Towards Flexible Active and Online Learning With Neural Networks

Jordan T. Ash

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Ryan P. Adams

November 2020

Abstract

Deep learning has elicited breakthrough successes on a wide array of machine learning tasks. Outside of the fully-supervised regime, however, many deep learning algorithms are brittle and unable to reliably perform across model architectures, dataset types, and optimization parameters. As a consequence, these algorithms are not easily usable by non-machine-learning experts, limiting their ability to meaningfully impact science and society.

This thesis addresses some nuanced pathologies around the use of deep learning for active and passive online learning. We propose a practical active learning approach for neural networks that is robust to environmental variables: Batch Active learning by Diverse Gradient Embeddings (BADGE). We also discuss the deleterious generalization effects of warm-starting the optimization of neural networks in sequential environments, and why this is a major problem for deep learning. We introduce a simple method that remedies this problem, and discuss some important ramifications of its application.

Acknowledgements

There are many people I would like to thank for my growth and development as a researcher. My first experience with machine learning came from Michael Littman, who was gracious in allowing me to spend time in his research group while I was an undergraduate. In preparing this section, I looked back on my first email to Michael, where I asked if he would supervise me for a semester-long research project. Despite my overly-enthusiastic, freshman-style prose, and what can be generously described as a tenuous understanding of machine learning, Michael kindly gave me a place in his lab.

I began my time at Princeton as a Master's student. Princeton's computer science Master's is a fantastic primer to graduate-level work; it is a fully-funded program, and can serve as a means for non-traditional or first-generation college students to gain exposure and confidence in academic research. In its absence, I do not think that I would be in a position to be writing this document.

I was advised over this period by Rob Schapire, who graciously continued to work with me even as he was transitioning to Microsoft Research. As a researcher, Rob's clarity in thought and communication are things to which I continually aspire. As a mentor, Rob has been a source of valuable career advice, and has introduced me to people who became meaningful collaborators. In large part, Rob's encouragement led me to apply to PhD programs.

My PhD work began under the advisement of Barbara Engelhardt. Working with Barbara was my first experience with truly interdisciplinary research. Her expertise in machine learning and statistical genetics have inspired my interest in building machine learning tools that are practically useful rather than autotelic.

I am concluding my PhD as a student of Ryan Adams, in the Laboratory for Intelligent Probabilistic Systems. Ryan's candor and research acumen have largely influenced the way in which I think about research and academia, and have refined my ability to identify impactful problems worth studying.

Over the course of my PhD, I had the opportunity to collaborate with several gifted researchers at MSR NYC. I am particularly thankful to have worked with John Langford, Akshay Krishnamurthy, Chicheng Zhang, and Alekh Agarwal. Their expertise helped me grow as a PhD student, and I am thrilled to continue working with them as a Postdoc.

This PhD would not have been possible without the support of a variety of machine learning graduate students. Emily Denton, Will Whitney, and Mikael Henaff helped introduce me to deep learning and to Lua Torch (now thankfully replaced by PyTorch) while I was an intern at Facebook AI Research in 2016. At Princeton, I collaborated and had invaluable conversations with Diana Cai, Alex Beatson, Tianju Xue, Greg Gunderson, David Zoltowski, Xingyuan Sun, Farhan Damani, Niranjani Prasad, Geoffrey Roeder, Sulin Liu, and the unparalleled LIPS "Jewish subset": Ari Seff, Sam Barnett, and Yaniv Ovadia. It has been a great privilege to learn and work alongside such gifted colleagues.

To Mom and Dad.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Deep Learning Strengths and Weaknesses

In recent years, terms like deep learning and machine learning have moved from research jargon to regular vernacular. Cutting-edge deep learning research consistently garners attention from the media [1, 2, 3], behemoth technology companies allocate significant resources toward machine learning endeavors [4, 5], and the economic and sociological consequences of artificial intelligence systems are frequent topics of debate among policymakers and pundits [6, 7, 8, 9].

Is this hype warranted? The recent excitement around deep learning may lead one to believe it is a relatively new technology, but this is not the case: neural networks have existed, in some form, along with the tools typically used to optimize them, for over half a century [10, 11].

The popularity of deep learning today is due largely to the availability of both powerful computation and huge, annotated datasets. In 2010, researchers at Stanford released the ImageNet database, consisting of 1.2 million images of 1,000 object classes, collected using extensive manual effort [12]. Since its inception, the ImageNet project has run an annual software contest to evaluate image-classification algorithms on the ImageNet database. In 2012, a deep neural network termed AlexNet drastically outperformed competing strategies, reinvigorating an otherwise relatively stale landscape of deep learning research [13].

AlexNet was an engineering triumph, demonstrating that neural network training could be significantly accelerated by GPUs. Since AlexNet, the ImageNet competition has been consistently won by convolutional neural networks of varying architecture, often trained with very specific hyperparameter and regularization settings. According to some measurements, recent winning models surpass even human performance [14].

An optimistic takeaway from the ImageNet competition is that neural networks can perform extremely well on machine learning problems. A more honest message, however, is that the success of these models relies more than we would like on their inclusion of meaningful inductive bias (like convolution), the availability of massive amounts of annotated data (like ImageNet), and the identifiability of high-performing hyperparameters (like learning rates and regularization settings).

Searching for optimal model architectures and hyperparameters is an iterative process, generally requiring both human expertise and large amounts of computation. This poses a problem: we want deep learning to enable breakthrough technologies and accelerate scientific innovation, but engineers and scientists may not have the expertise

to easily benefit from research in this space. If deep learning is to make a meaningful impact on society and science, it is necessary to take steps towards making its performance less reliant on ideal hyperparameters and large amounts of labeled data.

