doi: 10.1093/comjnl/bxz078

Pruning Neural Networks Using Multi-Armed Bandits

Salem Ameen and Sunil Vadera*

School of Computing, Science and Engineering, University of Salford, Manchester, M5 4WT,UK

**Corresponding author: S.Vadera@salford.ac.uk*

The successful application of deep learning has led to increasing expectations of their use in embedded systems. This, in turn, has created the need to find ways of reducing the size of neural networks. Decreasing the size of a neural network requires deciding which weights should be removed without compromising accuracy, which is analogous to the kind of problems addressed by multi-armed bandits (MABs). Hence, this paper explores the use of MABs for reducing the number of parameters of a neural network. Different MAB algorithms, namely ϵ-greedy, win-stay, lose-shift, UCB1, KL-UCB, BayesUCB, UGapEb, successive rejects and Thompson sampling are evaluated and their performance compared to existing approaches. The results show that MAB pruning methods, especially those based on UCB, outperform other pruning methods.

Keywords: neural networks; multi-armed bandits; pruning weights

Received 16 November 2018; revised 26 April 2019; editorial decision 25 June 2019
Handling editor: Fionn Murtagh

1. INTRODUCTION

The use of deep learning has led to dramatic improvements in performance in many pattern recognition applications [1–7]. These deep learning models consist of neural networks with a large number of parameters that require a significant amount of memory and computational resource [8]. Hence, there is renewed interest in developing algorithms for reducing the size of neural networks while retaining their predictive power. A survey of the literature reveals four categories of algorithms: direct methods, network pruning (NP) methods, use of regularization and methods that utilize sensitivity analysis, which are summarized below.

Direct methods [9] work by assessing the effect of setting weights to zero and removing weights that have little impact on performance. NP methods [10–13] are based on the view that very small weights are the least important and can be removed without affecting performance. Regularization methods [14, 15] extend the loss function L_i (such as mean square error) to include an additional term, $R(W)$, that aims to reduce the magnitude of weights and promote generalization [14, 15]:

$$\frac{1}{N}\sum_{i=1}^{N} L_i + \frac{\lambda}{N} R\left(W\right),$$

where $\lambda > 0$ is a parameter that can be set to a value that reflects the weight given to the regularization term and N is the number of examples in the training set. There are many types of regularization functions $R(W)$, and two that have been widely used are the sum of the squares of the weights (L2-norm) and the sum of the absolute values of the weights (L1-norm).

Sensitivity analysis methods [16–22] aim to assess the effect of perturbing the weights on the loss function. These include a method due to Le Cun *et al.* [19], known as optimal brain damage (OBD), that approximates the change in loss δL with the Taylor series [19]:

$$\delta L = \sum_i g_i \delta w_i + \frac{1}{2}\sum_i h_{ii}\delta w_i^2 + \frac{1}{2}\sum_{i \neq j} h_{ij}\delta w_i \delta w_j + O\left(||\delta W||^3\right), \tag{1}$$

where the $\delta w_i, \delta w_j$ are the weight perturbations, g_i are the components of the gradient of the loss function L with respect to the weights W and h_{ij} are elements of a Hessian matrix H:

$$g_i = \frac{\delta L}{\delta w_i} \text{ and } h_{ij} = \frac{\delta^2 L}{\delta w_i \delta w_j}.$$

Le Cun *et al.* [19] note that computing the Hessian matrix is computationally expensive, and to simplify the computation, they assume that the change in loss can be approximated by the diagonal elements of the Hessian, resulting in a simplification of equation (1) to

$$\delta L = \sum_i h_{ii}\delta w_i^2.$$

Given this simplification, the saliency s_k of a weight w_k can then be computed by [19]:

$$s_k = \frac{1}{2} h_{kk} w_k^2,$$

where the second-order derivatives, h_{kk}, are computed in a manner similar to the way the gradient is computed in back-propagation. In a later study, Hassibi *et al.* [20] also use equation (1) to develop a method known as optimal brain surgeon (OBS) but argue that it is not necessary to make the simplifying assumption that the non-diagonal elements of the Hessian matrix are zero.

The above methods for pruning neural networks have different merits. The direct methods are of $O(NP^3)$, where P is the number of weights and N is the size of the training set, and hence are considered to be intractable [9]. Regularization can lead to weights decaying towards zero, although as Collins and Kohli [23] and Gupta *et al.* [24] show, they may not reduce the weights to zero. One approach to address this might be to apply magnitude-based pruning methods to remove any small weights after regularization. However, Hassibi *et al.* [20] show that pruning based on magnitude can lead to removing important weights, and Srinivas and Babu [25] conclude that methods based on sensitivity analysis find it difficult to prune deep neural networks.

This paper explores an alternative approach based on the observation that there is a trade-off in deciding which weights to remove. That is, having too many weights can lead to overfitting, but equally, removing too many weights can result in underfitting the data. Although direct methods are optimal, evaluating the merits of weights by using all the training data to assess their value is not computationally feasible except for small data sets and networks. Equally, utilizing only a small sample of the data and trials may not lead to an accurate assessment of the value of a weight. Hence, this paper develops and evaluates an algorithm based on the use of multi-armed bandits (MABs), which have been successfully applied to problems where trials are carried out to explore the merits of various options [26–38].

The rest of this paper is organized as follows: Section 2 presents the background on MABs, Section 3 develops a new pruning algorithm that utilizes MABs, Section 4 presents the results of an empirical evaluation and Section 5 concludes the paper.

