# Evaluating Population Based Training on Small Datasets

Frode Tennebø and Marius Geitle
Østfold University College, Halden, Norway
{frodet, mariusge}@hiof.no

**Abstract**

Recently, there has been an increased interest in using artificial neural networks in the severely resource-constrained devices found in Internet-of-Things networks, in order to perform actions learned from the raw sensor data gathered by these devices. Unfortunately, training neural networks to achieve optimal prediction accuracy requires tuning multiple hyper-parameters, a process which has traditionally taken many times the computation time of a single training run of the neural network. In this paper, we empirically evaluate the Population Based Training algorithm, a method which simultaneously both trains and tunes a neural network, on datasets of similar size to what we might encounter in an IoT scenario. We determine that the population based training algorithm achieves prediction accuracy comparable to a traditional grid or random search on small datasets, and achieves state-of-the-art results for the *Biodeg* dataset.

## 1 Introduction

Over the last few years, we have seen a significant increase in research and development of novel Internet-of-Things (IoT) devices, and they are now used for controlling industrial machinery [1], monitoring crowds [2], managing energy in smart buildings [3], and in autonomous vehicles [4], to name a few.

Because IoT devices, especially in the edge, are typically severely constrained in terms of energy, computation, storage, and bandwidth resources, early approaches to using machine learning (ML) in IoT networks were often cloud-centric, in which predictions were made in the cloud. This approach required transferring all necessary data to the cloud, which prevented use-cases in which the necessary bandwidth and latency allowances were not available. Lately, prediction has mostly been moved to the nodes, where possible, but this move still requires the training to be done before deployment or in the cloud and therefore prevents using ML in situations where the data is not available before deployment, and the available bandwidth is insufficient for transfer to the cloud.

Among the most successful ML algorithms for learning from complex sensor signals are artificial neural networks (ANN); however, the majority of research into ANNs in the last decade has focused on improving the predictive performance on large datasets using an abundance of computational resources. This trend has resulted in a situation where even training a small state-of-the-art ANN on a small dataset can be too resource-demanding for an IoT node.

One of the reasons for the large resource requirements of training an ANN is that conventional approaches for tuning the hyper-parameters consist of an outer optimization loop based on strategies like random search [5], Bayesian optimization [6], and evolutionary computation [7]. The time required to tune the parameters is then the product of the training time and the number of hyper-parameter configuration evaluations. Recently, a promising new algorithm called Population Based Training (PBT) was proposed by Jaderberg et al. [8], which reduces the tuning time by both training and tuning the neural network simultaneously. However, the method was developed and tested on cases with an abundance of resources, and it is unclear whether it works well for small datasets.

*This paper was presented at the NIK 2019 conference. For more information, see http://www.nik.no/.*

In this paper, we empirically evaluate how well PBT performs using small datasets, and we have discovered that the PBT algorithm can achieve results similar to a simple grid-search approach, both using the gradient descent (GD) optimization algorithm to update the weights.

The remainder of this paper is structured as follows: in Section 2, we give a brief overview of related work, then describe the PBT implementation in Section 3. We go on to describe our two experiments in Section 4. Finally, we discuss the results and future work in Section 5.

# 2 Related work

Training the ML models outside of the cloud is a natural progression of the recent trend of moving services out of the cloud and closer to the consumers. An early example of this is the concept of edge computing, which involves processing the data closer to the user before transferring it to the cloud [9]. Training ML models on edge nodes can be difficult due to resource constraints, and doing so when the IoT network has only weaker nodes might require novel techniques such as federated learning [10], transfer learning [11] or online learning [12].

More recently, however, the concept of fog computing has been proposed [13]. Fog computing can be thought of as extending the concept of the cloud out and towards the edge by using more powerful processing gateways. With these more powerful nodes available, it is worth considering training ANNs locally, as these are powerful models, and prediction is typically fast.

Traditional approaches which involved automatically tuning the hyper-parameters of ANNs has so far involved either grid search [14], random search [5], Bayesian optimization [6], or evolutionary computation [7]. However, these approaches require independent trials of each hyper-parameter configuration being evaluated, causing a linear increase in the computation time required to train and tune a network. Additionally, optimizing schedules using these approaches requires manually specifying the model beforehand, often at the cost of increasing the number of hyper-parameters that must be optimized.

The PBT algorithm solves these problems by continually updating the parameters and hyper-parameters simultaneously, in addition to improving the accuracy of the model. However, it is unclear how well PBT will handle the higher noise in the evaluation of individual networks and the increased risk of overfitting caused by the smaller datasets.

# 3 Population based training

The PBT algorithm is a combined training and tuning algorithm proposed by Jaderberg et al. [8]. The outline of the algorithm can be seen in Algorithm 1.

It employs a population of N "cooperating" individuals, each exploring the problem space with randomly initialized hyperparameters. For the training algorithm, we use the standard *traingd* gradient descent with the backpropagation algorithm from MATLAB's *Deep Learning Toolbox* in a feed-forward neural network. For every 20 epochs, the entire population is evaluated on the validation set, and all individuals can have their weights and hyperparameters updated externally by two methods:

```
Initialize population P with random hyperparameters and weights
while not end of training do
  for p_i ∈ P do
    train p_i one step
  loop
  if ready for evaluation(P) then
    evaluate (P)
    for p_i ∈ P do
      p_i ← exploit (p_i, P)
      if exploited then
        p_i ← explore (p_i)
      end if
    loop
  end if
loop
```

Algorithm 1: High-level Population Based Training algorithm.

