DEEP BATCH ACTIVE LEARNING BY DIVERSE, UNCERTAIN GRADIENT LOWER BOUNDS

Anonymous authors
Paper under double-blind review

ABSTRACT

We design a new algorithm for batch active learning with deep neural network models. Our algorithm, Batch Active learning by Diverse Gradient Embeddings (BADGE), samples groups of points that are disparate and high-magnitude when represented in a hallucinated gradient space, a strategy designed to incorporate both predictive uncertainty and sample diversity into every selected batch. Crucially, BADGE trades off between diversity and uncertainty without requiring any hand-tuned hyperparameters. While other approaches sometimes succeed for particular batch sizes or architectures, BADGE consistently performs as well or better, making it a useful option for real world active learning problems.

1 INTRODUCTION

Deep neural networks represent the state-of-the-art supervised learning models to-date, but, as these models are quite data-hungry, their successes have been limited to domains where large amounts of labeled data are available. A promising approach for minimizing labeling effort is *active learning*, a supervised learning protocol where labels can be requested by the algorithm in a sequential feedback-driven fashion. Active learning algorithms aim to identify and label only the maximally informative samples, so that a high-performing classifier can be trained with minimal labeling effort. As such, a robust active learning algorithm for deep neural networks may considerably expand the domains where these models are applicable.

How should a practical, general-purpose, label-efficient active learning algorithm for deep neural networks be designed? Theory for active learning suggests a version-space-based approach (Balcan et al., 2006), but when using highly-expressive models, these algorithms degenerate to querying every example. Further, the computational overhead of training deep neural networks preclude algorithms that update the model to best fit the data after each label query, as is often done (exactly or approximately) for linear methods (Beygelzimer et al., 2010; Cesa-Bianchi et al., 2009). Unfortunately, the theory provides little guidance for these models.

One option is to use the network's uncertainty to inform a query strategy, for example by labeling samples for which the model is least confident. However, in a batch setting this creates a pathological scenario where data in the batch are nearly identical, a clear inefficiency. Remedying this issue, we could select samples to maximize batch diversity, but this might choose points that provide little useful new information to the model.

For these reasons, methods that exploit just uncertainty or diversity do not consistently work well across model architectures, batch sizes, or datasets. An algorithm that performs well when using a ResNet, for example, might perform poorly when using a multilayer-perceptron. A diversity-based approach might work well when the batch size is very large, but poorly when the batch size is small. Even what in practice constitutes a "big" or "small" batch size is largely a function of the statistical properties of the data in question. These weaknesses pose a major problem for real, practical batch active learning situations, where data are unfamiliar and potentially unstructured. There is no way to know which active learning algorithm is best to use.

Further, in a real active learning scenario, every change of hyperparameters typically causes the algorithm to label examples not chosen under other hyperparameters, provoking substantial labeling inefficiency. That is, hyperparameter sweeps in active learning can be label expenive. Because of this, active learning algorithms need to "just work", given fixed hyperparameters, to a greater extent than is typical for supervised learning.

Based on these observations, we design an approach which creates diverse batches of examples about which the current model is uncertain. We measure uncertainty as the gradient magnitude with respect to parameters in the final (output) layer, which is computed using the most likely label according to the model. To capture diversity, we collect a batch of examples where these gradients span a diverse set of directions. More specifically, we build up the batch of query points based on these hallucinated gradients using the k-MEANS++ initialization (Arthur and Vassilvitskii, 2007), which simultaneously captures both the magnitude of a candidate gradient and its distance from previously included points in the batch. We name the resulting approach Batch Active learning by Diverse Gradient Embeddings (BADGE).

We show that BADGE is robust to architecture choice, batch size, and dataset, generally performing as well as or better than the best baseline across our experiments, which vary all of the aforementioned environmental conditions. We begin by introducing our notation and setting, followed by a description of the BADGE algorithm in Section 3, and experiments in Section 4. We defer discussion of related work to Section 5.

2 NOTATION AND SETTING

Define $[K] := \{1, 2, \ldots, K\}$. Denote by $\mathcal{X}$ the instance space, and $\mathcal{Y}$ the label space. In this work we consider multiclass classification, so that $\mathcal{Y} = [K]$. Denote by D the distribution from which examples are drawn, $D_{\mathcal{X}}$ the unlabeled data distribution, and $D_{\mathcal{Y}|\mathcal{X}}$ the conditional distribution over labels given examples. We consider the pool-based active learning setup, where the learner receives an unlabeled dataset U sampled according to $D_{\mathcal{X}}$, and can request labels sampled according to $D_{\mathcal{Y}|\mathcal{X}}$ for any $x \in U$. We use $\mathbb{E}_D$ to denote expectation under the data distribution D. Given a classifier $h : \mathcal{X} \to \mathcal{Y}$, which maps examples to labels, and a labeled example (x, y), we denote the $0/1$ error of h on (x, y) as $\ell_{01}(h(x), y) = I(h(x) \neq y)$. The performance of a classifier h is measured by its expected $0/1$ error, i.e. $\mathbb{E}_D[\ell_{01}(h(x), y)] = \Pr_{(x,y)\sim D}(h(x) \neq y)$. The goal of pool-based active learning is to find a classifier with a small expected $0/1$ error, using as few label queries as possible. Given a set S of labeled examples (x, y), where each $x \in S$ is picked from U, followed by a label query, we use $\mathbb{E}_S$ as the sample averages over S.

In this paper, we consider classifiers h parameterized by underlying neural networks f of fixed architecture, with the weights in the network denoted by θ. We abbreviate the classifier with parameters θ as h_θ since the architectures are fixed in any given context, and our classifiers take the form $h_\theta(x) = \operatorname{argmax}_{y\in[K]} f(x; \theta)_y$, where $f(x; \theta) \in \mathbb{R}^K$ is a vector of scores assigned to candidate labels, given the example x and parameters θ. We optimize the parameters by minimizing the cross-entropy loss $\mathbb{E}_S[\ell_{\mathrm{CE}}(f(x; \theta), y)]$ over the labeled examples, where $\ell_{\mathrm{CE}}(p, y) = \sum_{i=1}^{K} I(y = i) \ln \frac{1}{p_i}$.

3 ALGORITHM

BADGE, described in Algorithm 1, starts by drawing an initial set of M examples uniformly at random from U and asking for their labels. It then proceeds iteratively, performing two main computations at each step t: a *gradient embedding computation* and a *sampling step*. Specifically, at each step t, for every x in the pool U, we compute the label $\hat{y}(x)$ preferred by the current model, and the gradient g_x of the loss on $(x, \hat{y}(x))$ with respect to the parameters of the last layer of the network. Given these *gradient embedding* vectors $\{g_x : x \in U\}$, BADGE selects a set of points by sampling via the k-MEANS++ initialization scheme (Arthur and Vassilvitskii, 2007). The algorithm queries for the labels of these examples, retrains the model and proceeds to the next iteration.

We now describe the main computations — the embedding and sampling steps — in more detail.

Algorithm 1 BADGE: Batch Active learning by Diverse Gradient Embeddings

Require: Neural network $f(x; \theta)$, unlabeled pool of examples U, initial number of examples M, number of iterations T, number of examples in a batch B.

Labeled dataset $S \leftarrow M$ examples drawn uniformly at random from U together with queried labels.
Train an initial model θ_1 on S by minimizing $\mathbb{E}_S[\ell(f(x; \theta), y)]$.
for $t = 1, 2, \ldots, T$: **do**
  For all examples x in $U \setminus S$:
    1. Compute its hypothetical label $\hat{y}(x) = h_{\theta_t}(x)$.
    2. Compute gradient embedding $g_x = \frac{\partial}{\partial \theta_{\text{out}}} \ell(f(x; \theta), \hat{y}(x))|_{\theta=\theta_t}$, where θ_{out} refers to parameters of the final (output) layer.
  Compute S_t, a random subset of $U \setminus S$, using the k-MEANS++ seeding algorithm, on $\{g_x : x \in U \setminus S\}$, and query for their labels.
  $S \leftarrow S \cup S_t$.
  Train a model θ_{t+1} on S by minimizing $\mathbb{E}_S[\ell_{\text{CE}}(f(x; \theta), y)]$.
end for
return Final model θ_{T+1}.

The gradient embedding. Since deep neural networks are optimized using gradient-based methods, we capture uncertainty about an example through the lens of gradients. In particular, we consider the model uncertain about an example if knowing the label induces a large gradient of the loss with respect to the model parameter, and hence a large update to the model. A difficulty with this reasoning is that we need to know the label to compute the gradient. As a proxy, we compute the gradient as if the model's current prediction on the example is the true label. We show in Proposition 1 that, assuming a common structure satisfied by most natural loss functions, the gradient norm with respect to the last layer using this label provides a lower bound on the gradient norm induced by any other label. In addition, under that assumption, the length of this hypothetical gradient vector captures the uncertainty of the model on the example: if the model is highly certain about the example's label, then the example's gradient embedding will have a small norm (see example below). Thus, the gradient embedding conveys information both about the model's uncertainty and potential update direction upon receiving a label at an example.

The sampling step. We want the newly-acquired labeled dataset to induce large and diverse changes to the model. To this end, we want the selection procedure to favor both sample magnitude and batch diversity. Specifically, we want to avoid the pathology of, for example, selecting a batch of k similar samples where even just a single single label could alleviate our uncertainty on all remaining samples.

A natural way of making this selection without introducing additional hyperparameters is to sample from a k-DPP (Kulesza and Taskar, 2011). That is, to select a batch of k points with a probability proportional to the determinant of their Gram matrix. In this process, when the batch size is very low, the selection will naturally favor points with a large length, which corresponds to uncertainty in our space. When the batch size is large, the sampler chooses points that are diverse because linear dependence makes the Gram determinant equal to zero.

Unfortunately, sampling from a k-DPP is not trivial. Many sampling algorithms (Kang, 2013; Anari et al., 2016) rely on MCMC, where mixing time poses a significant computational hurdle. The state-of-the-art algorithm of (Dereziński, 2018) has a high-order polynomial running time in the batch size and the embedding dimension. To overcome this computational hurdle, we suggest instead sampling using the k-MEANS++ seeding algorithm (Arthur and Vassilvitskii, 2007), originally made to produce a good initialization for k-means clustering. k-MEANS++ seeding selects centroids by sampling points in proportion to their squared distances from the nearest centroid that has already been chosen. For completeness, we give a formal description of the k-MEANS++ seeding algorithm in Appendix A.

This simple sampler tends to produce diverse points similar to k-DPP. As shown in Figure 1, switching between the two samplers does not affect the active learner's statistical performance while greatly improving the computational performance. A thorough comparison on the running times and test accuracies of BADGE and k-DPP based gradient embedding sampling can be found in Appendix G.

Example: multiclass classification with softmax activation. Consider a neural network f where the last nonlinearity is a softmax, i.e. $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$. Specifically, f is parametrized by $\theta = (W, V)$, where $\theta_{\text{out}} = W = (W_1^\top, \ldots, W_K^\top)^\top \in \mathbb{R}^{K \times d}$ are the weights of the last layer, and V consists of weights of all previous layers. This means that $f(x; \theta) = \sigma(W \cdot g(x; V))$, where g is the nonlinear function that maps an input x to the output of the network's penultimate layer. Let us fix an unlabeled sample x and define $p_i = f(x; \theta)_i$. With this notation, we have

$$\ell_{\text{CE}}(f(x; \theta), y) = \ln(\sum_{j=1}^{K} e^{W_j \cdot g(x;V)}) - W_y \cdot g(x; V).$$

Define $g_x^y = \frac{\partial}{\partial W} \ell_{\text{CE}}(f(x; \theta), y)$ for a label y and $g_x = g_x^{\hat{y}}$ as the gradient embedding in our algorithm, where $\hat{y} = \text{argmax}_{i \in [K]} p_i$. Then the i-th block of g_x is equal to

$$(g_x)_i = \frac{\partial}{\partial W_i} \ell_{\text{CE}}(f(x; \theta), \hat{y}) = (p_i - I(\hat{y} = i)) g(x; V), \tag{1}$$

Based on this expression, we have the following observations:

1. Each block of g_x is a scaling of $g(x; V)$, the output of the network's penultimate layer. In this respect, g_x captures x's representation information similar to that of (Sener and Savarese, 2018).
2. Proposition 1 in Appendix B shows that the norm of g_x is a lower bound on the norm of the loss gradient induced by the example with true label y with respect to the weights in the last layer, that is $\|g_x\| \leq \|g_x^y\|$. This suggests that the norm of g_x conservatively estimates the example's influence on the current model.
3. If the current model θ is highly confident about x, i.e. vector p is skewed towards a standard basis vector e_j, then $\hat{y} = j$, and vector $(p_i - I(\hat{y} = i))_{i=1}^{K}$ has a small length. Therefore, g_x has a small length as well. Consequently, such high-confidence examples tend to have gradient embeddings of small magnitude, which are unlikely to be repeatedly selected by k-MEANS++ at iteration t.

