Expert Systems With Applications 207 (2022) 117938

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Quality of randomness and node dropout regularization for fitting neural networks

Aki Koivu [a,b,*], Joona-Pekko Kakko [b], Santeri Mäntyniemi [a,b], Mikko Sairanen [b]

[a] University of Turku, Faculty of Technology, Department of Computing, Turku, 20014, Finland
[b] PerkinElmer Wallac Oy, Mustionkatu 6, Turku, 20750, Finland

ARTICLE INFO

Keywords:
Quality of randomness
Node dropout
Neural network
True random number
Quantum random number generation

ABSTRACT

Quality of randomness in generating random numbers is an attribute manifested by a sufficiently random process, and a sufficiently large sample size. To assess it, various statistical tests for it have been proposed in the past. The application area for random number generation is wide in natural sciences, and one of the more prominent and widely adopted is machine learning, where bounded randomness or stochastic random number generation has been utilized in various tasks. The artificial neural networks used for example in deep learning use random number generation for weight initialization, optimization and in methods that aim to reduce the overfitting phenomena of these models. One of these methods include node dropout, which has been widely adopted. The method's internal logic is heavily dictated by a random number generator it utilizes. This study investigated the relationship of quality of randomness and the node dropout regularization in terms of reducing overfitting of neural networks. Our experimentation included five different random number generators, which output were tested for quality of randomness by various statistical tests. These sets of random numbers were then used to dictate the internal logic of a node dropout layer in a neural network model, in four different classification tasks. The impact of data size and relevant hyperparameters were tested, and the overall amount of overfitting, which was compared against the randomness results of a generator. The results suggest that true random number generation in node dropout can be both advantageous and disadvantageous, depending on the dataset and prediction problem at hand. These findings suggest that fitting neural networks in general can be improved by adding random number generation experimentation to modelling.

1. Introduction

When fitting probabilistic models into data, nondeterminism can work in your favour. When properly bounded, nondeterministic algorithms can quickly arrive to a feasible solution, while deterministic algorithms arrive to the best, worst and average solution in substantially larger amount of time (Floyd, 1967).

The current formulation of the feed-forward neural networks leverage this bounded nondeterministic or stochastic behaviour in multiple different ways. Stochastic optimization algorithms such as stochastic gradient descent (SGD) (Ruder, 2016) along with random weight initialization are used for the purpose of optimizing or fitting the model to training data. This way, large datasets can be processed in a feasible amount of time, because all of the possible points of the search space are not calculated and evaluated.

Bounded random number generation has also been leveraged to improve generalizability in neural networks, in the form of noise injection (Matsuoka, 1992) and node dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012). These mechanisms try to reduce the over-fitting phenomena (Tetko, Livingstone, & Luik, 1995), where the model completely fits into training data while reducing generalizability to perform adequately with unseen data.

Quality of randomness in a random number generation can vary. Pseudorandom numbers can appear random, but they are produced by a deterministic process (Vadhan et al., 2012). This is the case for most random numbers generated for fitting neural networks or any stochastic machine learning algorithms, because the hardware and software they are calculated with are deterministic by design. On the other hand, true random numbers in theory can be derived from processes that are observed to be truly random. Examples of these include quantum mechanical phenomena such as the photoelectric effect with beam splitters (Jennewein, Achleitner, Weihs, Weinfurter, & Zeilinger, 2000)

* Corresponding author at: University of Turku, Faculty of Technology, Department of Computing, Turku, 20014, Finland.
E-mail addresses: aki.i.koivu@utu.fi (A. Koivu), joona-pekko.kakko@perkinelmer.com (J. Kakko), sjjman@utu.fi (S. Mäntyniemi), mikko.sairanen@perkinelmer.com (M. Sairanen).

https://doi.org/10.1016/j.eswa.2022.117938
Received 10 February 2022; Received in revised form 2 May 2022; Accepted 19 June 2022
Available online 26 June 2022

and the avalanche breakdown in semiconducting circuits (Lampert, Wahby, Leonard, & Levis, 2016).

Quality of randomness is an essential metric for systems that leverage randomness for their primary function (Demirhan & Bitirim, 2016). There exists physical devices that measure truly random natural processes in order to produce true random numbers, that can be used in deterministic environments such as computers. Because of this, successful applications for these devices have been shown to be cryptographic encryption and lottery machines (Demirhan & Bitirim, 2016; Stipcevic, 2012). While the true randomness status of the numbers generated by these devices can be contested not to be fully random due to microscale production variations that affect the measurement devices, in practise they produce true random numbers according to the many statistical tests designed to measure quality of randomness in a set of numbers (Marsaglia et al., 2002; Ritter, 1996).

Artificial neural networks utilize random number generation in various steps during the fitting or training process (Hopfield, 1982). The nondeterminism is lost after the training phase is completed, and using the model for inference is a deterministic process, therefore they are commonly described as being stochastic because of their common optimization methods (Spall, 2005). The effect of quality of randomness on fitting feed-forward neural networks can be easily demonstrated by assigning same initial weights to two hidden units in a hidden layer. If their initial states are completely the same and no randomness is introduced, both of them are updated in the same deterministic manner (Goodfellow, Bengio, & Courville, 2016). Pseudorandom number generation has been the standard for neural network-related optimization in the past, however how their quality of randomness affects the optimization process is an open research topic (Bird, Ekárt, & Faria, 2020; Heese, Wolter, Mücke, Franken, & Piatkowski, 2021).

Node dropout can be considered one of the most prominent regularization method for neural networks, where the aim is to randomly choose hidden nodes to become inactive during a training epoch (Hinton et al., 2012). The core idea of this is to force adjacent hidden nodes in a hidden layer to learn different weights and biases. This forced diversity between hidden nodes is what disables the network from learning the most optimal fit for training data, and the product of this is better generalizability to unseen data. The random selection of hidden nodes utilizes uniform pseudorandom number generation, and is therefore an ideal candidate for quality of randomness investigation in a neural network context.

In this study, we investigate the relationship of quality of randomness and neural network-related regularization method node dropout. Popular pseudorandom number generator algorithms are compared against true random number generators, which are based on observing some natural phenomena. Results are derived using multiple datasets, random number generators and neural network node dropout hyperparameters. This novel investigation proposes that quality of randomness could be impactful to modelling outcomes in terms of generalizability. If neural network models utilizing node dropout during training could be improved by simply using more random sequences to dictate the dropout process, it would have implications to fitting better neural networks in general. The structure of the paper will be the following: 1. lists the prior literature of the subject, 2. describes the materials and methods used in the study, 3. will showcase the results and 4. contains the discussion and conclusions of the study.