- *Exploit*: Replaces the weights and hyperparameters of the worst individuals from the bottom 20% quantile with randomly sampled individuals from the top 20% quantile. In their paper, Jaderberg et al. [8] named this *truncation selection* and noted that it, in general, gave better results than *binary tournament* and *T-test selection*.
- *Explore*: Proposes new hyperparameters to potentially better explore the problem space of an individual which has in the current evaluation already been a target for exploitation. Two methods were tried. We have only reported on *perturb*, in which each hyperparameter is independently and randomly scaled by a factor of either 0.8 or 1.2. We also implemented *resample*, in which each hyperparameter is resampled from the original prior distribution, but results from this test are not included as it did not give an improved performance in our tests.

The principles of *exploit* and *explore* very much resemble inheritance and mutation, respectively, as used in evolutionary computation (EC). Unlike EC, the crossover operation is not part of PBT. Instead, weights are inherited and hyperparameters are mutated. However, the rate of mutation is decoupled from the rate of learning, where learning is performed using standard training methods, in our case gradient descent. This process is similar to Lamarckian evolution with the Baldwin effect [15].

When the entire population is trained to completion, in our case either 600 or 2,000 epochs in total, the best ranked individual is selected and evaluated on the test set. This structure also makes the PBT algorithm well suited for parallel training.

# 4 Experiments

For our experiments, we wanted to see whether we could train models which achieve similar or improved predictive performance with PBT when compared to the traditional grid search with gradient descent (GSGD) approach for classification problems with a small number of examples. In order to make the comparison fair, we made sure that the total number of epochs was the same when using GSGD compared to a given population size N of PBT, e.g. for PBT with N=20 individuals which are trained for 600 epochs, the grid search was divided into 20 discrete values which are also trained for 600 epocs. Early stopping was not enabled in any of our two experiments to make sure

that the number of epochs was identical. The architecture was also the same for both experiments.

The first experiment was done with an implementation of PBT where only learning rate was tuned, to get an impression of how well it performed on various datasets. The second experiment expanded on this to tune two hyperparameters, learning rate and regularisation level, to see how it affected a small, unbalanced dataset.

While Jaderberg et al. [8] have used a variety of specialized measurements for measuring the performance of the trained models, we are interested in an automated approach for any classification dataset. While accuracy is one of the most used ranking metrics for classification performance, it can easily mischaracterize the predictive performance when used with unbalanced datasets [16]. Therefore, we opted to optimize the Area Under the Receiver Operating Characteristics Curve (AUC) for all datasets since AUC has been suggested as a better measure than accuracy [17].

## 4.1 Datasets

We have used four small, binary classification datasets retrieved from UCI[1], which contains between 1 055 and 11 055 instances. Additionally, we also used the *HIGGS* dataset, also from UCI, which contains 11 million instances, and are likely larger than what would be relevant when training the ANN in an IoT node. These sets are unbalanced to a variable degree, as shown in Table 1.

| Dataset | # instances | # features | Minority share |
|---|---|---|---|
| Biodeg [17] | 1,055 | 41 | 34 % (356) |
| HIGGS [18] | 11,000,000 | 21/28[2] | 47% |
| HIGGS subset | 100,00 | 21 | 47% |
| Phishing websites [19] | 11,055 | 30 | 44% (4,898) |
| Spambase [20] | 4,601 | 57 | 39% (1,813) |
| Wilt [21] | 4,839 | 6 | 5% (261) |

Table 1: Overview of the datasets used, the number of samples, number of features, and minority share in percent (absolute numbers) for each dataset.

These datasets were selected because they were of small size, with a moderate number of features and without missing values. The *HIGGS* dataset was used as a reference as it is nearly balanced and much larger. However, only 100,000 instances and 21 features were used in our experiment due to run-time constraints. Each dataset was divided into three parts: 20% for the test set, 16% for the validation set, and 64% for the training set.

## 4.2 Basic PBT experiment

We ran the initial PBT experiment with 10, 20, and 40 individuals in each population for 600 and 2,000 epochs. Each experiment was repeated 20 times on a network with three hidden layers, each consisting of 20 nodes. The results are reported as the arithmetic average of all trials.