1.2 Contributions

In this thesis, we attempt to robustify the use of neural networks for sequential learning scenarios. In particular, we identify and seek to remedy some nuanced pathologies around the use of deep learning in these passive and active online learning regimes.

In Chapter 2 we discuss a motivating example, designing a machine learning solution to a statistical genomics problem. We use convolutional neural networks and canonical correlation analysis to identify meaningful relationships in paired genotype and phenotype data. At the end of the chapter, we note that in this space, labeled data are expensive to acquire, limiting the size of datasets we use and, as a consequence, the effectiveness of models trained on them.

Chapter 3 introduces an algorithm intended to alleviate problems of this variety. Our hyperparameter-free, batch-mode active learning approach, which we call BADGE, is built to request labels only for examples that are most informative to the model. We propose a novel embedding for unlabeled data that somewhat decouples each sample's identity from the learner's uncertainty about its label. Once in this space, we use a sampler that can strike a balance between batch diversity and predictive uncertainty. We show that while the relative performance between baseline acquisition

functions is often sensitive to things like model architecture, data type, and batch size, BADGE is consistently high performing.

In Chapter 4 we discuss how, while some active learning acquisition functions are more efficient than others, the primary computational bottleneck in deep active learning is model train time. This is because virtually all deep active learning work retrains the network from scratch each time new data are appended to the training set [15, 16, 17]. The more efficient approach, which is to initialize network parameters to those found in the previous round of active learning, tends to cause significantly worse generalization performance in comparison to randomly-initialized models. We show that this is not just a problem in active learning, but also passive online learning. We propose a simple hybrid initialization that remedies this inefficiency without sacrificing performance.

We defer our discussion of related work to Chapter 5. Finally, Chapter 6 overviews our findings and identifies interesting avenues for future work. The main chapters of this thesis have been published or are in preparation in the following venues:

[Chapter 2]: Jordan T. Ash*, Gregory Darnell*, Daniel Munro*, and Barbara E. Engelhardt. Joint analysis of gene expression levels and histological images identifies genes associated with tissue morphology. *Nature Communications*, 2020.

[Chapter 3]: Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John

Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *International Conference on Learning Representations*, 2020. (talk)

[Chapter 4]: Jordan T. Ash and Ryan P. Adams. On Warm-Starting Neural Network Training. *arXiv preprint*, 2020.

Chapter 2

A motivating example

The success of deep learning in the past several years has spurred a plethora of exciting research on its application to problems in medicine [18, 19, 20, 21]. This chapter broadens research in this space by introducing a machine learning approach for studying associations between paired genotype and phenotype data.

Our goal is to associate histological image phenotypes with high-dimensional genomic markers; the limitations to incorporating histological image phenotypes in genomic studies are that relevant image features are difficult to identify and extract in an automated way. We use convolutional autoencoders and sparse canonical correlation analysis (CCA) on histological images and gene expression levels from paired samples to find subsets of genes whose expression values in a tissue sample correlate with subsets of morphological features from the corresponding sample image.

2.1 Introduction

Histological and histopathological images—high-resolution microscopic images of healthy or diseased tissue samples that have been sectioned and stained—are essential for identifying and characterizing complex phenotypes. Pathologists study tissues using histological imaging techniques for scientific research on cellular morphology and tissue structure and for medical practice. For example, visual inspection of biopsied tissue is a major component of cancer diagnosis, since cancer is known to affect the morphological properties of tissues, including extracellular structure and cell size, shape, and organization [22].

There has been considerable research in computationally analyzing pathological image data to develop automated cancer diagnoses. Earlier approaches typically involved the extraction of predetermined morphological, textural, and fractal image features from histological images [23]. The resulting image feature vectors then are used to classify the pathological status of the sample [24]. Because this feature extraction relies on human-defined features, challenges arise as a result of cross-tumor heterogeneity and the variance inherent in histology and pathology [25].

Complementary to visual inspection of histological images, gene expression can be measured to study cellular activity on the molecular level. Bulk gene expression levels have been used to characterize and understand cellular differences between sample tissues [26], disease phenotypes [27], environments [28], or exposures [29]. Current work has mainly focused on finding genotypes and gene expression levels associated with disease phenotypes [30, 26]. Single cell imaging studies have begun to study the connection between expression and cellular morphology [31, 32], but throughput,

number of transcripts imaged, number of cells, and image analysis pose challenges to this technology as an all-purpose solution currently. More generally, analyses to identify sets of genes whose expression levels are correlated with cellular physiology and tissue phenotypes would enable investigation into both basic cellular biology and drivers of cellular morphology associated with disease. Here, we are interested in identifying genes and genotypes associated with stained images of tissue sections.

Association studies involving histological image data, in contrast to predictive and diagnostic studies, have not been broadly undertaken, despite their importance. This is largely because it is not clear how to identify biologically relevant features automatically from histological images. Previous work on this subject involved extracting hand-engineered features from images and computing pairwise correlations with gene expression data [33]. Methods exist to analyze histological images automatically, but often these methods extract image features that are not associated with genomic features [34].

In this work, we address this difficulty in a framework called ImageCCA, which relies on the automatic extraction of image features using a convolutional autoencoder (CAE) [35]. A CAE is an unsupervised deep learning method that produces a small set of numeric features that can be used to characterize an image [35]. These image feature representations are intended to capture variance in the image as a whole, but, as discussed later, we can also produce image features that are predictive of class labels, such as tumor versus healthy samples.