For the special case of binary logistic regression ($K = 2$ and $g(x; V) = x$), we provide further justifications on why BADGE yields better updates than vanilla uncertainty sampling in Appendix C.

4 EXPERIMENTS

We evaluate the performance of BADGE against several algorithms in the literature. In our experiments, we seek to answer the following question: how robust are the learning algorithms to choices of neural network architecture, batch size, and dataset?

To ensure a comprehensive comparison among all algorithms, we evaluate them in a batch mode active learning setup, with $M = 100$ being the number of initial random labeled examples, and batch size B varying from $\{100, 1000, 10000\}$. The following is a list of the baseline algorithms evaluated: the first algorithm performs representative sampling; the next three algorithms are uncertainty based; the last algorithm is a hybrid of representativeness and uncertainty based approaches.

1. CORESET: active learning with coreset selection (Sener and Savarese, 2018), where the embedding of each example is computed as the network's output of the penultimate layer, and the samples at each round are selected using a greedy furthest-first traversal conditioned on all labeled examples.

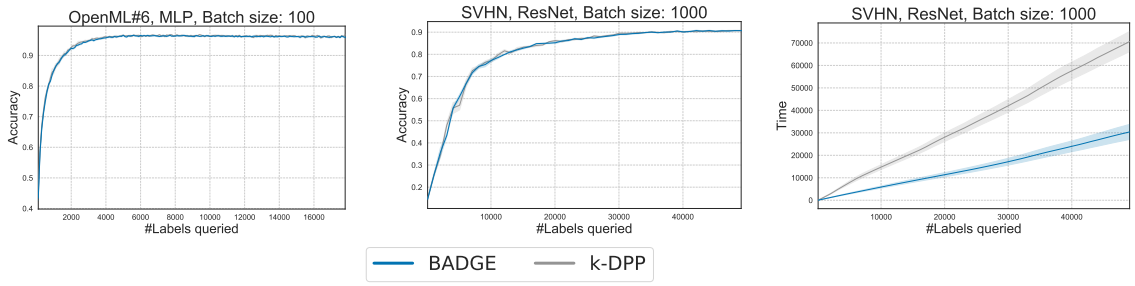

Figure 1: **Left and middle**: Learning curves for BADGE versus k-DPP sampling with gradient embeddings for different scenarios. **Right:** A run time comparison (seconds) for BADGE versus k-DPP sampling corresponding to the middle scenario. The performance of the two sampling approaches nearly perfectly overlap.

2. CONF (Confidence Sampling (Wang and Shang, 2014)): uncertainty-based active learning algorithm that selects B examples with the smallest predicted probability of a class according to $\max_{i=1}^{K} f(x;\theta)_i$.
3. MARG (Margin Sampling): uncertainty-based active learning algorithm that selects the bottom B examples sorted according to the example's multiclass margin, defined as $f(x;\theta)_{\hat{y}} - f(x;\theta)_{y'}$, where $\hat{y}$ and y' are the indices of the largest and second largest entries of $f(x;\theta)$.
4. ENTROPY (Wang and Shang, 2014): uncertainty-based active learning algorithm that selects the top B examples according to the entropy of the example's predictive class probability distribution, defined as $H((f(x;\theta)_y)_{y=1}^{K})$, where $H(p) = \sum_{i=1}^{K} p_i \ln \frac{1}{p_i}$.
5. ALBL (Active Learning by Learning (Hsu and Lin, 2015)): A bandit-style meta-active learning algorithm that selects between CORESET and CONF at every round.
6. RAND: the naive baseline of randomly selecting k examples to query at each round.

We consider three neural architectures: a two-layer Perceptron with ReLU activations (MLP), an 18-layer convolutional ResNet (He et al., 2016), and an 11-layer VGG network (Simonyan and Zisserman, 2014). We evaluate our algorithms using three image datasets, SVHN (Netzer et al., 2011), CIFAR10 (Krizhevsky, 2009) and MNIST (LeCun et al., 1998), [1] and four non-image datasets from the OpenML repository (#6, #155, #156, and #184). [2] For the image datasets, the embedding layer in the MLP is 256. For the openML datasets, the embedding dimensionality of the MLP is 1024, as more capacity is helpful to fit the training data. We train the models using the cross entropy loss and the Adam variant of SGD until the training accuracy of the algorithm exceeds 99%. We use a learning rate of 0.001 for image data, and of 0.0001 for non-image data. The model is retrained from scratch every time new samples are queried. All experiments are repeated five times. Baselines use implementations from the libact library (Yang et al., 2017).

Learning curves. Here we show some examples of learning curves that highlight some of the phenomena we observe related to the fragility of active learning algorithms with respect to batch size, architecture, and dataset. Often, we see that in early rounds of training, it is better to do diversity sampling, and later in training, it is better to do uncertainty sampling. This kind of event is demonstrated in the first panel of Figure 2, which shows CORESET outperforming confidence-based methods at first, but then doing worse

[1] Because MNIST is a dataset that is extremely easy to classify, we only use MLPs, rather than convolutional networks, to better study the differences between active learning algorithms.

[2] The OpenML datasets are from `openml.org`, and are selected on two criteria: first, they have at least 10000 samples; second, neural networks have a significantly smaller test error rate when compared to linear models.

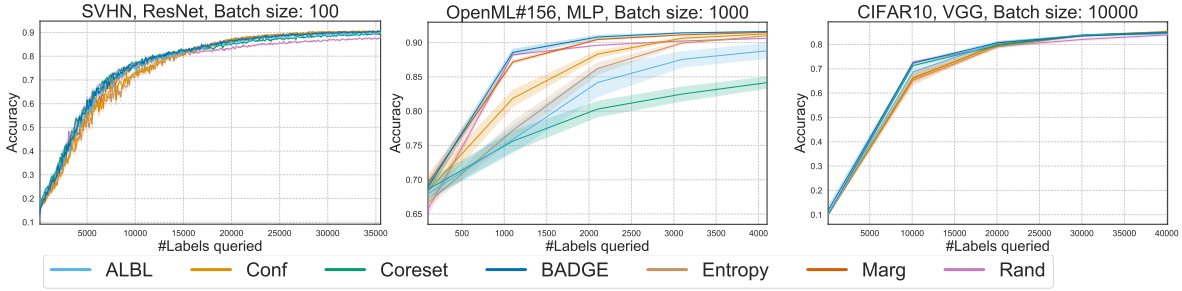

Figure 2: Active learning test accuracy versus the number of total labeled samples for a range of conditions.

than these methods later on. BADGE does about as well as representative sampling when that strategy does best, and as well as uncertainty sampling once those methods start outpacing CORESET. This suggests that BADGE is a good choice regardless of labeling budget.

Separately, we notice that diversity sampling only seems to work well when either the model has good priors (inductive biases) built in, or when the data are easy to learn. Otherwise, penultimate layer representations are not meaningful, and diverse sampling can be deleterious. For this reason, CORESET often performs worse than random on sufficiently complex data when not using a convolutional network (Figure 2, second panel). Even when batch size is large and the model has helpful inductive biases, the uncertainty information in BADGE can give it an advantage over pure diversity approaches (Figure 2, third panel) Comprehensive plots of this kind, spanning architecture, dataset, and batch size are in Appendix D.

Pairwise comparisons. We next give a comprehensive comparison over all pairs of algorithms over all datasets (D), batch sizes (B), model architectures (A), and label budgets (L). From the learning curves, it can be observed that when the label budgets are large enough, the algorithms eventually reach similar performance, hence the comparison between algorithms in the large sample limit is uninteresting. For this reason, for each combination of (D, B, A), we select a set of labeling budgets L where learning is still progressing. Specifically, we compute n_0, the smallest number of labels where RAND's accuracy reaches 99% of its final accuracy, and choose label budget L from $\left\{M + 2^{i-1}B : i \in [\lfloor \log((n_0 - M)/B) \rfloor]\right\}$. The calculation of scores in the penalty matrix P follows the protocol: for each (D, B, A, L) combination and algorithms i, j, we have 5 test errors (one for each repeated run) $\left\{e_i^1, \ldots, e_i^5\right\}$ and $\left\{e_j^1, \ldots, e_j^5\right\}$ for each algorithm respectively.We compute the z score as $z = \frac{\sqrt{5}\hat{\mu}}{\hat{\sigma}}$, where

$$\hat{\mu} = \frac{1}{5}\sum_{l=1}^{5}(e_i^l - e_j^l), \hat{\sigma} = \sqrt{\frac{1}{4}\sum_{l=1}^{5}(e_i^l - e_j^l - \hat{\mu})^2}.$$

	BADGE	ALBL	Coreset	Conf	Marg	Entropy	Rand
BADGE	0.0	22.58	24.32	30.28	12.79	33.83	24.41
ALBL	1.22	0.0	11.02	12.42	1.99	19.65	11.97
Coreset	4.28	8.03	0.0	20.42	11.68	20.89	15.06
Conf	1.89	9.06	16.86	0.0	1.29	15.66	13.68
Marg	2.65	20.39	24.0	22.4	0.0	25.08	20.76
Entropy	3.9	6.3	13.58	5.45	1.79	0.0	11.61
Rand	2.77	15.01	18.52	21.37	8.63	24.59	0.0
	2.39	11.62	15.47	16.05	5.45	19.96	13.93

Figure 3: A pariwise penalty matrix over all experiments. Element $P_{i,j}$ corresponds roughly to the number of times algorithm i outperforms algorithm j. Column-wise averages at the bottom show overall performance (lower is better).

Algorithm i is said to *beat* algorithm j in this setting if $z > 1.96$, and similarly algorithm j beats algorithm i if $z < -1.96$. For each (D, B, A) combination, suppose there are $n_{D,B,A}$ different values of L, then for each L, if algorithm i beats algorithm j, we accumulate a penalty of $1/n_{D,B,A}$ to $P_{i,j}$; otherwise, if algorithm j beats algorithm i, we accumulate a penalty of $1/n_{D,B,A}$ to $P_{j,i}$. The choice of the penalty value $1/n_{D,B,A}$ is to ensure that every (D, B, A) combination gets equal share in the aggregated matrix. Intuitively, each row i

indicates the number of settings where algorithm i beats other algorithms; and each column j indicates the number of settings where algorithm j gets beaten by other algorithms.

The penalty matrix in Figure 3 summarizes all experiments, showing that BADGE generally outperforms baselines. Matrices grouped by batch size and architecture are in Apppendx E, each suggesting that BADGE outperforms other algorithms.

Cumulative distribution functions of normalized errors. For each (D, B, A, L) combination, we have five average errors for each algorithm i: $\bar{e}_i = \frac{1}{5}\sum_{l=1}^{5} e_i^l$. To ensure that the errors of these algorithms are on the same scale in all settings, we compute the normalized error of every algorithm i, defined as $\mathrm{ne}_i = \bar{e}_i/\bar{e}_r$, where r is the index of the RAND algorithm. By definition, the normalized errors of the RAND algorithm are identically 1 in all settings. Same as in the generation of penalty matrices, for each (D, B, A) combination, we only consider a subset of L values from the set $\left\{M + 2^{i-1}B : i \in [\lfloor \log((n_0 - M)/B) \rfloor]\right\}$; in addition, we assign a weight proportional to $1/n_{D,B,A}$ to each (D, B, A, L) combination, where there are $n_{D,B,A}$ different L values for this combination of (D, B, A). We then plot the cumulative distribution functions (CDFs) of the normalized errors of all algorithms: for a value of x, the y value is the total weight of settings where the algorithm has normalized error at most x; in general, an algorithm that has a higher CDF value has better performance.

