
BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning

Andreas Kirsch* Joost van Amersfoort* Yarin Gal
OATML
Department of Computer Science
University of Oxford
{andreas.kirsch, joost.van.amersfoort, yarin}@cs.ox.ac.uk

Abstract

We develop BatchBALD, a tractable approximation to the mutual information between a batch of points and model parameters, which we use as an acquisition function to select multiple informative points jointly for the task of deep Bayesian active learning. BatchBALD is a greedy linear-time $1 - 1/e$ -approximate algorithm amenable to dynamic programming and efficient caching. We compare BatchBALD to the commonly used approach for batch data acquisition and find that the current approach acquires similar and redundant points, sometimes performing worse than randomly acquiring data. We finish by showing that, using BatchBALD to consider dependencies within an acquisition batch, we achieve new state of the art performance on standard benchmarks, providing substantial data efficiency improvements in batch acquisition.

1 Introduction

A key problem in deep learning is data efficiency. While excellent performance can be obtained with modern tools, these are often data-hungry, rendering the deployment of deep learning in the real-world challenging for many tasks. Active learning (AL) [7] is a powerful technique for attaining data efficiency. Instead of a-priori collecting and labelling a large dataset, which often comes at a significant expense, in AL we iteratively acquire labels from an expert only for the most informative data points from a pool of available unlabelled data. After each acquisition step, the newly labelled points are added to the training set, and the model is retrained. This process is repeated until a suitable level of accuracy is achieved. The goal of AL is to minimise the amount of data that needs to be labelled. AL has already made real-world impact in manufacturing [33], robotics [5], recommender systems [1], medical imaging [17], and NLP [30], motivating the need for pushing AL even further.

In AL, the informativeness of new points is assessed by an *acquisition function*. There are a number of intuitive choices, such as model uncertainty and mutual information, and, in this paper, we focus on BALD [18], which has proven itself in the context of deep learning [12, 29, 19]. BALD is based on mutual information and scores points based on how well their label would inform us about the true model parameter distribution. In deep learning models [15, 31], we generally treat the parameters as point estimates instead of distributions. However, Bayesian neural networks have become a powerful alternative to traditional neural networks and do provide a distribution over their parameters. Improvements in approximate inference [4, 11] have enabled their usage for high dimensional data such as images and in conjunction with BALD for Bayesian AL of images [12].

In practical AL applications, instead of single data points, batches of data points are acquired during each acquisition step to reduce the number of times the model is retrained and expert-time is

*joint first authors

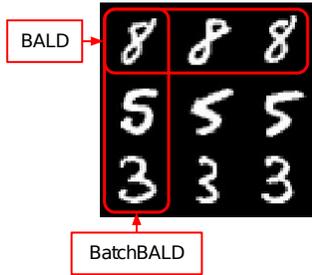


Figure 1: *Idealised acquisitions of BALD and BatchBALD.* If a dataset were to contain many (near) replicas for each data point, then BALD would select all replicas of a single informative data point at the expense of other informative data points, wasting data efficiency.

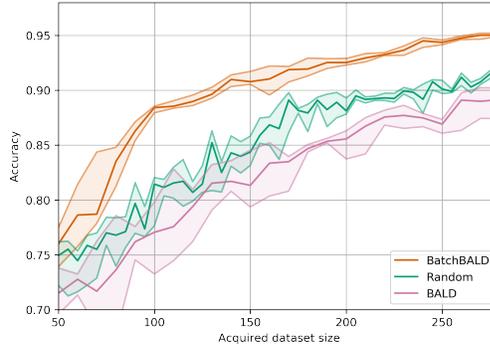


Figure 2: *Performance on Repeated MNIST with acquisition size 10.* See section 4.1 for further details. BatchBALD outperforms BALD while BALD performs worse than random acquisition due to the replications in the dataset.

requested. Model retraining becomes a computational bottleneck for larger models while expert time is expensive: consider, for example, the effort that goes into commissioning a medical specialist to label a single MRI scan, then waiting until the model is retrained, and then commissioning a new medical specialist to label the next MRI scan, and the extra amount of time this takes.

In Gal et al. [12], *batch acquisition*, i.e. the acquisition of multiple points, takes the top b points with the highest BALD acquisition score. This naive approach leads to acquiring points that are individually very informative, but not necessarily so jointly. See figure 1 for such a batch acquisition of BALD in which it performs poorly whereas scoring points jointly ("BatchBALD") can find *batches* of informative data points. Figure 2 shows how a dataset consisting of repeated MNIST digits (with added Gaussian noise) leads BALD to perform worse than random acquisition while BatchBALD sustains good performance.

Naively finding the best batch to acquire requires enumerating all possible subsets within the available data, which is intractable as the number of potential subsets grows exponentially with the acquisition size b and the size of available points to choose from. Instead, we develop a greedy algorithm that selects a batch in linear time, and show that it is at worst a $1 - 1/e$ approximation to the optimal choice for our acquisition function. We provide an open-source implementation².

The main contributions of this work are:

1. *BatchBALD*, a data-efficient active learning method that acquires *sets* of high-dimensional image data, leading to improved data efficiency and reduced total run time, section 3.1;
2. a greedy algorithm to select a batch of points efficiently, section 3.2; and
3. an estimator for the acquisition function that scales to larger acquisition sizes and to datasets with many classes, section 3.3.

2 Background

2.1 Problem Setting

The Bayesian active learning setup consists of an unlabelled dataset $\mathcal{D}_{\text{pool}}$, the current training set $\mathcal{D}_{\text{train}}$, a Bayesian model \mathcal{M} with model parameters $\omega \sim p(\omega | \mathcal{D}_{\text{train}})$, and output predictions $p(y | x, \omega, \mathcal{D}_{\text{train}})$ for data point x and prediction $y \in \{1, \dots, c\}$ in the classification case. The conditioning of ω on $\mathcal{D}_{\text{train}}$ expresses that the model has been trained with $\mathcal{D}_{\text{train}}$. Furthermore, an oracle can provide us with the correct label \tilde{y} for a data point in the unlabelled pool $x \in \mathcal{D}_{\text{pool}}$. The goal is to obtain a certain level of prediction accuracy with the least amount of oracle queries. At each acquisition step,

²<https://github.com/BlackHC/BatchBALD>

a batch of data points $\{\mathbf{x}_1^*, \dots, \mathbf{x}_b^*\}$ is selected using an acquisition function a which scores a candidate batch of unlabelled data points $\{\mathbf{x}_1, \dots, \mathbf{x}_b\} \subseteq \mathcal{D}_{\text{pool}}$ using the current model parameters $p(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})$:

$$\{\mathbf{x}_1^*, \dots, \mathbf{x}_b^*\} = \underset{\{\mathbf{x}_1, \dots, \mathbf{x}_b\} \subseteq \mathcal{D}_{\text{pool}}}{\arg \max} a(\{\mathbf{x}_1, \dots, \mathbf{x}_b\}, p(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})). \quad (1)$$

2.2 BALD

BALD (*Bayesian Active Learning by Disagreement*) [18] uses an acquisition function that estimates the mutual information between the model predictions and the model parameters. Intuitively, it captures how strongly the model predictions for a given data point and the model parameters are coupled, implying that finding out about the true label of data points with high mutual information would also inform us about the true model parameters. Originally introduced outside the context of deep learning, the only requirement on the model is that it is Bayesian. BALD is defined as:

$$\mathbb{I}(y; \boldsymbol{\omega} \mid \mathbf{x}, \mathcal{D}_{\text{train}}) = \mathbb{H}(y \mid \mathbf{x}, \mathcal{D}_{\text{train}}) - \mathbb{E}_{p(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})} [\mathbb{H}(y \mid \mathbf{x}, \boldsymbol{\omega}, \mathcal{D}_{\text{train}})]. \quad (2)$$

Looking at the two terms in equation (2), for the mutual information to be high, the left term has to be high and the right term low. The left term is the entropy of the model prediction, which is high when the model’s prediction is uncertain. The right term is an expectation of the entropy of the model prediction over the posterior of the model parameters and is low when the model is overall certain for each draw of model parameters from the posterior. Both can only happen when the model has many possible ways to explain the data, which means that the posterior draws are disagreeing among themselves.