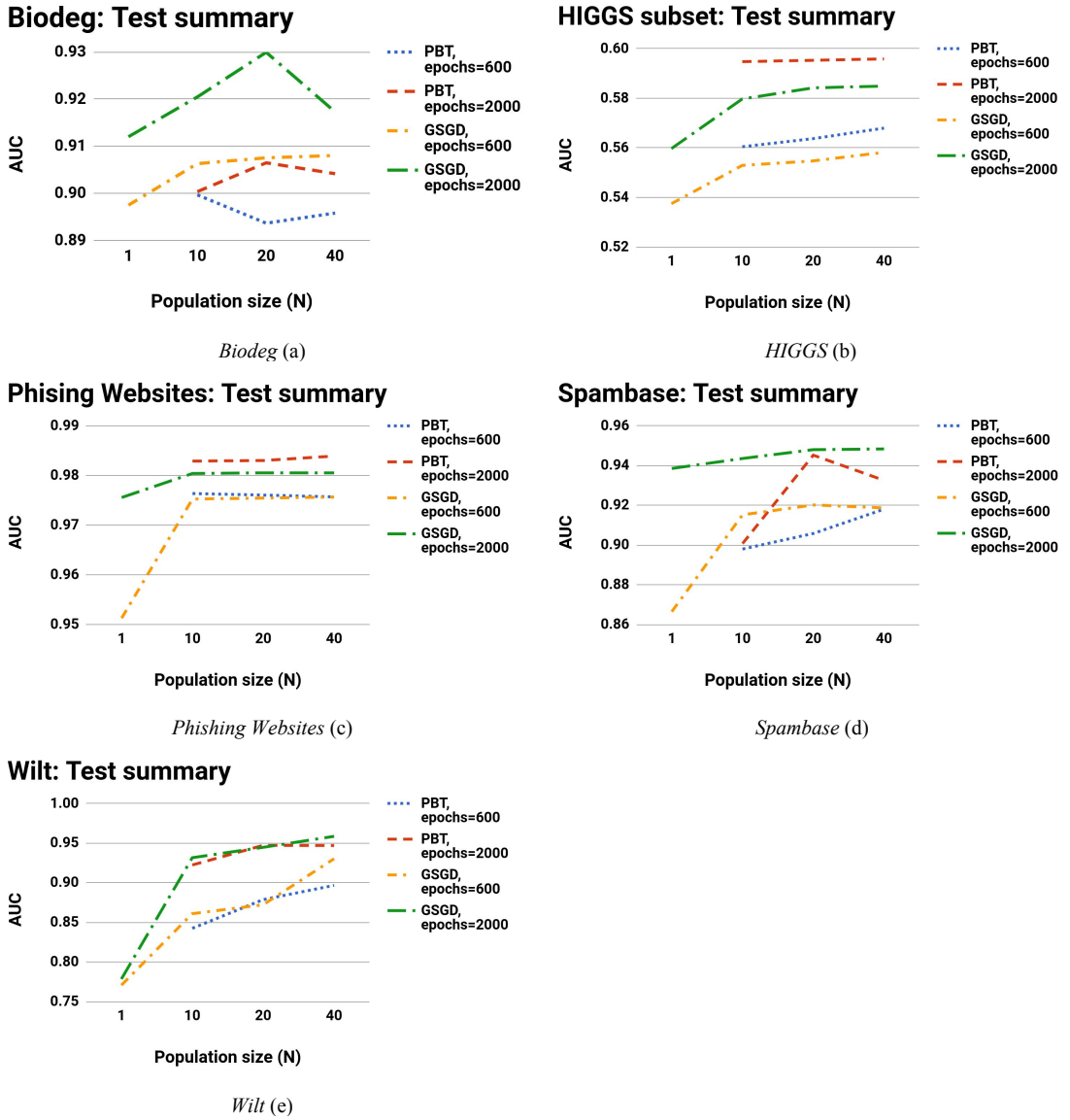
The only hyperparameter to be optimized for this first experiment was the learning rate (LR), $\eta$. For PBT, it was randomly initialized from a uniform distribution between 0.0001 and 0.1, and for GSGD the same range was uniformly divided to match the respective number of individuals in PBT. The results can be seen in Table 2 and Figure 1.

[1] http://archive.ics.uci.edu/ml/index.php

[2] 21 features are low-level, kinematic properties measured by the particle detectors in an accelerator, and seven features are functions of the first 21 features: high-level features derived by physicists to help discriminate between the two classes.

| Epochs | 600 | | | | 2,000 | | | |
|---|---|---|---|---|---|---|---|---|
| **Population size (N)** | **1** | **10** | **20** | **40** | **1** | **10** | **20** | **40** |
| **Dataset** | | | | | | | | |
| Biodeg, PBT | | 0.900 | 0.894 | 0.896 | | 0.900 | **0.906** | 0.904 |
| Biodeg, GSGD | 0.897 | 0.906 | 0.908 | 0.908 | 0.912 | 0.920 | **0.930** | 0.917 |
| HIGGS subset, PBT | | 0.560 | 0.564 | 0.568 | | 0.595 | 0.595 | **0.596** |
| HIGGS subset, GSGD | 0.537 | 0.553 | 0.555 | 0.558 | 0.560 | 0.580 | 0.584 | **0.585** |
| Phishing websites, PBT | | 0.976 | 0.976 | 0.976 | | 0.983 | 0.983 | **0.984** |
| Phishing websites, GSGD | 0.951 | 0.975 | 0.975 | 0.976 | 0.976 | 0.980 | 0.981 | **0.981** |
| Spambase, PBT | | 0.898 | 0.906 | 0.918 | | 0.901 | **0.945** | 0.933 |
| Spambase, GSGD | 0.867 | 0.915 | 0.920 | 0.919 | 0.939 | 0.944 | **0.948** | 0.948 |
| Wilt, PBT | | 0.843 | 0.879 | 0.897 | | 0.922 | **0.947** | 0.947 |
| Wilt, GSGD | 0.771 | 0.861 | 0.872 | 0.930 | 0.779 | 0.932 | 0.945 | **0.959** |

Table 2: AUC results of the initial PBT implementation and GSGD rounded to three significant digits for all datasets for four different population sizes and two different run-times in epochs.