We use sparse canonical correlation analysis (CCA) to identify correlated sets of genes and learned histological image features. CCA finds linear mappings from paired

samples into a shared low-dimensional space for which these observations maximally correlate [36]. Because we expect only a small subset of genes to contribute to a particular kind of morphological variation, we use a sparse form of CCA to identify small subsets of genes and image features whose values correlate most strongly with each other. CCA can be thought of as jointly modeling and partitioning the contributors to variance in the gene expression levels and image features. A single CCA component—capturing variation in samples due to a subset of genes and image features—implicitly removes this variation from signals captured in other components. We interpret the variation captured in CCA components by examining both the enriched molecular functions of their selected genes and the cellular morphology of their selected image features.

This chapter proceeds as follows. First, we give an overview of ImageCCA for the joint analysis of paired gene expression and histological image data. Next, we apply this framework to three datasets with histological images and gene expression levels on paired samples. We demonstrate the biological significance of the resulting associations using functional analyses of the subsets of genes that correlate with image features.

2.2 ImageCCA for gene-image associations

ImageCCA correlates extracted image features with paired gene expression levels to faclilitate the study of associations between cellular morphology and gene expression levels. In our scenario, the model has access to paired samples $X \in \{x_1, x_2, ..., x_n\}$ and

$Y \in \{y_1, y_2, ..., y_n\}$, where X is a dataset of images of tissues and Y is a corresponding dataset of gene expressions. Each image $x_i \in \mathbb{R}^{d \times d \times 3}$ is a color (three channel), $d \times d$-pixel photograph of cells from a particular tissue. Gene expressions for those same cells are represented in the corresponding sample $y_i \in \mathbb{R}^m$, where each dimension denotes the amount by which a particular gene is expressed.

The goal of ImageCCA is to correlate image features with gene expression data. Solving this problem successfully would allow us to garner insight into the genetic underpinnings of cellular morphologies. We propose to learn concise image representations of each image x_i, then to use sparse canonical correlation analysis to correlate this learned representation with the corresponding sample y_i.

2.2.1 Representation Learning

Our method has both supervised and unsupervised variants, which differ only in the way image representations are learned. In the unsupervised variant, image representations are learned using a convolutional autoencoder (Figure 2.1).

The CAE is used to embed these images into a 1,024-dimensional space [13]. Embeddings from the CAE are learned with the objective of reconstructing the original image as accurately as possible—the model's loss is the ℓ_2 distance between original and reconstructed images, relying on the 1,024 representative image features learned by the encoder. The low-dimensional representation encodes visual properties of images without regard to annotations like cancer status or tissue type. In this feature space, images with similar morphological features tend to be closer to each

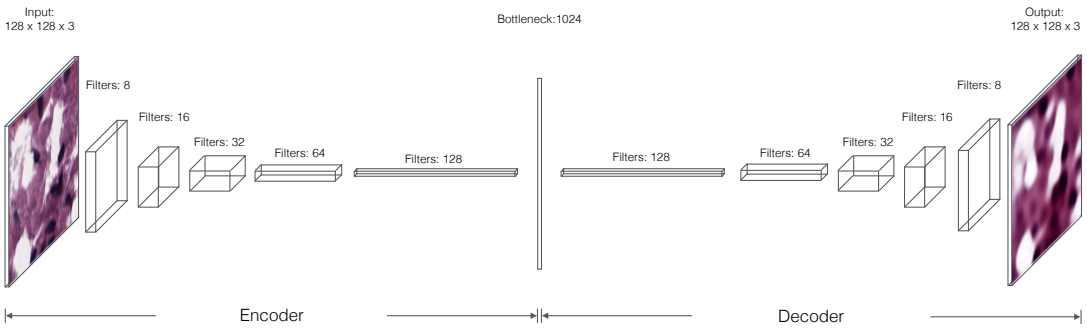


Figure 2.1: A convolutional autoencoder trained to reconstruct tissue samples from the BRCA dataset. Each convolutional layer of the encoder includes 5×5 filters followed by 2×2 max pooling and rectified linear (ReLU) activations. The final layer of the encoder, which produces our embedding, is fully connected to a layer of 1024 units. Each convolutional layer in the decoder is upsampled $2\times$ before again applying ReLU nonlinearities. The first convolutional layer of the decoder is linearly projected and reshaped from the bottleneck layer.

other in Euclidean space, while images with dissimilar features tend to be farther apart (Figure 2.3).

Many of the datasets we consider also include annotations for each image, specifying, for example, cancer type. The feature representation from the CAE quantifies many types of histological variance, but we are often interested primarily in the morphological differences between these kinds of pathological states. To capture these differences in our embeddings, we added a multilayer perceptron (MLP) to the pre-trained encoding portion of the CAE, and trained the MLP to distinguish histological images according to the labels in the dataset. The MLP adds a supervision signal to the feature extraction process. The modified image representation will identify features that are useful for classification—for example, distinguishing morphological

features of tumors versus healthy tissues—rather than for image reconstruction. In this setting we use penultimate layer activations of the MLP to represent samples.

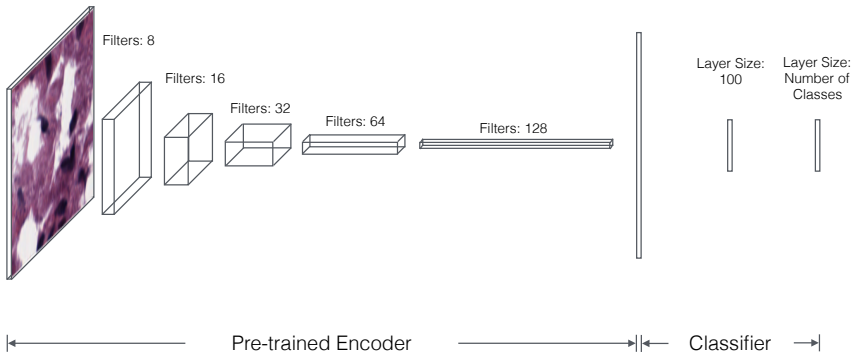


Figure 2.2: Architecture of the feature extraction model for supervised ImageCCA. The pre-trained encoder is attached to two fully-connected layers to allow for classification. The first classification layer features 128 ReLU units, and the second has as many neurons as there are classes with softmax activation (for multi-class problems) or a single sigmoid unit (for binary classification problems).