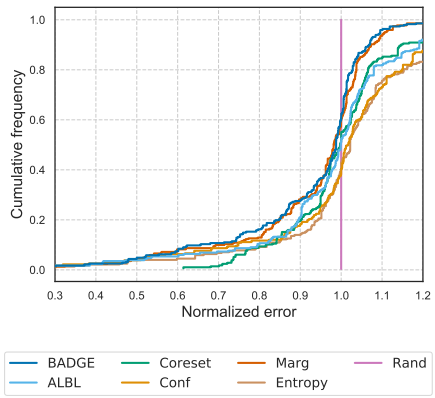

Figure 4: Cumulative distribution function of the normalized errors of all algorithms.

We show the generated CDFs in Figures 4, 21 and 22. We can see from Figure 4 that, BADGE has the best overall performance. In addition, from Figures 21 and 22 in Appendix F, we can conclude that, when the batch sizes are small (100 or 1000), or when the MLP model is used, both BADGE and MARG outperform the rest. However, in the regime when the batch size is large (10000), MARG's performance degrades, while BADGE, ALBL and CORESET are the best performing approaches.

5 RELATED WORK

Active learning is a well-studied problem (Settles, 2010; Dasgupta, 2011; Hanneke, 2014). There are two major strategies for active learning algorithms: representative sampling and uncertainty sampling.

In representative sampling, the algorithm selects a batch of examples that are representative of the unlabeled set to ask for labels. The high-level idea is that the set of examples chosen, once labeled, can act as a surrogate for the full dataset. Performing loss minimization on the surrogate suffices to ensure a low error with respect to the full dataset. In the context of deep learning, (Sener and Savarese, 2018; Geifman and El-Yaniv, 2017) select representative examples based on core-set construction, a fundamental problem in computational geometry. Inspired by generative adversarial learning, (Gissin and Shalev-Shwartz, 2019) selects samples that are maximally indistinguishable from the pool of unlabeled examples.

On the other hand, uncertainty sampling is based on a different principle: it tries to select new samples that maximally reduce the uncertainty the algorithm has on the target classifier. In the context of linear classification, (Tong and Koller, 2001; Schohn and Cohn; Tur et al., 2005) propose uncertainty sampling methods that query examples that lie closest to the current decision boundary. Some uncertainty sampling approaches have theoretical guarantees on consistency (Hanneke, 2014; Balcan et al., 2006). Such methods have also been recently generalized to deep learning. For instance, (Gal et al., 2017) uses Dropout as an approximation of the posterior of the model parameters, and develop information-based uncertainty reduction

criteria; inspired by recent advances on adversarial examples generation, (Ducoffe and Precioso, 2018) uses the distance between an example and one of its adversarial examples as an approximation of its distance to the current decision boundary, and uses it as the criterion of label queries.

There are several existing approaches that support a hybrid of representative sampling and uncertainty sampling. For example, Baram et al. (2004); Hsu and Lin (2015) present meta-active learning algorithms that can combine the advantages of different active learning algorithms. Inspired by expected loss minimization, Huang et al. (2010) develops label query criteria that balances between the representativeness and informativeness of examples. Another method for this is Active Learning by Learning (Hsu and Lin, 2015), which can select whether to exercise a diversity based algorithm or an uncertainty based algorithm at each round of training as a sequential decision process.

There is also a large body of literature on batch mode active learning, where the learner is asked to select a batch of samples within each round (Guo and Schuurmans, 2008; Wang and Ye, 2015; Chen and Krause, 2013; Wei et al., 2015). In these works, batch selection is often formulated as an optimization problem, with objectives based on (upper bounds of) average log-likelihood, average squared loss, etc.

A different query criterion based on expected gradient length (EGL) has been proposed in the literature (Settles et al., 2008). In recent work, (Huang et al., 2016) show that the EGL criterion is related to the T-optimality criterion in experimental design; in addition, they show that the samples selected by EGL are very different from those by entropy-based uncertainty criterion. (Zhang et al., 2017a) uses the EGL criterion in active sentence and document classification with CNNs. These works differ most substatially from BADGE in that they do not take into account the diversity of the examples queried within each batch.

There are many theoretical works that focus on the related problem of adaptive subsampling for fully-labeled datasets in regression settings (Han et al., 2016; Wang et al., 2018; Ting and Brochu, 2018). Empirical studies of batch stochastic gradient descent also employ adaptive sampling to "emphasize" hard or representative examples (Zhang et al., 2017b; Chang et al., 2017). These works aim at reducing computation costs or finding a better local optimal solution, as opposed to reducing label costs. Nevertheless, our work is inspired by their sampling criteria, which also emphasizes samples that induce large updates to the model.

As mentioned earlier, our sampling criterion has resemblance to sampling from k-determinantal point processes (k-DPPs) (Kulesza and Taskar, 2011). Note that in multiclass classification settings, our gradient-based embedding of an example can be viewed as the outer product of the original embedding in the penultimate layer and a probability score vector that encodes the uncertainty information on this example (see Section 3). In this view, the penultimate layer embedding characterizes the diversity of each example, whereas the probability score vector characterizes the quality of each example. The k-DPP is also a natural probabilistic tool for sampling that trades off between quality and diversity (See Kulesza et al., 2012, Section 3.1).

6 Discussion

We have established that BADGE is empirically an effective deep active learning algorithm across different architectures and batch sizes, performing similar to or better than other active learning algorithms. A fundamental remaining question is: "Why?" While deep learning is notoriously difficult to analyze theoretically, there are several intuitively appealing properties of BADGE:

1. The definition of uncertainty (a lower bound on the gradient magnitude of the last layer) guarantees some update of parameters.
2. It optimizes for diversity as well as uncertainty, eliminating a failure mode of choosing many identical uncertain examples in a batch, and does so without requiring any hyperparameters.
3. The randomization associated with the k-MEANS++ initialization sampler implies that even for adversarially constructed datasets it eventually converges to a good solution.

The combination of these properties appears to generate the robustness that we observe empirically.

REFERENCES

Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 65–72, 2006.

Alina Beygelzimer, Daniel J Hsu, John Langford, and Tong Zhang. Agnostic active learning without constraints. In *Advances in Neural Information Processing Systems*, pages 199–207, 2010.

Nicolo Cesa-Bianchi, Claudio Gentile, and Francesco Orabona. Robust bounds for classification via selective sampling. In *Proceedings of the 26th annual international conference on machine learning*, pages 121–128. ACM, 2009.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1193–1200, 2011.

Byungkon Kang. Fast determinantal point process sampling with application to clustering. In *Advances in Neural Information Processing Systems*, pages 2319–2327, 2013.

Nima Anari, Shayan Oveis Gharan, and Alireza Rezaei. Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes. In *Conference on Learning Theory*, pages 103–115, 2016.

Michał Dereziński. Fast determinantal point processes via distortion-free intermediate sampling. *arXiv preprint arXiv:1811.03717*, 2018.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=H1aIuk-RW`.

Dan Wang and Yi Shang. A new active labeling method for deep learning. In *2014 International joint conference on neural networks (IJCNN)*, pages 112–119. IEEE, 2014.

Wei-Ning Hsu and Hsuan-Tien Lin. Active learning by learning. In *Twenty-Ninth AAAI conference on artificial intelligence*, 2015.

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.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yao-Yuan Yang, Shao-Chuan Lee, Yu-An Chung, Tung-En Wu, Si-An Chen, and Hsuan-Tien Lin. libact: Pool-based active learning in python. *arXiv preprint arXiv:1710.00379*, 2017.

Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

Sanjoy Dasgupta. Two faces of active learning. *Theoretical computer science*, 412(19):1767–1781, 2011.

Steve Hanneke. Theory of disagreement-based active learning. *Foundations and Trends® in Machine Learning*, 7(2-3):131–309, 2014.

Yonatan Geifman and Ran El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

Daniel Gissin and Shai Shalev-Shwartz. Discriminative active learning, 2019. URL `https://openreview.net/forum?id=rJl-HsR9KX`.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. pages 45–66, 2001.

Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. Citeseer.

Gokhan Tur, Dilek Hakkani-Tür, and Robert E Schapire. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171–186, 2005.

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.

Melanie Ducoffe and Frederic Precioso. Adversarial active learning for deep networks: a margin based approach. *arXiv preprint arXiv:1802.09841*, 2018.

Yoram Baram, Ran El Yaniv, and Kobi Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5(Mar):255–291, 2004.

Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. In *Advances in neural information processing systems*, pages 892–900, 2010.

Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In *Advances in neural information processing systems*, pages 593–600, 2008.

Zheng Wang and Jieping Ye. Querying discriminative and representative samples for batch mode active learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(3):17, 2015.

Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 160–168, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/chen13b.html`.

Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*, pages 1954–1963, 2015.

Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In *Advances in neural information processing systems*, pages 1289–1296, 2008.

Jiaji Huang, Rewon Child, and Vinay Rao. Active learning for speech recognition: the power of gradients. *arXiv preprint arXiv:1612.03226*, 2016.

Ye Zhang, Matthew Lease, and Byron C Wallace. Active discriminative text representation learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017a.

Lei Han, Kean Ming Tan, Ting Yang, and Tong Zhang. Local uncertainty sampling for large-scale multi-class logistic regression. *arXiv preprint arXiv:1604.08098*, 2016.

HaiYing Wang, Rong Zhu, and Ping Ma. Optimal subsampling for large sample logistic regression. *Journal of the American Statistical Association*, 113(522):829–844, 2018.

Daniel Ting and Eric Brochu. Optimal subsampling with influence functions. In *Advances in Neural Information Processing Systems*, pages 3650–3659, 2018.

Cheng Zhang, Hedvig Kjellstrom, and Stephan Mandt. Determinantal point processes for mini-batch diversification. *arXiv preprint arXiv:1705.00607*, 2017b.

Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, pages 1002–1012, 2017.

Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

Stephen Mussmann and Percy S Liang. Uncertainty sampling is preconditioned stochastic gradient descent on zero-one loss. In *Advances in Neural Information Processing Systems*, pages 6955–6964, 2018.

A THE k-MEANS++ SEEDING ALGORITHM

Here we briefly review the k-MEANS++ seeding algorithm by (Arthur and Vassilvitskii, 2007). Its basic idea is to perform sequential sampling of k centers, where each new center is sampled from the ground set with probability proportional to the squared distance to its nearest center. It is shown in (Arthur and Vassilvitskii, 2007) that the set of centers returned is guaranteed to approximate the k-means objective function in expectation, thus ensuring diversity.

Algorithm 2 The k-MEANS++ seeding algorithm (Arthur and Vassilvitskii, 2007)

Require: Ground set $G \subset \mathbb{R}^d$, size k.
Ensure: Center set C of size k.
  $C_1 \leftarrow \{c_1\}$, where c_1 is sampled uniformly at random from G.
  for $t = 2, \ldots, k$: **do**
    Define $D_t(x) := \min_{c \in C_{t-1}} \|x - c\|_2$.
    $c_t \leftarrow$ Sample x from G with probability $\frac{D_t(x)^2}{\sum_{x \in G} D_t(x)^2}$.
    $C_t \leftarrow C_{t-1} \cup \{c_t\}$.
  end for
  return C_k.

B GRADIENT NORM LOWER BOUND

Recall that we are in the following setting: the network f has the form of $f(x;\theta) = \sigma(W \cdot g(x;V))$, where $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$, and the loss function is the cross entropy loss $\ell_{\mathrm{CE}}(p, y) = \sum_{i=1}^K I(y = i) \ln \frac{1}{p_i}$. In addition, $\hat{y} := \operatorname{argmax}_{y \in [K]} f(x;\theta)_y$. We have the following proposition.