*Biodeg* (a) *HIGGS* (b)

*Phishing Websites* (c) *Spambase* (d)

*Wilt* (e)

Figure 1: Graphical representation of AUC (y-axis) for the *Biodeg* (a), *HIGGS subset* (b), *Phishing Websites* (c), *Spambase* (d), and *Wilt* (e) datasets for PBT and GSGD. The x-axis shows the population size for PBT or the equivalent number of grid values for GSGD. Only LR has been optimized.

*4.2.1 Results*

For the *HIGGS* and *Phishing Websites* datasets, PBT scores better than GSGD. These two datasets are also the two largest with 100,000 and 11,000 instances, respectively. This indicates that PBT is more sensitive to the number of samples than GSGD. Jaderberg et al. [8] got their results using either very large datasets or generating as many samples as they needed.

For the other three datasets, PBT scores worse or comparable to GSGD. This result could be caused by overfitting because of the low number of training instances in combination with a relative high architecture complexity. However, it could also be caused by the AUC metric being too sensitive to the small number of examples in the dataset, as described by Hanczar et al. [22].

While PBT achieves better predictive performance on one dataset, and only slightly worse results on three of five datasets compared to the state-of-the-art results for these datasets, see Table 6, the results are still impressive given that the state-of-the-art results were achieved using methodology which was optimized for each dataset.

Even though the PBT and GSGD optimize the hyper-parameters with respect to the AUC, the predictive performance of the models could still suffer when the models are trained on imbalanced datasets. In our next experiment, we will address this by attempting to balance the dataset using oversampling.

## 4.3 PBT with oversampling and $L_2$ regularisation

Several strategies have been developed to deal with class imbalance, and Buda et al. [23] have discussed several of these methods and their use for training convoluted neural networks and concluded that imbalance is detrimental to classification performance. They also use AUC as an evaluation metric and justify this by arguing that overall accuracy metric is associated with notable difficulties for imbalanced data. Chawla et al. [24] note that oversampling can lead to overfitting for general machine learning techniques. To mitigate this and other problems, they developed Synthetic Minority Oversampling TEchnique (SMOTE). This technique works by adding synthetic data. However, they only compared against undersampling and did not consider the case of training a neural network. Therefore we will consider regular random, minority oversampling to deal with class imbalance for our next experiment.

In [14], Goodfellow et al. discuss several methods to handle overfitting and some of the computationally less expensive methods are drop out, $L_1$ and $L_2$ regularisation. We have used a variant of $L_2$ regularisation which modifies the normal performance function, the mean sum of squares of the network errors, by adding the term mean sum of the squares of the network weights and biases as described in Equation 2.

$$mse_w = \frac{1}{N}(1-\lambda)\sum_{i=1}^{N}(t_i-\alpha_i)^2 + \frac{1}{n}\lambda\sum_{j=1}^{n}w_j^2 \tag{2}$$

This is from the MATLAB 2018a implementation, which is similar, but not identical, to how others like Krogh and Hertz [25] and Goodfellow et al. [14] describe $L_2$ regularisation, but the general principle, and therefore the result should be similar.

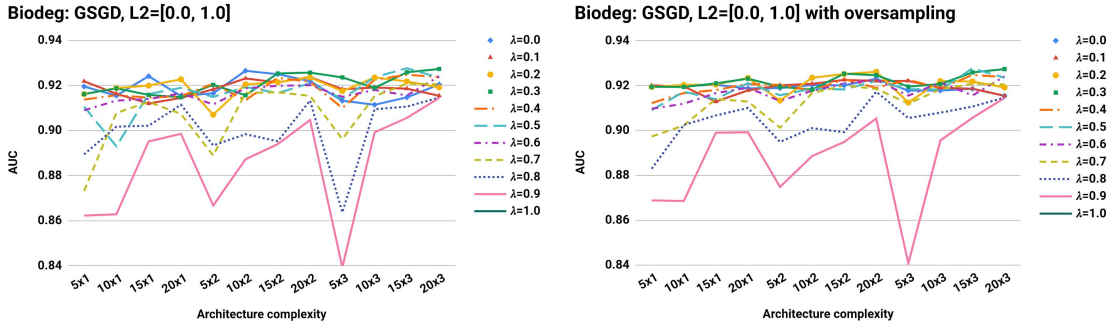
Based on the results of the previous experiment, we opted to focus only on the *Biodeg* dataset since this dataset achieved the worst performance in our N experiment, when compared to GSGD. In order to keep the complexities down, we looked at only two variants for this experiment, one with only oversampling and one with both oversampling and $L_2$ regularisation.

For this experiment we trained PBT for 600 epochs with a population of N=20 individuals for 20 independent trials, but to rule out any overfitting in the relatively large architecture in the first experiment, we tested multiple different architectures; one, two and three hidden layers with five, ten, fifteen, and twenty nodes. For GSGD, we used the same architectures as for PBT and the same single level grid search for learning rate as in the first experiment. Also, the hyperparameter distribution is the same as described in the first experiment for both PBT and GSGD. The results can be seen in Table 3 and Figure 3a.