BALD was originally intended for acquiring individual data points and immediately retraining the model. This becomes a bottleneck in deep learning, where retraining takes a substantial amount of time. Applications of BALD [11, 19] usually acquire the top b . This can be expressed as summing over individual scores:

$$a_{\text{BALD}}(\{\mathbf{x}_1, \dots, \mathbf{x}_b\}, p(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})) = \sum_{i=1}^b \mathbb{I}(y_i; \boldsymbol{\omega} \mid \mathbf{x}_i, \mathcal{D}_{\text{train}}), \quad (3)$$

and finding the optimal batch for this acquisition function using a greedy algorithm, which reduces to picking the top b highest-scoring data points.

2.3 Bayesian Neural Networks (BNN)

In this paper we focus on BNNs as our Bayesian model because they scale well to high dimensional inputs, such as images. Compared to regular neural networks, BNNs maintain a distribution over their weights instead of point estimates. Performing exact inference in BNNs is intractable for any reasonably sized model, so we resort to using a variational approximation. Similar to Gal et al. [12], we use MC dropout [11], which is easy to implement, scales well to large models and datasets, and is straightforward to optimise.

3 Methods

3.1 BatchBALD

We propose *BatchBALD* as an extension of BALD whereby we jointly score points by estimating the mutual information between a *joint of multiple data points* and the model parameters:³

$$a_{\text{BatchBALD}}(\{\mathbf{x}_1, \dots, \mathbf{x}_b\}, p(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})) = \mathbb{I}(y_1, \dots, y_b; \boldsymbol{\omega} \mid \mathbf{x}_1, \dots, \mathbf{x}_b, \mathcal{D}_{\text{train}}). \quad (4)$$

This builds on the insight that independent selection of a batch of data points leads to data inefficiency as correlations between data points in an acquisition batch are not taken into account.

To understand how to compute the mutual information between a set of points and the model parameters, we express $\mathbf{x}_1, \dots, \mathbf{x}_b$, and y_1, \dots, y_b through joint random variables $\mathbf{x}_{1:b}$ and $y_{1:b}$ in a product probability space and use the definition of the mutual information for two random variables:

$$\mathbb{I}(y_{1:b}; \boldsymbol{\omega} \mid \mathbf{x}_{1:b}, \mathcal{D}_{\text{train}}) = \mathbb{H}(y_{1:b} \mid \mathbf{x}_{1:b}, \mathcal{D}_{\text{train}}) - \mathbb{E}_{p(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})} \mathbb{H}(y_{1:b} \mid \mathbf{x}_{1:b}, \boldsymbol{\omega}, \mathcal{D}_{\text{train}}). \quad (5)$$

³ We use the notation $\mathbb{I}(x, y; z \mid c)$ to denote the mutual information between the *joint of the random variables* x, y and the random variable z conditioned on c .



$$\sum_i \mathbb{I}(y_i; \omega | \mathbf{x}_i, \mathcal{D}_{\text{train}}) = \sum_i \mu^*(y_i \cap \omega) \quad \mathbb{I}(y_1, \dots, y_b; \omega | \mathbf{x}_1, \dots, \mathbf{x}_b, \mathcal{D}_{\text{train}}) = \mu^*\left(\bigcup_i y_i \cap \omega\right)$$

(a) BALD

(b) BatchBALD

Figure 3: *Intuition behind BALD and BatchBALD using I-diagrams [35]. BALD overestimates the joint mutual information. BatchBALD, however, takes the overlap between variables into account and will strive to acquire a better cover of ω . Areas contributing to the respective score are shown in grey, and areas that are double-counted in dark grey.*

Algorithm 1: Greedy BatchBALD $1 - 1/e$ -approximate algorithm

Input: acquisition size b , unlabelled dataset $\mathcal{D}_{\text{pool}}$, model parameters $p(\omega | \mathcal{D}_{\text{train}})$

- 1 $A_0 \leftarrow \emptyset$
- 2 **for** $n \leftarrow 1$ **to** b **do**
- 3 **foreach** $\mathbf{x} \in \mathcal{D}_{\text{pool}} \setminus A_{n-1}$ **do** $s_{\mathbf{x}} \leftarrow a_{\text{BatchBALD}}(A_{n-1} \cup \{\mathbf{x}\}, p(\omega | \mathcal{D}_{\text{train}}))$
- 4 $\mathbf{x}_n \leftarrow \arg \max_{\mathbf{x} \in \mathcal{D}_{\text{pool}} \setminus A_{n-1}} s_{\mathbf{x}}$
- 5 $A_n \leftarrow A_{n-1} \cup \{\mathbf{x}_n\}$
- 6 **end**

Output: acquisition batch $A_n = \{\mathbf{x}_1, \dots, \mathbf{x}_b\}$

Intuitively, the mutual information between two random variables can be seen as the intersection of their information content. In fact, Yeung [35] shows that a signed measure μ^* can be defined for discrete random variables x, y , such that $\mathbb{I}(x; y) = \mu^*(x \cap y)$, $\mathbb{H}(x, y) = \mu^*(x \cup y)$, $\mathbb{E}_{p(y)} \mathbb{H}(x | y) = \mu^*(x \setminus y)$, and so on, where we identify random variables with their counterparts in information space, and conveniently drop conditioning on $\mathcal{D}_{\text{train}}$ and \mathbf{x}_i .

Using this, BALD can be viewed as the sum of individual intersections $\sum_i \mu^*(y_i \cap \omega)$, which double counts overlaps between the y_i . Naively extending BALD to the mutual information between y_1, \dots, y_b and ω , which is equivalent to $\mu^*(\bigcap_i y_i \cap \omega)$, would lead to selecting *similar* data points instead of diverse ones under maximisation.