Proposition 1. *For all $y \in \{1, \ldots, K\}$, denote by $g_x^y = \frac{\partial}{\partial W} \ell_{\mathrm{CE}}(f(x;\theta), y)$. Then*

$$\|g_x^y\|^2 = (\sum_{i=1}^K p_i^2 + 1 - 2p_y)\|g(x;V)\|^2.$$

Consequently, $\hat{y} = \operatorname{argmin}_{y \in [K]} \|g_x^y\|$.

Proof. Observe that by Equation (1),

$$\|g_x^y\|^2 = \sum_{i=1}^K (p_i - I(y = i))^2 \|g(x;V)\|^2 = (\sum_{i=1}^K p_i^2 + 1 - 2p_y)\|g(x;V)\|^2.$$

The second part follows from the fact that $\hat{y} = \operatorname{argmax}_{y \in [K]} p_y$. □

C BADGE FOR BINARY LOGISTIC REGRESSION

We consider instantiating BADGE for binary logistic regression, where $\mathcal{Y} = \{-1, +1\}$. Given a linear classifier w, we define the predictive probability of w on x as $p_w(y|x, \theta) = \sigma(yw \cdot x)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid funciton.

Recall that $\hat{y}(x)$ is the hallucinated label:

$$\hat{y}(x) = \begin{cases} +1 & p_w(+1|x, \theta) > 1/2 \\ -1 & p_w(+1|x, \theta) \leq 1/2. \end{cases}$$

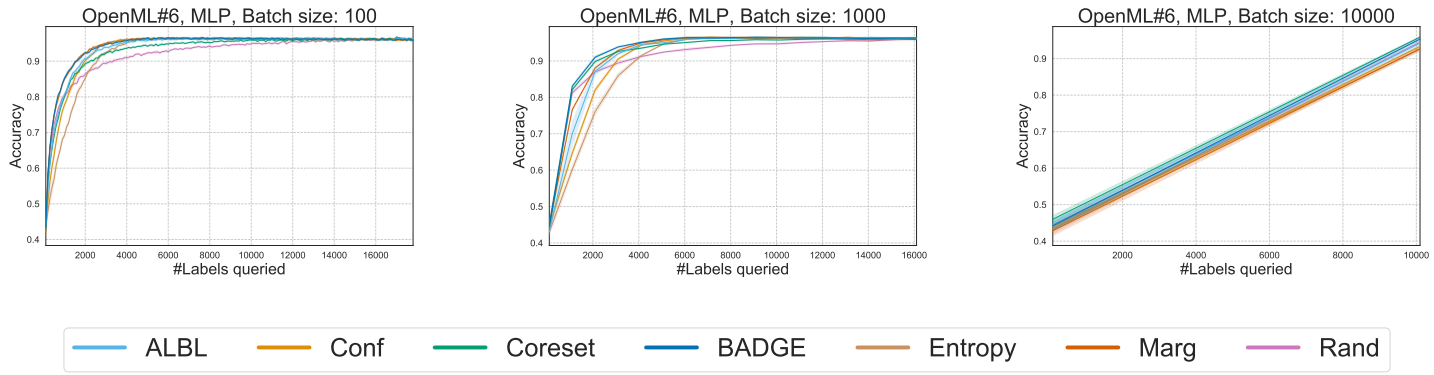

Figure 5: Full learning curves for OpenML #6 with MLP.

The logistic loss of classifier w on example (x, y) is defined as:

$$\ell(w, (x, y)) = \ln(1 + \exp(-\hat{y}(x) w \cdot x)).$$

Now, given model w and example x, we define $\hat{g}_x = \frac{\partial}{\partial w} \ell(w, (x, \hat{y})) = (1 - p_w(\hat{y}|x, \theta)) \cdot (-\hat{y} \cdot x)$ as the loss gradient induced by the example with hallucinated label, and $\tilde{g}_x = \frac{\partial}{\partial w} \ell(w, (x, y)) = (1 - p_w(y|x, \theta)) \cdot (-y \cdot x)$ as the loss gradient induced by the example with true label.

Suppose that BADGE only selects examples from region $S_w = \{x : w \cdot x = 0\}$, then as $p_w(+1|x, \theta) = p_w(-1|x, \theta) = \frac{1}{2}$, we have that for all x in S_w, $\hat{g}_x = s_x \cdot g_x$ for some $s_x \in \{\pm 1\}$. This implies that, sampling from a DPP induced by $\hat{g}_x$'s is equivalent to sampling from a DPP induced by g_x's. It is noted in Mussmann and Liang (2018) that uncertainty sampling (i.e. sampling from $D_{|S_w}$) implicitly performs preconditioned stochastic gradient descent on the expected 0-1 loss. In addition, it has been using DPP sampling over gradients may reduce the variance of the updates Zhang et al. (2017b); this suggests that BADGE, when restricted sampling over low-margin regions (S_w), improves over uncertainty sampling by collecting examples that together induce lower-variance updates on the gradient direction of expected 0-1 loss.

D ALL LEARNING CURVES

We plot all learning curves (test accuracy as a function of the number of labeled example queried) in Figures 5 to 11. In addition, we zoom into regions of the learning curves that discriminates the performance of all algorithms in Figures 12 to 18.

E PAIRWISE COMPARISONS OF ALGORITHMS

In addition to Figure 3 in the main text, we also provide penalty matrices (Figures 19 and 20), where the results are aggregated by conditioning on a fixed batch size (100, 1000 and 10000) or on a fixed neural network model (MLP, ResNet and VGG). It can be seen that, uncertainty-based methods (e.g. MARG) perform well only in small batch size regimes (100) or when using MLP models; representative sampling based methods (e.g. CORESET) only perform well in large batch size regimes (1000) or when using ResNet models. In contrast, BADGE's performance is competitive across all batch sizes and neural network models.

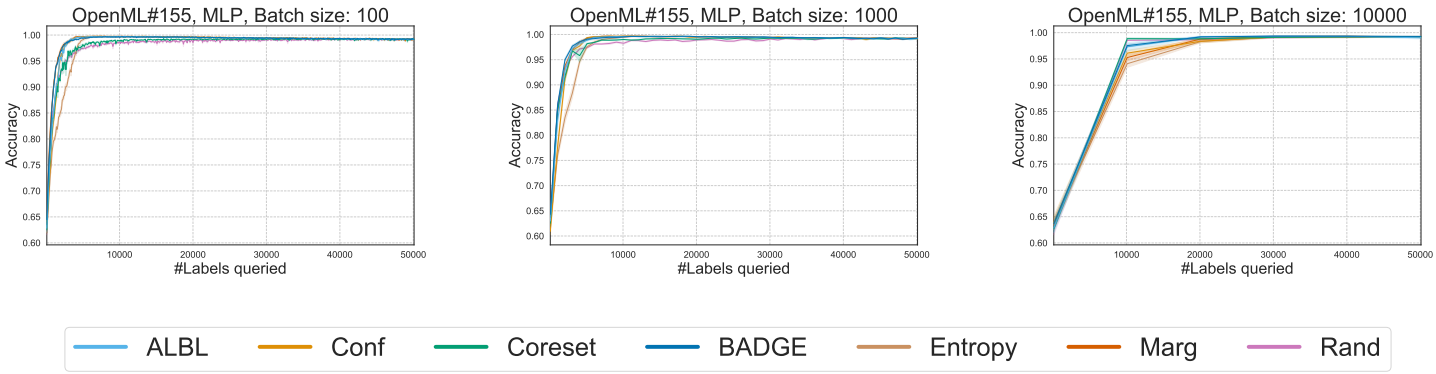

Figure 6: Full learning curves for OpenML #155 with MLP.

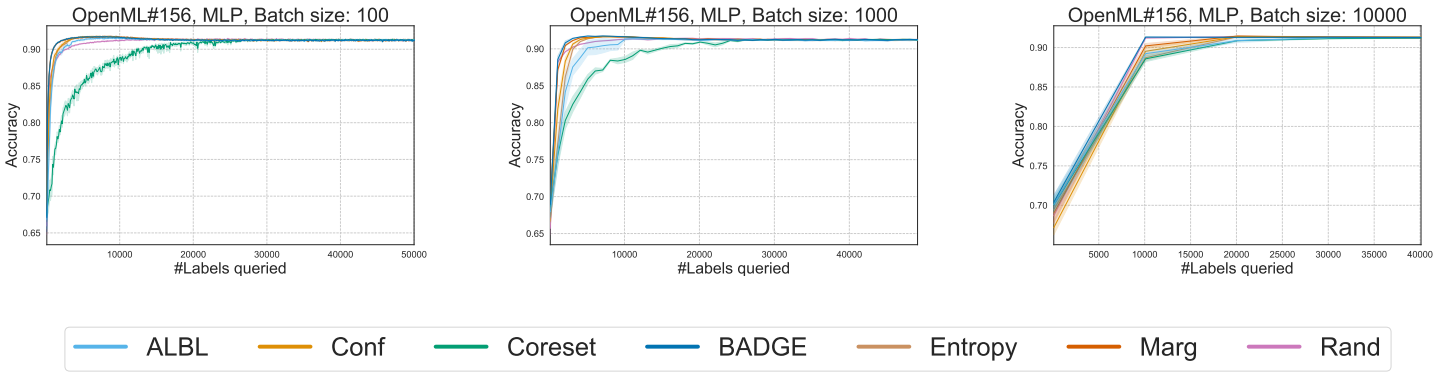

Figure 7: Full learning curves for OpenML #156 with MLP.

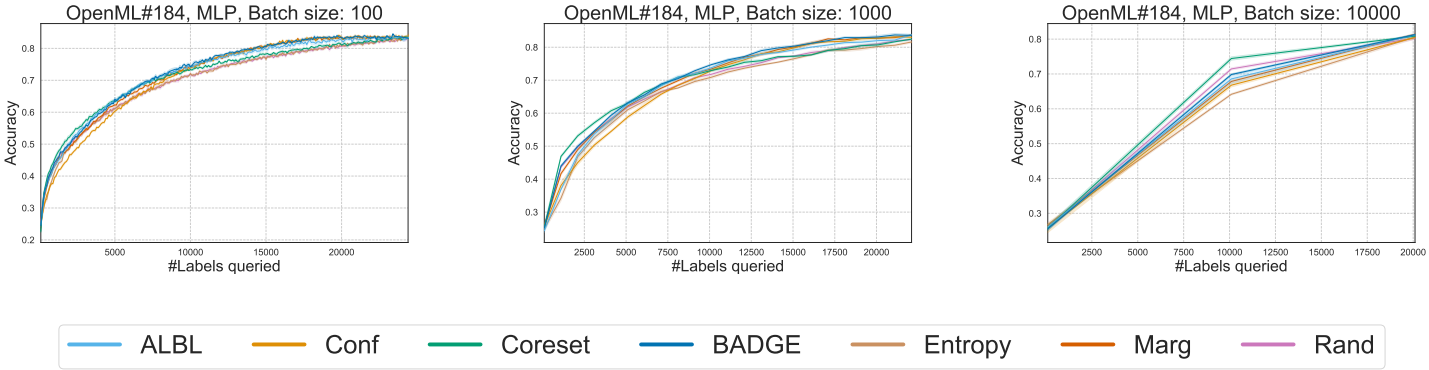

Figure 8: Full learning curves for OpenML #184 with MLP.

F CDFs OF NORMALIZED ERRORS OF DIFFERENT ALGORITHMS

In addition to Figure 4 that aggregates over all settings, we show here the CDFs of normalized errors by conditioning on fixed batch sizes (100, 1000 and 10000) in Figure 21, and show the CDFs of normalized errors by conditioning on fixed neural network models (MLP, ResNet and VGG) in Figure 22.

G COMPARISON OF k-MEANS++ AND k-DPP IN BATCH SELECTION

In Figures 23 to 29, we give running time and test accuracy comparisons between k-MEANS++ and k-DPP for selecting examples based on gradient embedding in batch mode active learning. We implement the k-DPP sampling using the MCMC algorithm from (Kang, 2013), which has a time complexity of $O(\tau \cdot (k^2 + kd))$ and space complexity of $O(kd + k^2)$, where τ is the number of sampling steps. We set τ as $\lfloor 5k \ln k \rfloor$ in our experiment. The comparisons for batch size 10000 are not shown here as the implementation of k-DPP sampling runs out of memory.