| Architecture | AUC | | | | Accuracy in % | | | |
|---|---|---|---|---|---|---|---|---|
| | PBT, LR (SD) | PBT, LR, oversampling (SD) | GSGD, LR (SD) | GSGD, LR, oversampling, (SD) | PBT, LR (SD) | PBT, LR, oversampling (SD) | GSGD, LR (SD) | GSGD, LR, oversampling (SD) |
| 5x1 | 0.895 (0.015) | 0.905 (0.012) | 0.900 (0.025) | 0.909 (0.016) | 80.3 (2.40) | 82.1 (1.81) | 83.1 (3.03) | 82.9 (2.57) |
| 10x1 | 0.897 (0.013) | **0.911 (0.012)** | 0.912 (0.026) | 0.918 (0.013) | 80.9 (1.76) | 83.0 (2.24) | 84.4 (2.47) | 83.9 (3.00) |
| 15x1 | 0.895 (0.019) | 0.908 (0.012) | 0.917 (0.016) | **0.926 (0.020)** | 80.9 (2.48) | 82.9 (1.67) | 84.9 (2.56) | 85.5 (3.37) |
| 20x1 | 0.899 (0.011) | 0.909 (0.013) | 0.915 (0.019) | 0.924 (0.016) | 81.2 (1.68) | 82.7 (1.47) | 84.5 (1.95) | 84.1 (2.33) |
| 5x2 | 0.900 (0.015) | 0.910 (0.009) | 0.917 (0.018) | 0.923 (0.022) | 81.9 (2.01) | **83.4 (1.12)** | 85.2 (1.94) | **85.5 (2.97)** |
| 10x2 | 0.904 (0.015) | 0.910 (0.013) | 0.919 (0.028) | 0.915 (0.028) | **82.2 (1.87)** | 83.0 (1.49) | 84.9 (2.85) | 84.7 (3.25) |
| 15x2 | **0.904 (0.011)** | 0.911 (0.014) | 0.910 (0.022) | 0.921 (0.022) | 81.6 (1.84) | 83.0 (1.64) | 84.2 (1.85) | 84.1 (3.30) |
| 20x2 | 0.902 (0.012) | 0.908 (0.017) | **0.924 (0.024)** | 0.925 (0.019) | 81.9 (1.91) | 82.5 (1.59) | 85.2 (2.30) | 84.6 (2.50) |
| 5x3 | 0.900 (0.016) | 0.903 (0.024) | 0.916 (0.023) | 0.915 (0.024) | 82.2 (2.19) | 82.9 (2.60) | **85.3 (2.63)** | 84.5 (2.81) |
| 10x3 | 0.900 (0.020) | 0.910 (0.013) | 0.915 (0.016) | 0.918 (0.017) | 81.8 (1.99) | 82.9 (1.53) | 84.6 (2.18) | 84.3 (3.40) |
| 15x3 | 0.898 (0.019) | 0.907 (0.011) | 0.917 (0.024) | 0.923 (0.017) | 81.4 (2.14) | 82.5 (1.56) | 85.1 (2.70) | 84.0 (2.51) |
| 20x3 | 0.899 (0.016) | 0.911 (0.012) | 0.922 (0.026) | 0.921 (0.013) | 81.5 (2.00) | 83.1 (1.72) | 84.4 (3.35) | 84.4 (2.61) |

Table 3: AUC and accuracy results with standard deviation in parenthesis for the *Biodeg* dataset. Only learning rate was optimised for PBT. For GSGD, single level grid search on learning rate was done. Results are with and without oversampling rounded to three significant digits for twelve different architectures.

In order to find out how $L_2$ regularisation works on this dataset, we compared GSGD with only $L_2$ regularisation and with both oversampling and $L_2$ regularisation with levels of $\lambda$=[0.0, 1.0] in steps of 0.1. We randomly initialized $\eta$ from a uniform distribution between 0.0001 and 0.1. The results can be seen in Figure 2.




(a) (b)

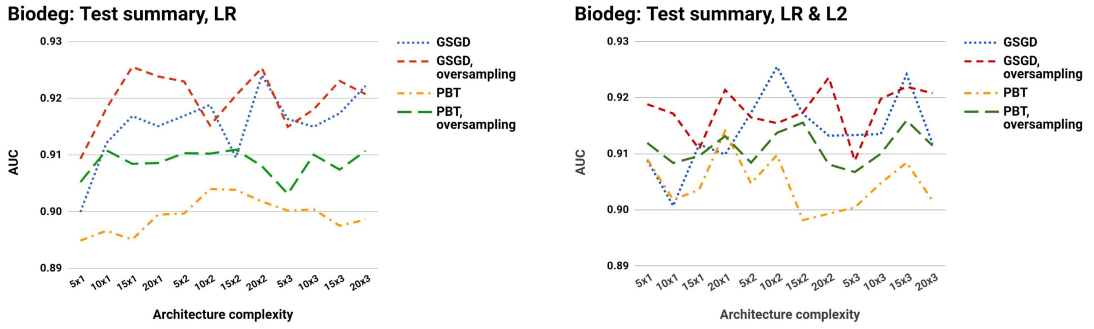
Figure 2: Graphical representation of AUC (y-axis) for GSGD for various architectures and levels of $L_2$ regularisation with (b) and without (a) oversampling for different architecture complexities (x-axis). $\lambda$=1.0 is not shown as it is equivalent to random classification with AUC ~= 0.5.

