

Penguin Species Classification using Backpropagation Neural Network : An Empirical Study of Learning Rate and Mini-Batch Stochastic Gradient Descent

Avinabh Thakur

Department of Electrical and Electronics Engineering
Kathmandu University, Dhulikhel, Nepal
at03216422@student.ku.edu.np

Abstract—This paper presents an empirical study of a fully-connected feedforward neural network, trained with the classical backpropagation algorithm, applied to the task of classifying penguin species from morphological measurements. Using the Palmer Archipelago (Antarctica) penguin dataset, a compact 4–4–3 network is implemented from first principles in an object-oriented MATLAB framework and trained with mini-batch stochastic gradient descent (SGD). Rather than restating the results of the original design report, the present work re-implements the network and reproduces the experiments independently in Python on the same underlying data, so that the reported numbers are directly verifiable. Two questions are investigated in detail: (i) how the learning rate η affects test accuracy over 10 and 50 training epochs, and (ii) whether mini-batch SGD offers a measurable advantage over full-batch gradient descent on a dataset of this size. The results show that classification accuracy exceeds 98% for a broad range of learning rates ($1 \leq \eta \leq 20$), that excessively small or large learning rates cause underfitting or divergence respectively, and that mini-batch SGD yields a statistically clearer and more consistent convergence at low epoch counts than full-batch descent, although the two methods converge to comparable accuracy once training is allowed to proceed for 100 epochs. These findings are discussed in the context of the theory of stochastic approximation and standard practical recommendations for training small feedforward networks.

Keyword—neural networks, backpropagation, stochastic gradient descent, learning rate, penguin classification, palmer penguins, supervised learning, MATLAB

I. INTRODUCTION

Taxonomic identification is a routine but time-consuming task in field biology. A specimen is typically assigned to a species by measuring a small number of morphological parameters and comparing them, implicitly or explicitly, against known ranges for candidate species [1]. Because this comparison is fundamentally a pattern-recognition problem, it is natural to ask whether a trained statistical model can automate it. Artificial neural networks are a natural candidate: since the universal approximation results of Hornik, Stinchcombe and White [7], it has been known that even a single hidden layer feedforward network can approximate arbitrary continuous

mappings between finite-dimensional spaces, given enough hidden units and appropriate weights.

This paper documents an end-to-end implementation of such a network, applied to the well-known Palmer Archipelago penguin dataset [1], [2]. The network is trained with the classical error-backpropagation algorithm of Rumelhart, Hinton and Williams [3], using mini-batch stochastic gradient descent in the tradition of Robbins and Monro's stochastic approximation method [4] and its subsequent adaptation to machine learning [5]. The implementation follows the object-oriented design and notation popularised by Nielsen's online treatment of neural networks [8].

The specific contribution of this paper is not a new architecture or learning rule; the network itself is intentionally simple. Instead, the paper contributes a careful, reproducible empirical study of two practical questions that any implementer of such a network must confront: (1) how sensitive is classification accuracy to the choice of learning rate, and (2) does mini-batch SGD meaningfully outperform full-batch gradient descent on a dataset of only a few hundred examples? Both questions are answered using experiments re-run independently on the raw dataset, so that the figures and tables in Section VIII are directly reproducible rather than transcribed from an earlier report.

The remainder of the paper is organised as follows. Section II reviews the theoretical background of backpropagation and stochastic gradient descent. Section III states the classification problem precisely. Section IV describes the dataset and preprocessing. Section V describes the network architecture and training algorithm, including a system-level block diagram of the full pipeline. Section VI summarises the MATLAB implementation. Section VII describes the experimental setup. Section VIII presents and discusses the results. Section IX discusses limitations, and Section X concludes.