At the time of writing this paper, the topic of applying true random noise to neural networks is relatively unexplored. In 2018, Fan et al. proposed a neural network model to accurately produce pseudorandom numbers while investigating differences in pseudorandom and true quantum random numbers (Fan & Wang, 2018). Next year, Alcin et al. proposed a neural network model that could statistically produce true random numbers (Alcin, Koyuncu, Tuna, Varan, & Pehlivan, 2019). In 2020, Bird et al. investigated pseudorandom and true quantum random number generators or QRNG for the initialization of neural networks, and proposed a Quantum Random Tree classifier (Bird et al., 2020). In the paper, the authors report that the QRNG initialization produced mixed results over pseudorandom, as some differences were unremarkable and some were significant (Bird et al., 2020). In 2021, Heese et al. continued the investigation of initializing neural networks with PRNG and QRNG numbers, and they did not find reproducible statistical difference between the two (Heese et al., 2021). As far as the authors know, this study is the first to address true random numbers to be applied to the node dropout regularization process.

2. Materials and methods

In this section, materials and methods used in the study are listed. In order to assess the quality of randomness of a number sequence, the statistical tests used in this study are listed. Also, the random number generators used for calculating those sequences are described. Datasets for neural network modelling experiments are mentioned, the neural network models used, and lastly the utilized software and hardware.

2.1. Quality of randomness testing

Various statistical tests have been proposed in the past that aim to qualify and quantitate randomness in a set of random bits (Marsaglia et al., 2002; Ritter, 1996). As most of the tests evaluate randomness from different aspects, they are usually done in unison as a battery of tests. Some focus more on sub-sequences within the whole set of random bits, while others calculate more global statistics of the whole set. Notable implementations of these tests include Diehard (Marsaglia, 2008), Dieharder (Brown, Eddelbuettel, & Bauer, 2018), TestU01 (L'ecuyer & Simard, 2007) and the NIST Statistical Test Suite (National Institute of Standards and Technology, 2010). For our experimentation, NIST Suite was selected due to the fact that it can operate on bit streams and therefore 64-bit integers, while TestU01 and Dieharder operate strictly on 32-bit sequences (Vigna, 2016).

NIST Suite contains 15 different statistical tests for quality of randomness, they are listed in Table 1. Frequency test investigated the proportion of zeros and ones for the entire sequence, while block frequency test checks proportions within M-bit blocks. Run test calculates the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits, while run test (longest run of ones) does this for the longest run of ones within M-bit blocks. Binary matrix rank test ranks the disjoint sub-matrices of the entire sequence, and checks for their linear dependence to the original sequence. Discrete Fourier transform test calculates the peak heights in Fourier space and attempts to detect periodic patterns. Overlapping and the non-overlapping template matching tests look for the occurrences of pre-specified target strings. Maurer's universal statistical test is used to test the number of bits between matching patterns. The authors of NIST Suite suggested certain sample sizes to be used with certain sequence block sizes (National Institute of Standards and Technology, 2010), which we also used in our experimentation. Linear complexity test investigated the length of a linear feedback shift register or LFSR (Canteaut, 2005). Serial test calculates the frequency of all possible overlapping m-bit patterns across the entire sequence. Approximate entropy test on the other hand tests for the frequency of all possible overlapping M-bit patterns across the entire sequence. Cumulative Sums test can be executed forwards and backwards, and it checks for the maximal excursion of a random walk defined by the cumulative sum of adjusted (−1, +1) digits in the sequence. Random excursions test calculates the number of cycles having exactly K visits in a cumulative sum random walk, and lastly random excursion variant test calculates the number of times that a particular state is visited in a cumulative sum random walk.

As the authors noted in the companion publication that no set of tests can give an absolute answer of quality of randomness (National Institute of Standards and Technology, 2010). Also, using NIST only for testing randomness has been criticized in the past (Hurley-Smith

Table 1
Statistical tests in the NIST Statistical Test Suite (National Institute of Standards and Technology, 2010).

Test
1. Frequency Test
2. Block Frequency Test
3. Run Test
4. Run Test (Longest Run of Ones)
5. Binary Matrix Rank Test
6. Discrete Fourier Transform (Spectral) Test
7. Non-overlapping Template Matching Test
8. Overlapping Template Matching Test
9. Universal Statistical Test
10. Linear Complexity Test
11. Serial Test
12. Approximate Entropy Test
13. Cumulative Sums (Forward & Backward)
14. Random Excursion Test
15. Random Excursion Variant Test

& Hernandez-Castro, 2020). For our experimentation however they are sufficient, as they give a heuristic approximation for ranking the different random number generators, so that the investigation can be conducted.

2.2. Random number generators

Pseudorandom number generators were selected based on their popularity in machine learning-related software libraries. These included Mersenne Twister (Matsumoto & Nishimura, 1998) or MT and Philox (Salmon, Moraes, Dror, & Shaw, 2011). MT uses matrix linear recurrence over a finite binary field in order to generate random variables. The popular implementation of MT, MT19937, has a word length of 32-bits, which is derived from the Mersenne prime of $2^{19937}-1$. While MT19937 can pass the Diehard tests and most TestU01 tests given enough derived samples (L'ecuyer & Simard, 2007), its state vector is 624 long, meaning that if the generator's output is observed as a sequence of 624 integer values, one can duplicate the generator and successfully predict the next random values the generator produces.

The second pseudorandom number generator Philox is counter-based, meaning that it only uses an integer counter as its internal state. Its state vector consists of a 256-bit value which is encoded as a 4-element unsigned integer array and a 128-bit value encoded as a 2-element unsigned integer array. The 64-bit implementation uses wide multiples of two 32-bit numbers in order to produce 64-bit numbers. Since the schema of Philox involves the creation of multiple independent streams, it is commonly used with hardware such as GPUs that can parallelize calculation. Philox has been tested to perform well with the TestU01 suite (Salmon et al., 2011).

True random number generators used in the study were the Quantis QRNG USB-device by ID Quantique (Quantique, 2021), ANU QRNG, a website hosted by The Australian National University (Symul, Assad, & Lam, 2011) and HotBits, a website hosted by Fourmilab (Fourmilab, 1996). All of these random number sources leverage natural phenomena in generating values. Quantis exploits a quantum optics process where photons are sent one by one onto a semi-transparent surface, and they have a naturally occurring 50/50 chance of either reflect off or pass through. Within the device, this is implemented by one light source, a 45-degree mirror and two light collection components at each end of possible photon traversal. The detected photons are then encoded as 0 and 1 bits accordingly. The schema of this is showcased in Fig. 1. These types of quantum device applications are known to be affected by thermal noise (Saulson, 1990), where the calculation outcome is highly affected by the environmental temperature. However, the effect of this should be lesser on light-based calculation (Arrazola et al., 2021). The manufacturer states that the contribution of the thermal noise in deriving random bits is less than 1%, and the device