2. BACKGROUND ON MABS

The term MAB refers to a framework that is based on modelling a gambler who faces a collection of slot machines and needs to select which machines to play in order to maximize returns. Prior to each lever pull, the gambler will know the expected return or pay off based on the previous history of rewards and will be able to use this to decide which arm to pull next in order to achieve his or her goal. Typically, the goal can be to maximize the cumulative reward [39] or to find the best arm [40]. When the aim is to maximize the cumulative reward, a key decision for the gambler is to decide whether to exploit the best arm to date or explore other arms with the hope of gaining greater reward. In contrast, when the aim is to find the best arm, a key decision is to select an arm that will lead to high confidence that the arm ultimately selected will indeed be the best one to exploit. Given the goal of the algorithm developed in this paper is to remove weights that do not impact the performance of a neural network, we utilize MABs for identifying the best arms given a fixed budget defined by the number of lever pulls.

There are several alternative strategies for selecting the next arm to pull which can be grouped into three categories: random exploration [41], optimistic exploration [42] and Bayesian bandits [43, 44]. The following subsections summarize algorithms in these categories.

2.1. Random exploration

In random exploration methods [41], arms are pulled randomly, expected returns are calculated and a strategy for deciding when the best arm should be exploited and when other arms are tried is employed. In the simplest algorithm, the next arm is chosen randomly for a fixed number of times, an average reward computed and the best arm selected and exploited repeatedly. A more informed strategy, known as ϵ-greedy [37, 39, 45], pulls (i.e. exploits) the current best arm with a probability 1-ϵ and otherwise pulls another arm randomly (i.e. explores). More formally, given k arms, the ϵ-greedy algorithm selects the next arm a_{t+1} as follows:

$$a_{t+1} = \begin{cases} argmax\,[\mu_t(1), \ldots \mu_t(k)] & \text{with prob } 1 - \epsilon \\ select\ randomly\ from \{1 \ldots k\} & \text{with prob } \epsilon, \end{cases}$$

where $\mu_t(i)$ denotes the average reward for arm i obtained over t rounds.

Selecting a suitable ϵ for this algorithm can be challenging. If ϵ is large then, it will waste time pulling random arms without gaining much, but if ϵ is too small, then the learning process will be slow [45]. Hence, some authors have proposed a strategy of decaying ϵ over time [39]. For example, White [46] proposes decaying ϵ by

$$\frac{1}{\log(t + \phi)},$$

where ϕ is a small number and t is the number of rounds to date.

Another technique, known as WinStay, LoseShift (WSLS) [47, 48], changes the probability of selecting an arm depending on whether it results in a reward or not in the current round. If a selected arm results in a reward (i.e. wins), then its probability of being selected in the future is increased; otherwise, it is reduced. More formally, let $P_t(a)$ be the probability of choosing arm a at time t, then the WSLS update equations are

$$P_{t+1} = \begin{cases} P_t(a) + \beta(1 - P_t(a)) & \text{if } a \text{ wins} \\ P_t(a) - \beta P_t(a) & \text{otherwise,} \end{cases}$$

where β is a scaling parameter for rewarding the winner and penalizing the losers.

2.2. Optimistic explorations

As mentioned above, prior to pulling the next arm, a player will know the expected reward for each arm based on the history of lever pulls. A simple approach to selecting the next arm is to use an arm with the largest reward. However, this ignores the fact that early estimates of the rewards may be inaccurate, making it ineffective, irrespective of whether we wish to maximize cumulative reward or, as in our case, for the best arm problem. Thus, the main idea for optimistic exploration is to maintain confidence bounds on the expected rewards and to select an arm with the largest upper bound. This ensures that there is sufficient exploration at the start, knowing that the bounds will tighten as the number of lever pulls increase. MAB methods that adopt this approach are known as upper confidence bound (UCB) algorithms [39, 49]. More formally, UCB algorithms aim to select the next arm, a_{t+1} as follows:

$$a_{t+1} = \underset{i \epsilon \{1...K\}}{\text{argmax}}(\mu_i + P_{f_i}),$$

where μ_i is the expected reward for arm a_i and P_{f_i} is a padding function that is used to provide an upper bound for the reward for an arm a_i.

One of the earliest and most widely cited UCB algorithms, known as UCB1, uses the following selection function [39]:

$$a_{t+1} = \underset{i \epsilon \{1...K\}}{\text{argmax}} \left(\mu_i + \sqrt{\frac{2 \log t}{n_i}} \right), \tag{2}$$

where n_i is the number of times arm i has been chosen and t is the total number of rounds. UCB1 begins by playing each arm once to create an initial estimate. Then, for each iteration t, an arm is selected using equation 2. Initially, when arms have only been pulled a few times, the padding function in equation 2 allows exploration, but as the number of rounds increases, the extent of the padding function reduces, leading to greater exploitation of the arms that return the largest rewards.

KL-UCB [50, 51] presents an alternative approach where the padding function is derived from the Kullback–Leibler (KL) divergence measure, leading to a selection function where the next arm to pull is given by

$$a_{t+1} = \underset{i \epsilon \{1...K\}}{\text{argmax}}\, q(i)$$

$$q(i) = \max_{q \epsilon \{\mu_i ... 1\}} \left(d(\mu_i, q) \leq \frac{\log t + c . \log \log t}{n_i} \right),$$

where c is a constant and d is the KL divergence measure defined by [52]

$$d(p,q) = p \log \frac{p}{q} + (1-p) \log \left(\frac{1-p}{1-q} \right).$$

Both UCB1 and KL-UCB focus on the upper bounds of individual bandits. In contrast, some recent MAB algorithms also utilize the information in the lower bounds. One such algorithm is the unified gap-based exploration algorithm (UGapEb) [53]. As with the above algorithms, it maintains the lower and upper bounds (l_i, u_i) for each arm, but, in addition, it also maintains a set S_m with the top m arms, which are selected on the basis of the gap, g_i between the lower bound of an arm and the best upper bound amongst the arms:

$$g_i = \max_{j \neq i} u_j - l_i$$

Given this set, the algorithm considers whether an arm that is not currently in S_m can make it into the set in the next round. To assess this, it picks two arms: an arm in Sm with the lowest lower bound and an arm that is not in Sm but which has the highest upper bound. From these, it selects the arm with the largest uncertainty (i.e. $u_i - l_i$) and repeats the process a fixed number of times (fixed budget setting).