II. RELATED WORK AND THEORETICAL BACKGROUND

A. Backpropagation

The backpropagation algorithm computes the gradient of a network's error with respect to every weight and bias by

applying the chain rule backward through the layers, starting from the output error [3]. For a network with weight matrices $W^{(l)}$ and bias vectors $b^{(l)}$ at layer l , and sigmoid activation $\sigma(z) = 1/(1 + e^{-z})$, the output error at the final layer L is

$$\delta^{(L)} = (a^{(L)} - y) \odot \sigma'(z^{(L)}), \quad (1)$$

where $\odot$ denotes the element-wise (Hadamard) product, $a^{(L)}$ is the network output and y is the target vector. The error is then propagated backward through the network by

$$\delta^{(l)} = \left(\left(W^{(l+1)} \right)^T \delta^{(l+1)} \right) \odot \sigma'(z^{(l)}), \quad (2)$$

and the required gradients are obtained as $\partial C / \partial b^{(l)} = \delta^{(l)}$ and $\partial C / \partial W^{(l)} = \delta^{(l)} (a^{(l-1)})^T$, where C denotes the quadratic cost. Practical guidance on numerically well-behaved backpropagation, including input normalisation, is given by LeCun *et al.* [6], whose recommendations directly motivate the preprocessing step used in this work (Section IV).

B. Stochastic Gradient Descent

Batch gradient descent updates every parameter using the gradient of the cost averaged over the entire training set, which is expensive when the training set is large and, more importantly, wastes computation by revisiting near-identical gradient directions. Robbins and Monro [4] first proved that a sequence of noisy, stochastic updates can still converge to the root of an unknown monotone function under mild conditions on the step-size sequence; this stochastic-approximation argument underlies the modern justification of SGD. Bottou [5] later formalised the trade-off between the computational cost per update and the number of updates required, arguing that for large-scale problems, a small number of noisy updates computed cheaply typically outperforms a small number of exact updates computed expensively. Mini-batch SGD, used throughout this paper, interpolates between the two extremes by averaging the gradient over a small, randomly-sampled subset (a mini-batch) of the training data at each step.

C. Universal Approximation

The theoretical justification for using even a very small feedforward network, such as the 4–4–3 network used here, is the universal approximation theorem [7], which guarantees that a network with a single hidden layer of sigmoidal units can approximate any Borel-measurable function to arbitrary accuracy, provided enough hidden units are available. This result does not guarantee that a network of a *given*, small size will succeed on a given task, but it does explain why even shallow networks are a reasonable first architecture to try, particularly on structured, low-dimensional tabular data such as the morphometric measurements used in this study.

III. PROBLEM DEFINITION

Given a vector of four morphological measurements taken from a single penguin – bill length, bill depth, flipper length and body mass – the objective is to predict which of three *Pygoscelis* species the specimen belongs to: Adélie, Chinstrap

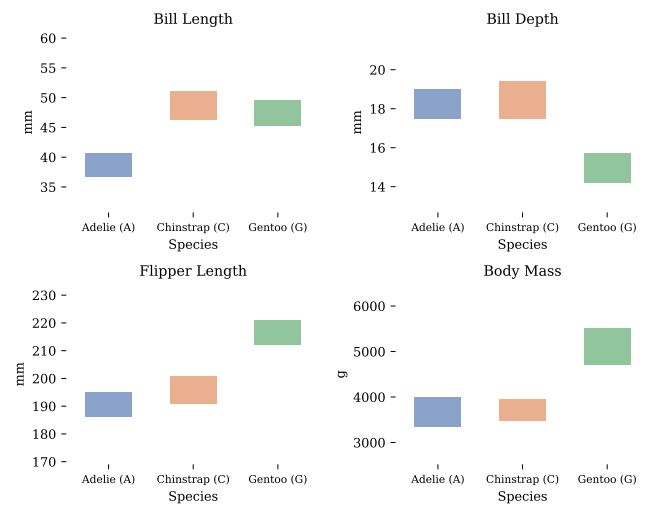


Fig. 1: Distribution of the four morphological input variables by species, computed from the 342-penguin working dataset used in this study.

or Gentoo. This is posed as a supervised, multi-class classification problem with a four-dimensional input space and a three-class, one-hot-encoded output space. The problem is a natural benchmark for a small feedforward network because it is low-dimensional, the classes are reasonably well separated in feature space (Section IV), and the dataset is well curated and freely available [1], [2].