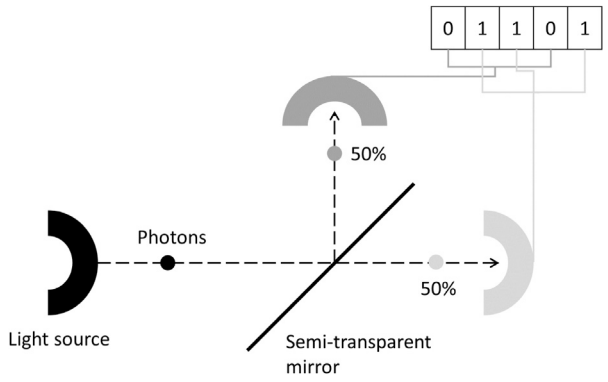

Fig. 1. Quantum optic random number generation of Quantis (Quantique, 2021). When photons make contact with the semi-transparent mirror, there is roughly 50% probability that it will pass the mirror, and 50% probability that it bounces of the mirror surface. Light collectors at each end of the trajectories measure the photons, and encode them as 0 and 1 bits accordingly.

has acquired various certifications related to producing statistically true random numbers (Quantique, 2021). For our testing, the ambient temperature was 22.6 °C, which is within the storage temperature range of the general specifications for the device.

ANU QRNG website provides open data quantum random numbers that can be accessed. These numbers are either generated beforehand or live within a lab in The Australian National University, by using lasers to measure vacuum fluctuations that are proposed to be truly random (Symul et al., 2011). In such measurement, broadband measurements of the vacuum field are made, which can be detected in the radio-frequency sidebands of a single-mode laser (Symul et al., 2011). This digitized photocurrent is then processed with appropriate algorithms in order to reduce noise coming from the environment and measuring setup (Haw et al., 2015). The authors claim that this random number generation scheme has high environmental immunity, which is a similar claim as what has been stated about Quantis. However, it should be noted that the quality of both Quantis RNG and ANU QRNG have been criticized in the past (Hurley-Smith & Hernandez-Castro, 2020), where the authors stated that Quantis RNG did not pass various tests without off-device post-processing of random numbers, while ANU QRNG did not pass three tests in TestU01.

Lastly, HotBits website also provides true random numbers by measuring the interval of two Cæsium-137 radiation decay events from a radiation source (National Institute of Standards and Technology, 2009). The natural uncertainty of a Cæsium-137 nucleus decay event enables this process, and they are detected by a Geiger counter. By calculating the time interval between two individual, consecutive pairs of decay events, bit encoding can be calculated if the first or the second interval is longer. Even comparisons are discarded. Also, in order to reduce any systematic errors stemming from the radiation source or the measuring process, the encoding is reversed for consecutive comparisons. The schema of this measurement logic is showcased in Fig. 2. TRNGs based on radioactive decay have been criticized in the past for their low bit rate and the need for post-processing (Herrero-Collantes & Garcia-Escartin, 2017).

2.3. Study data

Four datasets commonly used for machine learning research were selected for the study; Iris (Anderson, 1936), Wine (Aeberhard, Coomans, & de Vel, 1992) Dry Bean (Koklu & Ozkan, 2020), and Adult (Kohavi et al., 1996). The selection was made to confirm that given enough epochs, overfitting during training would occur. Iris represents the simplest dataset in terms of feature dimensionality and number of observations, while Adult is the most complex of the four. All of the datasets are similar in terms of the prediction task, multivariate classification. Data description is listed in Table 2.

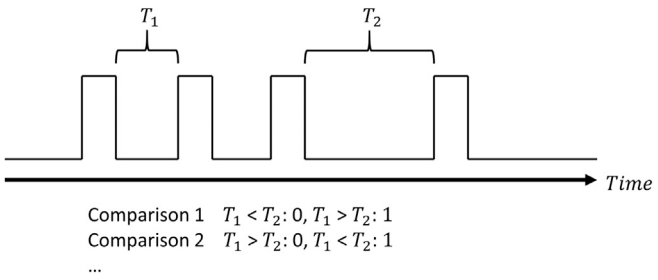

Fig. 2. The process behind HotBits. The time intervals of two consecutive radiation decay event pairs T_1 and T_2 are calculated, and then they are compared. If T_1 is shorter, this is encoded as 0, and as 1 of T_1 is longer. This logic is reversed at every consecutive comparison.

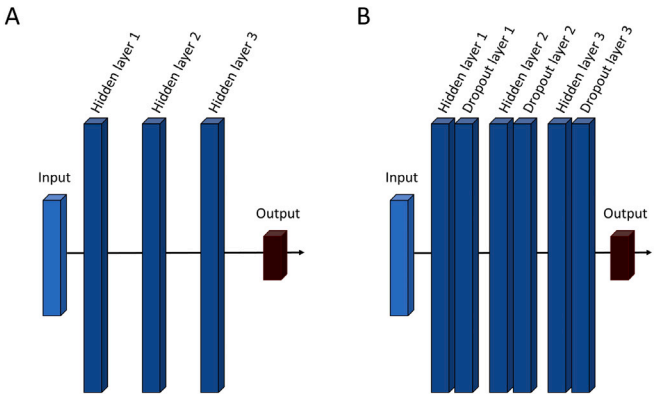

Fig. 3. 3A depicts the simplistic network architecture of the baseline model without dropout, while 3B depicts the model used to investigate node dropout's association to randomness. Input and output layers are adjusted to fit each dataset and their learning task, while the design of hidden layers stay the same.

Table 2
Description of the data used in the study.

Attribute	Iris	Wine	Dry bean	Adult
Observations	150	178	13611	48842
Features	4	13	17	14
Feature types	Real	Integer, Real	Integer, Real	Categorical, Integer
Target classes	3	3	7	2
Missing values	No	No	No	Yes

2.4. Neural networks & node dropout

Neural network models were chosen to be as simplistic as possible in order to mitigate the effect that choosing a specific architecture would affect the modelling results. For all datasets, a deep feed-forward neural network containing 3 hidden layers were used, all containing 100 nodes using the ReLU activation (Agarap, 2018). As all of the modelling tasks included in this investigation were classification, softmax activation (Goodfellow, Bengio, & Courville, 2018) was used in the output layer. The number of output nodes would be adjusted to fit the data in question, this adjustment was also done for the input layer. Categorical cross-entropy (Murphy, 2012) was used as the loss metric, and Adam (Kingma & Ba, 2014) was used as the optimizer. Parameters of the optimizer were set to the default values of TensorFlow 2 (Abadi et al., 2015). Random seeds related to the model were fixed.

For utilizing node dropout, node dropout layers after each hidden layer were added to the network architecture. The schematic view of both architectures is presented in Fig. 3. Instead of using pseudorandom number generators of the library, the default implementation of TensorFlow 2 dropout layer was extended in a way that a set of random numbers could be given as input, and the random decisions made by the layer during training would be anchored to those values. This enabled us to conduct repeatable experiments with various sets of random values. The amount of hidden nodes to be dropped was chosen to be one of the hyperparameters that would be iterated during the study, in order to assess its affect on the results. The second hyperparameter was the number of epochs, which dictates how many node dropout-related random decision are needed during training.