The bounds (l_i, u_i) for an arm a_i, with T_i lever pull at round t, are defined as follows and can be derived using the Chernoff–Hoeffding bounds [53]:

$$(l_i, u_i) = (\mu_i - \beta_i, u_i = \mu_i + \beta_i)$$

$$\beta_i = b \sqrt{\frac{a}{T_i(t-1)}}, \tag{3}$$

where a and b are user provided parameters that need tuning to improve performance. Audibert and Bubeck [40] argue that finding suitable parameters for algorithms such as UGapEb can be challenging and propose a parameter-free algorithm, successive rejects (SR), that involves $K - 1$ phases in which the available arms are attempted for a fixed number of times and the least promising arm is removed following each phase. The number of rounds n_k, in the k^{th} phase, is set with a view to achieving the theoretical lower bound for the best arm problem [40]:

$$n_k = \left\lceil \frac{1}{\overline{\log}(K)} . \frac{n-K}{K+1-k} \right\rceil,$$

where n is the total number of rounds and

$$\overline{\log}(K) = \frac{1}{2} + \sum_{i=2}^{K} \frac{1}{i}$$

2.3. Bayesian bandits

In the Bayesian approach, the potential reward from each arm is represented by a probability distribution that is updated in a Bayesian fashion. If $P(R)$ is the prior probability of a reward, then the goal is to compute a posterior distribution $P(R|h_t)$ where h_t is the history of rewards and actions.

One of the first algorithms to adopt a Bayesian approach was Thompson sampling [26, 27, 52]. Given s_a, the number of times an arm results in a reward, and f_a, the number of times an arm fails to deliver a reward, the probability distribution for an arm is defined by the beta distribution [43, 44]:

$$P(x) = \frac{(1-x)^{\beta-1}x^{\alpha-1}}{B(\alpha,\beta)},$$

where α is set to $s_a + 1$, β is set to $f_a + 1$ and $B(\alpha,\beta)$ is a normalizing constant.

In a more recent development that uses a Bayesian approach, Kaufmann *et al.* [54] propose an algorithm BayesUCB, in which the quantiles of a distribution are estimated to increasingly tight bounds and used to determine the next step:

$$a_{t+1} = \underset{i\epsilon\{1\ldots K\}}{\text{argmax}}\, q_i(t)$$

$$q_i(t) = Q\left(1 - \frac{1}{t}, \lambda_i^{t-1}\right),$$

where Q is the standard quantile function and λ_i^{t-1} denotes the posterior distribution of the mean reward for the i^{th} arm.

3. A MULTI-ARMED PRUNING ALGORITHM

To utilize the MAB algorithms described in Section 2, we need to define the arms and the reward function. The arms, a_k, are defined as the weights $w_{i,j}$ from a weight matrix W connecting the neurons in the layers of the network, and pulling an arm is considered to be equivalent to setting a weight to zero. The reward is defined as the difference between the accuracy of a network before and after removing a weight and is based on applying the network on a random sample of the data. The weight selected, together with the reward, becomes part of the history, which is then used by a MAB algorithm to select the next weight and the process repeated for a fixed number of rounds.

The reward function used varies depending on the type of bandit algorithm used. For UCB1, ϵ-greedy, KL-UCB and WSLS, the reward is computed by first calculating the difference in loss:

$$\delta L = L(D|W) - L(D|W^{'}),$$

where L is the loss when the network is applied on the sample of the data D, W denotes the weights and $W^{'}$ the weights after pruning. The reward is then computed using

$$R_{a_t,t} = \frac{max(0, Threshold + \delta L)}{Constant},$$

where the value of *Threshold* determines how much loss in performance can be tolerated when pruning. For example, suppose pruning results in a slightly worse performance, giving -0.05 for δL, then a threshold of 0.1 would still result in a reward. The divisor, *Constant*, is defined in a way that ensures that the reward is bounded between zero and one.

The Thompson sampling and BayesUCB algorithms assume Bernoulli rewards, and hence the reward is one if δL is larger than zero and zero otherwise.

In summary, given the number of rounds T and loss function L, the main steps of the MAB algorithm for pruning a neural network are

1. Initialize the round number: $t = 1$.
2. Let D be a random sample from the training data.
3. If any weight w_{ji} has yet to be played, then let the selected indices be (j, i), else use a MAB selection policy to return the indices (j, i) based on the current history.
4. Perform forward propagation with the initial weights W to obtain $L(D|W)$.
5. Save the selected weight w_{ji} in case it needs to be reinstated and set w_{ji} to zero to obtain W'.
6. Perform forward propagation with the revised weights W' to obtain $L(D|W')$.
7. Compute the change in loss and reward for round t:
$$\begin{aligned}\delta L &= L(D|W) - L(D|W^{'})\\ R_{ji,t} &= Reward(\delta L).\end{aligned}$$
8. Update the average reward for the selected arm:
$$\mu_{ji} = (n_{ji} - 1)/n_{ji} * \mu_{ji} + 1/n_{ji} * R_{ji,t}.$$
9. Increment the number of rounds t and repeat from Step 2 for T rounds after resetting the selected weight w_{ji}.
10. Rank the weights based on the rewards μ_{ji} and select a user defined proportion to be set to zero to obtain the pruned network.