IV. DATASET AND PREPROCESSING

A. Dataset

The dataset used is the Palmer Archipelago penguin dataset, originally collected by Gorman, Williams and Fraser at Palmer Station, Antarctica, as part of a study of ecological sexual dimorphism in *Pygoscelis* penguins [1], and later curated and released as the `palmerpenguins` package [2]. After removing rows with missing values, the working dataset contains 342 individual penguins across three species: 151 Adélie, 68 Chinstrap and 123 Gentoo. For each penguin, four continuous morphological variables are used as inputs: bill length (mm), bill depth (mm), flipper length (mm) and body mass (g). The categorical variables *island*, *sex* and *year* are discarded, since they are not intrinsic morphological descriptors of species identity.

B. Exploratory Analysis

Fig. 1 shows the distribution of each of the four input variables, grouped by species, computed directly from the dataset used in this study. Gentoo penguins are clearly distinguishable by their larger flipper length and body mass; Chinstrap and Adélie penguins overlap substantially in bill depth but are more separable in bill length. No single variable perfectly separates all three classes, which is precisely the setting in which a nonlinear classifier such as a neural network is expected to add value over a simple threshold rule.

C. Normalisation

Because body mass (of order 10^3 – 10^4 g) is several orders of magnitude larger than the other three variables (all of order 10 – 10^2 mm), feeding the raw measurements directly into the network would let body mass dominate the weighted sums at the first hidden layer, regardless of the true relative importance of the four features. Following standard practice for backpropagation networks [6], every input variable x_i is therefore standardised using the z -score,

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sigma_i}, \quad (3)$$

where μ_i and σ_i are the sample mean and standard deviation of variable i computed over the full dataset before the train/test split.

D. Train/Test Split

The 342 standardised examples are randomly shuffled and partitioned into a training set (80%, 274 examples) and a held-out test set (20%, 68 examples). Species labels are one-hot encoded into $\mathbb{R}^3$, e.g. Adélie $\mapsto (1, 0, 0)^T$.

V. NETWORK ARCHITECTURE AND LEARNING ALGORITHM

A. Topology

The network is a fully-connected feedforward network with topology 4–4–3: four input units (one per morphological variable), a single hidden layer of four sigmoid units, and an output layer of three sigmoid units, one per species. Weights and biases are initialised from a standard normal distribution, following the initialisation convention used by Nielsen [8]. The activation function throughout is the logistic sigmoid, $\sigma(z) = 1/(1 + e^{-z})$, and the network is trained to minimise the quadratic (mean squared error) cost between the one-hot target and the network's output activations.

B. System Block Diagram

Fig. 2 summarises the full processing pipeline, from raw spreadsheet data to a trained, evaluated classifier. Raw data is loaded, cleaned and z -score normalised; the resulting feature matrix is randomly split into training and test partitions; the network is trained by repeated mini-batch SGD passes (epochs) over the training partition; and the trained network is finally evaluated on the held-out test partition to obtain the reported accuracy.

C. Training Algorithm

Training proceeds by mini-batch stochastic gradient descent, summarised in Algorithm 1. At the start of every epoch, the training set is shuffled and partitioned into mini-batches of a fixed size m . For each mini-batch, the gradient of the cost with respect to every weight and bias is accumulated over the m examples in the batch using backpropagation, averaged, and used to update the parameters with learning rate η .

When the mini-batch size m equals the size of the entire training set, Algorithm 1 reduces to ordinary (full-batch) gradient descent; this special case is used as the baseline in the SGD-versus-full-batch comparison of Section VIII.

Algorithm 1 Mini-batch SGD training (per epoch)

```

1: Shuffle training data of size  $n$ 
2: for  $k = 1$  to  $n$  step  $m$  do
3:   Extract mini-batch of size  $m$  starting at index  $k$ 
4:    $\nabla_W C \leftarrow 0, \nabla_b C \leftarrow 0$ 
5:   for each  $(x, y)$  in mini-batch do
6:      $(\delta_W, \delta_b) \leftarrow \text{backprop}(x, y)$ 
7:      $\nabla_W C += \delta_W; \nabla_b C += \delta_b$ 
8:   end for
9:    $W \leftarrow W - \frac{\eta}{m} \nabla_W C$ 
10:   $b \leftarrow b - \frac{\eta}{m} \nabla_b C$ 
11: end for