It can be seen from the figures that, although k-DPP and k-MEANS++ are based on different sampling criteria, the classification accuracies of their induced active learning algorithm are similar. In addition, when large batch sizes are required (e.g. $k = 1000$), the running times of k-DPP sampling are generally much higher than those of k-MEANS++.

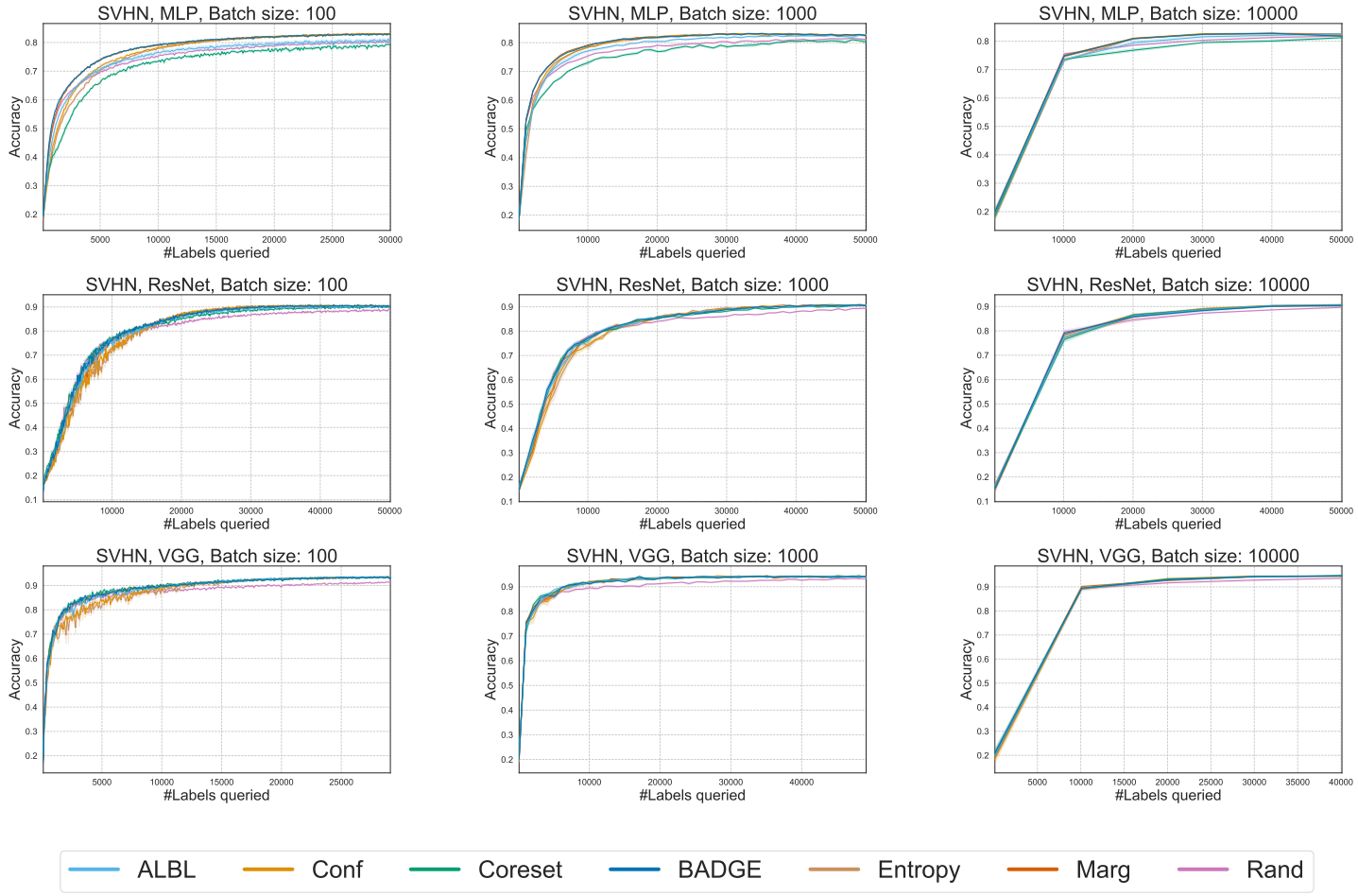

Figure 9: Full learning curves for SVHN with MLP, ResNet and VGG.

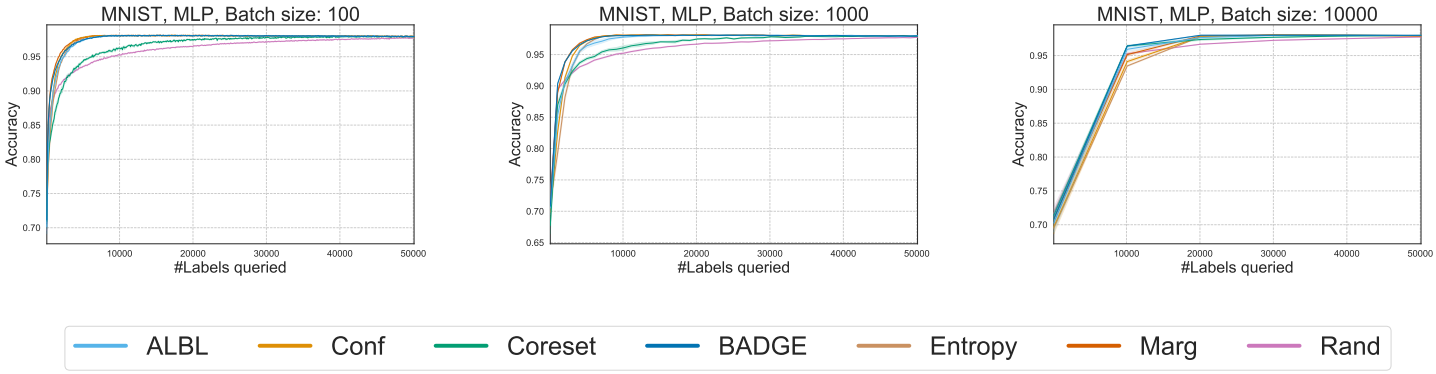

Figure 10: Full learning curves for MNIST with MLP.

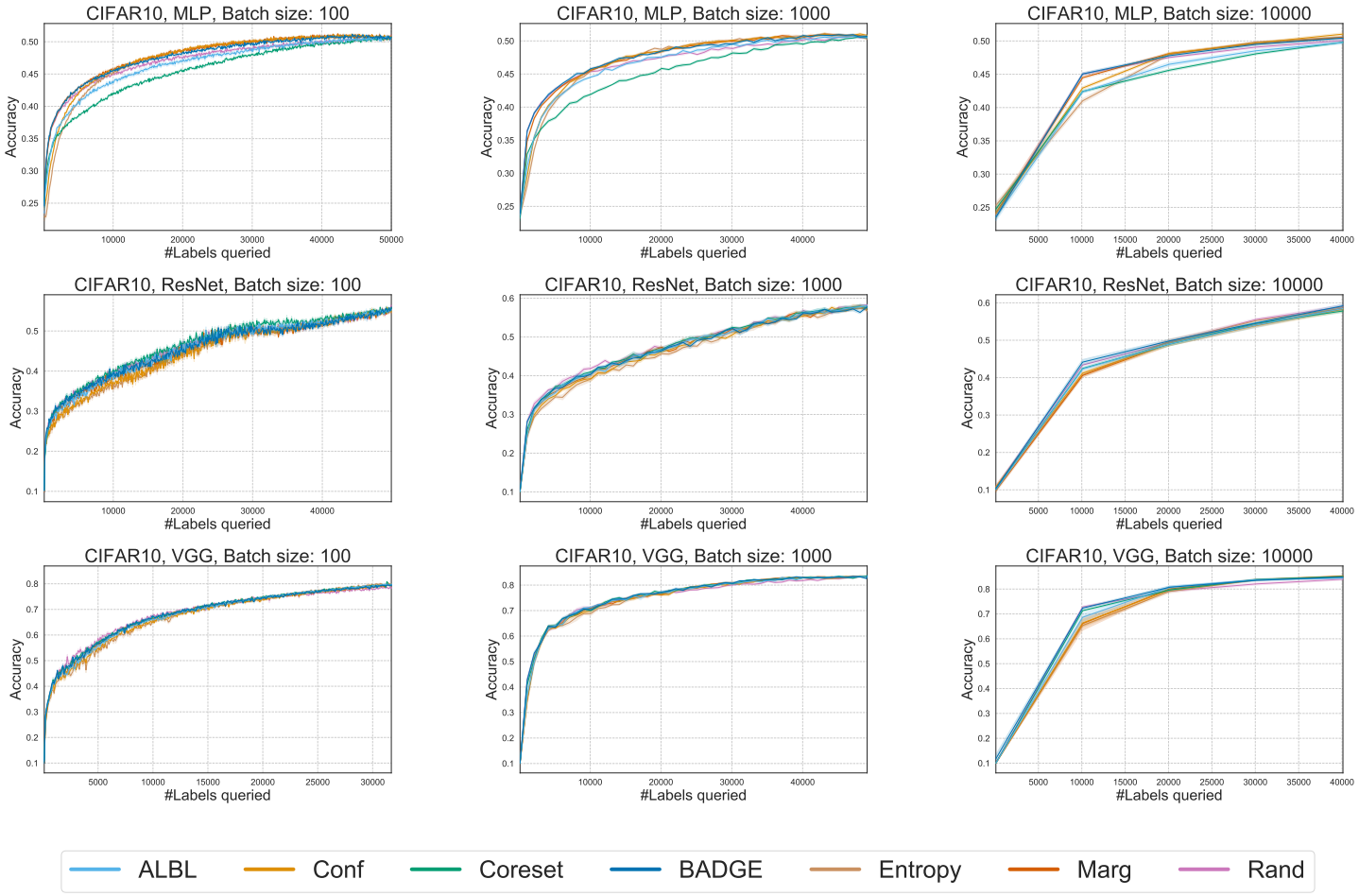

Figure 11: Full learning curves for CIFAR10 with MLP, ResNet and VGG.

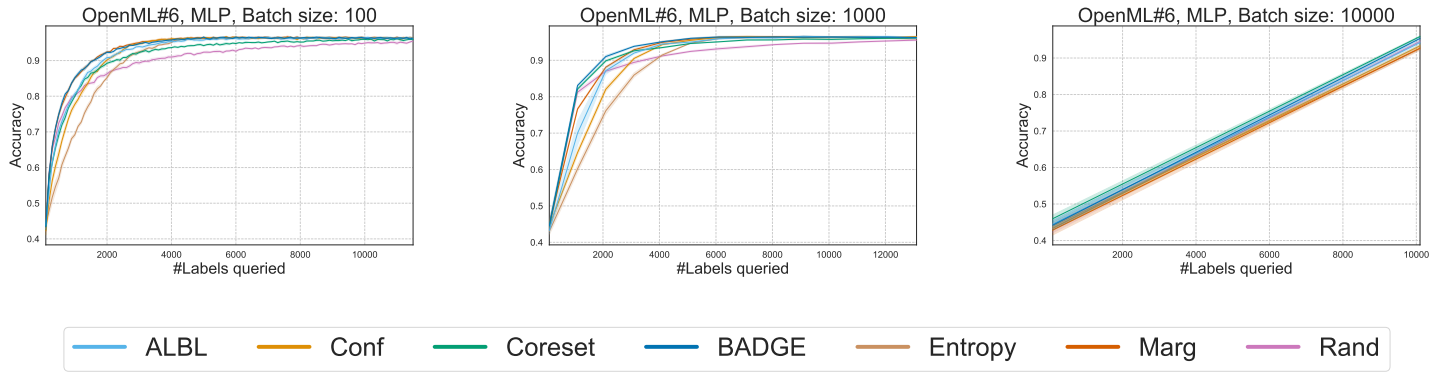

Figure 12: Zoomed-in learning curves for OpenML #6 with MLP.