$\lambda$=1, $\lambda$=0.9, $\lambda$=0.8, and to a lesser extent also $\lambda$=0.7, particularly without oversampling, have a clear, negative impact on the performance for all architectures. Particularly noticeable is this for the less complex architectures with only five nodes in each layer. Otherwise, there is little difference between the results with or without

oversampling. For PBT's *exploit* we, therefore, decided to draw $\lambda$ from a uniform distribution between 0.0 and 0.7, and for *explore,* we truncated $\lambda$ to 0.7. GSGD was run as a grid search with $\lambda$=[0.0, 0.7] in steps of 0.23… and $\eta$=[0.0001, 0.1001] in steps of 0.025. The result can be seen in Table 4 and Figure 3b.

| **Architecture** | **PBT, LR & $L_2$ (SD)** | **PBT, LR & $L_2$, oversampling (SD)** | **GSGD, LR & $L_2$ (SD)** | **GSGD, LR & $L_2$, oversampling (SD)** |
|---|---|---|---|---|
| 5x1 | 0.909 (0.023) | 0.912 (0.021) | 0.909 (0.022) | 0.919 (0.020) |
| 10x1 | 0.902 (0.023) | 0.908 (0.020) | 0.901 (0.029) | 0.917 (0.021) |
| 15x1 | 0.904 (0.021) | 0.910 (0.027) | 0.912 (0.021) | 0.911 (0.017) |
| 20x1 | **0.914 (0.020)** | 0.913 (0.019) | 0.910 (0.023) | 0.921 (0.020) |
| 5x2 | 0.905 (0.013) | 0.908 (0.020) | 0.917 (0.015) | 0.916 (0.023) |
| 10x2 | 0.910 (0.019) | 0.914 (0.016) | **0.925 (0.013)** | 0.915 (0.013) |
| 15x2 | 0.898 (0.023) | **0.916 (0.017)** | 0.917 (0.017) | 0.917 (0.015) |
| 20x2 | 0.899 (0.019) | 0.908 (0.017) | 0.913 (0.021) | **0.924 (0.019)** |
| 5x3 | 0.900 (0.030) | 0.907 (0.029) | 0.913 (0.026) | 0.909 (0.017) |
| 10x3 | 0.905 (0.025) | 0.910 (0.023) | 0.914 (0.019) | 0.920 (0.026) |
| 15x3 | 0.908 (0.017) | 0.916 (0.018) | 0.924 (0.017) | 0.922 (0.019) |
| 20x3 | 0.902 (0.027) | 0.911 (0.022) | 0.912 (0.016) | 0.921 (0.021) |

Table 4: AUC and standard deviation results of PBT and GSGD for the *Biodeg* dataset. For PBT both learning rate and $L_2$ regularisation is tuned. For GSGD a 5 x 4 grid search was used. Results are with and without oversampling rounded to three significant digits for twelve different architectures.




(a) (b)

Figure 3: Graphical representation of AUC (y-axis) for PBT and GSGD with LR optimization with (b) and without (a) $L_2$ regularisation optimization, with and without oversampling for different architecture complexities (x-axis).

### *4.3.1 Results*

From Figure 3a, we can see that oversampling alone gives a better overall performance for PBT while it does not show overall improvements for GSGD.

Comparing Figure 3a and 3b, $L_2$ regularisation appears to overall have a positive effect on PBT without oversampling, but not with oversampling nor for GSGD. The p-value for conducting an independent two-tailed Student's t-test comparing the best results of PBT and GSGD with and without oversampling from Table 3 and 4 are given in Table 5.

Based on this, we can therefore reject the alternative hypothesis that there is a statistically significant difference between the best results optimizing only the learning rate and optimizing both learning rate and $L_2$ regularisation term within a 5% significance level.

| Results | LR (SD) (from Table 3) | LR & $L_2$ (SD) (from Table 4) | *p*-value |
|---|---|---|---|
| PBT | 0.904 (0.011) | 0.914 (0.020) | 0.081 |
| PBT, oversampling | 0.911 (0.012) | 0.916 (0.017) | 0.307 |
| GSGD | 0.924 (0.024) | 0.925 (0.013) | 0.816 |
| GSGD, oversampling | 0.926 (0.020) | 0.924 (0.019) | 0.760 |

Table 5: Overview of the best AUC and standard deviation results from the *Biodeg* datasets for PBT and GSGD with only LR optimization and with both LR and $L_2$ regularisation optimization, with and without oversampling and the calculated *p*-value.

This indicates that oversampling and $L_2$ regularisation has little or no effect on the *Biodeg* dataset. This possibility is supported by Figure 2, where there is little difference in results between $\lambda$=[0.0, 0.7] with and without oversampling.

Similarly, the *p*-values comparing the best results of PBT with the best result of GSGD (bold in Table 4) are 0.042 and 0.169 without and with oversampling, respectively. We can, therefore, accept the alternative hypothesis that there is a statistically significant difference between PBT and GSGD without oversampling, but not with oversampling, within a 5% significance level.

Our overall best results (bold in Table 4) has as 95% CI at [0.909, 0.923] for PBT and [0.919, 0.931] for GSGD. Both PBT and GSGD are better than state-of-the-art results listed in Table 6.

# 5 Discussion and future work

In this paper, we have proposed a general PBT configuration to train neural networks on classification datasets. We then evaluated the PBT configuration on a selection of binary classification problems using small datasets in a scenario that is relevant for IoT applications. We have shown that our PBT configuration outperforms GSGD on the larger of these datasets. But on the smallest datasets, PBT performs comparable or only slightly worse than GSGD. Using the Biodeg dataset, we have also shown that it can optimize both the learning rate $\eta$ and the penalty term $\lambda$ of $L_2$ regularisation simultaneously, and we have established that oversampling, when relevant, has a positive effect on PBT.