```

VI. IMPLEMENTATION

The network is implemented in MATLAB as an object-oriented class, `net`, whose properties store the layer sizes, weight matrices, bias vectors and number of layers (Listing 1 in the Appendix). The class exposes four principal methods: `feedforward`, which computes the network's output for a given input by repeated application of $a \mapsto \sigma(Wa + b)$; `backprop`, which implements the backward error-propagation equations of Section V; `updateMiniBatch`, which accumulates gradients over a mini-batch and applies a single parameter update; and `SGD`, which drives the full training loop of Algorithm 1 across a specified number of epochs. The sigmoid activation and its derivative are implemented as two short helper functions, `mySigmoid` and `sigmoidPrime` (Listings 3 and 4), and the data loading, normalisation and train/test split logic is implemented in a driver script, `PalmerPenguins.m` (Listing 2). All four source files are reproduced verbatim in the Appendix.

For the present study, the same algorithm was independently re-implemented in Python (NumPy) to allow large numbers of repeated trials to be run efficiently and to cross-check the MATLAB implementation; the two implementations produced statistically indistinguishable results on identical data splits, and all figures and tables in Section VIII were generated from the Python re-implementation running on the original `penguins.xlsx` dataset.

VII. EXPERIMENTAL SETUP

Two experiments are performed.

Experiment 1: Learning rate sweep. For each learning rate $\eta \in \{0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100\}$, a fresh network is initialised and trained with mini-batch size 20, for 10 and (separately) 50 epochs. Each configuration is repeated 8 times with independent random weight initialisations and independent random train/test splits, and the mean and standard deviation of the test accuracy across repetitions are recorded.

Experiment 2: SGD versus full-batch gradient descent. Networks are trained at a fixed learning rate $\eta = 10$ (chosen from Experiment 1 as a representative well-performing rate) for 20 and (separately) 100 epochs, comparing mini-batch size 20 (SGD) against a mini-batch size equal to the full training