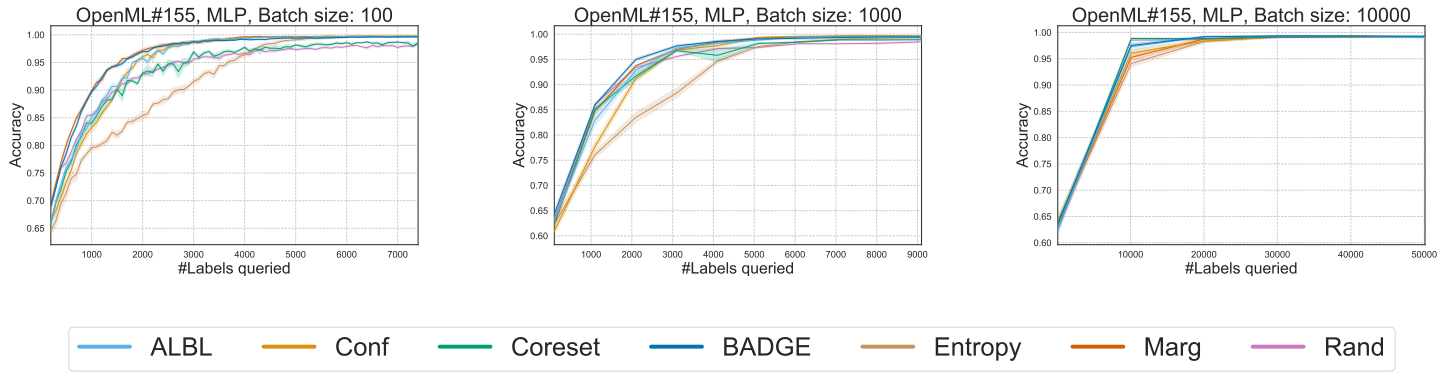

Figure 13: Zoomed-in learning curves for OpenML #155 with MLP.

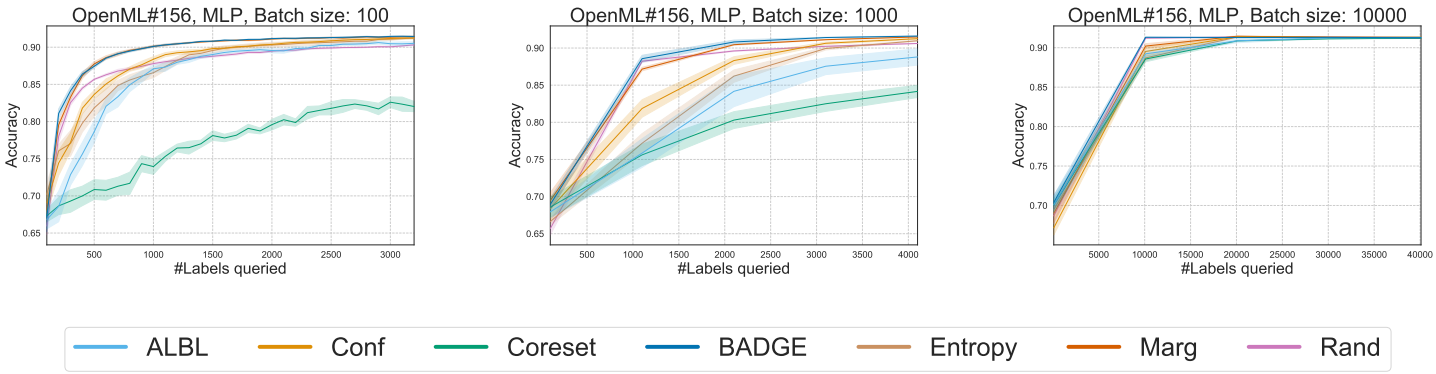

Figure 14: Zoomed-in learning curves for OpenML #156 with MLP.

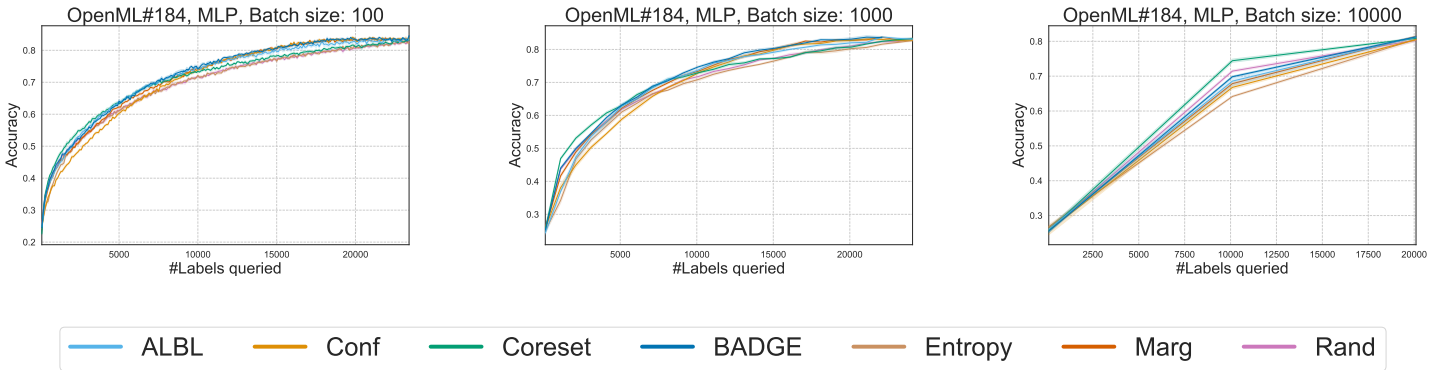

Figure 15: Zoomed-in learning curves for OpenML #184 with MLP.

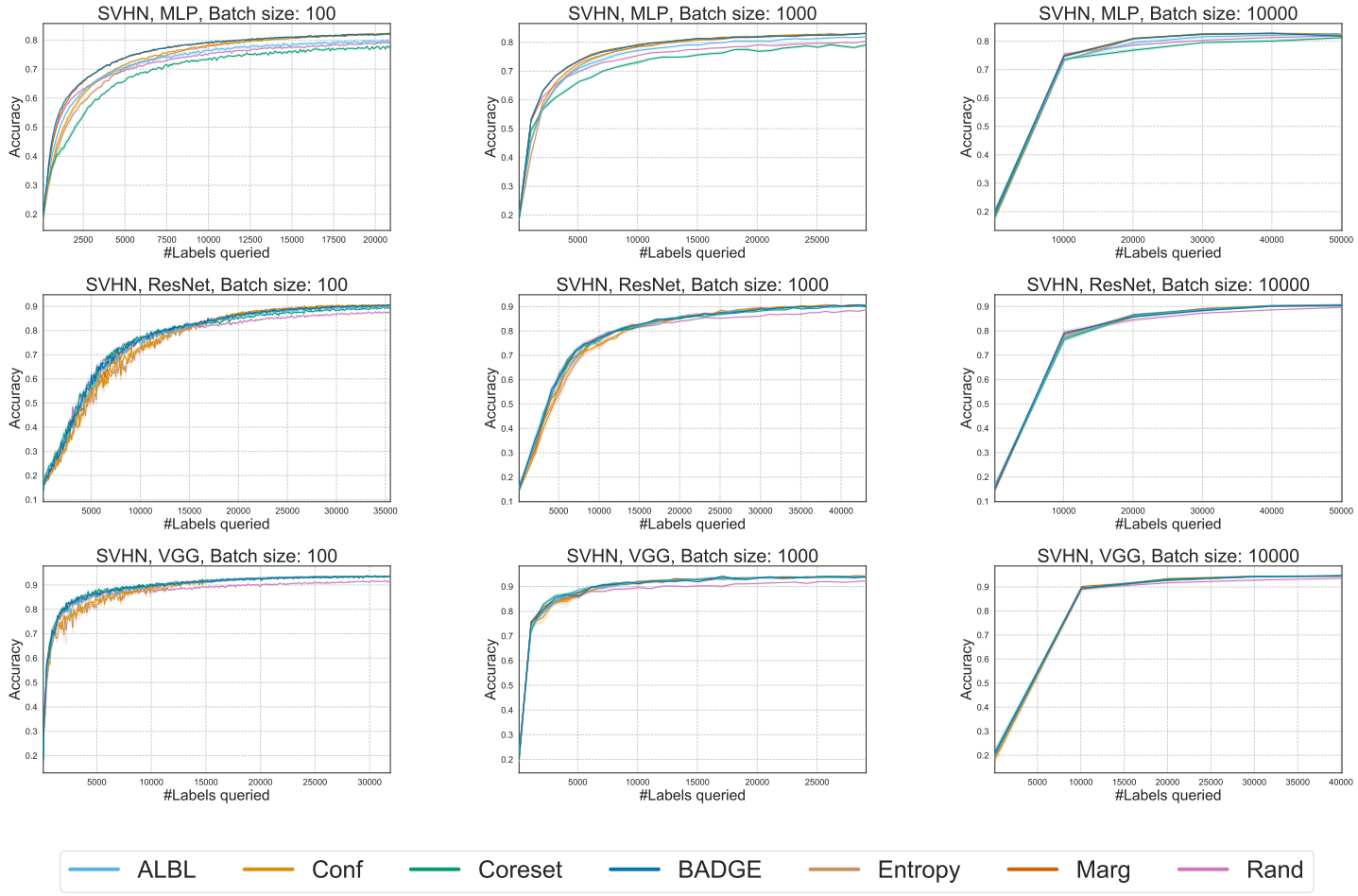

Figure 16: Zoomed-in learning curves for SVHN with MLP, ResNet and VGG.

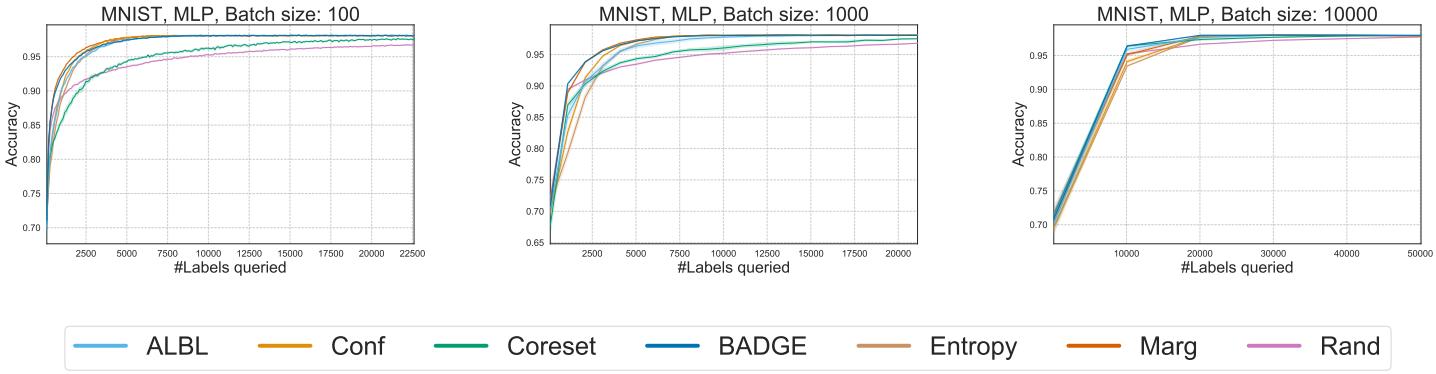

Figure 17: Zoomed-in learning curves for MNIST with MLP.