2.2.2 Canonical Correlation Analysis

Canonical Correlation Analysis identifies linear mappings from paired samples into a shared low-dimensional space for which these observations maximally correlate [36].

CCA is given an $n \times d$ matrix of image embeddings X^e (representing n samples of dimensions d produced by the encoding portion autoencoder), and a corresponding $n \times g$ matrix of gene expression levels Y (representing n samples of dimension g). The first iteration of CCA aims to identify vectors a_1 and b_1 that maximize $corr(X^e a_1, Y b_1)$. The procedure is repeated $k \leq \min\{d, g\}$ times. Proceeding iterations i of CCA

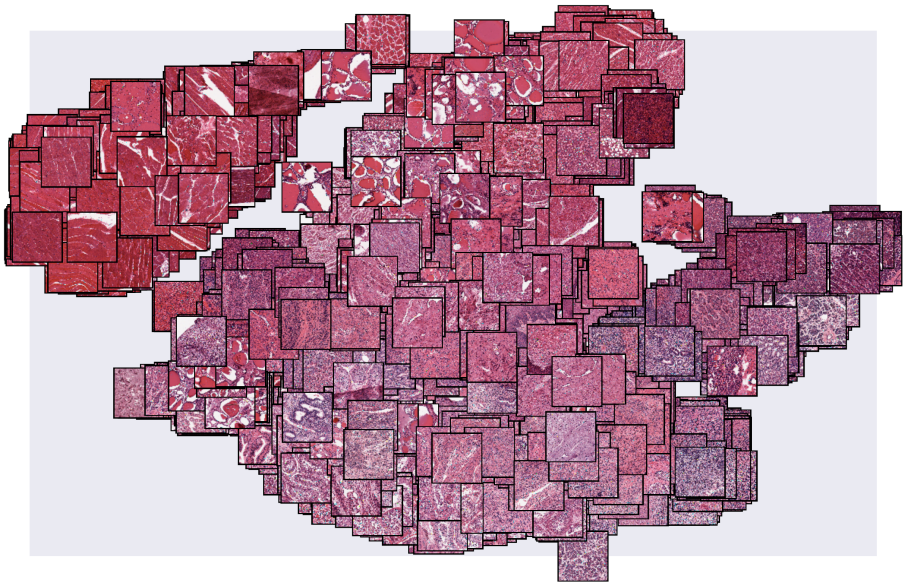


Figure 2.3: Embeddings of GTEx histological images (discussed later) learned by a CAE. Samples are visualized using t-SNE. Images with similar morphological features are closer together, with muscle tissues forming a cluster distinct from the remaining tissue types in the upper left corner

follow this same procedure, under the additional constraint that the i-th set of CCA components, Xa_i and Yb_i, are uncorrelated with previous components (i.e. $Xa_i \cdot Xa_j = 0,\ Yb_i \cdot Yb_j = 0,\ \forall i \neq j$).

In its typical form, CCA results in canonical variates a_i and b_i that are dense. We employ a sparse version of CCA which incorporates L_1 penalties λ_1 and λ_2 respectively on the values in a_i and b_i. The advantage of the resulting sparsification is twofold. First, it regularizes the components, better ensuring $corr(Aa_i, Bb_i)$ to be large even when samples in A and B are different from but similarly-distributed to those in X^e and Y respectively. Second, it makes results more interperatable—for the i-th set of canonical variables, non-zero elements of b_i correspond to genes that best correlate

with learned image features that have non-zero elements in a_i. This interpretability informs our discussion in the next section.

2.3 Results

We applied ImageCCA to three datasets. In each, we use Gene Ontology (GO) [37] analysis to identify functions of the group of genes selected by sparse CCA (genes that correspond to non-zero values of b_i) for a particular component. GO terms were identified using the `topGO` [38], `org.Hs.eg.db` [39] and `GO.db` [40] R packages. We present these GO terms and show examples of images that most strongly represent the image features selected by that same component.

2.3.1 Tissue Datasets

BRCA First, we applied our method to data from the Cancer Genome Atlas (TCGA) Breast Invasive Carcinoma (BRCA) study [41]. We used 1,541 histological images from 1,106 tissue biopsy samples, taken from 1,073 breast cancer patients. Of these, 1,502 image samples were of primary tumors, 7 were of metastatic tumors, and 32 were of normal tissue. Primary and metastatic tumor samples were grouped into a single tumor class label, in contrast to a normal label, to train the classifier in the supervised version of our approach. Details of the data collection and preparation can be found in the original study [41].