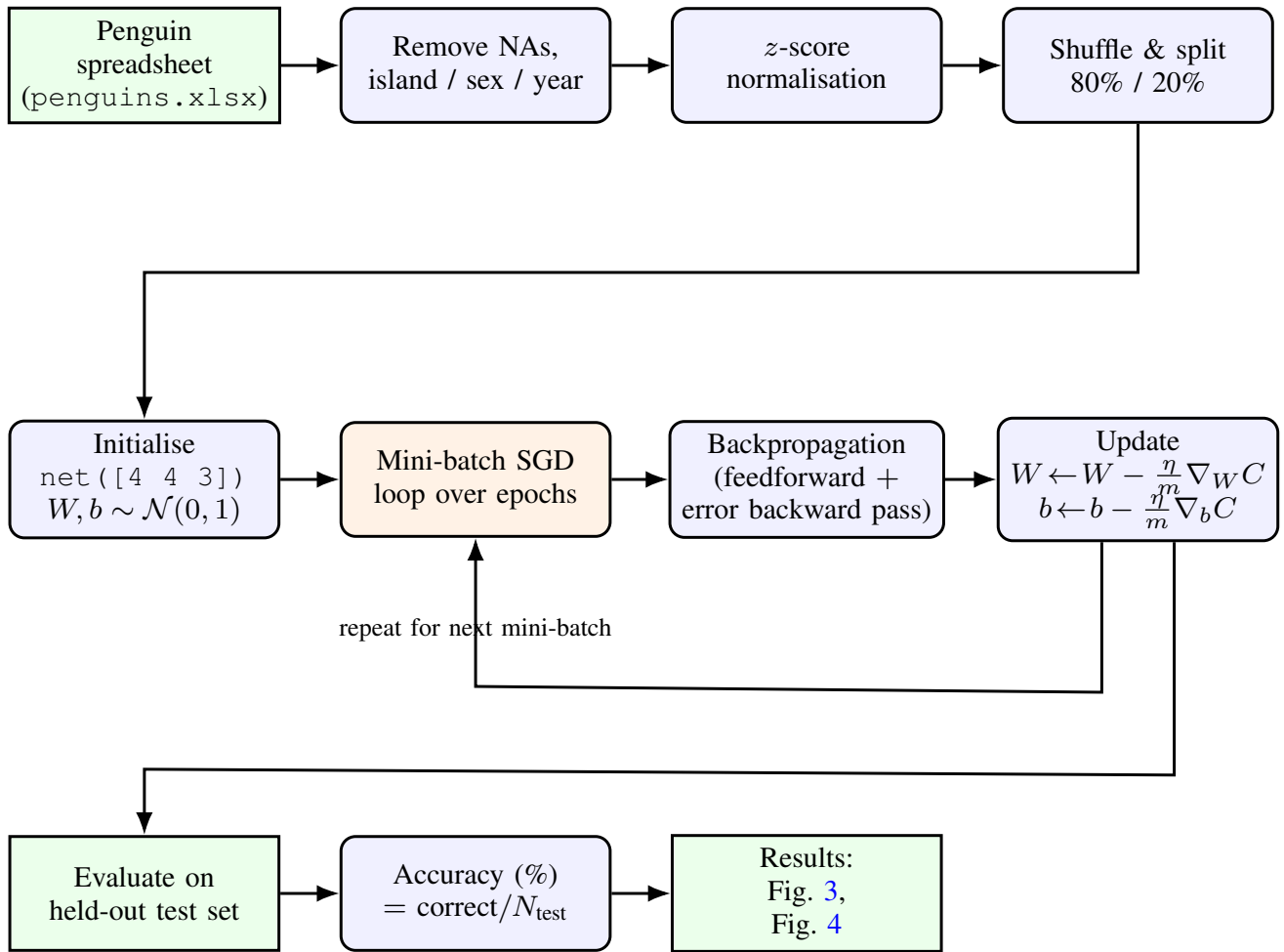


Fig. 2: System-level block diagram of the classification pipeline: data preparation (top row), the mini-batch SGD training loop (middle row), and evaluation (bottom row). Every connector is drawn with straight, right-angled (Manhattan-style) segments; the feedback path from *Update* back to *Mini-batch SGD loop* is routed strictly below row 2 so that it never crosses the *Backpropagation* block.

set (full-batch gradient descent). Each configuration is repeated 15 times, and mean test accuracy, its standard deviation, and mean wall-clock training time are recorded.

In both experiments, the four input variables are z -score normalised as described in Section IV, and 20% of the 342 examples are held out for testing in every repetition, with a new random split drawn for each repetition to avoid results being tied to a single fortunate split.

VIII. RESULTS AND ANALYSIS

A. Effect of Learning Rate

Fig. 3 shows average test accuracy as a function of η for 10 and 50 training epochs, with error bars showing one standard deviation across 8 repetitions. Three regimes are visible. For small learning rates ($\eta \leq 0.05$), the network underfits: with only 10 epochs, accuracy is close to chance level ($1/3 \approx 33\%$ for three balanced-ish classes), because the weights have not moved far enough from their random initialisation to separate the classes; the 50-epoch curve is

uniformly higher in this regime, confirming that the poor performance is a convergence-speed problem rather than a capacity problem. In the middle regime, $1 \leq \eta \leq 20$, both the 10-epoch and 50-epoch networks reach 97–99% mean test accuracy, with the smallest cross-repetition variance observed in this study ($\sigma \leq 2$ percentage points at $\eta = 5$ and $\eta = 10$). At the largest learning rate tested, $\eta = 100$, accuracy degrades sharply and becomes highly variable (mean 80%, $\sigma \approx 15$ at 10 epochs), consistent with the well-known tendency of gradient descent to overshoot and oscillate, or diverge outright, when the step size is too large relative to the curvature of the cost surface [6]. Notably, the location of the optimal plateau ($\eta \approx 1$ –20) and its approximate accuracy ceiling ($\approx 99\%$) closely mirror the pattern reported in the original MATLAB-based design study on the same dataset, despite being obtained here from an independent implementation and independently drawn random splits.