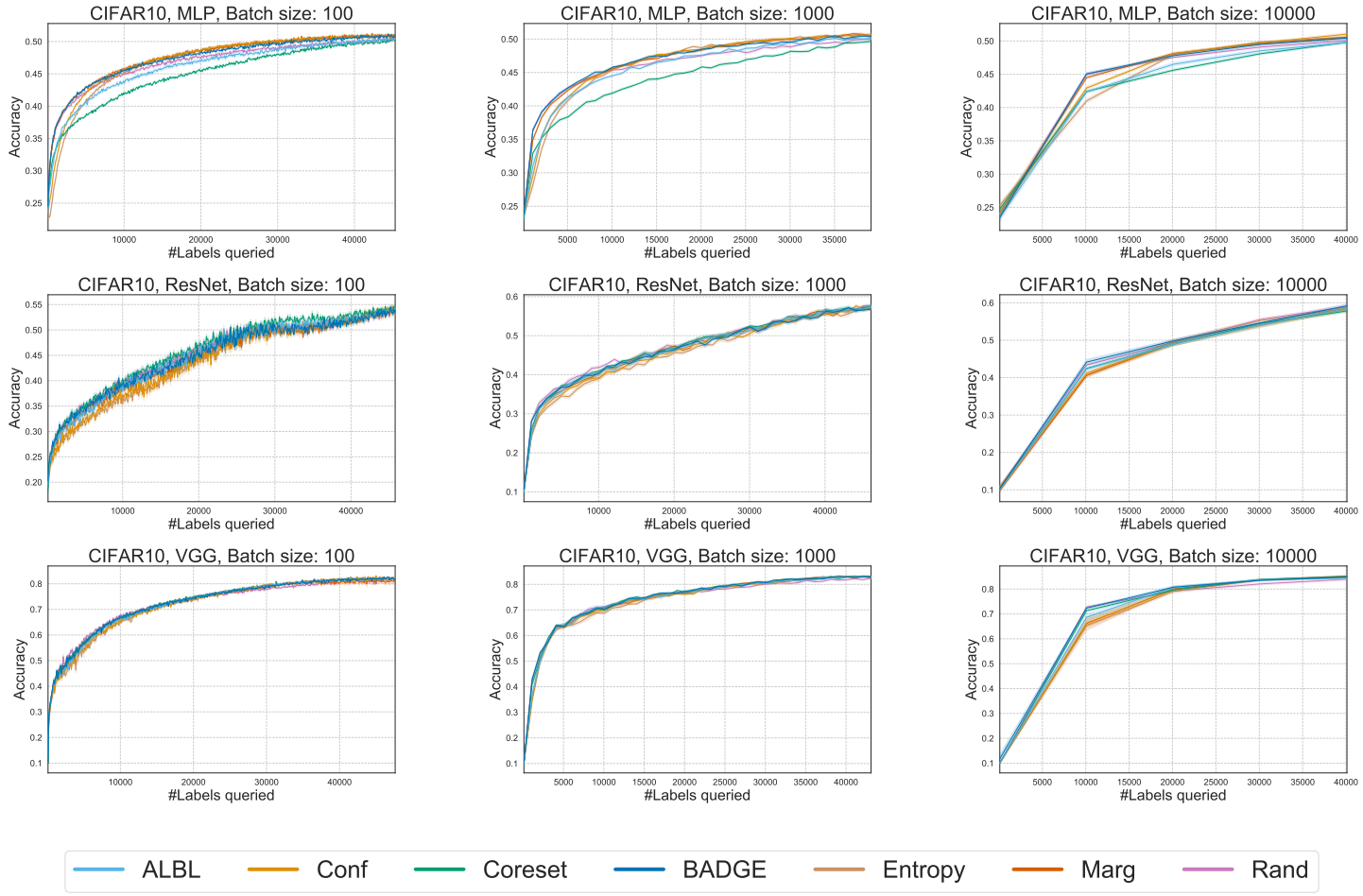

Figure 18: Zoomed-in learning curves for CIFAR10 with MLP, ResNet and VGG.

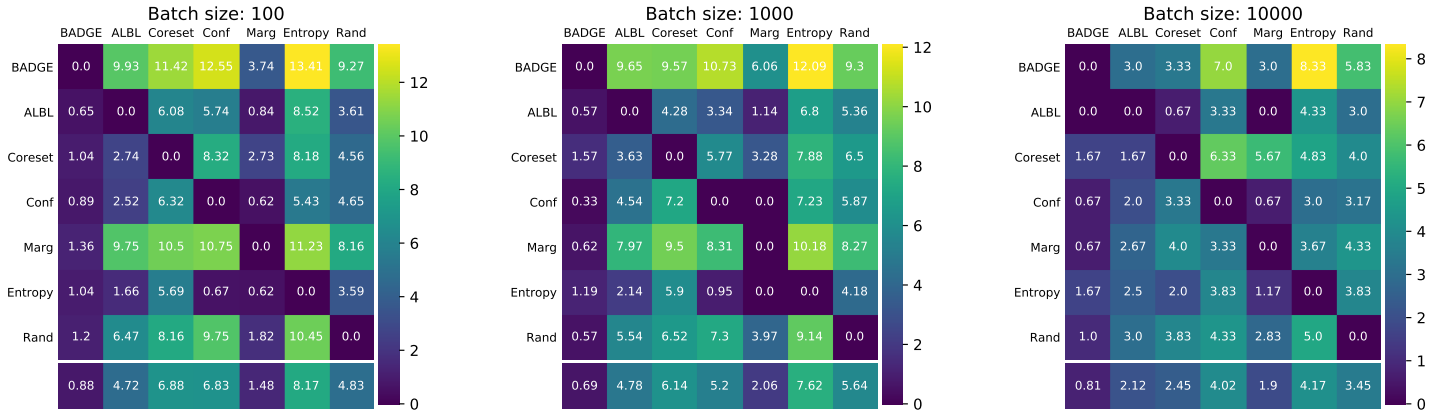

Figure 19: Pairwise penalty matrices of the algorithms, grouped by different batch sizes. Element i, j corresponds roughly to the number of times algorithm i outperforms algorithm j. Column-wise averages at the bottom show aggregate performance (lower is better). From left to right: batch size = 100, 1000, 10000.

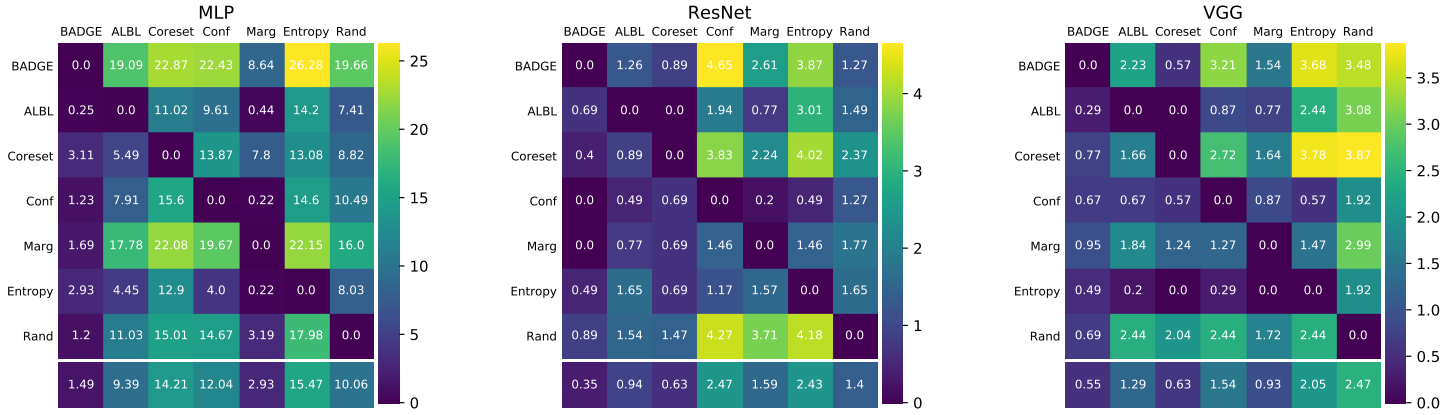

Figure 20: Pairwise penalty matrices of the algorithms, grouped by different neural network models. Element i, j corresponds roughly to the number of times algorithm i outperforms algorithm j. Column-wise averages at the bottom show aggregate performance (lower is better). From left to right: MLP, ResNet and VGG.

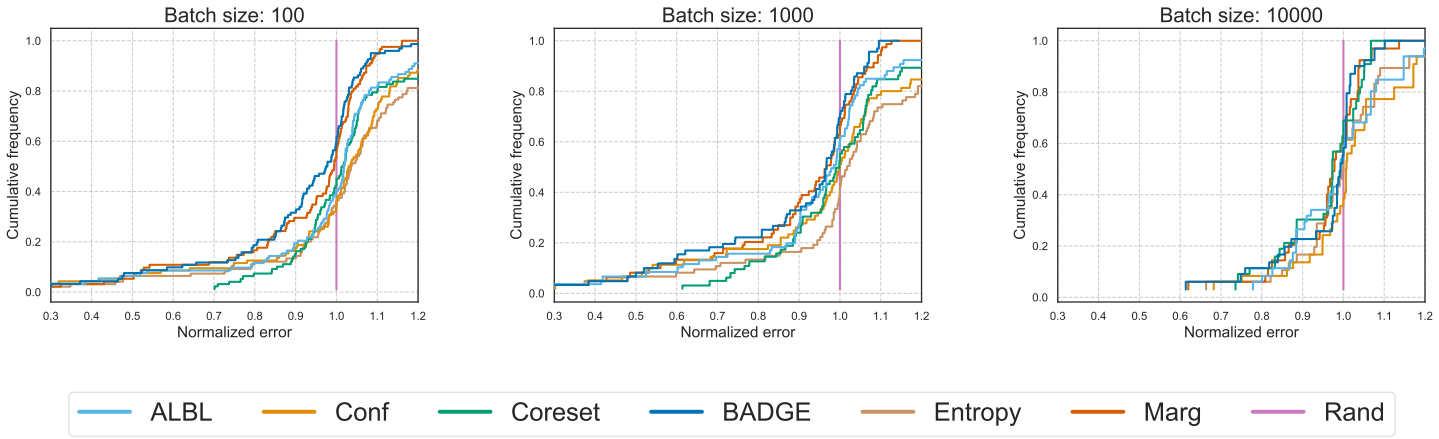

Figure 21: CDFs of normalized errors of the algorithms, group by different batch sizes. Higher CDF indicates better performance. From left to right: batch size = 100, 1000, 10000.

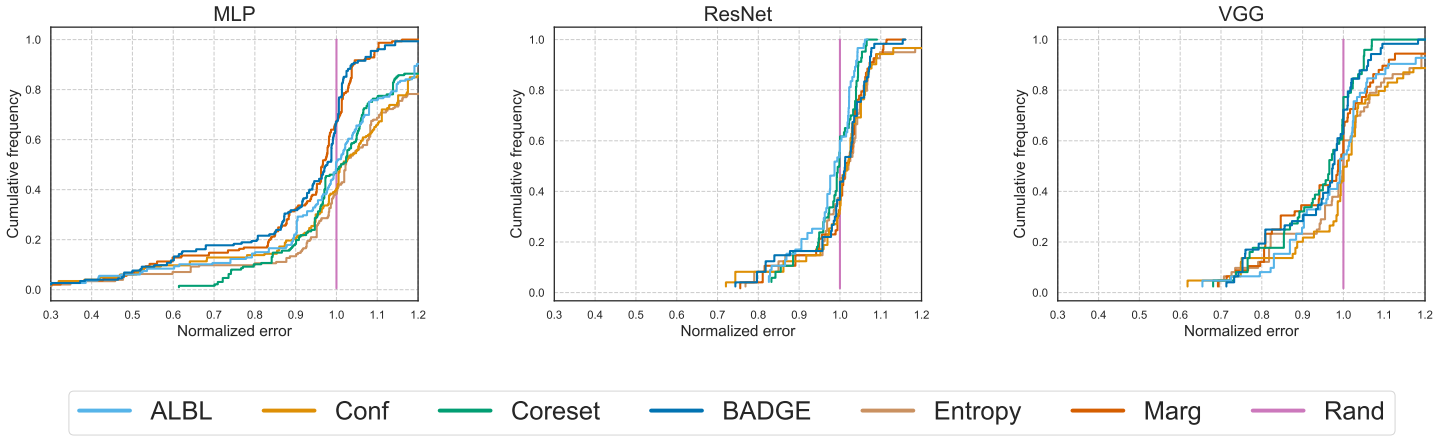

Figure 22: CDFs of normalized errors of the algorithms, group by different neural network models. Higher CDF indicates better performance. From left to right: MLP, ResNet and VGG.