BatchBALD, on the other hand, takes overlaps into account by computing $\mu^*(\bigcup_i y_i \cap \omega)$ and is more likely to acquire a more diverse cover under maximisation:

$$\begin{aligned} \mathbb{I}(y_1, \dots, y_b; \omega | \mathbf{x}_1, \dots, \mathbf{x}_b, \mathcal{D}_{\text{train}}) &= \mathbb{H}(y_{1:b} | \mathbf{x}_{1:b}, \mathcal{D}_{\text{train}}) - \mathbb{E}_{p(\omega | \mathcal{D}_{\text{train}})} \mathbb{H}(y_{1:b} | \mathbf{x}_{1:b}, \omega, \mathcal{D}_{\text{train}}) \quad (6) \\ &= \mu^*\left(\bigcup_i y_i\right) - \mu^*\left(\bigcup_i y_i \setminus \omega\right) = \mu^*\left(\bigcup_i y_i \cap \omega\right) \quad (7) \end{aligned}$$

This is depicted in figure 3 and also motivates that $a_{\text{BatchBALD}} \leq a_{\text{BALD}}$, which we prove in appendix B.1. For acquisition size 1, BatchBALD and BALD are equivalent.

3.2 Greedy approximation algorithm for BatchBALD

To avoid the combinatorial explosion that arises from jointly scoring subsets of points, we introduce a greedy approximation for computing BatchBALD, depicted in algorithm 1. In appendix A, we prove that $a_{\text{BatchBALD}}$ is submodular, which means the greedy algorithm is $1 - 1/e$ -approximate [24, 23].

In appendix B.2, we show that, under idealised conditions, when using BatchBALD and a fixed final $|\mathcal{D}_{\text{train}}|$, the active learning loop itself can be seen as a greedy $1 - 1/e$ -approximation algorithm, and that an active learning loop with BatchBALD and acquisition size larger than 1 is bounded by an active learning loop with individual acquisitions, that is BALD/BatchBALD with acquisition size 1, which is the ideal case.

3.3 Computing $a_{\text{BatchBALD}}$

For brevity, we leave out conditioning on $\mathbf{x}_1, \dots, \mathbf{x}_n$, and $\mathcal{D}_{\text{train}}$, and $\mathbf{p}(\boldsymbol{\omega})$ denotes $\mathbf{p}(\boldsymbol{\omega} | \mathcal{D}_{\text{train}})$ in this section. $a_{\text{BatchBALD}}$ is then written as:

$$a_{\text{BatchBALD}}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\}, \mathbf{p}(\boldsymbol{\omega})) = \mathbb{H}(y_1, \dots, y_n) - \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} [\mathbb{H}(y_1, \dots, y_n | \boldsymbol{\omega})]. \quad (8)$$

Because the y_i are independent when conditioned on $\boldsymbol{\omega}$, computing the right term of equation (8) is simplified as the conditional joint entropy decomposes into a sum. We can approximate the expectation using a Monte-Carlo estimator with k samples from our model parameter distribution $\hat{\boldsymbol{\omega}}_j \sim \mathbf{p}(\boldsymbol{\omega})$:

$$\mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} [\mathbb{H}(y_1, \dots, y_n | \boldsymbol{\omega})] = \sum_{i=1}^n \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} [\mathbb{H}(y_i | \boldsymbol{\omega})] \approx \frac{1}{k} \sum_{i=1}^n \sum_{j=1}^k \mathbb{H}(y_i | \hat{\boldsymbol{\omega}}_j). \quad (9)$$

Computing the left term of equation (8) is difficult because the unconditioned joint probability does not factorise. Applying the equality $\mathbf{p}(y) = \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} [\mathbf{p}(y | \boldsymbol{\omega})]$, and, using sampled $\hat{\boldsymbol{\omega}}_j$, we compute the entropy by summing over all possible configurations $\hat{y}_{1:n}$ of $y_{1:n}$:

$$\mathbb{H}(y_1, \dots, y_n) = \mathbb{E}_{\mathbf{p}(y_1, \dots, y_n)} [-\log \mathbf{p}(y_1, \dots, y_n)] \quad (10)$$

$$= \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_1, \dots, y_n | \boldsymbol{\omega})} [-\log \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} [\mathbf{p}(y_1, \dots, y_n | \boldsymbol{\omega})]] \quad (11)$$

$$\approx - \sum_{\hat{y}_{1:n}} \left(\frac{1}{k} \sum_{j=1}^k \mathbf{p}(\hat{y}_{1:n} | \hat{\boldsymbol{\omega}}_j) \right) \log \left(\frac{1}{k} \sum_{j=1}^k \mathbf{p}(\hat{y}_{1:n} | \hat{\boldsymbol{\omega}}_j) \right). \quad (12)$$

3.4 Efficient implementation

In each iteration of the algorithm, $\mathbf{x}_1, \dots, \mathbf{x}_{n-1}$ stay fixed while \mathbf{x}_n varies over $\mathcal{D}_{\text{pool}} \setminus A_{n-1}$. We can reduce the required computations by factorizing $\mathbf{p}(y_{1:n} | \boldsymbol{\omega})$ into $\mathbf{p}(y_{1:n-1} | \boldsymbol{\omega}) \mathbf{p}(y_n | \boldsymbol{\omega})$. We store $\mathbf{p}(\hat{y}_{1:n-1} | \hat{\boldsymbol{\omega}}_j)$ in a matrix $\hat{P}_{1:n-1}$ of shape $c^{n-1} \times k$ and $\mathbf{p}(y_n | \hat{\boldsymbol{\omega}}_j)$ in a matrix \hat{P}_n of shape $c \times k$. The sum $\sum_{j=1}^k \mathbf{p}(\hat{y}_{1:n} | \hat{\boldsymbol{\omega}}_j)$ in (12) can be then be turned into a matrix product:

$$\frac{1}{k} \sum_{j=1}^k \mathbf{p}(\hat{y}_{1:n} | \hat{\boldsymbol{\omega}}_j) = \frac{1}{k} \sum_{j=1}^k \mathbf{p}(\hat{y}_{1:n-1} | \hat{\boldsymbol{\omega}}_j) \mathbf{p}(y_n | \hat{\boldsymbol{\omega}}_j) = \left(\frac{1}{k} \hat{P}_{1:n-1} \hat{P}_n^T \right)_{\hat{y}_{1:n-1}, \hat{y}_n}. \quad (13)$$

This can be further sped up by using batch matrix multiplication to compute the joint entropy for different \mathbf{x}_n . $\hat{P}_{1:n-1}$ only has to be computed once, and we can recursively compute $\hat{P}_{1:n}$ using $\hat{P}_{1:n-1}$ and \hat{P}_n , which allows us to sample $\mathbf{p}(y | \hat{\boldsymbol{\omega}}_j)$ for each $\mathbf{x} \in \mathcal{D}_{\text{pool}}$ only once at the beginning of the algorithm.

For larger acquisition sizes, we use m MC samples of $y_{1:n-1}$ as enumerating all possible configurations becomes infeasible. See appendix C for details.

Monte-Carlo sampling bounds the time complexity of the full BatchBALD algorithm to $O(bc \cdot \min\{c^b, m\} \cdot |\mathcal{D}_{\text{pool}}| \cdot k)$ compared to $O(c^b \cdot |\mathcal{D}_{\text{pool}}|^b \cdot k)$ for naively finding the exact optimal batch and $O((b+k) \cdot |\mathcal{D}_{\text{pool}}|)$ for BALD⁴.

4 Experiments

In our experiments, we start by showing how a naive application of the BALD algorithm to an image dataset can lead to poor results in a dataset with many (near) duplicate data points, and show that BatchBALD solves this problem in a grounded way while obtaining favourable results (figure 2).

⁴ b is the acquisition size, c is the number of classes, k is the number of MC dropout samples, and m is the number of sampled configurations of $y_{1:n-1}$.