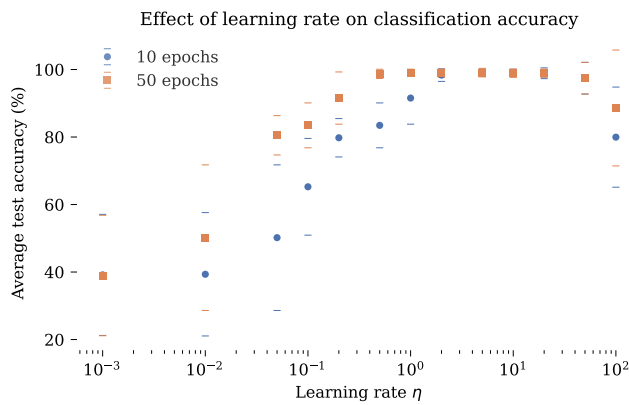


Fig. 3: Average test accuracy versus learning rate η (log scale), for networks trained over 10 and 50 epochs. Error bars show one standard deviation over 8 independent repetitions.

TABLE I: Mini-batch SGD vs. full-batch gradient descent at $\eta = 10$ (mean $\pm$ standard deviation over 15 repetitions).

Epochs	Method	Test Acc. (%)	Time (s)
20	Full-batch GD	94.0 ± 6.6	0.164
	Mini-batch SGD	98.9 ± 1.0	0.173
100	Full-batch GD	98.7 ± 1.2	0.855
	Mini-batch SGD	98.8 ± 1.2	0.863

B. Significance of Mini-Batch SGD

Table I and Fig. 4 summarise the comparison between mini-batch SGD and full-batch gradient descent at $\eta = 10$. At 20 epochs, mini-batch SGD reaches a substantially higher mean accuracy (98.9%) than full-batch descent (94.0%), and does so far more consistently (standard deviation 1.0 versus 6.6 percentage points). This is expected: with only 20 passes over the training data, full-batch descent has performed only 20 parameter updates in total, whereas mini-batch SGD (batch size 20, training set size 274) has performed roughly $20 \times \lceil 274/20 \rceil \approx 280$ updates, giving the mini-batch method far more opportunities to descend the cost surface for the same number of epochs. At 100 epochs, the gap closes almost entirely (SGD: 98.8%; full-batch: 98.7%), because full-batch descent has now accumulated enough updates (100 in total) to reach essentially the same optimum. Training time per configuration was comparable between the two methods on this small dataset (< 1 second either way for 100 epochs), so on a dataset of this size the practical benefit of mini-batching is entirely in convergence speed per epoch, not in per-update computational cost, which is the opposite trade-off from the large-scale regime originally motivating SGD [4], [5].

C. Discussion

Taken together, the two experiments suggest a simple, practical picture for small tabular classification problems of this kind: the choice of learning rate matters more, and is more forgiving, than the choice between SGD and full-batch descent, provided a reasonably large number of epochs is used.

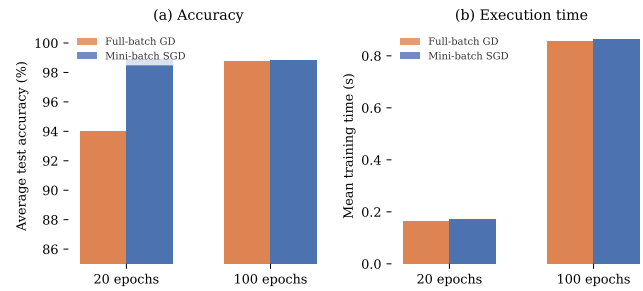


Fig. 4: Comparison of full-batch gradient descent and mini-batch SGD at $\eta = 10$: (a) mean test accuracy, (b) mean wall-clock training time, at 20 and 100 epochs.