This algorithm has been implemented with Step 3 invoking the selection policy of a specific MAB algorithm such as UCB1, KL-UCB, Bayes-UCB, UGapEb and Thompson sampling and Step 7 utilizing the relevant reward function.[1] Its worth noting that this algorithm inherits the extensive body of

[1] All the implementations are available on GitHub at https://github.com/SalemAmeen/codepaper/tree/master/codepaper.

research on the theoretical convergence properties of the different MAB algorithms. Readers interested in these properties are referred to the survey paper by Burtini *et al.* [45].

4. EMPIRICAL EVALUATION

The MAB pruning algorithms were evaluated by carrying out two sets of experiments. In the first, the NNSYSID package [55] was used to build neural networks for 12 data sets from the UCI machine learning repository [56] whose characteristics are summarized in Appendix A. The inputs and outputs of the neural networks reflected the features and class of the data sets. For consistency, each network adopted two hidden layers, with each hidden layer utilizing 20 neurons. The data were randomly divided into 60:40 training:testing sets. The models were trained using stochastic gradient descent with a batch size of 10 examples, momentum of 0.9 and weight decay of 0.005. The learning rate was initialized to 0.01, and the Softmax function was used as the activation function. Once trained, the neural networks were pruned using the different methods and their performance analysed on the testing set. In the second set of experiments, the LeNet deep learning model, with two convolutional layers and two fully connected layers, was adopted and trained on the MNIST data set [57]. The model was then pruned using the different methods and their performance compared on the MNIST testing data. The following subsections present the results from the two sets of experiments.

4.1. Results for the UCI data sets

The empirical evaluation of the eight MAB methods on the UCI data was carried out in comparison with the following four algorithms:

1. Random pruning (RP), in which the weights are sampled randomly for a fixed number of times and the weights removed are selected based on their average effect on reducing the error.[2]
2. An NP method [58], which removes weights that are below a user specified threshold value.
3. OBD [19], which, as described in Section 1, is derived using a Taylor series approximation for the change in error that would occur if the weights were perturbed but makes some simplifying assumptions about the off-diagonal elements of the resulting Hessian matrix.
4. OBS [20, 21], which also adopts a Taylor series approximation for the change in error but does not make the assumptions made by OBD.

The experiments aimed to address what happens to the accuracy when a network is pruned using the different methods and to compare the computational time of the methods. Table 1 presents the percentage error rate for each of the pruning methods on data sets from the UCI repository after pruning 10% of the weights after 1800 rounds of the MAB methods. The ϵ-greedy and UGapEb algorithms both have hyper parameters. In these experiments, for ϵ-greedy, we set ϵ to 0.5 to allow sufficient opportunities for exploring alternative arms. For UGapEb, we set the parameters a and b (in equation 3) to 0.25 and 1, respectively, based on the recommendations in Terayama *et al.* [59]. The average error rate, presented in the last row, shows that UCB1 and UGapE methods perform well relative to the other methods for pruning.

To compare the methods more formally, we adopted the methodology recommended by Demsar [60], who advocates use of a non-parametric test introduced by Friedman [61] to determine if there is a difference amongst the methods, and if so, to follow up with the Nemenyi test [62] to assess if one method is significantly better than another.

Table 3 presents the average rank of the 13 methods over the 12 data sets, with UCB1 being ranked the most effective in terms of reducing the error rate and NP ranked the least effective. Applying the Friedman test results in a *P*-value of 1.2×10^{-12} confirming that there is a significant difference amongst some of the methods and hence the Nemnyi *post hoc* test was carried out. The critical difference (CD) for the Nemenyi test was calculated using [62]:

$$CD = q_{\alpha,k}\sqrt{\frac{K(K+1)}{6N}},$$

where α is the confidence level, which was set to 0.05, K is the number of models (or classifiers) and N is the number of measurements (data sets). To compute $q_{\alpha,K}$, the studentized range statistic for infinite degrees of freedom divided by $\sqrt{2}$ was used.

Figure 1 displays the results, where the methods are plotted according to their average rank. The best ranked methods are to the right, and the line at the top indicates the CD. The coloured lines group the methods that are not significantly different at the 0.05 level. These results show that

- The UCB family of methods performed significantly better than NP. Thompson sampling for pruning is also significantly better than NP.
- The performance of BayesUCB and KL-UCB is very similar, which is consistent with the theoretical results due to Kaufmann *et al.* [63].
- Although the bandit-based methods have a higher average rank, the Nemenyi test does not distinguish these methods significantly from OBD or OBS in terms of minimizing the error.

2 Note that RP is equivalent to setting ϵ to 1 in ϵ-greedy.

TABLE 1. The table presents the computed percentage error rate for the datasets, where the column labelled Model presents the error before pruning and the remaining columns present the error when 10% of the weights are pruned using the different pruning methods.