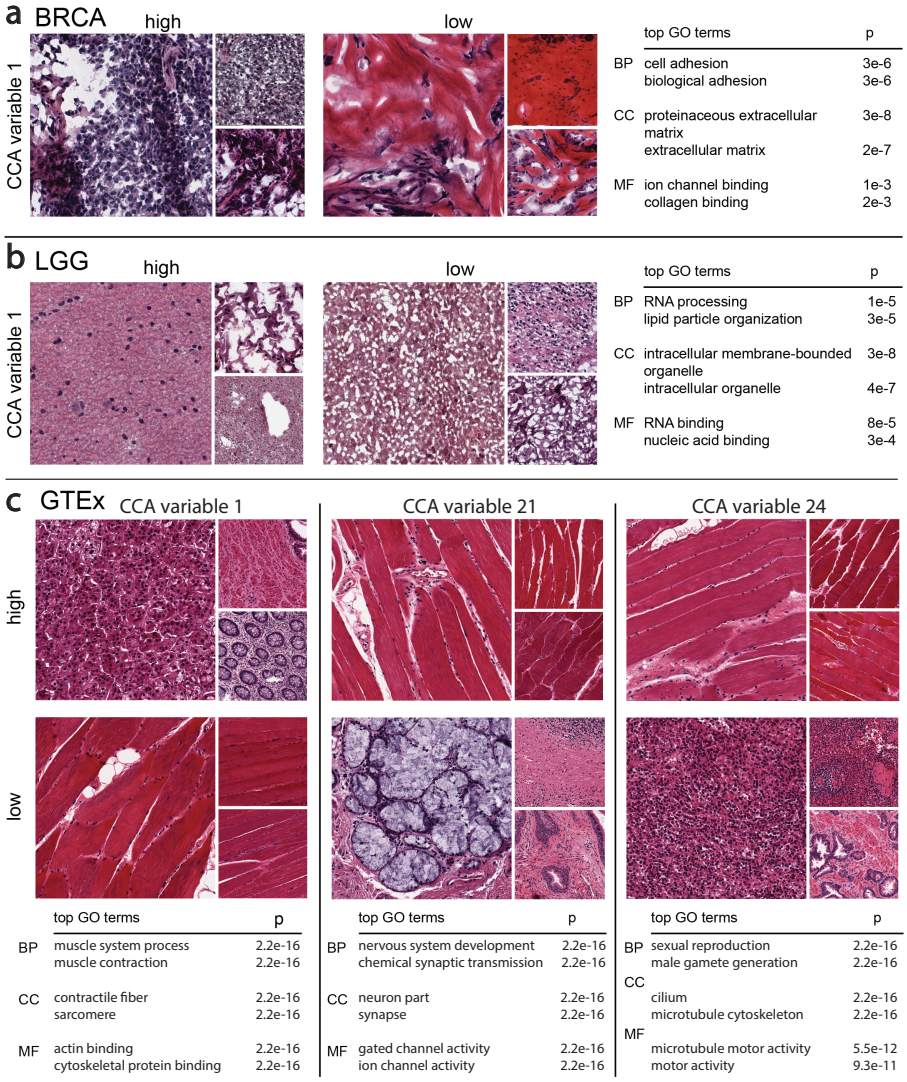
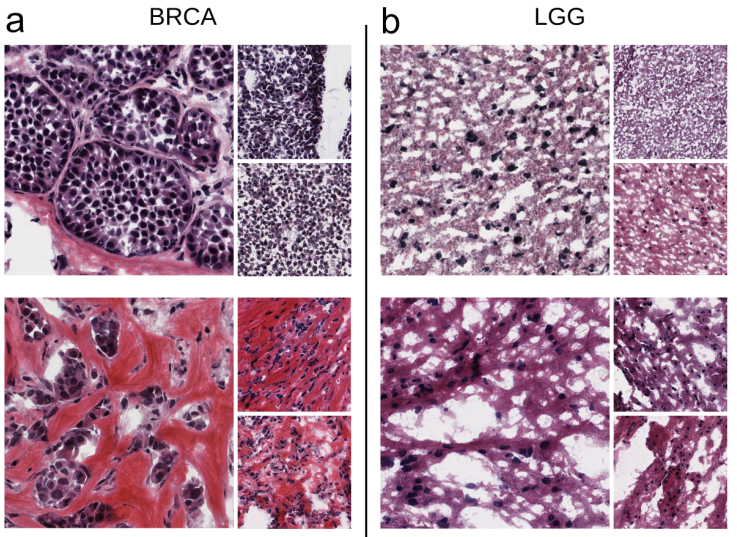


Figure 2.4: Results using ImageCCA for three different datasets. We report images sampled from those with the most extreme (top and bottom 10%) CCA variable values, and top two GO terms that are most enriched with the corresponding genes with extreme loading values in the same component. BP is Biological Process; CC is Cellular Component; MF is Molecular Function. The p-values reported are uncorrected Fisher's exact test. Panel a: the first component of the BRCA ImageCCA results; Panel b: the first component of the LGG ImageCCA results; Panel c: Three interesting components of the GTEx ImageCCA results.

Figure 2.4a shows the top CCA component was enriched for genes involved in *cell adhesion* found in the *proteinaceous extracellular matrix* with molecular functions related to *ion channel binding* and *collagen binding*. The p-values reported for all GO terms are from Fisher's exact test. Figure 2.4a also displays images associated with the most extreme positive and negative values of the component, demonstrating dramatic differences in the structure of stained tissues. In particular, images with high magnitude positive values have well-differentiated nuclei (dark purple spots) and minimal extracellular connective tissue, whereas images with high magnitude negative values have few nuclei and a dramatic presence of extracellular connective tissue (pink colors). This component appears to capture differences in the extracellular connective tissue structure, reflected in the extreme-valued histological images and the GO functional terms enriched in the subset of non-zero genes. Components estimated using supervised ImageCCA, which refines learned representations using an MLP stacked on the pre-trained encoder, are well correlated with those from their unsupervised counterpart (Figure 2.5a).

LGG Next, we applied ImageCCA to samples from the TCGA Brain Lower Grade Glioma (LGG) study data, which includes both primary and recurrent tumor types [42]. These data include 484 images from 401 tissue biopsy samples taken from 392 lower grade glioma patients. Of these, 471 images were derived from 388 primary tumor samples, and 13 images were derived from 13 recurrent tumor samples. The class labels used for supervised ImageCCA were primary tumor and recurrent tumor.