2.5. Software and hardware

For testing random number sequences, NIST statistical test suite (National Institute of Standards and Technology, 2010) and Dieharder (Brown et al., 2018) were used. For NIST, the python implementation by Steven Kho Ang, version 1.3 was used (Ang, 2021). Modelling experiments were conducted using Python 3.6.7 (Van Rossum & Drake, 2009), Tensorflow 2.6.0 (Abadi et al., 2015), Numpy 1.19.5 (Harris et al., 2020) and Pandas 0.23.4 (McKinney et al., 2010). For data processing of bit data, bitstring 3.1.9 package was used (Griffiths, 2021). For producing Fig. 4, the R package ggplot2 was used (Wickham, 2016).

The modelling experiments were calculated using either the NVIDIA® RTX™ 2080 Ti GPU, or the Intel® Core™ i9-9940X CPU. For Quantis random bit data, Quantis USB device (Quantique, 2021) was used.

2.6. Experiment description

In order to investigate the relationship between quality of randomness of random numbers used in the internal logic of node dropout and a neural network's ability to overfit, the study was designed to include multiple datasets and multiple random data generators. First, all of the generators were used to calculate a random set of 64-bit integers, where the quantity corresponds to the maximum number of random number generation events which are needed for fitting a neural network. The maximum number of epochs included in our investigation determines the amount of RNG events. After this, the sets were tested with the NIST Suite by converting and combining them to bit sequences, after which they were ranked based on the number of tests they pass. This gives us a heuristic, qualitative result of the quality of randomness for each set. This step was followed by the neural network fitting experiments.

One unit in our model fitting experimentation was a 10-fold cross validation process, which was iterated in terms of used datasets, random number sets and neural network parameters. In cross validation, the data is first partitioned into 10 folds. Then, these folds are iterated over in a way that one fold is the test set for the duration of the iteration, and the rest are combined as training data. In the next iteration, the next subsequent fold is deemed as the test data, and the rest as training data. This way, 10 model fits are done during the experiment, and 10 test set accuracies are calculated, which gives an accurate representation of model performance. Four common machine learning open datasets Wine, Iris, Dry Bean and Adult were used to fit a neural network with the same architecture, excluding the input and output layers, which were dataset-specific. Initializations of those models were also the same. The five random number sets used to anchor randomness during each epoch for the fixed dropout layers were MT, Philox, Quantis QRNG, ANU QRNG and HotBits. Model parameters node dropout amount (10% to 50% by 10% increments) and number of epochs (10,100,1000 and 10 000) were iterated to see their effect. Also, for assessing the usefulness of node dropout in general, a baseline model without dropout layers was calculated. This amounted in a total of 16 cross-validation experiments for the baseline neural network model, and 400 experiments for node dropout model. Fig. 4 contains the flowchart for the experiments. These results are inspected in terms of training data size, number of epochs and amount of node dropout. Finally, for each random number generator the average amount of model overfitting is investigated. This is done by calculating the amount of overfitting acc^{dif}, where accuracy produced by the test data is subtracted from the training data accuracy result. These results are compared against how many randomness tests the generator in question passed.

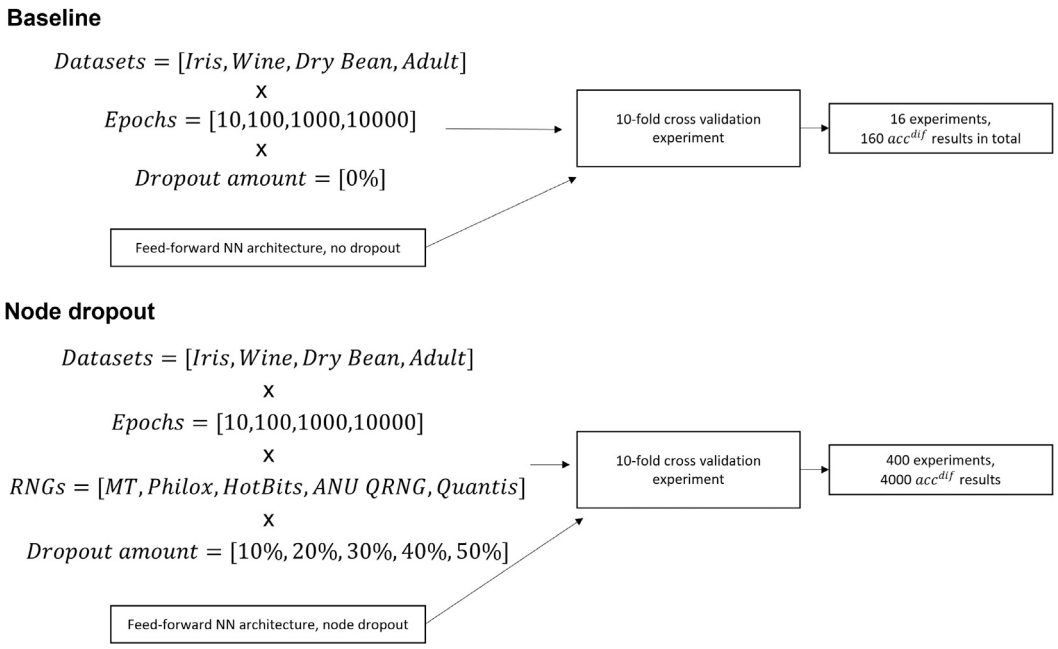

Fig. 4. Flowchart for the baseline and node dropout experiments.

3. Results

In this section, the study results are showcased. Firstly, the randomness results of each random number generator are described. Then, the neural network modelling experiment results are listed.

3.1. Randomness testing

All of the five random number generators were used to produce the 300 000 random integers needed for fitting a neural network. This means that when the integers are converted to bits, the number of bits can vary. Also, while the QRNG and Philox generators operate in 64-bit, MT operates with 32-bit precision which actively halves the number of bits. These sets are partitioned to 300, 3000, 30 000 and 300 000 sets, which in turn represent the maximum amount of needed random seeds with 10, 100, 1000 and 10 00 epochs when 3 hidden layers are utilized in the neural networks. This way, when investigating a certain model training experiment, we can also assess the number of statistical tests the used random number set would have passed. Table 3 presents the numbers of total bits contained in each random number set of each generator.

Table 4 contain the results of the statistical testing done to each set of random bits, derived from the random integers generated by a random number generator. The results show that the number of passed tests depends on the sample size of the tested values, for example tests 2,7 and 9 fail mostly with 30 000 and 300 000 numbers. Comparing pseudo-random generators with true random ones, MT produces the worst overall result of 24 tests passed, while Philox passes a similar amount of tests when compared to true random number generators. However, due to the 32-bit word length of MT, the comparison is not favourable towards it, as the random numbers do not generate enough bits for the same amount of tests to be conducted. Tests 10, 11 and 12 with 30 000 numbers are not calculated due to insufficient bits. MT is however a highly utilized random number generator in ML implementations, so its inclusion is justified in this comparison, as most practitioners utilize it in ML experiments, knowingly or unknowingly.