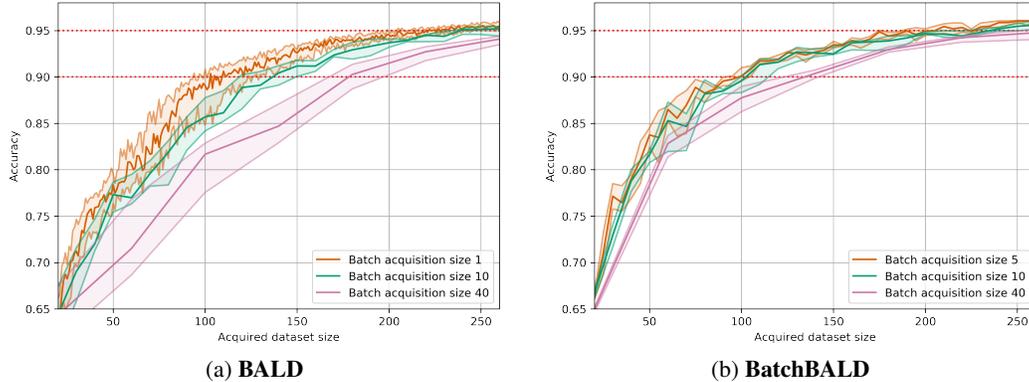


Figure 4: *Performance on MNIST for increasing acquisition sizes.* BALD’s performance drops drastically as the acquisition size increases. BatchBALD maintains strong performance even with increasing acquisition size.

We then illustrate BatchBALD’s effectiveness on standard AL datasets: MNIST and EMNIST. EMNIST [6] is an extension of MNIST that also includes letters, for a total of 47 classes, and has a twice as large training set. See appendix F for examples of the dataset. We show that BatchBALD provides a substantial performance improvement in these scenarios, too, and has more diverse acquisitions. Finally, we look at BatchBALD in the setting of transfer learning, where we finetune a large pretrained model on a more difficult dataset called CINIC-10 [8], which is a combination of CIFAR-10 and downsampled ImageNet.

In our experiments, we repeatedly go through active learning loops. One active learning loop consists of training the model on the available labelled data and subsequently acquiring new data points using a chosen acquisition function. As the labelled dataset is small in the beginning, it is important to avoid overfitting. We do this by using early stopping after 3 epochs of declining accuracy on the validation set. We pick the model with the highest validation accuracy. Throughout our experiments, we use the Adam [21] optimiser with learning rate 0.001 and betas 0.9/0.999. All our results report the median of 6 trials, with lower and upper quartiles. We use these quartiles to draw the filled error bars on our figures.

We reinitialize the model after each acquisition, similar to Gal et al. [12]. We found this helps the model improve even when very small batches are acquired. It also decorrelates subsequent acquisitions as final model performance is dependent on a particular initialization [9].

When computing $p(y|x, \omega, \mathcal{D}_{\text{train}})$, it is important to keep the dropout masks in MC dropout consistent while sampling from the model. This is necessary to capture dependencies between the inputs for BatchBALD, and it makes the scores for different points more comparable by removing this source of noise. We do not keep the masks fixed when computing BALD scores because its performance usually benefits from the added noise. We also do not need to keep these masks fixed for training and evaluating the model.

In all our experiments, we either compute joint entropies exactly by enumerating all configurations, or we estimate them using 10,000 MC samples, picking whichever method is faster. In practice, we compute joint entropies exactly for roughly the first 4 data points in an acquisition batch and use MC sampling thereafter.

4.1 Repeated MNIST

As demonstrated in the introduction, naively applying BALD to a dataset that contains many (near) replicated data points leads to poor performance. We show how this manifests in practice by taking the MNIST dataset and replicating each data point in the training set two times (obtaining a training set that is three times larger than the original). After normalising the dataset, we add isotropic Gaussian noise with a standard deviation of 0.1 to simulate slight differences between the duplicated data points in the training set. All results are obtained using an acquisition size of 10 and 10 MC

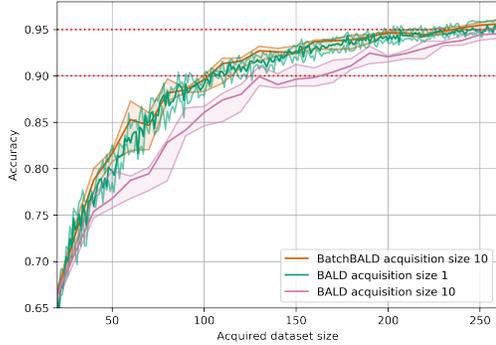


Figure 5: *Performance on MNIST*. BatchBALD outperforms BALD with acquisition size 10 and performs close to the optimum of acquisition size 1.

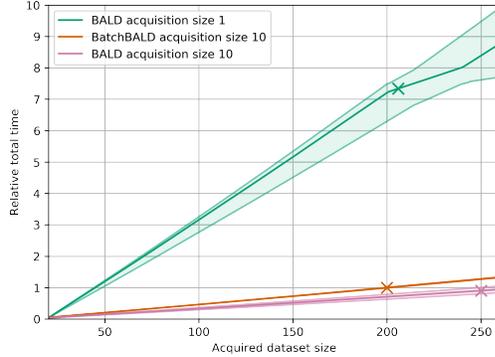


Figure 6: *Relative total time on MNIST*. Normalized to training BatchBALD with acquisition size 10 to 95% accuracy. The stars mark when 95% accuracy is reached for each method.

Table 1: *Number of required data points on MNIST until 90% and 95% accuracy are reached. 25%-, 50%- and 75%-quartiles for the number of required data points when available.*

	90% accuracy	95% accuracy
BatchBALD	70 / 90 / 110	190 / 200 / 230
BALD ⁶	120 / 120 / 170	250 / 250 / >300
BALD [12]	145	335

dropout samples. The initial dataset was constructed by taking a balanced set of 20 data points⁵, two of each class (similar to [12]).

Our model consists of two blocks of [convolution, dropout, max-pooling, relu], with 32 and 64 5x5 convolution filters. These blocks are followed by a two-layer MLP that includes dropout between the layers and has 128 and 10 hidden units. The dropout probability is 0.5 in all three locations. This architecture achieves 99% accuracy with 10 MC dropout samples during test time on the full MNIST dataset.

The results can be seen in figure 2. In this illustrative scenario, BALD performs poorly, and even randomly acquiring points performs better. However, BatchBALD is able to cope with the replication perfectly. In appendix D, we look at varying the repetition number and show that as we increase the number of repetitions BALD gradually performs worse. In appendix E, we also compare with Variation Ratios [10], and Mean STD [20] which perform on par with random acquisition.

4.2 MNIST

For the second experiment, we follow the setup of Gal et al. [12] and perform AL on the MNIST dataset using 100 MC dropout samples. We use the same model architecture and initial dataset as described in section 4.1. Due to differences in model architecture, hyper parameters and model retraining, we significantly outperform the original results in Gal et al. [12] as shown in table 1.

We first look at BALD for increasing acquisition size in figure 4a. As we increase the acquisition size from the ideal of acquiring points individually and fully retraining after each points (acquisition size 1) to 40, there is a substantial performance drop.