In the unsupervised setting, ImageCCA components selected an average of 228 genes and 31 image features. In the supervised setting, these components selected an average of 399 genes and 5 image features.



	top GO terms	p
BP	extracellular matrix organization	3e-12
	extracellular structure organization	4e-12
CC	extracellular matrix	2e-16
	proteinaceous extracellular matrix	1e-15
MF	integrin binding	1e-5
	cell adhesion molecule binding	2e-4

	top GO terms	p
BP	establishment of cell polarity	1e-4
	organelle fusion	2e-4
CC	bounding membrane of organelle	1e-4
	organelle membrane	4e-4
MF	GTP binding	1e-3
	poly(A) RNA binding	2e-3

Figure 2.5: Results using a CAE with an MLP, to add supervision to estimate an image embedding. We report images sampled from those with the most extreme (top and bottom 10%) loading values, and top two GO terms that are most enriched with the corresponding genes with extreme loading values in the same component. BP is *Biological Process*; CC is *Cellular Component*; MF is *Molecular Function*. The p-values reported are uncorrected Fisher's exact test. Panel a: the first component of the BRCA data; Panel b: the first component of the LGG data.

We performed GO term enrichment tests with the subsets of non-zero genes for each unsupervised LGG component. Enriched terms for the first component are

indicative of RNA metabolism (Fig 2.4b). In particular, this component included genes enriched for *RNA processing*, *lipid particle organization*, and *regulation of DNA metabolic process*.

Table 2.1 shows GO annotations for the first 9 CCA components computed with ImageCCA. The second component included genes enriched for *synaptic transmission*, *synaptic signaling*, *trans-synaptic signaling*, and *cell-cell signaling*. This second component included 77 genes and 38 image features. Many of the genes in this list are only expressed in brain tissues.

GTEx Finally, we ran our method on data from the Genotype-Tissue Expression (GTEx) project [26]. These data include 2,221 samples from 499 different individuals of 29 different types of non-diseased tissue. Each component included an average of 1,054 genes and 148 image features. We did not run ImageCCA in the supervised setting for GTEx.

In the GTEx dataset, many of the unsupervised ImageCCA components capture image features and genes specific to a tissue. For example, the first component differentiates skeletal muscle tissue on one extreme from pancreatic tissues on the other extreme via muscle-specific genes (Figure 2.4c); the two tissue types have distinct morphology.

The twenty-first component from unsupervised ImageCCA distinguishes cerebellum and cerebral cortex tissues from other tissue types (Figure 2.4c). The extreme valued cerebellum and cerebral cortex images include tissues with uniform neurons and densely packed nuclei, while images on the other extreme are either skeletal or pancreatic muscle. Genes in this component are enriched for terms related to

Table 2.1: Enriched GO terms for genes selected by sparse CCA in the LGG data. Enriched Biological Process GO terms were found separately for each gene set contributing to the first nine CCA components for the LGG data. Only the four most enriched terms per gene set are shown. Uncorrected p-values for the Fisher's exact test are reported.

CCA var	GO ID	term	p-value
1	GO:0006396	RNA processing	1.4e-5
	GO:0034389	lipid particle organization	2.9e-5
	GO:0051052	regulation of DNA metabolic process	6.7e-5
	GO:0051054	positive regulation of DNA metabolic processes	8.6e-5
2	GO:0007268	synaptic transmission	1.3e-23
	GO:0099536	synaptic signaling	1.3e-23
	GO:0099537	trans-synaptic signaling	1.3e-23
	GO:0007267	cell-cell signaling	5.6e-18
3	GO:0007272	ensheathment of neurons	5.4e-8
	GO:0008366	axon ensheathment	5.4e-8
	GO:0042552	myelination	8e-7
	GO:0032060	bleb assembly	1.4e-5
4	GO:0006396	RNA processing	6.5e-11
	GO:0090304	nucleic acid metabolic process	5.5e-10
	GO:0034641	cellular nitrogen compound metabolic pro...	7.5e-9
	GO:0006807	nitrogen compound metabolic process	1.1e-8
5	GO:0035589	G-protein coupled purinergic nucleotide ...	2.8e-7
	GO:0035590	purinergic nucleotide receptor signaling...	2e-6
	GO:0035588	G-protein coupled purinergic receptor si...	2.4e-6
	GO:0035587	purinergic receptor signaling pathway	8.3e-6
6	GO:0044802	single-organism membrane organization	3.6e-6
	GO:0006810	transport	7.3e-6
	GO:1902578	single-organism localization	7.3e-6
	GO:0044765	single-organism transport	7.4e-6
7	GO:0006811	ion transport	8.9e-7
	GO:0030029	actin filament-based process	9.3e-7
	GO:0044765	single-organism transport	1.5e-6
	GO:0048771	tissue remodeling	2.5e-6
8	GO:0010001	glial cell differentiation	1.6e-5
	GO:0048709	oligodendrocyte differentiation	3.1e-5
	GO:0042063	gliogenesis	8.7e-5
	GO:0042552	myelination	1.2e-4
9	GO:0006955	immune response	3.9e-29
	GO:0002376	immune system process	1.9e-27
	GO:0006952	defense response	2e-21
	GO:0002682	regulation of immune system process	1.1e-20

synaptic function , including *gated channel activity, chemical synaptic transmission*, and *anterograde trans-synaptic signaling*, and *synaptic membrane*. There are 1,360 genes associated with this component, and these genes tend to be expressed primarily in cerebellum and cerebral cortex.