Out of the three true random number generators, Quantis performed the best with 29 tests passed. However, it failed test 15 with 3 different data sizes, when no other generator did. Surprisingly, the pseudorandom generator Philox performed better with tests 1, 5, 10, 11 and 13 when compared to ANU QRNG and HotBits, implicating that measuring a natural phenomena does not always produce more favourable randomness results. This can be caused by the bias originating from the measurement process. Out of the 15 statistical tests, frequency test or test 1 and cumulative sums or test 13 were failed the most by generators in general. Failure of test 1 also meant that test 3 was not calculated for those cases, making it the test with the most missing results.

Table 3
Description of the random value sets created by different generators. As the values are sampled as integers and then converted to bits, the number of total bits in a set can differ.

Data generator	Size	Bits
MT (32-bit)	300	9344
	3000	92994
	30 000	930092
	300 000	9299174
Philox (64-bit)	300	18883
	3000	188997
	30 000	1889984
	300 000	18898134
ANU QRNG (64-bit)	300	18917
	3000	188939
	30 000	1889924
	300 000	18900233
Quantis (64-bit)	300	18884
	3000	188955
	30 000	1890702
	300 000	18900083
HotBits (64-bit)	300	18941
	3000	189085
	30 000	1889660
	300 000	18900116

3.2. Model fitting

Fig. 5 demonstrates the mean(acc^{dif}) results over different epochs and amount of node dropout used. Results under 0 imply that some overfitting has occurred, while results over 0 mean that the fitted model

Table 4
P-value results of the statistical tests 1 to 15 per random number generator, using 300, 3000, 30 000 & 300 000 numbers. Significance level <0.01 for not truly random sequence, these results are in bold. Number of passed tests are listed with the generator names. Random excursion tests 14 and 15 are either passed if all of the states are truly random with <0.01 significance, and failed otherwise. Tests 11 and 13 are considered passed only if both of the associated p-values are not significant.

Test name	MT (24)	Philox (29)	ANUQ (27)	Quantis (29)	HotBits (26)
1. Frequency Test	<0.01	0.032	0.007	0.126	<0.01
	<0.01	<0.01	<0.01	<0.01	<0.01
	<0.01	<0.01	<0.01	<0.01	<0.01
	<0.01	<0.01	<0.01	<0.01	<0.01
2. Block Frequency Test	0.18	0.248	0.428	0.589	0.103
	0.025	0.447	0.064	0.674	0.683
	<0.01	0.055	0.021	0.025	0.035
	<0.01	<0.01	<0.01	<0.01	<0.01
3. Run Test	NA[a]	0.451	0.343	0.562	0.440
	NA[a]	NA[a]	NA[a]	NA[a]	NA[a]
	NA[a]	NA[a]	NA[a]	NA[a]	NA[a]
	NA[a]	NA[a]	NA[a]	NA[a]	NA[a]
4. Run Test (Longest Run of Ones)	0.079	0.226	0.923	0.129	0.338
	<0.01	<0.01	<0.01	<0.01	<0.01
	0.485	0.107	0.428	0.021	0.125
	<0.01	<0.01	<0.01	0.013	<0.01
5. Binary Matrix Rank Test	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	0.702	0.587	0.607	0.289	0.753
	0.227	0.625	0.706	0.617	0.857
	0.221	0.657	<0.01	0.755	0.214
6. Discrete Fourier Transform (Spectral) Test	0.279	0.486	0.574	0.635	0.948
	0.837	0.575	0.583	0.592	0.240
	0.073	0.901	0.968	0.384	0.075
	0.419	0.749	0.185	0.429	0.524
7. Non-overlapping Template Matching Test	0.527	0.546	0.307	0.320	0.949
	0.059	0.208	0.121	0.024	0.248
	<0.01	<0.01	<0.01	<0.01	<0.01
	<0.01	<0.01	<0.01	<0.01	<0.01
8. Overlapping Template Matching Test	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	<0.01	<0.01	<0.01	<0.01	<0.01
9. Universal Statistical Test	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	0.854	0.502	0.935	0.125	0.508
	<0.01	<0.01	<0.01	0.755	0.236
10. Linear Complexity Test	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	0.36	0.285	0.762	0.394
	0.609	0.326	0.31	0.537	<0.01
11. Serial Test	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	0.081, 0.495	<0.01, 0.018	0.031, 0.532	<0.01, 0.031
	<0.01, 0.382	<0.01, 0.30	<0.01, 0.509	0.015, 0.291	<0.01, 0.074
12. Approximate Entropy Test	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	NA[b]	NA[b]	NA[b]	NA[b]
	NA[b]	<0.01	<0.01	<0.01	<0.01
	<0.01	<0.01	<0.01	<0.01	<0.01
13. Cumulative Sums (Forward & Backward)	0.008, 0.014	0.057, 0.050	0.003, 0.013	0.027, 0.232	<0.01, <0.01
	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01
	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01
	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01	<0.01, <0.01
14. Random Excursion Test	Passed	Passed	Passed	Passed	Passed
	Passed	Passed	Passed	Passed	Passed
	Passed	Passed	Passed	Passed	Passed
	Passed	Passed	Passed	Passed	Passed
15. Random Excursion Variant Test	Passed	Passed	Passed	Failed	Passed
	Passed	Passed	Passed	Failed	Passed
	Passed	Passed	Passed	Failed	Passed
	Passed	Passed	Passed	Passed	Passed

NA[a] Test fails due to failure of the Frequency test, NA[b] Number of bits is below the suggested minimum (National Institute of Standards and Technology, 2010).

performed better with test set data, when compared to the accuracy achieved with training data. The no dropout baseline method results showcase how without node dropout, the calculated mean(acc^{dif}) in most cases decreases as the number of epochs is increased. Wine and Adult datasets are good examples of this. In the 1000 epochs plot, the baseline result for Iris remains near zero, while all node dropout methods seem to produce overfitting. It is probable that the small number of observations in the Iris dataset introduces some added variability to the results, as results for the two bigger datasets are more consistent. For Dry Bean, the baseline results over all epochs remain near zero. This result speaks of the dataset's quality and consistency, as the 10 different iterations of the cross validation experiment produce roughly the same performance of the training and test sets.