BatchBALD, in figure 4b, is able to maintain performance when doubling the acquisition size from 5 to 10. Performance drops only slightly at 40, possibly due to estimator noise.

The results for acquisition size 10 for both BALD and BatchBALD are compared in figure 5. BatchBALD outperforms BALD. Indeed, BatchBALD with acquisition size 10 performs close to the ideal with acquisition size 1. The total run time of training these three models until 95% accuracy is

⁵These initial data points were chosen by running BALD 6 times with the initial dataset picked randomly and choosing the set of the median model. They were subsequently held fixed.

⁶reimplementation using reported experimental setup

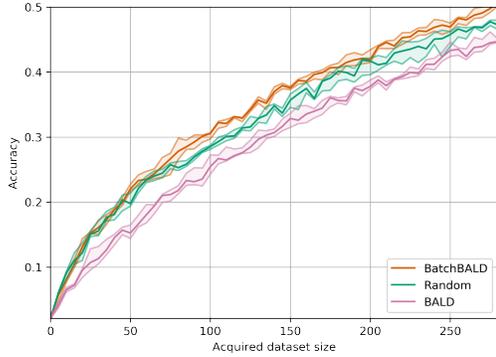


Figure 7: *Performance on EMNIST*. BatchBALD consistently outperforms both random acquisition and BALD while BALD is unable to beat random acquisition.

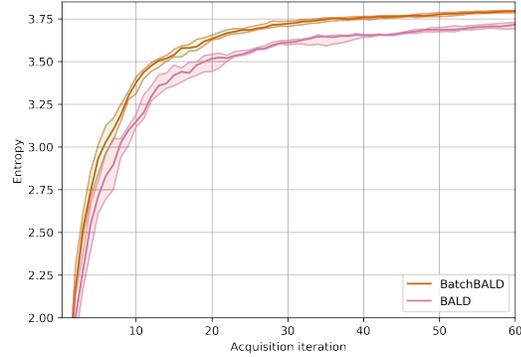


Figure 8: *Entropy of acquired class labels over acquisition steps on EMNIST*. BatchBALD steadily acquires a more diverse set of data points.

visualized in figure 6, where we see that BatchBALD with acquisition size 10 is much faster than BALD with acquisition size 1, and only marginally slower than BALD with acquisition size 10.

4.3 EMNIST

In this experiment, we show that BatchBALD also provides a significant improvement when we consider the more difficult EMNIST dataset [6] in the *Balanced* setup, which consists of 47 classes, comprising letters and digits. The training set consists of 112,800 28x28 images balanced by class, of which the last 18,800 images constitute the validation set. We do not use an initial dataset and instead perform the initial acquisition step with the randomly initialized model. We use 10 MC dropout samples.

We use a similar model architecture as before, but with added capacity. Three blocks of [convolution, dropout, max-pooling, relu], with 32, 64 and 128 3x3 convolution filters, and 2x2 max pooling. These blocks are followed by a two-layer MLP with 512 and 47 hidden units, with again a dropout layer in between. We use dropout probability 0.5 throughout the model.

The results for acquisition size 5 can be seen in figure 7. BatchBALD outperforms both random acquisition and BALD while BALD is unable to beat random acquisition. Figure 8 gives some insight into why BatchBALD performs better than BALD. The entropy of the categorical distribution of acquired class labels is consistently higher, meaning that BatchBALD acquires a more diverse set of data points. In figure 15, the classes on the x-axis are sorted by number of data points that were acquired of that class. We see that BALD undersamples classes while BatchBALD is more consistent.

4.4 CINIC-10

CINIC-10 is an interesting dataset because it is large (270k data points) and its data comes from two different sources: CIFAR-10 and ImageNet. To get strong performance on the test set it is important to obtain data from both sets. Instead of training a very deep model from scratch on a small dataset, we opt to run this experiment in a transfer learning setting, where we use a pretrained model and acquire data only to finetune the original model. This is common practice and suitable in cases where data is abundant for an auxiliary domain, but is expensive to label for the domain of interest.

For the CINIC-10 experiment, we use 160k training samples for the unlabelled pool, 20k validation samples, and the other 90k as test samples. We use an

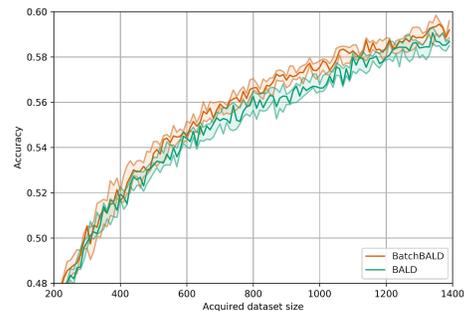


Figure 9: *Performance on CINIC-10*. BatchBALD outperforms BALD from 500 acquired samples onwards.

ImageNet pretrained VGG-16, provided by PyTorch [25], with a dropout layer before a 512 hidden unit (instead of 4096) fully connected layer. We use 50 MC dropout samples, acquisition size 10 and repeat the experiment for 6 trials. The results are in figure 9, with the 59% mark reached at 1170 for BatchBALD and 1330 for BALD (median).

5 Related work

AL is closely related to Bayesian Optimisation (BO), which is concerned with finding the global optimum of a function [32], with the fewest number of function evaluations. This is generally done using a Gaussian Process. A common problem in BO is the lack of parallelism, with usually a single worker being responsible for function evaluations. In real-world settings, there are usually many such workers available and making optimal use of them is an open problem [13, 2] with some work exploring mutual information for optimising a multi-objective problem [16].

Maintaining diversity when acquiring a batch of data has also been attempted using constrained optimisation [14] and in Gaussian Mixture Models [3]. In AL of molecular data, the lack of diversity in batches of data points acquired using the BALD objective has been noted by Janz et al. [19], who propose to resolve it by limiting the number of MC dropout samples and relying on noisy estimates.

A related approach to AL is semi-supervised learning (also sometimes referred to as weakly-supervised), in which the labelled data is commonly assumed to be fixed and the unlabelled data is used for unsupervised learning [22, 26]. Wang et al. [34], Sener and Savarese [28], Samarth Sinha [27] explore combining it with AL.

6 Scope and limitations

Unbalanced datasets BALD and BatchBALD do not work well when the test set is unbalanced as they aim to learn well about all classes and do not follow the density of the dataset. However, if the test set is balanced, but the training set is not, we expect BatchBALD to perform well.

Unlabelled data BatchBALD does not take into account any information from the unlabelled dataset. However, BatchBALD uses the underlying Bayesian model for estimating uncertainty for unlabelled data points, and semi-supervised learning could improve these estimates by providing more information about the underlying structure of the feature space. We leave a semi-supervised extension of BatchBALD to future work.

Noisy estimator A significant amount of noise is introduced by MC-dropout’s variational approximation to training BNNs. Sampling of the joint entropies introduces additional noise. The quality of larger acquisition batches would be improved by reducing this noise.

7 Conclusion

We have introduced a new batch acquisition function, BatchBALD, for Deep Bayesian Active Learning, and a greedy algorithm that selects good candidate batches compared to the intractable optimal solution. Acquisitions show increased diversity of data points and improved performance over BALD and other methods.