Data set	**Model**	**ϵ-greedy**	**WSLS**	**UCB1**	**KL-UCB**	**BayesUCB**	**OBD**	**OBS**	**TS**	**NP**	**UGapEb**	**SR**	**RP**
Banknote	16	16	17	15	15	15	16	18	16	18	15	16	17
Blood TraNS.	12	12	18	12	12	12	12	13	12	13	12	12	14
Credit Approval	11	11	10	10	11	11	13	14	11	20	10	11	13
Haberman	20	20	23	20	20	20	20	20	20	21	20	21	21
Liver Disorders	30	30	35	30	30	30	30	37	30	32	30	31	32
MAGIC Gamma	20	23	25	20	20	20	21	26	20	22	20	21	23
Mammographic	19	17	19	17	17	18	19	20	19	19	18	19	19
MONK's	30	33	35	30	30	30	30	35	30	33	30	30	31
Connectionist	16	18	16	16	16	16	16	16	17	18	16	16	16
Spam Base	14	16	16	14	15	15	14	17	15	18	14	15	15
SPECTF Heart	20	21	23	20	20	20	20	22	20	23	20	20	21
Tic-Tac-Toe	22	23	25	21	22	22	22	25	22	24	22	22	23
Average	19.17	20.00	21.83	18.75	19.00	19.08	19.42	21.92	19.33	21.75	18.92	19.50	20.42

TABLE 2. Run-time performance of the pruning methods in seconds.

Data set	**ϵ-greedy**	**WSLS**	**UCB1**	**KL-UCB**	**BayesUCB**	**OBD**	**OBS**	**TS**	**UGapEb**	**SR**	**RP**
Banknote	8.5	21.3	12.5	25.5	29.3	210.8	214.5	12.5	23.3	150.5	4.6
Blood TraNS.	8.2	20.9	11.0	23.5	24.4	218.0	220.1	12.1	23.3	149.4	4.5
Credit Approval	9.4	19.8	11.9	23.1	23.9	469.3	470.0	20.1	27.1	169.3	4.5
Haberman	44.8	58.1	48.9	57.6	61.2	201.8	102.4	42.0	68.1	458.4	19.0
Liver Disorders	8.9	19.6	11.5	23.4	24.2	290.0	296.5	12.6	22.1	148.5	4.7
MAGIC Gamma	8.3	19.3	12.0	22.2	23.3	301.0	303.7	13.3	23.2	142.3	4.3
Mammographic	50.0	56.8	42.7	66.6	74.0	190.8	195.7	48.2	60.5	452.0	20.3
MONK's	9.3	20.4	13.3	23.5	23.3	276.5	180.0	12.4	24.4	151.4	4.5
Connectionist	8.2	19.0	12.7	22.3	22.3	1024.6	1100.0	11.8	27.7	139.2	4.4
Spam Base	14.3	22.0	11.4	24.8	26.1	570.2	577.8	13.4	25.9	155.5	5.9
SPECTF Heart	8.5	19.5	15.0	23.0	25.5	920.0	998.4	15.1	22.7	157.3	4.4
Tic-Tac-Toe	8.3	19.5	11.0	23.4	22.6	404.4	487.2	12.0	24.7	153.4	4.9
Average	15.56	26.35	17.83	29.91	31.68	423.12	428.86	18.79	31.08	202.27	**7.17**

Table 2 presents the run-time performance of these methods showing that the UCB family, ϵ-greedy, WSLS and Thompson sampling have the best run-time performance. OBD and OBS are computationally intensive given the need to compute the Hessian matrix. Thus, UCB methods achieve better performance on average than OBD and OBS but in significantly less time.

4.2. Results for the MNIST data set

The MNIST (Modified National Institute of Standards and Technology) data set is a well-known collection of handwritten digits that has been used in evaluating many handwriting recognition algorithms [57]. One of the most widely adopted deep learning architecture for this data set is the LeNet model [57]. In this model, the network has two 5×5 convolutional layers with 20 and 50 filters, respectively, and two fully connected layers with 500 and 10 neurons in the output layer. This results in the first two layers (the convolutional layers) having 500 and 25,000 weights, respectively, and the third and fourth layers (fully connected layers) having 2,500,000 and 500 nodes, respectively. The base line accuracy of this model is 98.06%, so it provides a good example for assessing which methods can best remove weights without adversely affecting the level of accuracy. To assess this, we applied a selection of MAB methods, with the number of rounds set to 150,000, to prune 50% of the weights in the second and third layers, which have the most weights. The methods we selected include one from the UCB1 family given their performance is similar, Thompson sampling and NP.

Table 4 presents the results, where the first column lists the pruning methods, the second column presents the accuracy obtained after pruning the first fully connected layer of LeNet

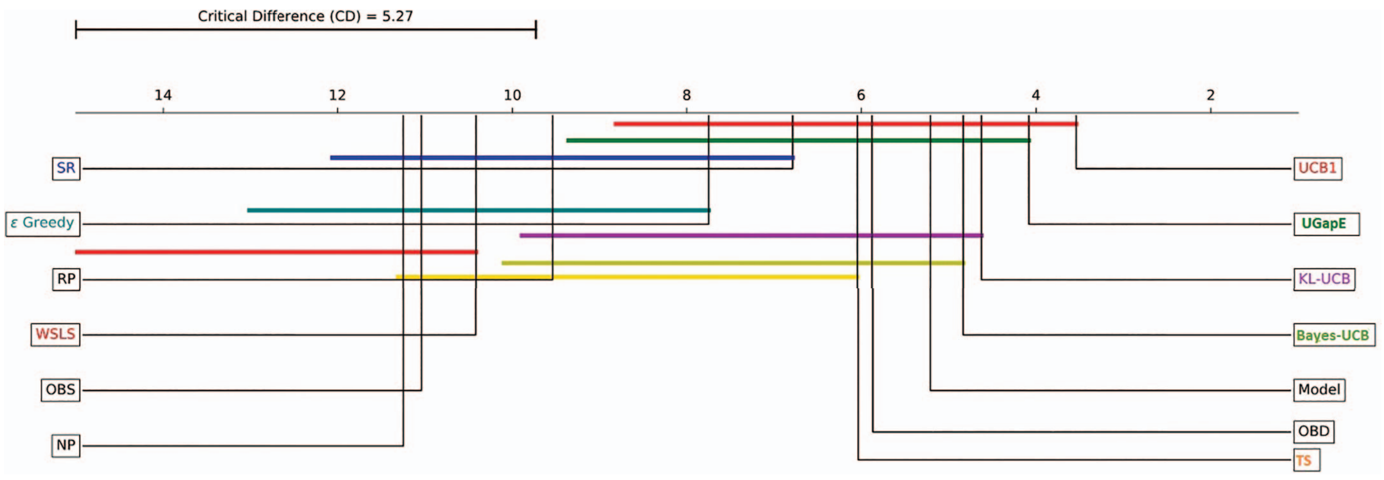

FIGURE 1. Comparison of all classifiers against each other with the Nemenyi test at $\alpha = 0.05$. The coloured lines group methods that cannot be distinguished based on the critical distance.