The two small datasets Iris and Wine produced biggest variation in results, this is to be expected as the number of observations during

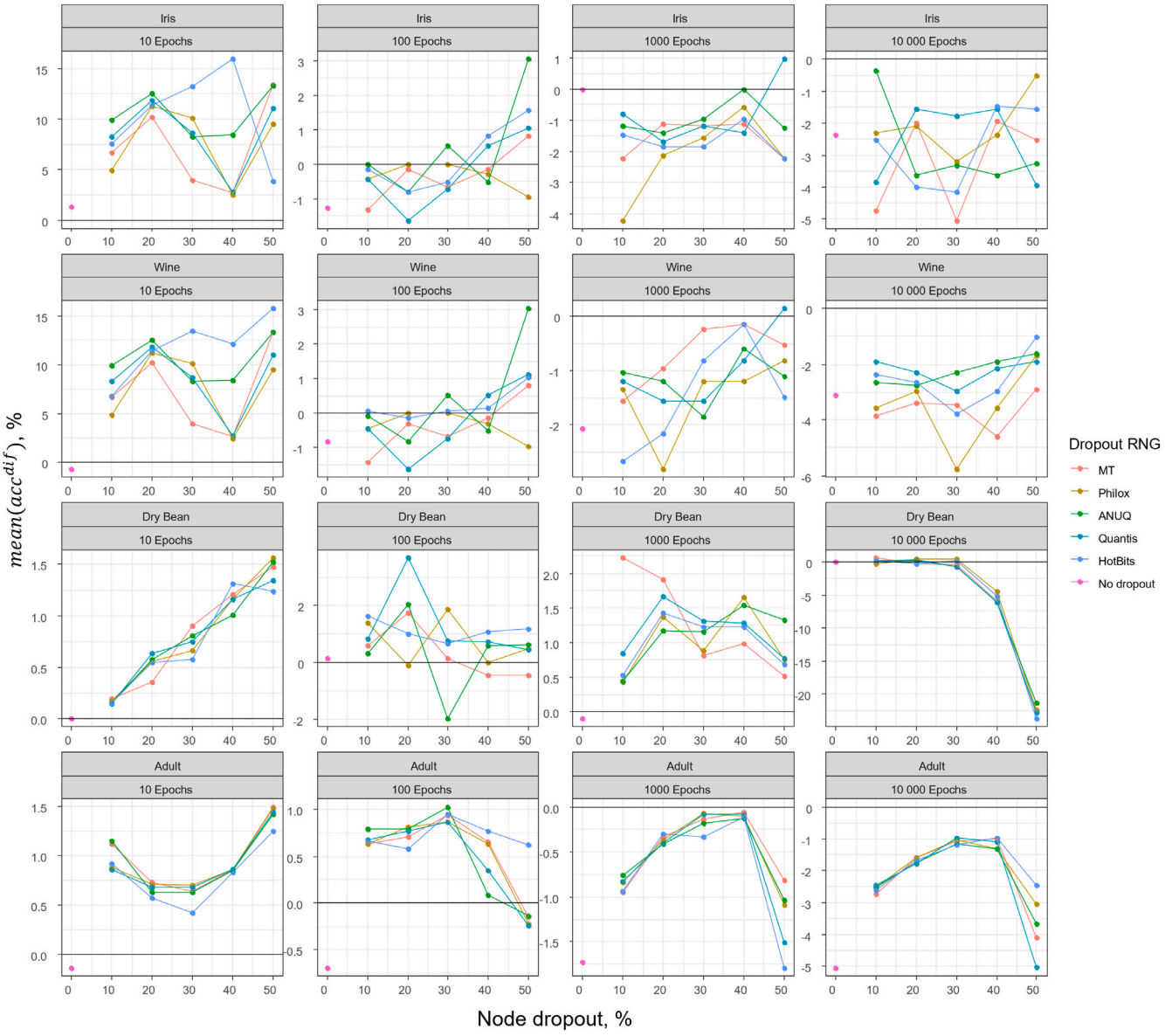

Fig. 5. Mean accuracy difference results, plotted by the amount of node dropout and epochs used over all four datasets. Different random number generators used with node dropout are highlighted in different colours, and also the benchmark result of no node dropout utilized. The zero line indicates overfitting results on the negative side, and increased test data accuracy over training data accuracy on the positive side.

model fitting is at it's lowest. Every node dropout experiment performed better than the baseline at 10 epochs for both datasets. At 100 epochs, only node dropout of 0.3 and 0.4 were able to achieve the same performance, as 0.1, 0.2 and 0.5 for some random number generators performed worse. For this many epochs and this level of complexity of the learning task, 0.1 and 0.2 were not enough to produce consistent improvements. Aside from Philox, node dropout of 0.5 over-performed the baseline.

For epochs of 1000 and 10 000, node dropout methods performed either on par or worse when compared to the baseline. This showcases the fact that the optimal number of epochs per dataset and learning task differs. If the amount of epochs is set unfeasibly large, the neural network model will overfit to the training data, regardless of how much node dropout is used, or what random number generators were used to dictate it's functionality.

The results calculated for the other two datasets, Adult and Dry Bean, differ from the two smaller datasets in terms of epochs more than 10, most probably due to their significantly different size. For 10 epochs and both datasets, every node dropout method performs better when compared to the baseline. Dry bean seems to clearly benefit from node dropout at 1000 epochs, and at 100 epochs for most experiments as well. At 10 000 epochs, generally minor improvements in performance can be seen with dropout amount from 0.1 to 0.3, however with 0.4 and 0.5 amounts the experiment performances get significantly worse when compared to the baseline. This dictates a saturation point, where too much dropout with too many epochs results in a situation where the model to struggles to learn the underlying signal, as too much variance is introduced by the node dropout method.

For the Adult dataset, at 100 epochs the node dropout experiments generated significantly better models when compared to the baseline, which seems to overfit more as more epochs are calculated. This is also true for experiments calculated with 1000 and 10 000 epochs, excluding dropout amount of 0.5 where performance seems to deteriorate again. This similar saturation point can be seen with Dry Bean.

When different generators over different epochs and dataset are compared, there does not seem to be a clear general pattern indicating

Table 5
The comparison of model fitting results from the 10-fold cross validation experiments per dataset and generator over different epochs used. For the no node dropout baseline results, the unit is mean(acc^{dif}), while for the node dropout results, the unit is the difference from the baseline result. Different random number generators used for node dropout are listed, along with their number of passed NIST tests.

