature-based detector.

8.3 Threshold tuning would not recover most errors

Measuring each error by its distance from the 0.5 threshold: 6 fall within 0.1 of the boundary, 21 between 0.1 and 0.4, and 20 more than 0.4 away. The right panel of Figure 4 shows where those 20 sit — approximately 15 false negatives clustered at a predicted probability near 0.0, and 5 false positives near 0.98.

The 15 confident false negatives are the most serious result in this analysis. These are intrusions the network assigns near-zero probability of being anomalous. No adjustment to the decision threshold reaches them; recovering them would require different features or a different model. Only 6 of 47 errors are genuine near-misses that threshold tuning could plausibly convert.

9. Conclusions

1. The MLP is an effective intrusion detector. Test accuracy of 0.9938 and ROC-AUC of 0.9989 against a 0.5339 majority-class baseline, with 20 missed intrusions and 27 false alarms across 7,558 records.

2. It matches tree ensembles on detection and loses on false alarms. Against Assignment 04 on identical test data, its false negatives are level with Gradient Boosting and Random Forest, while its false positives run roughly six times higher — the same detection at six times the alert burden.

3. Learning rate mattered; batch size did not. Mean validation loss fell monotonically across the learning-rate range while batch-size differences remained inside the measured run-to-run variance.

4. The tuning grid could not separate its top configurations. The best four sat within 0.00096 validation loss of one another, while re-running a single configuration under different seed state moved it by 0.0043. The selected model is defensible but not demonstrably superior to its neighbors, and the default configuration was already near the achievable ceiling.

5. Epoch count is determined by learning rate, not chosen independently. Early stopping settled on 22 to 34 epochs at the highest learning rate and 91 to 100 at the lowest.

6. Errors are concentrated rather than diffuse. ICMP echo traffic produces 16 of 27 false positives, and session protocols account for 17 of 20 false negatives. Fifteen of the false negatives are confident errors that no threshold adjustment would recover.

10. Limitations

The two configurations at learning rate 0.0001 with batch sizes 64 and 128 reached the 100-epoch ceiling with validation loss still decreasing, so they were truncated rather than converged and do not constitute fair comparisons against the configurations that terminated on early stopping.

Architecture was held constant at 64 and 32 hidden units with dropout 0.3 throughout. Layer width, depth, and dropout rate were never searched, so the false-positive gap against the tree ensembles may reflect this specific network rather than the model family in general.

Each grid cell was trained exactly once. Given that run-to-run variance was measured at 0.0043 in validation loss — larger than the spread among the top four configurations — reliable ranking would require several runs per cell with variance reported.

The decision threshold remained at 0.5 and was never tuned. The error analysis indicates this costs relatively little, since 41 of 47 errors lie more than 0.1 from the boundary, but a precision-recall curve would quantify the available trade-off properly.

NSL-KDD is a curated research benchmark derived from traffic captured in 1999. Attack techniques, protocol usage, and baseline traffic patterns have changed substantially since, so performance measured here should not be read as an estimate of performance on a contemporary production network.

The Assignment 04 comparison assumes both assignments partitioned the data under the same random seed. Partition sizes and class rates match exactly, but row-level identity of the two test sets was not verified directly.