TABLE 3. Average rank of the methods.

Name of method	Mean rank
UCB1	3.54
UGapEb	4.08
KL-UCB	4.63
BayesUCB	4.83
Model before pruning	5.21
OBD	5.88
TS	6.04
SR	6.79
ϵ -greedy	7.75
RP	9.54
WSLS	10.42
OBS	11.04
NP	11.25

TABLE 4. The table shows the percentage accuracy after pruning 50% of the first fully connected layer (FC) and 50% of the second convolutional layer (Conv) in LeNet.

Method	FC	Conv
TS	98.30	98.10
ϵ -greedy	98.08	98.07
UCB1	98.40	98.20
RP	93.30	95.47
NP	59.20	49.50

and the last column presents the accuracy after pruning the second convolution layer of LeNet. The results show that the use of the bandit algorithms maintains accuracy but NP results in a significant decline in accuracy.

5. CONCLUSION AND FUTURE WORK

Pruning neural networks has re-emerged as a significant research topic following the emergence of deep neural networks which can be memory intensive and computationally demanding. Pruning neural networks can involve various trade-offs such as the size of a network versus accuracy and time spent in assessing the value of a weight versus accuracy of the assessment. This paper has explored the use of MABs for pruning neural networks. Several MAB methods, namely UCB1, UGapEb, BayesUCB, KL-UCB, ϵ-greedy, WSLS, SR and Thompson sampling were utilized for pruning neural networks and compared with existing methods.

In terms of retaining accuracy, the results on data sets from the UCI repository show that UCB1 and UGapEb are the most effective methods for pruning when compared to existing methods such as OBD, OBS and NP. Computationally, UCB1 is also significantly less demanding than methods such as OBS and OBD.

The bandit methods were also applied to Le Cun's deep learning model to examine their effectiveness and performed well in terms of maintaining the accuracy of the original model.

There are several directions for future research on the use of MAB algorithms for pruning neural networks. First, there are MAB algorithms, such as the SR algorithm, that are designed to guarantee the theoretical lower bound for the best arm problem. As our evaluation of SR indicates, these algorithms can be computationally demanding for deep neural networks. Hence, the development of best arm MAB algorithms that ensure near optimal regret bounds, perform well empirically and are also computationally feasible for pruning neural networks remains an open challenge. A second area of interesting work is to evaluate the use of MABs at different layers of granularity, such as for pruning neurons and feature maps.

In conclusion, this study shows that MABs based on UCB offer an effective way of pruning neural networks. Given the growth and scale of applications of deep neural networks, pruning them remains an open challenge, and we have therefore shared the implementations on GitHub so others can also build upon the work presented in this paper.

ACKNOWLEDGMENTS

The authors are grateful to the reviewers whose comments have led to improvements to several improvements to the paper including the inclusion of the SR and UGapEb algorithms in this study.

REFERENCES

[1] Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012) ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, Lake Tahoe, Nevada, USA, December, pp. 1106–1114.

[2] Zeiler, M.D. and Fergus, R. (2014) Visualizing and understanding convolutional networks. In *Computer Vision—ECCV 2014*, pp. 818–833. Springer.

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) Going deeper with convolutions. In *IEEE Conf. Computer Vision and Pattern Recognition, CVPR 2015*, Boston, MA, USA, June 7–12, 2015, pp. 1–9.

[4] Simonyan, K. and Zisserman, A. (2015) *Very deep convolutional networks for large-scale image recognition*. The 3rd International Conference on Learning Representations https://arxiv.org/abs/1409.1556.

[5] He, K., Zhang, X., Ren, S. and Sun, J. (2016) *Deep residual learning for image recognition*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, pp. 770–778.

[6] Montúfar, G., Pascanu, R., Cho, K. and Bengio, Y. (2014) On the number of linear regions of deep neural networks. In *Proc. 27th Int. Conf. Neural Information Processing Systems*, Vol. 2, Cambridge, MA, USA, NIPS'14, pp. 2924–2932. MIT Press.

[7] Ameen, S. and Vadera, S. (2017) A convolutional neural network to classify american sign language fingerspelling from depth and colour images. *Expert Syst.*, 34, e12197.

[8] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A. and Dally, W.J. (2016) *EIE: efficient inference engine on compressed deep neural network*. ACM/IEEE 43rd Annual International Symposium on Computer Architecture, Seoul, South Korea, pp. 243–254.

[9] Reed, R. (1993) Pruning algorithms—a survey. *IEEE Trans. Neural Netw.*, 4, 740–747.

[10] Chauvin, Y. (1989) A back-propagation algorithm with optimal use of hidden units. In Touretzky, D.S. (ed.) *Advances in Neural Information Processing Systems*, Vol. 1, pp. 519–526. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[11] Weigend, D. (1991) Back-propagation, weight-elimination and time series prediction. In *Proc. 1990 Connectionist Models Summer School*, pp. 105–116.