Iris							Wine					
		Node Dropout, difference from baseline						Node Dropout, difference from baseline				
Epochs	Baseline	MT	HotBits	ANU QRNG	Philox	Quantis	Baseline	MT	HotBits	ANU QRNG	Philox	Quantis
10	1.33	−6.04	−9.05	−9.17	−6.31	−7.19	−0.74	−8.12	−12.67	−11.24	−8.39	−9.26
100	−1.26	−0.96	−1.44	−1.70	−0.92	−1.01	−0.81	−0.47	−1.05	−1.24	−0.47	−0.58
1000	<0.01	1.57	1.67	0.96	2.15	0.83	−2.07	−1.39	−0.62	−0.92	−0.61	−1.08
10000	−2.37	0.87	0.37	0.47	−0.28	0.16	−3.11	0.53	−0.55	−0.86	0.40	−0.86
	Mean	−1.14	−2.11	−2.36	−1.34	−1.80	Mean	−2.36	−3.72	−3.57	−2.27	−2.94
Dry bean							Adult					
		Node Dropout, difference from baseline						Node Dropout, difference from baseline				
Epochs	Baseline	MT	HotBits	ANU QRNG	Philox	Quantis	Baseline	MT	HotBits	ANU QRNG	Philox	Quantis
10	<0.01	−0.83	−0.77	−0.81	−0.82	−0.80	−0.14	−1.10	−0.93	−1.07	−1.06	−1.04
100	0.13	−0.19	−0.99	−0.19	−0.60	−1.17	−0.69	−1.25	−1.41	−1.21	−1.24	−1.18
1000	−0.10	−1.39	−1.12	−1.23	−1.12	−1.27	−1.73	−1.27	−1.04	−1.23	−1.24	−1.15
10000	−0.05	5.70	5.73	5.51	5.18	5.78	−5.05	−2.96	−3.27	−2.97	−3.15	−2.79
	Mean	0.82	0.71	0.82	0.66	0.63	Mean	−1.65	−1.66	−1.62	−1.67	−1.54

that using one over the other would result in improved generalizability. However, with optimal learning parameters in terms of dataset and learning task, the quality of randomness seems to affect added generalizability imposed by utilizing node dropout. Table 5 showcases the mean(acc^{dif}) results of each baseline model, and how much each random number generator-based node dropout model differs from these results. This way, the bigger the difference is on the negative side, that much better generalizability the results provide. For Iris and Wine, the results in Fig. 5 indicate that node dropout was beneficial to use, when the number of epochs was either 10 or 100. For those experiments, ANU QRNG provided the best improvement over baseline, excluding Wine at 10 epochs, where HotBits had the best improvement of −12.67 over the baseline. MT on the other hand produced the worst improvement during these experiments, tied with Philox at 100 epochs for Wine.

For Dry Bean, the results in Fig. 5 showcased that at 10 000 epochs there is a clear saturation point in performance, where node dropout was not beneficial. At 10 epochs, MT produced the best result of −0.83. At 100 epochs, Quantis reached the best improvement of −1.17. Lastly, at 1000 epochs, MT had the best improvement of −1.39. For Adult, the benefits of node dropout can be clearly seen with epochs 10 and 100. MT had the best result of −1.10 at 10 epochs, while HotBits had the best result of −1.41 at 100 epochs.

In Table 6, the mean results from Table 5 and the number of passed NIST tests of a random number generator were tested for correlation. For Wine and Adult datasets, the Pearson correlation was 0.07 and 0.40 respectively. This indicates that on average, using a random number generator with improved randomness to dictate the node dropout process reduced generalizability for these datasets. For the Iris and Dry Bean datasets, the correlation result was −0.17 and −0.80 respectively. This would suggest that on average, using more random numbers to dictate the node dropout process had the effect of improved generalizability, especially with the Dry Bean dataset.

Table 6
Correlation results from the 10-fold cross validation experiments per dataset, over all generators and datasets. Overfitting statistic of baseline difference of mean(acc^{dif}) proposed in Table 5 are compared against the number of passed NIST tests with Pearson correlation.

RNG	Iris	Wine	Dry bean	Adult	Passed NIST tests
MT	−1.14	−2.36	0.82	−1.65	24
HotBits	−2.11	−3.72	0.71	−1.66	26
ANU QRNG	−2.36	−3.57	0.82	−1.62	27
Philox	−1.34	−2.27	0.66	−1.67	29
Quantis	−1.80	−2.94	0.63	−1.54	29
Pearson correlation	−0.17	0.07	−0.80	0.40	

4. Discussion and conclusions

The determination of true randomness can be undecipherable at times for multiple reasons. The statistical tests conducted can be disadvantageous to certain pseudonumber generators due to the fact that their core functionality is drawn from the same theory as the test. This amounts to the fact that the significance of a test result with different generators is not uniform. Tests can also assume a fixed sequence size as input, so generators that produce differing sizes need additional processing to be compatible, and in worst cases this processing can deteriorate the quality of randomness of that sequence. This is why randomness tests are usually described as being not theoretically complete, but useful in practise (Brown et al., 2018).

Dieharder was used to test all of the full random number sets in this study, MT did not pass 2 tests, while Philox did not pass 5 tests, ANU QRNG did not pass 3 tests, Quantis did not pass 4 tests and HotBits did not pass 5 tests. The authors suspect that the poor Dieharder performances of Philox, ANU QRNG, Quantis and HotBits were due to the fact that Dieharder operates on 32-bit sequences (Marsaglia, 2008), while the binary sequences were constructed from 64-bit integers. In practise this means that the 64-bit sequences are partitioned into two 32-bit sequences, which introduces non-uniformity to the overall bits processed by the tests.

Our experimentation showcases that the selection of data, epochs and the amount of node dropout affect the amount of overfitting demonstrated by the model. In small to modest datasets, iterating epochs by magnitudes had a significant impact on overfitting. At 1000 and 10 000 epochs, using node dropout did not have a beneficial effect, demonstrating the fact that the number of epochs should firstly be fine-tuned for the specific data in question. For larger datasets, this effect was not as dramatic, however a clear saturation point with high number of epochs and amount of dropout used was seen.

Increased quality of randomness was strongly correlated with a neural network model's ability to generalize within the Dry Bean dataset. There was also a mild association within the Iris dataset. At the same time, it had a moderate correlation within the Adult dataset to decrease generalization, and very mild association with Wine. Therefore, our study results support the notion of Bird et al. (2020) and Heese et al. (2021), that there would not be a reproducible general association, but that the effect is highly data-dependent. Differences between random number generators were found, and in most optimal experiments, more random generators would provide over 4% better accuracy results with the used test sets.

The significance of this finding is that potentially any neural network model that utilizes node dropout during training could be improved generalization-wise. By iterating multiple different random

number generators during training experiment, one could select the most optimal one, instead of strictly using just one for all experiments. As training neural networks is commonly a highly iterative process, this additional step could fit into the existing modelling workflows. Acquiring random number sequences from a source could be done in advance, and read from a database during training. Same subsequences could also be reused, giving the modelling step reproducibility, even with true random numbers.

In addition to node dropout, future work could also include an investigation of quality of randomness to stochastic depth method, which is a layer-level dropout method initially proposed for residual networks (Huang, Sun, Liu, Sedra, & Weinberger, 2016). Random selection of layers to drop is a similar selection problem when compared to hidden node dropout selection in node dropout.

The clear limitations of the study are the amount of datasets used, future work would include experimenting with a wider range of datasets, with various classification and regression tasks. This would improve the generalizability of our findings. Also, more complex data such as image and video should be investigated. At the time of writing this paper, this was not feasible with the hardware we had available.