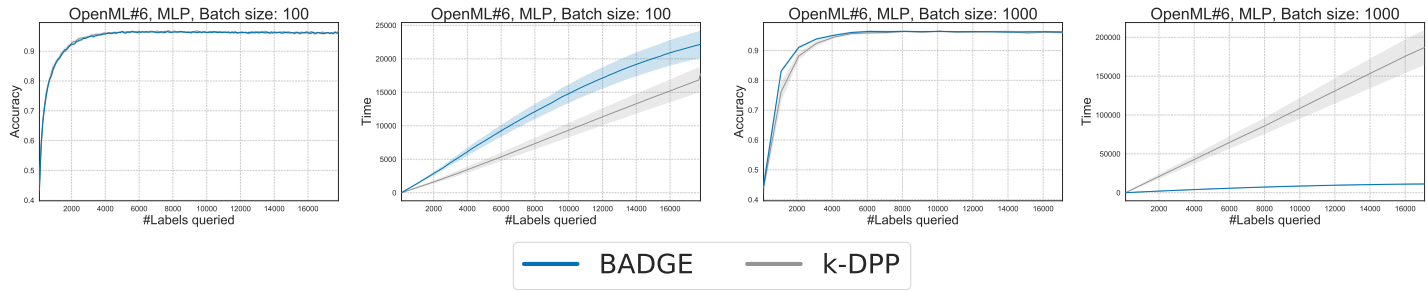

Figure 23: Learning curves and running times for OpenML #6 with MLP.

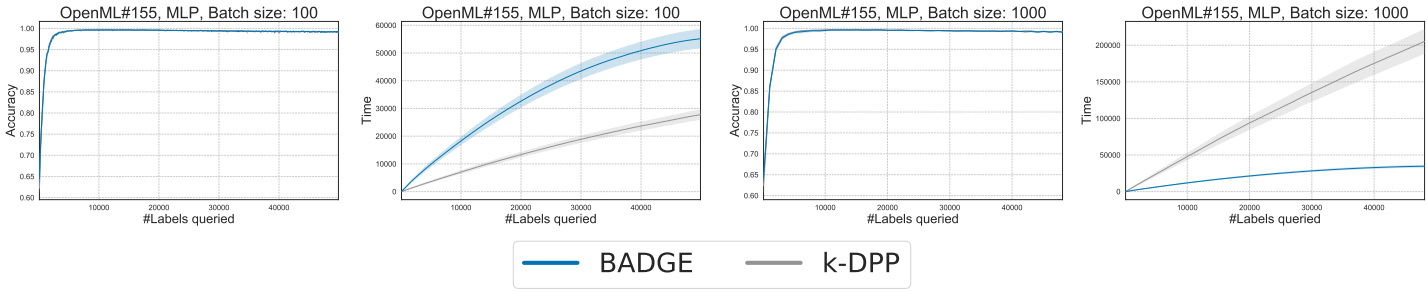

Figure 24: Learning curves and running times for OpenML #155 with MLP.

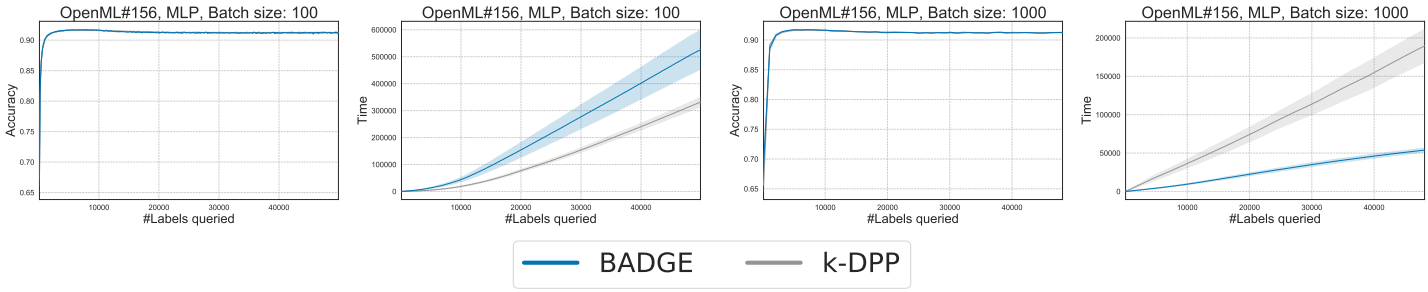

Figure 25: Learning curves and running times for OpenML #156 with MLP.

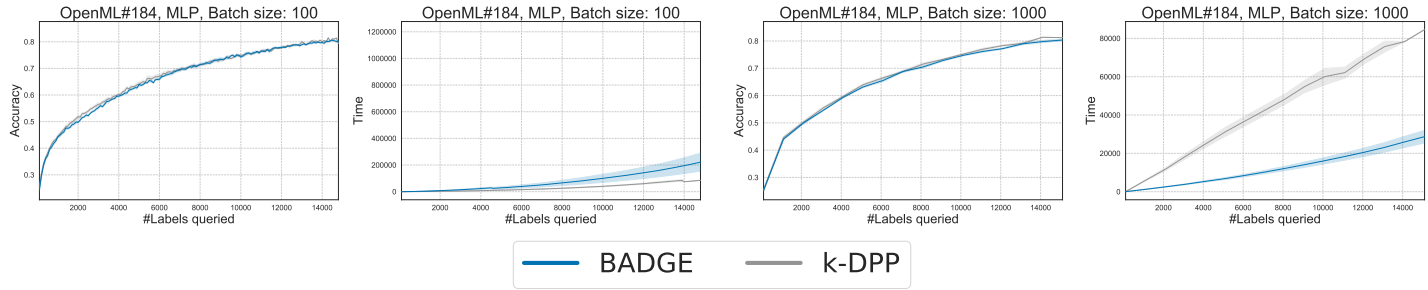

Figure 26: Learning curves and running times for OpenML #184 with MLP.

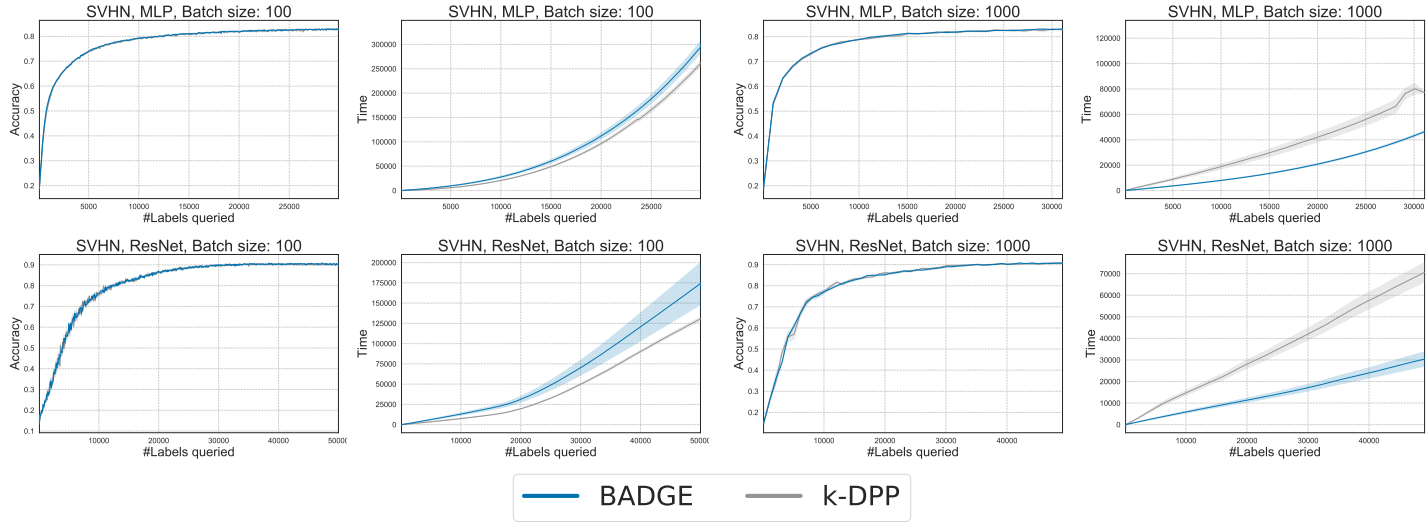

Figure 27: Learning curves and running times for SVHN with MLP and ResNet.

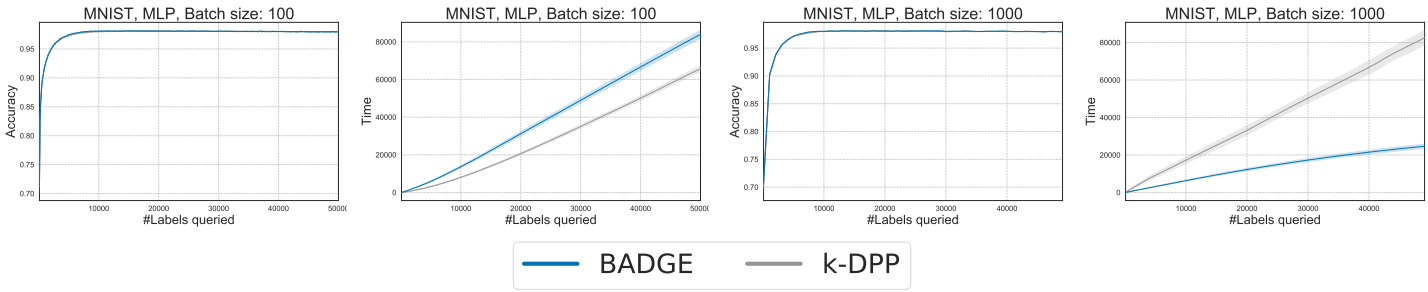

Figure 28: Learning curves and running times for MNIST with MLP.

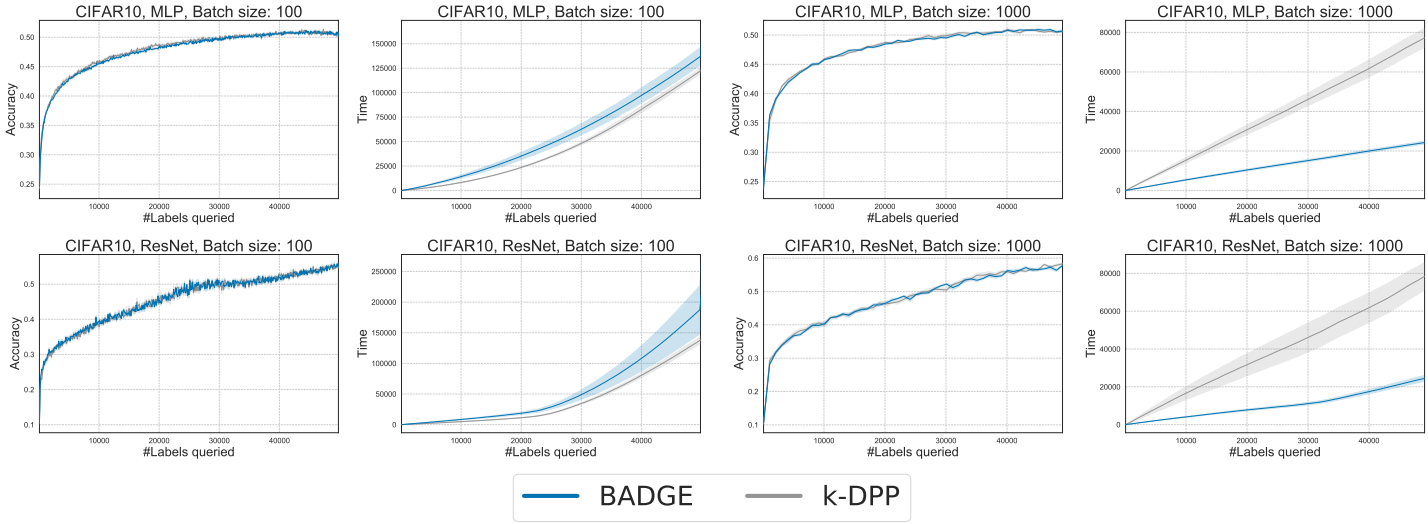

Figure 29: Learning curves and running times for CIFAR10 with MLP and ResNet.