[12] Weigend, A.S., Rumelhart, D.E. and Huberman, B.A. (1991) Generalization by weight-elimination applied to currency exchange rate prediction. In *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN-91)*, pp. 837–841. IEEE.

[13] Weigend, A.S., Rumelhart, D.E. and Huberman, B.A. (1990) Generalization by weight-elimination with application to forecasting. In *Proc. 1990 Conf. Advances in Neural Information Processing Systems*, Vol. 3, pp. 875–882. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[14] Hanson, S.J. and Pratt, L.Y. (1988) Comparing biases for minimal network construction with back-propagation. In *Proc. 1st Int. Conf. Neural Information Processing Systems*, pp. 177–185. MIT Press, Cambridge, MA, USA NIPS'88.

[15] Arbib, M.A. (1995) *The Handbook of Brain Theory and Neural Networks*. MIT Press.

[16] Nikolaev, N. and Iba, H. (2006) *Adaptive Learning of Polynomial Networks: Genetic Programming, Backpropagation and Bayesian Methods*. Springer.

[17] Finnoff, W., Hergert, F. and Zimmermann, H.G. (1993) Improving model selection by nonconvergent methods. *Neural Netw.*, 6, 771–783.

[18] Neuneier, R. and Zimmermann, H. (1998) How to train neural networks. *Neural networks: tricks of the trade*, Lecture Notes in Computer Science, Vol. 1524, pp. 373–423. Springer.

[19] Le Cun, Y., Denker, J.S. and Solla, S.A. (1990) Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 598–605. Morgan Kaufmann.

[20] Hassibi, B., Stork, D.G. and Wolff, G.J. (1993) Optimal brain surgeon and general network pruning. In *IEEE Int. Conf. Neural Networks*, pp. 293–299. IEEE Press, Piscataway, NJ.

[21] Hassibi, B., Stork, D.G., Wolff, G. and Watanabe, T. (1993) Optimal brain surgeon: extensions and performance comparisons. In *Proc. 6th Int. Conf. Neural Information Processing Systems*, pp. 263–270. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, NIPS'93.

[22] Hassibi, B. and Stork, D.G. (1993) Second order derivatives for network pruning: optimal brain surgeon. In *Advances in Neural Information Processing Systems*, Vol. 5, San Francisco, CA, USA, pp. 164–171. Morgan Kaufmann Publishers Inc.

[23] Collins, M.D. and Kohli, P. (2014) *Memory bounded deep convolutional networks*. https://arxiv.org/abs/1412.1442.

[24] Gupta, M., Jin, L. and Homma, N. (2004) *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*. John Wiley & Sons.

[25] Srinivas, S. and Babu, R.V. (2015) *Data-free parameter pruning for deep neural networks*. Proceedings of the British Machine Vision Conference, pp. 31.1–31.12.

[26] Thompson, W.R. (1933) On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25, 285–294.

[27] Thompson, W.R. (1935) On the theory of apportionment. *Amer. J. Math.*, 57, 450–456.

[28] Robbins, H. (1952) Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.*, 58, 527–535.

[29] Babaioff, M., Sharma, Y. and Slivkins, A. (2009) Characterizing truthful multi-armed bandit mechanisms: extended abstract. In *Proc. 10th ACM Conf. Electronic Commerce*, New York, NY, USA, EC'09, pp. 79–88. ACM.

[30] Devanur, N.R. and Kakade, S.M. (2009) The price of truthfulness for pay-per-click auctions. In *Proc. 10th ACM Conf. Electronic Commerce*, New York, NY, USA, EC'09, pp. 99–106. ACM.

[31] Gelly, S. and Wang, Y. (2006) Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*, Canada, December.

[32] Kleinberg, R. (2004) Nearly tight bounds for the continuum-armed bandit problem. In *Proc. 17th Int. Conf. Neural Information Processing Systems*, Cambridge, MA, USA, NIPS'04, pp. 697–704. MIT Press.

[33] Kleinberg, R., Slivkins, A. and Upfal, E. (2008) Multi-armed bandits in metric spaces. In *Proc. Fortieth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, STOC'08, pp. 681–690. ACM.

[34] Mnih, V., Szepesvári, C. and Audibert, J.-Y. (2008) Empirical Bernstein stopping. In *Proc. 25th Int. Conf. Machine Learning*, New York, NY, USA, ICML'08, pp. 672–679. ACM.

[35] Bubeck, S., Stoltz, G., Szepesvári, C. and Munos, R. (2008) Online optimization in x-armed bandits. In *Advances in Neural Information Processing Systems*, pp. 201–208.

[36] Busa-Fekete, R. and Kégl, B. Fast boosting using adversarial bandits. In *27th Int. Conf. Machine Learning (ICML 2010)*, pp. 143–150.

[37] Sutton, R.S. and Barto, A.G. (1998) *Reinforcement Learning: An Introduction*. MIT Press.

[38] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013) *Playing Atari with deep reinforcement learning*. https://arxiv.org/abs/1312.5602.

[39] Auer, P., Cesa-Bianchi, N. and Fischer, P. (2002) Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47, 235–256.

[40] Audibert, J.-Y. and Bubeck, S. (2010) Best arm identification in multi-armed bandits. In *COLT—23th Conf. Learning Theory—2010*, Haifa, Israel, June, p. 13.

[41] Agrawal, R. (1995) Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Adv. in Appl. Probab.*, 27, 1054–1078.

[42] Auer, P., Cesa-Bianchi, N., Freund, Y. and Schapire, R.E. (2002) The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32, 48–77.