This study provides the initial investigation to how quality of randomness affects the regularization impact of node dropout in neural network model fitting. Increasing statistical quality of randomness of decisions made by node dropout layers can have beneficial effect with datasets, when other parameters regarding training and node dropout have been fine-tuned.

CRediT authorship contribution statement

Aki Koivu: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Visualization. **Joona-Pekko Kakko:** Methodology, Investigation, Writing – review & editing. **Santeri Mäntyniemi:** Software, Data curation, Writing – review & editing. **Mikko Sairanen:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/.

Aeberhard, S., Coomans, D., & de Vel, O. (1992). *The classification performance of RDA*: *Tech. rep.*, (92-02), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.

Alcin, M., Koyuncu, I., Tuna, M., Varan, M., & Pehlivan, I. (2019). A novel high speed artificial neural network–based chaotic true random number generator on field programmable gate array. *International Journal of Circuit Theory and Applications*, *47*(3), 365–378.

Anderson, E. (1936). The species problem in iris. *Annals of the Missouri Botanical Garden*, *23*(3), 457–509.

Ang, S. K. (2021). NIST randomness testsuit. GitHub Repository, GitHub, https://github.com/stevenang/randomness_testsuite.

Arrazola, J., Bergholm, V., Brádler, K., Bromley, T., Collins, M., Dhand, I., et al. (2021). Quantum circuits with many photons on a programmable nanophotonic chip. *Nature*, *591*(7848), 54–60.

Bird, J. J., Ekárt, A., & Faria, D. R. (2020). On the effects of pseudorandom and quantum-random number generators in soft computing. *Soft Computing*, *24*(12), 9243–9256.

Brown, R. G., Eddelbuettel, D., & Bauer, D. (2018). *Dieharder* (pp. 305–27708). Duke University Physics Department Durham, NC.

Canteaut, A. (2005). Linear feedback shift register. In *Encyclopedia of cryptography and security* (pp. 355–358). Boston, MA: Springer US.

Demirhan, H., & Bitirim, N. (2016). Statistical testing of cryptographic randomness. *İstatistikçiler Dergisi: İstatistik Ve Aktüerya*, *9*(1), 1–11.

Fan, F., & Wang, G. (2018). Learning from pseudo-randomness with an artificial neural network–does god play pseudo-dice? *IEEE Access*, *6*, 22987–22992.

Floyd, R. W. (1967). Nondeterministic algorithms. *Journal of the ACM*, *14*(4), 636–644.

Fourmilab (1996). HotBits. https://www.fourmilab.ch/hotbits/.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* (p. 301). MIT Press.

Goodfellow, I., Bengio, Y., & Courville, A. (2018). Softmax units for multinoulli output distributions. In *Deep Learning*. MIT Press.

Griffiths, S. (2021). Bitstring. https://bitstring.readthedocs.io/.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Haw, J.-Y., Assad, S., Lance, A., Ng, N., Sharma, V., Lam, P. K., et al. (2015). Maximization of extractable randomness in a quantum random-number generator. *Physical Review A*, *3*(5), Article 054004.

Heese, R., Wolter, M., Mücke, S., Franken, L., & Piatkowski, N. (2021). On the effects of biased quantum random numbers on the initialization of artificial neural networks. arXiv preprint arXiv:2108.13329.

Herrero-Collantes, M., & Garcia-Escartin, J. C. (2017). Quantum random number generators. *Reviews of Modern Physics*, *89*(1), Article 015004.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558.

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *European conference on computer vision* (pp. 646–661). Springer.

Hurley-Smith, D., & Hernandez-Castro, J. (2020). Quantum leap and crash: Searching and finding bias in quantum random number generators. *ACM Transactions on Privacy and Security*, [ISSN: 2471-2566] *23*(3).

Jennewein, T., Achleitner, U., Weihs, G., Weinfurter, H., & Zeilinger, A. (2000). A fast and compact quantum random number generator. *Review of Scientific Instruments*, *71*(4), 1675–1680.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kohavi, R., et al. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.. In *Kdd, 96* (pp. 202–207).

Koklu, M., & Ozkan, I. A. (2020). Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, *174*, Article 105507.

Lampert, B., Wahby, R. S., Leonard, S., & Levis, P. (2016). Robust, low-cost, auditable random number generation for embedded system security. In *Proceedings of the 14th ACM conference on embedded network sensor systems CD-ROM* (pp. 16–27).

L'ecuyer, P., & Simard, R. (2007). TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, *33*(4), 1–40.

Marsaglia, G. (2008). The Marsaglia random number CDROM including the diehard battery of tests of randomness. http://www.stat.fsu.edu/pub/diehard/.

Marsaglia, G., Tsang, W. W., et al. (2002). Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, *7*(3), 1–9.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *8*(1), 3–30.

Matsuoka, K. (1992). Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, *22*(3), 436–440.

McKinney, W., et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th python in science conference, vol. 445* (pp. 51–56). Austin, TX.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT Press.

National Institute of Standards and Technology (2009). Radionuclide half-life measurements. https://www.nist.gov/pml/data/halflife-html.cfm.

National Institute of Standards and Technology (2010). NIST Statistical test suite, SP 800-22 Rev. 1a.

Quantique, I. (2021). Quantis random number generator. Carouge, Geneva, Switzerland, https://www.idquantique.com/random-number-generation/products/quantis-random-number-generator/.

Ritter, T. (1996). Randomness tests: A literature survey. http://www.ciphersbyritter.com/RES/RANDTEST.HTM.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Salmon, J. K., Moraes, M. A., Dror, R. O., & Shaw, D. E. (2011). Parallel random numbers: As easy as 1, 2, 3. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis* (pp. 1–12).

Saulson, P. R. (1990). Thermal noise in mechanical experiments. *Physical Review D*, *42*(8), 2437.

Spall, J. C. (2005). *Introduction to stochastic search and optimization: estimation, simulation, and control, vol. 65*. John Wiley & Sons.

Stipcevic, M. (2012). Quantum random number generators and their applications in cryptography. In *Advanced photon counting techniques VI, vol. 8375*. International Society for Optics and Photonics, Article 837504.

Symul, T., Assad, S., & Lam, P. K. (2011). Real time demonstration of high bitrate quantum random number generation with coherent laser light. *Applied Physics Letters*, *98*(23), Article 231103.

Tetko, I. V., Livingstone, D. J., & Luik, A. I. (1995). Neural network studies. 1. Comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, *35*(5), 826–833.

Vadhan, S. P., et al. (2012). *Pseudorandomness, vol. 7*. Now Delft.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

Vigna, S. (2016). An experimental exploration of Marsaglia's xorshift generators, scrambled. *ACM Transactions on Mathematical Software*, *42*(4), 1–23.

Wickham, H. (2016). *Use R!, Ggplot2: elegant graphics for data analysis* (2nd ed.). Cham, Switzerland: Springer International Publishing.