The twenty-fourth component in the unsupervised ImageCCA distinguishes testis tissue from muscle tissue (Figure 2.4c). Genes in this component are enriched for terms related to spermatogenesis, including *sexual reproduction, male gamete generation, spermatogenesis,* and *gamete generation*, and are expressed primarily in testis samples.

2.4 Discussion

In this project, we developed an analysis framework for paired histological images and gene expression levels to identify the sets of genes that are associated with specific features of tissue appearance. We applied this framework to three sets of paired histological image and gene expression data: breast carcinoma samples, lower grade glioma samples, and GTEx v6 tissue samples. Applying the ImageCCA framework to these data, and interpreting the components, we were able to find genes known to influence cellular morphology.

Our results demonstrate that biologically meaningful correlations exist and can be identified between gene expression levels and features extracted from histological images. We have shown that the framework introduced here can be applied to both pathological and healthy tissue samples, and to both single tissue types and a mixture of types, to detect correlations between gene expression and image features. We

note that we identify correlations here and do not make causal statements about the relationship between gene expression and cellular morphology; experiments that modify cell shape find changes in gene expression levels [43].

Still, the connection between variation in gene expression levels and in the corresponding tissue image suggests that one can be used to aid in the analysis and prediction of the other. A pathologist who visually inspects tissue images for diagnostic purposes could confirm each observation using predicted expression values of the genes linked to the visible feature of interest. Conversely, in some cases clinically significant values in a patient's gene expression profile could be used to generate an encoding of the visual properties of the associated histological image. This study begins to address the question of how regulation of gene expression in tissues relates to tissue morphology and downstream organismal phenotypes.

2.5 Making it work

ImageCCA's results are promising, but its successful application relies on several important decisions. In this section, we will overview how CCA hyperparameters are chosen and how we mitigate the effects of having relatively few samples available for each dataset.

2.5.1 Hyperparameter Tuning

Sparse CCA requires setting three hyperparameters: λ_1, and λ_2, the amount of sparsity regularization applied to the image feature and gene expression matrix

respectively, and k, the number of CCA components. To select values, we performed a hyperparameter search on the LGG dataset. For held-out data, we used learned CCA variates to predict the gene matrix from the image feature matrix, or vice versa. In particular, we consider matrices A and B, respectively of size $d \times k$ and $g \times k$, where each column is a canonical variate. We then measure $\frac{1}{k}\sum_{i=1}^{k} corr(x_i^e, y_i BA^+)$ and $\frac{1}{k}\sum_{i=1}^{k} corr(x_i^e AB^+, y_i)$, the average correlation between gene expressions or image features and their reconstructions, computed via a Moore-Penrose pseudoinverse of the corresponding canonicalized dataset (Figure 2.6).

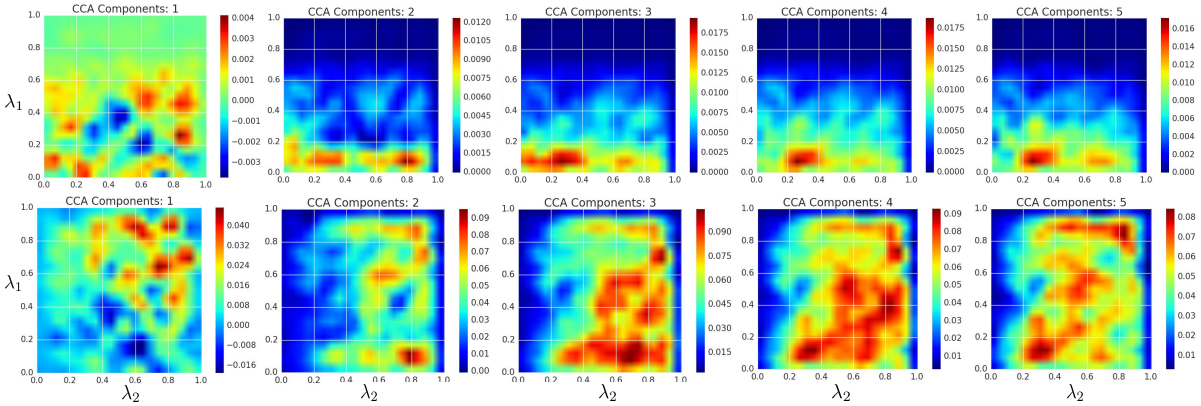


Figure 2.6: Correlation between features and reconstructed features for LGG data. Each heatmap shows correlation as a function of λ_1, λ_2, and the number of CCA components. Low values of λ increase the amount of sparsity used in CCA. Correlations were computed using held out data, different from those used to fit CCA. **Top row:** gene expression reconstructions. **Bottom row:** Image feature reconstructions.

In this work, we are primarily interested in selecting sparsity parameters that allow optimal reconstructions of images and gene expression levels (on held out data), that produce a small number of genes and image features per component, and that produce interpretable subsets of genes as quantified by GO term enrichment. Based

on these results, we fixed λ_1 (for image features) to 0.15 and λ_2 (for gene expression) to 0.05 for all experiments.

2.5.2 Image Subsampling

The datasets used in this study are, by machine learning standards, extremely small. Typical benchmark datasets used in deep learning research are in the tens of thousands or larger, while the largest dataset explored here includes only a couple thousand.

To combat this disadvantage, we train our autoencoder to reconstruct randomly-sampled 128 × 128-pixel crops of images, rather than their uncropped counterparts. Microscopy images are often more than a megapixel in size, so this approach allows us to artificially boost the number of samples available to the model.