While our method comes with additional computational cost during acquisition, BatchBALD is able to significantly reduce the number of data points that need to be labelled and the number of times the model has to be retrained, potentially saving considerable costs and filling an important gap in practical Deep Bayesian Active Learning.

Acknowledgements

The authors want to thank Binxin (Robin) Ru for helpful references to submodularity and the appropriate proofs. We would also like to thank the rest of OATML for their feedback at several stages of the project. AK is supported by the UK EPSRC CDT in Autonomous Intelligent Machines and Systems (grant reference EP/L015897/1). JvA is grateful for funding by the EPSRC (grant reference EP/N509711/1) and Google-DeepMind. Funding for computational resources was provided by the Allan Turing Institute and Google.

Author contributions

AK derived the original estimator, proved submodularity and bounds, implemented BatchBALD efficiently, and ran the experiments. JvA developed the narrative and experimental design, advised on debugging, structured the paper into its current form, and pushed it forward at difficult times. JvA and AK wrote the paper jointly.

References

- [1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, 2005.
- [2] Ahsan S Alvi, Binxin Ru, Jan Calliess, Stephen J Roberts, and Michael A Osborne. Asynchronous batch Bayesian optimisation with improved local penalisation. *arXiv preprint arXiv:1901.10452*, 2019.
- [3] Javad Azimi, Alan Fern, Xiaoli Zhang-Fern, Glencora Borradaile, and Brent Heeringa. Batch active learning via coordinated matching. *arXiv preprint arXiv:1206.6458*, 2012.
- [4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 1613–1622, 2015.
- [5] Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.
- [6] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.
- [7] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.
- [8] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.
- [9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.
- [10] Linton C Freeman. *Elementary applied statistics: for students in behavioral science*. John Wiley & Sons, 1965.
- [11] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.
- [12] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.
- [13] Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch Bayesian optimization via local penalization. In *Artificial Intelligence and Statistics*, pages 648–657, 2016.
- [14] Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In *Advances in neural information processing systems*, pages 593–600, 2008.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [16] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, pages 1492–1501, 2016.
- [17] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd international conference on Machine learning*, pages 417–424. ACM, 2006.
- [18] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

- [19] David Janz, Jos van der Westhuizen, and José Miguel Hernández-Lobato. Actively learning what makes a discrete sequence valid. *arXiv preprint arXiv:1708.04465*, 2017.
- [20] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.
- [21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [22] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.
- [23] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb):235–284, 2008.
- [24] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.
- [25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.
- [26] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in neural information processing systems*, pages 3546–3554, 2015.
- [27] Trevor Darrell Samarth Sinha, Sayna Ebrahimi. Variational adversarial active learning. *arXiv preprint arXiv:1904.00370*, 2019.
- [28] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.
- [29] Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, and Animashree Anandkumar. Deep active learning for named entity recognition. In *International Conference on Learning Representations*, 2018.
- [30] Aditya Siddhant and Zachary C Lipton. Deep Bayesian active learning for natural language processing: Results of a large-scale empirical study. *arXiv preprint arXiv:1808.05697*, 2018.
- [31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- [32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.
- [33] Simon Tong. *Active learning: theory and applications*, volume 1. Stanford University USA, 2001.
- [34] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2017.
- [35] Raymond W Yeung. A new outlook on shannon’s information measures. *IEEE transactions on information theory*, 37(3):466–474, 1991.

A Proof of submodularity

Nemhauser et al. [24] show that if a function is submodular, then a greedy algorithm like algorithm 1 is $1 - 1/e$ -approximate. Here, we show that $a_{\text{BatchBALD}}$ is submodular.

We will show that $a_{\text{BatchBALD}}$ satisfies the following equivalent definition of submodularity:

Definition A.1. A function f defined on subsets of Ω is called submodular if for every set $A \subset \Omega$ and two non-identical points $y_1, y_2 \in \Omega \setminus A$:

$$f(A \cup \{y_1\}) + f(A \cup \{y_2\}) \geq f(A \cup \{y_1, y_2\}) + f(A) \quad (14)$$

Submodularity expresses that there are "diminishing returns" for adding additional points to f .

Lemma A.2. $a_{\text{BatchBALD}}(A, p(\boldsymbol{\omega})) := \mathbb{I}(A; \boldsymbol{\omega})$ is submodular for $A \subset \mathcal{D}_{\text{pool}}$.

Proof. Let $y_1, y_2 \in \mathcal{D}_{\text{pool}}, y_1 \neq y_2$. We start by substituting the definition of $a_{\text{BatchBALD}}$ into (14) and subtracting $\mathbb{I}(A; \boldsymbol{\omega})$ twice on both sides, using that $\mathbb{I}(A \cup B; \boldsymbol{\omega}) - \mathbb{I}(B; \boldsymbol{\omega}) = \mathbb{I}(A; \boldsymbol{\omega} | B)$:

$$\mathbb{I}(A \cup \{y\}; \boldsymbol{\omega}) + \mathbb{I}(A \cup \{x\}; \boldsymbol{\omega}) \geq \mathbb{I}(A \cup \{x, y\}; \boldsymbol{\omega}) + \mathbb{I}(A; \boldsymbol{\omega}) \quad (15)$$

$$\Leftrightarrow \mathbb{I}(y; \boldsymbol{\omega} | A) + \mathbb{I}(x; \boldsymbol{\omega} | A) \geq \mathbb{I}(x, y; \boldsymbol{\omega} | A). \quad (16)$$

We rewrite the left-hand side using the definition of the mutual information $\mathbb{I}(A; B) = \mathbb{H}(A) - \mathbb{H}(A | B)$ and reorder:

$$\mathbb{I}(y; \boldsymbol{\omega} | A) + \mathbb{I}(x; \boldsymbol{\omega} | A) \quad (17)$$

$$= \underbrace{\mathbb{H}(y_1 | A) + \mathbb{H}(y_2 | A)}_{\geq \mathbb{H}(y_1, y_2 | A)} - \underbrace{(\mathbb{H}(y_1 | A, \boldsymbol{\omega}) + \mathbb{H}(y_2 | A, \boldsymbol{\omega}))}_{= \mathbb{H}(y_1, y_2 | A, \boldsymbol{\omega})} \quad (18)$$

$$\geq \mathbb{H}(y_1, y_2 | A) - \mathbb{H}(y_1, y_2 | A, \boldsymbol{\omega}) \quad (19)$$

$$= \mathbb{I}(x, y; \boldsymbol{\omega} | A), \quad (20)$$

where we have used that entropies are subadditive in general and additive given $y_1 \perp\!\!\!\perp y_2 | \boldsymbol{\omega}$. \square

Following Nemhauser et al. [24], we can conclude that algorithm 1 is $1 - 1/e$ -approximate.

B Connection between BatchBALD and BALD

In the following section, we show that BALD approximates BatchBALD and that BatchBALD approximates BALD with acquisition size 1. The BALD score is an upper bound of the BatchBALD score for any candidate batch. At the same time, BatchBALD can be seen as performing BALD with acquisition size 1 during each step of its greedy algorithm in an idealised setting.