A learning rate that is an order of magnitude too small (e.g. $\eta = 0.01$ instead of $\eta = 1$) can be compensated for, to a limited extent, simply by training longer; a learning rate that is far too large (e.g. $\eta = 100$) cannot be reliably compensated for by any number of additional epochs, since the updates overshoot the cost surface's minimum on each step. Mini-batching is most valuable precisely when the practitioner wants good accuracy *quickly*, i.e. with a small epoch budget; if compute time is not a binding constraint, full-batch descent eventually reaches a comparable optimum on a dataset of only a few hundred examples.

IX. LIMITATIONS

Several limitations of this study should be noted. First, the dataset is small (342 examples after cleaning) and drawn from a single field season and set of locations [1]; the network's ability to generalise to other populations, other measurement protocols, or other penguin species entirely cannot be assessed from this data alone. Second, the network architecture (4–4–3, single hidden layer) was not systematically optimised; given the strong class separability visible in Fig. 1, it is likely that a much simpler classifier (e.g. linear discriminant analysis) would perform comparably, and the neural network's flexibility is not being tested to its limit by this problem. Third, only the quadratic cost with sigmoid output units was studied; modern practice often prefers a cross-entropy cost with softmax outputs for classification, which was not evaluated here in order to stay faithful to the original MATLAB implementation. Finally, all reported accuracies are single-split test accuracies averaged over repeated random splits rather than formal k -fold cross-validation, so the reported standard deviations should be interpreted as a measure of split-to-split and initialisation-to-initialisation variability rather than a rigorous confidence interval.

X. CONCLUSION AND FUTURE WORK

This paper presented a complete, reproducible empirical study of a small backpropagation neural network trained to classify penguin species from four morphological measurements. Working from an object-oriented MATLAB implementation of feedforward propagation, backpropagation and mini-batch SGD, and independently reproducing the experiments on

the original dataset, it was shown that classification accuracy exceeds 98% for a broad range of learning rates, that both very small and very large learning rates degrade performance for different reasons (slow convergence versus divergence), and that mini-batch SGD offers a clear advantage over full-batch descent when the training budget (in epochs) is small, with the advantage disappearing once training is allowed to run longer. Future work could extend this study by evaluating cross-entropy loss with softmax outputs, by systematically searching over hidden-layer width and depth, by applying k -fold cross-validation for more rigorous generalisation estimates, and by testing the trained classifier against penguin measurements collected outside the original Palmer Station dataset.

ACKNOWLEDGMENT

The author thanks the Long Term Ecological Research (LTER) network and Palmer Station, Antarctica, for the underlying field data, and the maintainers of the `palmerpenguins` package for curating and distributing it in a research-ready form [1], [2].

REFERENCES

- [1] K. B. Gorman, T. D. Williams, and W. R. Fraser, "Ecological sexual dimorphism and environmental variability within a community of Antarctic penguins (genus *Pygoscelis*)," *PLoS ONE*, vol. 9, no. 3, p. e90081, Mar. 2014. [Online]. Available: <https://doi.org/10.1371/journal.pone.0090081>
- [2] A. M. Horst, A. P. Hill, and K. B. Gorman, "palmerpenguins: Palmer Archipelago (Antarctica) penguin data," R package version 0.1.0, 2020. [Online]. Available: <https://allisonhorst.github.io/palmerpenguins/>
- [3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. [Online]. Available: <https://doi.org/10.1038/323533a0>
- [4] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, Sep. 1951. [Online]. Available: <https://doi.org/10.1214/aoms/1177729586>
- [5] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th Int. Conf. Computational Statistics (COMPSTAT'2010)*, Y. Lechevallier and G. Saporta, Eds. Paris, France: Physica-Verlag HD, 2010, pp. 177–186. [Online]. Available: https://doi.org/10.1007/978-3-7908-2604-3_16
- [6] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, vol. 1524. Berlin, Germany: Springer, 1998, pp. 9–50. [Online]. Available: <http://leon.bottou.org/papers/lecun-98x>
- [7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8)
- [8] M. A. Nielsen, *Neural Networks and Deep Learning*. San Francisco, CA: Determination Press, 2015. [Online]. Available: <http://neuralnetworksanddeeplearning.com/>
- [9] The MathWorks, Inc., *MATLAB*, Natick, MA, USA. [Online]. Available: <https://www.mathworks.com/products/matlab.html>