When compared to the state-of-the-art results reported in the literature for these datasets, as summarised in Table 6, both PBT and GSGD give better results for *Biodeg*.

| Dataset | Best-in-class | Experiment 1 | Experiment 2 | |
|---|---|---|---|---|
| | | Best initial PBT (from Table 2) | Best PBT with LR (from Table 3) (SD) | Best PBT with LR & $L_2$ (from Table 4) (SD) |
| Biodeg | 0.887 [26] | 0.906 | 0.911 (0.012) | 0.916 (0.017) |
| HIGGS | 0.891 [27] | 0.596 (subset) | | |
| Phishing websites | 0.993 [28] | 0.984 | | |
| Spambase | 0.980 [29] | 0.945 | | |
| Wilt | 0.990 [30] | 0.947 | | |

Table 6: Overview of the datasets used giving state-of-the-art results found in published papers to the best of our abilities together with the best results from our experiments.

We found that PBT achieves better or similar results on four of five datasets compared to the state-of-the-art results for these datasets, see Table 6. Given that the state-of-the-art results were achieved using a methodology which was optimized for each dataset, this is quite impressive and strengthen the idea that state-of-the-art or near state-of-the-art results can be achieved in constrained IoT devices in the edge. The inferior performance on the *HIGGS* dataset, is likely because we only used a subset of

the complete dataset, in addition to only considering a small class of network architectures.

Jaderberg et al. [8] claim that PBT does not have wall-clock run time that is greater than that of a single optimization process and is also able to use fewer computational resources than conventional search methods. In a separate paper by Jaderberg et al. [31] describing how they used PBT to train a neural network to playing the game of *Quake III Arena Capture the Flag* against human players using reinforcement learning on in real-time on the same commodity workstations used by the human players without adversely affecting the frame-rate of the game.

This presents intriguing possibilities for using PBT to train neural network architectures on low-powered IoT devices such as smart light bulbs, microwaves, and thermostats; devices where powerful processors are not available, and training common neural network architectures are currently not possible, thereby enabling new technologies that require these low-powered devices to learn, in vivo, how to respond to complex sensor information which will not be available until after deployment and when methods based on online learning is not suitable.

Automated training after deployment also highlights an important difference between our work and that of Jaderberg et al. [8]. Training after deployment will likely prevent the oversight of a human machine learning practitioner and therefore will require the training method to consistently achieve satisfactory predictive performance. In our experiments, we only used a single setup for all datasets, and still achieved results that either surpass or are similar to the state-of-the-art on those datasets. This is in contrast to the experiments by Jaderberg et al. [8] where the PBT algorithm only served as a framework which were extensively adapted to every problem.

This paper serves as a starting point for creating a general PBT configuration for training neural networks on classification datasets, and further work is likely to yield significant improvements. In the remainder of this section we will outline some possible further work.

In our experiments, we used the error rate as a measure of fitness to calculate the gradient during GD. Cortes and Mori [32] have pointed out that directly optimizing performance using AUC instead of indirectly by minimizing the error rate, can result in substantially better AUC scores.

In our experiments, we only tuned at most two hyper-parameters; the learning rate and the $L_2$-regularisation level. But PBT can, in principle, tune any number of hyper-parameters, and Jaderberg et al. [8] have tuned up to four hyper-parameters in their experiments. The complexity of optimizing hyper-parameters grows exponentially with the number of tunable hyper-parameters for a given algorithm using a grid search. PBT utilizes a property similar to random search [5], which should lead to much faster convergence, since the good regions of the hyper-parameters are explored more. More distinct values are used for each hyper-parameter to identify the good regions of hyper-parameters and ensures that they are explored more. Therefore, more complex algorithms with more hyper-parameters might produce better results with PBT.

The PBT algorithm itself also has hyper-parameters such as the size of quantile, time of evaluation, size of population and perturbation factors. Further work should explore ways of dispensing with these, or provide generally justified values. An option is to treat these parameters as tunable by PBT itself, together with the architecture as described by Zoph and Le [33]. This could make the optimization process more automated and robust. This is a type of automation that is an emerging strategy in the neural architecture search, a subfield of AutoML, according to Elsken et al. [34].

# 6 References


[1] D. Kwon, M. R. Hodkiewicz, J. Fan, T. Shibutani, and M. G. Pecht, "IoT-Based Prognostics and Systems Health Management for Industrial Applications," *IEEE Access*, vol. 4. pp. 3659–3670, 2016 [Online]. Available: http://dx.doi.org/10.1109/access.2016.2587754

[2] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV-Based IoT Platform: A Crowd Surveillance Use Case," *IEEE Communications Magazine*, vol. 55, no. 2. pp. 128–134, 2017 [Online]. Available: http://dx.doi.org/10.1109/mcom.2017.1600587cm

[3] K. Akkaya, I. Guvenc, R. Aygun, N. Pala, and A. Kadri, "IoT-based occupancy monitoring techniques for energy-efficient smart buildings," *2015 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. 2015 [Online]. Available: http://dx.doi.org/10.1109/wcncw.2015.7122529

[4] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer Architectures for Autonomous Driving," *Computer*, vol. 50, no. 8. pp. 18–25, 2017 [Online]. Available: http://dx.doi.org/10.1109/mc.2017.3001256

[5] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *J. Mach. Learn. Res.*, vol. 13, no. Feb, pp. 281–305, 2012.