B.1 BALD as an approximation of BatchBALD

Using the subadditivity of information entropy and the independence of the y_i given $\boldsymbol{\omega}$, we show that BALD is an approximation of BatchBALD and is always an upper bound on the respective BatchBALD score:

$$a_{\text{BatchBALD}}(\{\mathbf{x}_1, \dots, \mathbf{x}_b\}, p(\boldsymbol{\omega} | \mathcal{D}_{\text{train}})) \quad (21)$$

$$= \mathbb{H}(y_1, \dots, y_b | \mathbf{x}_1, \dots, \mathbf{x}_b, \mathcal{D}_{\text{train}}) - \mathbb{E}_{p(\boldsymbol{\omega} | \mathcal{D}_{\text{train}})} [\mathbb{H}(y_1, \dots, y_b | \mathbf{x}_1, \dots, \mathbf{x}_b, \boldsymbol{\omega}, \mathcal{D}_{\text{train}})] \quad (22)$$

$$\leq \sum_{i=1}^b \mathbb{H}(y_i | \mathbf{x}_i, \mathcal{D}_{\text{train}}) - \sum_{i=1}^b \mathbb{E}_{p(\boldsymbol{\omega} | \mathcal{D}_{\text{train}})} [\mathbb{H}(y_i | \mathbf{x}_i, \boldsymbol{\omega}, \mathcal{D}_{\text{train}})] \quad (23)$$

$$= \sum_{i=1}^b \mathbb{I}(y_i; \boldsymbol{\omega} | \mathbf{x}_i, \mathcal{D}_{\text{train}}) = a_{\text{BALD}}(\{\mathbf{x}_1, \dots, \mathbf{x}_b\}, p(\boldsymbol{\omega} | \mathcal{D}_{\text{train}})) \quad (24)$$

B.2 BatchBALD as an approximation of BALD with acquisition size 1

To see why BALD with acquisition size 1 can be seen as an upper bound for BatchBALD performance in an idealised setting, we reformulate line 3 in algorithm 1 on page 4.

Instead of the original term $a_{\text{BatchBALD}}(A_{n-1} \cup \{\mathbf{x}\}, \mathbf{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}}))$, we can equivalently maximise

$$a_{\text{BatchBALD}}(A_{n-1} \cup \{\mathbf{x}\}, \mathbf{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})) - a_{\text{BatchBALD}}(A_{n-1}, \mathbf{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}})) \quad (25)$$

as the right term is constant for all $\mathbf{x} \in \mathcal{D}_{\text{pool}} \setminus A_{n-1}$ within the inner loop, which, in turn, is equivalent to

$$= \mathbb{I}(y_1, \dots, y_{n-1}, y; \boldsymbol{\omega} \mid \mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{x}, \mathcal{D}_{\text{train}}) - \mathbb{I}(y_1, \dots, y_{n-1}; \boldsymbol{\omega} \mid \mathbf{x}_{1:n-1}, \mathcal{D}_{\text{train}}) \quad (26)$$

$$= \mathbb{I}(y; \boldsymbol{\omega} \mid \mathbf{x}, y_1, \dots, y_{n-1}, \mathbf{x}_{1:n-1}, \mathcal{D}_{\text{train}}) \quad (27)$$

once we expand $A_{n-1} = \{\mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$. This means that, at each step of the inner loop, our greedy algorithm is maximising the mutual information of the individual available data points with the model parameters conditioned on all the additional data points that have already been picked for acquisition and the existing training set. Finally, assuming training our model captures all available information,

$$\geq \mathbb{I}(y; \boldsymbol{\omega} \mid \mathbf{x}, \mathcal{D}_{\text{train}} \cup \{(\mathbf{x}_1, \tilde{y}_1), \dots, (\mathbf{x}_{n-1}, \tilde{y}_{n-1})\}) \quad (28)$$

$$= a_{\text{BALD}}(\{\mathbf{x}\}, \mathbf{p}(\boldsymbol{\omega} \mid \mathcal{D}_{\text{train}} \cup \{(\mathbf{x}_1, \tilde{y}_1), \dots, (\mathbf{x}_{n-1}, \tilde{y}_{n-1})\})), \quad (29)$$

where $\tilde{y}_1, \dots, \tilde{y}_{n-1}$ are the actual labels of $\mathbf{x}_1, \dots, \mathbf{x}_n$. The mutual information decreases as $\boldsymbol{\omega}$ becomes more concentrated as we expand its training set, and thus the overlap of y and $\boldsymbol{\omega}$ will become smaller (in an information-measure-theoretical sense).

This shows that every step n of the inner loop in our algorithm is at most as good as retraining our model on the new training set $\mathcal{D}_{\text{train}} \cup \{(\mathbf{x}_1, \tilde{y}_1), \dots, (\mathbf{x}_{n-1}, \tilde{y}_{n-1})\}$ and picking x_n using a_{BALD} with acquisition size 1.

Relevance for the active training loop. We see that the active training loop as a whole is computing a greedy $1 - 1/e$ -approximation of the mutual information of all acquired data points over all acquisitions with the model parameters.

C Sampling of configurations

We are using the same notation as in section 3.3. We factor $\mathbf{p}(y_{1:n} \mid \boldsymbol{\omega})$ to avoid recomputations and rewrite $\mathbb{H}(y_{1:n})$ as:

$$\mathbb{H}(y_{1:n}) = \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_{1:n} \mid \boldsymbol{\omega})} [-\log \mathbf{p}(y_{1:n})] \quad (30)$$

$$= \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_{1:n-1} \mid \boldsymbol{\omega}) \mathbf{p}(y_n \mid \boldsymbol{\omega})} [-\log \mathbf{p}(y_{1:n})] \quad (31)$$

$$= \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_{1:n-1} \mid \boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_n \mid \boldsymbol{\omega})} [-\log \mathbf{p}(y_{1:n})] \quad (32)$$

To be flexible in the way we sample $y_{1:n-1}$, we perform importance sampling of $\mathbf{p}(y_{1:n-1} \mid \boldsymbol{\omega})$ using $\mathbf{p}(y_{1:n-1})$, and, assuming we also have m samples $\hat{y}_{1:n-1}$ from $\mathbf{p}(y_{1:n-1})$, we can approximate:

$$\mathbb{H}(y_{1:n}) = \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_{1:n-1})} \left[\frac{\mathbf{p}(y_{1:n-1} \mid \boldsymbol{\omega})}{\mathbf{p}(y_{1:n-1})} \mathbb{E}_{\mathbf{p}(y_n \mid \boldsymbol{\omega})} [-\log \mathbf{p}(y_{1:n})] \right] \quad (33)$$

$$= \mathbb{E}_{\mathbf{p}(y_{1:n-1})} \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} \mathbb{E}_{\mathbf{p}(y_n \mid \boldsymbol{\omega})} \left[-\frac{\mathbf{p}(y_{1:n-1} \mid \boldsymbol{\omega})}{\mathbf{p}(y_{1:n-1})} \log \mathbb{E}_{\mathbf{p}(\boldsymbol{\omega})} [\mathbf{p}(y_{1:n-1} \mid \boldsymbol{\omega}) \mathbf{p}(y_{1:n} \mid \boldsymbol{\omega})] \right] \quad (34)$$