[43] Srinivas, N., Krause, A., Kakade, S. and Seeger, M. (2010) Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proc. 27th Int. Conf. Machine Learning*, USA ICML'10, pp. 1015–1022. Omnipress.

[44] Kaufmann, E., Korda, N. and Munos, R. (2012) Thompson sampling: an asymptotically optimal finite-time analysis. *Algorithmic Learning Theory*, Lecture Notes in Computer Science, Vol. 7568, pp. 199–213. Springer, Berlin, Heidelberg.

[45] Burtini, G., Loeppky, J. and Lawrence, R. (2015) *A survey of online experiment design with the stochastic multi-armed bandit*. https://arxiv.org/abs/1510.00757.

[46] White, J. (2012) *Bandit Algorithms for Website Optimization*. O'Reilly Media, Inc.

[47] Nowak, M. and Sigmund, K. (1993) A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364, 56–58.

[48] Posch, M. (1997) *Win Stay—Lose Shift: An Elementary Learning Rule for Normal Form Games*. Report. Santa Fe Institute.

[49] Lai, T.L. and Robbins, H. (1985) Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6, 4–22.

[50] Maillard, O.-A., Munos, R. and Stoltz, G. (2011) Finite-time analysis of multi-armed bandits problems with Kullback–Leibler divergences. In *Proc. 24th Annual Conf. Learning Theory*, pp. 497–514.

[51] Garivier, A. and Cappé, O. (2011) The KL-UCB algorithm for bounded stochastic bandits and beyond. In Kakade, S.M. and von Luxburg, U. (eds)*Proc. 24th Annual Conf. Learning Theory, Budapest, Hungary, 09–11 June, Proceedings of Machine Learning Research*, Vol. 19, pp. 359–376.

[52] Agrawal, S. and Goyal, N. (2013) Further optimal regret bounds for thompson sampling. In *Sixteenth Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, pp. 99–107.

[53] Gabillon, V., Ghavamzadeh, M. and Lazaric, A. (2012) Best arm identification: a unified approach to fixed budget and fixed confidence. In *Proc. 25th Int. Conf. Neural Information Processing Systems*, Vol. 2, pp. 3212–3220.

[54] Kaufmann, E., Cappe, O. and Garivier, A. (2012) On Bayesian upper confidence bounds for bandit problems. In Lawrence, N.D. and Girolami, M. (eds)*Proc. Fifteenth Int. Conf. Artificial Intelligence and Statistics*, La Palma, Canary Islands, 21–23 April, Proceedings of Machine Learning Research, 22, pp. 592–600. PMLR.

[55] Norgaard, M., Ravn, O. and Poulsen, N. (2002) Nnsysid—toolbox for system identification with neural networks. *Math. Comput. Model. Dyn. Syst.*, 8, 1–20.

[56] Bache, K. and Lichman, M. (2013). *UCI machine learning repository*, University of California, Irvine, School of Information and Computer Sciences. http://archive.ics.uci.edu/ml (accessed June 24, 2018).

[57] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) Gradient-based learning applied to document recognition. *Proc. IEEE*, 86, 2278–2324.

[58] Han, S., Pool, J., Tran, J. and Dally, W.J. (2015) Learning both weights and connections for efficient neural networks. In *Proc. 28th Int. Conf. Neural Information Processing Systems*, Vol. 1, Cambridge, MA, USA, NIPS'15, pp. 1135–1143. MIT Press.

[59] Terayama, K., Iwata, H., Araki, M., Okuno, Y. and Tsuda, K. (2018) Machine learning accelerates MD-based binding pose prediction between ligands and proteins. *Bioinformatics*, 34, 770–778.

[60] Demšar, J. (2006) Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7, 1–30.

[61] Friedman, M. (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Amer. Statist. Assoc.*, 32, 675–701.

[62] Nemenyi, P. (1963) Distribution-free multiple comparisons. PhD Thesis, Princeton University.

[63] Kaufmann, E., Cappé, O. and Garivier, A. (2011) On the efficiency of Bayesian bandit algorithms from a frequentist point of view. In *Neural Information Processing Systems (NIPS)*. Curran Associates, Red Hook, NY.

Appendix A

This appendix summarizes the data used from the UCI repository where full details are available

Data set	No examples	No features	Brief description
Banknote	1372	4	These data include records of whether bank notes are forged or not together with wavelet features extracted from images.
Blood Trans.	748	4	These data record whether there has been a blood donation in a particular month together with the months since donation, frequency, total donated to date and time since first donation.
Haberman	306	3	The records in these data indicate whether a patient has survived following surgery for breast cancer. Features include age, year of operation and number of axillary nodes detected.
Liver Disorders	345	6	Liver disorders data that include results from blood samples and alcohol consumption per day.
MAGIC Gamma	19020	10	Data generated from a Monte Carlo simulation of the registration of high energy gamma particles in a ground-based atmospheric gamma telescope. The task is to predict the presence of a gamma signal.
Mammographic	961	5	Data to enable screening for breast cancer based on breast imaging data (from BI-RADS) and age.
MONK's	432	6	Widely used artificial benchmark data where the task is to classify a robot based on features such head shape, body shape, smiling or not, holding, jacket colour, wearing a tie or not.
Connectionist	208	59	Each example consists of the data obtained when sonar signals are bounced off a metal cylinder or rock at various angles. The task is to predict whether a signal obtained is metal or rock.
Spam base	4601	96	Data for detecting spam emails based on frequency of certain words and use of capitalization.
SPECTF Heart	267	44	These data include examples of classifying patients with normal or abnormal hearts based on features extracted from single proton emission computed tomography images.
Tic-Tac-Toe	958	9	Each example of these data consists of states of tic-tac-toe (Noughts and crosses) and resulting win or no win.