[6] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," *arXiv [cs.LG]*, 12-Dec-2010 [Online]. Available: http://arxiv.org/abs/1012.2599

[7] J. H. Holland, "Genetic Algorithms," *Sci. Am.*, vol. 267, no. 1, pp. 66–73, 1992.

[8] M. Jaderberg *et al.*, "Population Based Training of Neural Networks," *arXiv preprint arXiv:1711.09846*, 27-Nov-2017 [Online]. Available: http://arxiv.org/abs/1711.09846. [Accessed: 23-Jan-2018]

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5. pp. 637–646, 2016 [Online]. Available: http://dx.doi.org/10.1109/jiot.2016.2579198

[10] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019 [Online]. Available: http://dx.doi.org/10.1109/icc.2019.8761315

[11] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1. 2016 [Online]. Available: http://dx.doi.org/10.1186/s40537-016-0043-6

[12] T. Chen, Q. Ling, Y. Shen, and G. B. Giannakis, "Heterogeneous Online Learning for 'Thing-Adaptive' Fog Computing in IoT," *IEEE Internet of Things Journal*, vol. 5, no. 6. pp. 4328–4341, 2018 [Online]. Available: http://dx.doi.org/10.1109/jiot.2018.2860281

[13] M. Mukherjee, L. Shu, and D. Wang, "Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3. pp. 1826–1857, 2018 [Online]. Available: http://dx.doi.org/10.1109/comst.2018.2814571

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[15] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Parallel Problem Solving from Nature — PPSN III*, 1994, pp. 5–15.

[16] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview," in *Data Mining and Knowledge Discovery Handbook*, Second edition., O. Maimon and L. Rokach, Eds. Springer, 2009, pp. 875–886.

[17] K. Mansouri, T. Ringsted, D. Ballabio, R. Todeschini, and V. Consonni, “Quantitative structure-activity relationship models for ready biodegradability of chemicals,” *J. Chem. Inf. Model.*, vol. 53, no. 4, pp. 867–878, Apr. 2013.
[18] P. Baldi, P. Sadowski, and D. Whiteson, “Searching for exotic particles in high-energy physics with deep learning,” *Nat. Commun.*, vol. 5, p. 4308, Jul. 2014.
[19] R. M. Mohammad, F. Thabtah, and L. McCluskey, “An assessment of features related to phishing websites using an automated technique,” in *2012 International Conference for Internet Technology and Secured Transactions*, 2012, pp. 492–497.
[20] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, “Spambase dataset,” *UCI Machine Learning Repository*, 1999. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/spambase
[21] B. A. Johnson, R. Tateishi, and N. T. Hoan, “A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees,” *Int. J. Remote Sens.*, vol. 34, no. 20, pp. 6969–6982, Oct. 2013.
[22] B. Hanczar, J. Hua, C. Sima, J. Weinstein, M. Bittner, and E. R. Dougherty, “Small-sample precision of ROC-related estimates,” *Bioinformatics*, vol. 26, no. 6, pp. 822–830, Mar. 2010.
[23] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Netw.*, vol. 106, pp. 249–259, Jul. 2018.
[24] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *1*, vol. 16, pp. 321–357, Jun. 2002.
[25] A. Krogh and J. A. Hertz, “A Simple Weight Decay Can Improve Generalization,” in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. Morgan-Kaufmann, 1992, pp. 950–957.
[26] Z. Luo and M. Hauskrecht, “Group-Based Active Learning of Classification Models,” *Proc. Int. Fla. AI Res. Soc. Conf.*, vol. 2017, pp. 92–97, May 2017.
[27] P. Sadowski, J. Collado, D. Whiteson, and P. Baldi, “Deep Learning, Dark Knowledge, and Dark Matter,” in *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, 2015, pp. 81–87.
[28] T. Chin, K. Xiong, and C. Hu, “Phishlimiter: A Phishing Detection and Mitigation Approach Using Software-Defined Networking,” *IEEE Access*, vol. 6, pp. 42516–42531, 2018.
[29] M. Khalid, I. Ray, and H. Chitsaz, “Scalable Nonlinear AUC Maximization Methods,” *Proceedings of the European*, 2018 [Online]. Available: http://www.ecmlpkdd2018.org/wp-content/uploads/2018/09/82.pdf
[30] Y. Zhang, J. Wu, C. Zhou, and Z. Cai, “Instance cloned extreme learning machine,” *Pattern Recognit.*, vol. 68, pp. 52–65, Aug. 2017.
[31] M. Jaderberg *et al.*, “Human-level performance in first-person multiplayer games with population-based deep reinforcement learning,” 03-Jul-2018 [Online]. Available: http://dx.doi.org/10.1126/science.aau6249. [Accessed: 18-Aug-2019]
[32] C. Cortes and M. Mohri, “AUC Optimization vs. Error Rate Minimization,” in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2004, pp. 313–320.
[33] B. Zoph and Q. V. Le, “Neural Architecture Search with Reinforcement Learning,” *arXiv [cs.LG]*, 05-Nov-2016 [Online]. Available: http://arxiv.org/abs/1611.01578
[34] T. Elsken, J. H. Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” *arXiv [stat.ML]*, 16-Aug-2018 [Online]. Available: http://arxiv.org/abs/1808.05377