$$\approx -\frac{1}{m} \sum_{\hat{y}_{1:n-1}} \sum_{\hat{y}_n} \frac{\frac{1}{k} \sum_{\hat{\boldsymbol{\omega}}_j} \mathbf{p}(\hat{y}_{1:n-1} \mid \hat{\boldsymbol{\omega}}_j) \mathbf{p}(\hat{y}_n \mid \hat{\boldsymbol{\omega}}_j)}{\mathbf{p}(\hat{y}_{1:n-1})} \log \left(\frac{1}{k} \sum_{\hat{\boldsymbol{\omega}}_j} \mathbf{p}(\hat{y}_{1:n-1} \mid \hat{\boldsymbol{\omega}}_j) \mathbf{p}(\hat{y}_n \mid \hat{\boldsymbol{\omega}}_j) \right) \quad (35)$$

$$= -\frac{1}{m} \sum_{\hat{y}_{1:n-1}} \sum_{\hat{y}_n} \frac{\left(\hat{\mathbf{P}}_{1:n-1} \hat{\mathbf{P}}_n^T \right)_{\hat{y}_{1:n-1}, \hat{y}_n}}{\left(\hat{\mathbf{P}}_{1:n-1} \mathbb{1}_{k,1} \right)_{\hat{y}_{1:n-1}}} \log \left(\frac{1}{k} \left(\hat{\mathbf{P}}_{1:n-1} \hat{\mathbf{P}}_n^T \right)_{\hat{y}_{1:n-1}, \hat{y}_n} \right), \quad (36)$$

where we store $\mathbf{p}(\hat{y}_{1:n-1} \mid \hat{\boldsymbol{\omega}}_j)$ in a matrix $\hat{\mathbf{P}}_{1:n-1}$ of shape $m \times k$ and $\mathbf{p}(\hat{y}_n \mid \hat{\boldsymbol{\omega}}_j)$ in a matrix $\hat{\mathbf{P}}_n$ of shape $c \times k$ and $\mathbb{1}_{k,1}$ is a $k \times 1$ matrix of 1s. Equation (36) allows us to cache $\hat{\mathbf{P}}_{1:n-1}$ inside the inner loop of algorithm 1 and use batch matrix multiplication for efficient computation.

D Ablation study on Repeated MNIST

To better understand the effect of redundant data points on BALD and BatchBALD, we run the RMNIST experiment with an increasing number of repetitions. The results can be seen in figure 10. We use the same setup as in section 4.1. BatchBALD performs the same on all repetition numbers (100 data points till 90%). BALD achieves 90% accuracy at 120 data points (0 repetitions), 160 data points (1 repetition), 280 data points (2 repetitions), 300 data points (4 repetitions). This shows that BALD and BatchBALD behave as expected.

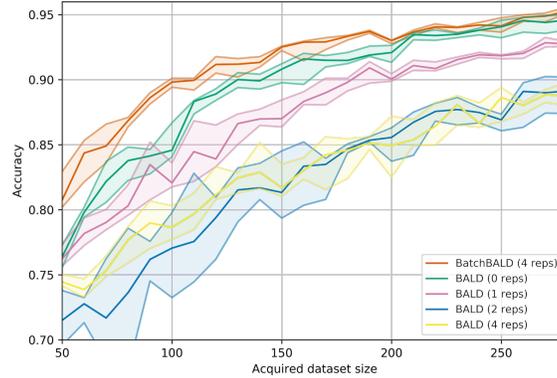


Figure 10: *Performance of BALD on Repeated MNIST for increasing amount of repetitions.* We see that BALD performs worse as the number of repetitions is increased, while BatchBALD outperforms BALD with zero repetitions.

E Additional results for Repeated MNIST

We show that BatchBALD also outperforms Var Ratios [10] and Mean STD [20].

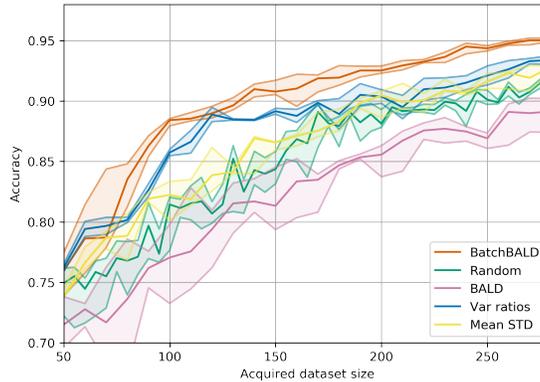


Figure 11: *Performance on Repeated MNIST.* BALD, BatchBALD, Var Ratios, Mean STD and random acquisition with acquisition size 10 and 10 MC dropout samples.

F Example visualisation of EMNIST



Figure 12: Examples of all 47 classes of EMNIST

G Entropy and class acquisitions including random acquisition

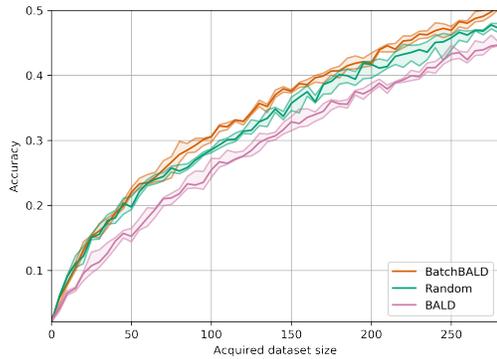


Figure 13: Performance on EMNIST. BatchBALD consistently outperforms both random acquisition and BALD while BALD is unable to beat random acquisition.

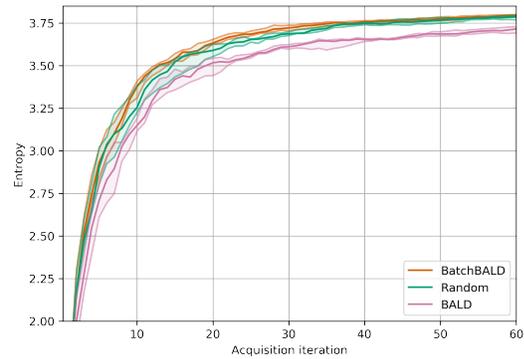


Figure 14: Entropy of acquired class labels over acquisition steps on EMNIST. BatchBALD steadily acquires a more diverse set of data points than BALD.

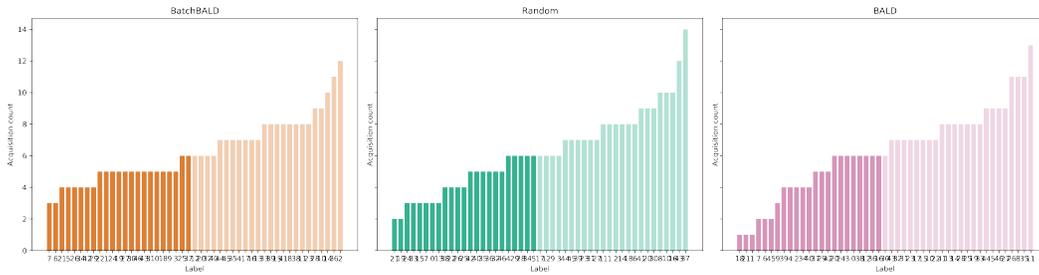


Figure 15: Histogram of acquired class labels on EMNIST. BatchBALD left and BALD right. Classes are sorted by number of acquisitions. Several EMNIST classes are underrepresented in BALD and random acquisition while BatchBALD acquires classes more uniformly. The histograms were created from all acquired points at the end of an active learning loop