Once the network was trained, each image was represented by randomly sampling a hundred 128 × 128 windows from it, embedding each using the encoding section of the CAE, and averaging those feature encodings (for $m = 100$ subsamples, $i \in \{1, \ldots, n\}$ samples, and $j \in \{1, \ldots, p\}$ image features):

$$x^e_{i,j} = \frac{1}{m} \sum_{\ell=1}^{m} \hat{x}^e_{i,j,\ell}, \tag{2.1}$$

where $x^e_{i,j}$ is the average learned representation for sample i and dimension j, and $\hat{x}^e_{i,j,\ell}$ is the ℓ-th sampling of the representation for sample i and dimension j.

Because the CAE is trained to reconstruct images as accurately as possible, some variance of the encoded samples are inevitably used to represent the locations of structures in image, while the remainder is used to represent the physical properties of

those structures. This averaged bag of features representation allows us to essentially integrate away the location-based information, while keeping information about the image properties in which we are most interested. The 1024-dimensional feature vector was the mean encoded feature value across the 100 image windows of each image. Finally, we whitened the averaged image representations using PCA [44]. We use the 1024 whitened features to represent the images in CCA. This procedure decorrelates each dimension of the feature space, which is helpful for interpreting the results of CCA.

2.5.3 Moving Forward

If we had access to much larger datasets on which to train our models, we would expect ImageCCA to provide even more biologically meaningful associations. In general, however, labeling medical data (in this case sequencing genes that correspond to our images) can be enormously expensive.

When constructing a dataset, we would want to maximize the performance of our models given a fixed labeling budget. How can we efficiently select samples that are maximally informative? This problem is referred to as active learning. In the next chapter, we will design an algorithm for this task when using neural network models. Chiefly, we will show that our approach is high performing and robust to hyperparameter choices and environmental conditions.

Chapter 3

Batch Active Learning by Diverse, Uncertain Gradient Lower Bounds

In this chapter, 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 uncertainty and diversity 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.

3.1 Introduction

In recent years, deep neural networks have produced state-of-the-art results on a variety of important supervised learning tasks. However, many of these 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 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 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 in which these models are applicable.

How should we design a practical, general-purpose, label-efficient active learning algorithm for deep neural networks? Theory for active learning suggests a version-space-based approach [45, 46], which explicitly or implicitly maintains a set of plausible models, and queries examples for which these models make different predictions. But when using highly expressive models like neural networks, these algorithms degenerate to querying every example. Further, the computational overhead of training deep neural networks precludes approaches that update the model to best fit data after each label query, as is often done (exactly or approximately) for linear methods [47, 48]. 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. In a batch setting, however, this creates a pathological scenario where samples in the selected 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 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. Further, what even constitutes a "large" 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.

Moreover, 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 expensive. As a result, 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 [49], 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 3.3 and experiments in Section 3.4. Our discussion of related work can be found in Section 5.1.

3.2 Notation and setting

Denote by $\mathcal{X}$ the instance space and by $\mathcal{Y}$ the label space. In this work we consider multiclass classification, so in a K-class problem, $\mathcal{Y} = [K]$. Denote by D the distribution from which examples are drawn and by $D_{\mathcal{X}}$ the unlabeled data distribution. We consider the pool-based active learning setup, where the learner has access to an unlabeled dataset U that is sampled according to $D_{\mathcal{X}}$ and can request labels for any $x \in U$. Given a classifier $h : \mathcal{X} \to \mathcal{Y}$, which maps from 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 accuracy, i.e. $1 - \mathbb{E}_D[\ell_{01}(h(x), y)] = \Pr_{(x,y)\sim D}(h(x) = y)$. The goal of

active learning is to find a classifier with large expected 0/1 accuracy using as few label queries as possible.

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, and our classifiers take the form $h_\theta(x) = \text{argmax}_{y \in [K]} f(x;\theta)_y$, where $f(x;\theta) \in \mathbb{R}^K$ is a vector of scores assigned to candidate labels, given the example x and parameters θ. At train time, we optimize model parameters by minimizing the cross-entropy loss $\mathbb{E}_S[\ell_{\text{CE}}(f(x;\theta), y)]$ over available labeled examples, where $\ell_{\text{CE}}(p, y) = \sum_{i=1}^{K} I(y = i) \ln 1/p_i$.

3.3 Algorithm

BADGE, described in Algorithm 1, starts by drawing an initial set of M examples uniformly at random from U and querying their labels. It then proceeds iteratively, performing two main computations at each step t: a *gradient embedding* computation and a *sampling* computation. Specifically, at each step t, for every x in the unlabeled pool U, we first 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 [49]. The algorithm queries the labels of these examples, retrains the model, and repeats.

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.

1: Labeled dataset $S \leftarrow M$ examples drawn uniformly at random from U together with queried labels.
2: Train an initial model θ_1 on S by minimizing $\mathbb{E}_S[\ell_{\text{CE}}(f(x;\theta), y)]$.
3: **for** $t = 1, 2, \ldots, T$: **do**
4: For all examples x in $U \setminus S$:

1. Compute its hypothetical label $\hat{y}(x) = h_{\theta_t}(x)$.
2. Compute gradient embedding $g_x = \frac{\partial}{\partial \theta_{\text{out}}} \ell_{\text{CE}}(f(x;\theta), \hat{y}(x))|_{\theta=\theta_t}$, where θ_{out} refers to parameters of the final (output) layer.

5: 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.
6: $S \leftarrow S \cup S_t$.
7: Train a model θ_{t+1} on S by minimizing $\mathbb{E}_S[\ell_{\text{CE}}(f(x;\theta), y)]$.
8: **end for**
9: **return** Final model θ_{T+1}.

The gradient embedding. Neural networks are optimized using gradient-based methods, so 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 parameters 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 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. The length of this hypothetical gradient vector captures a notion of 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

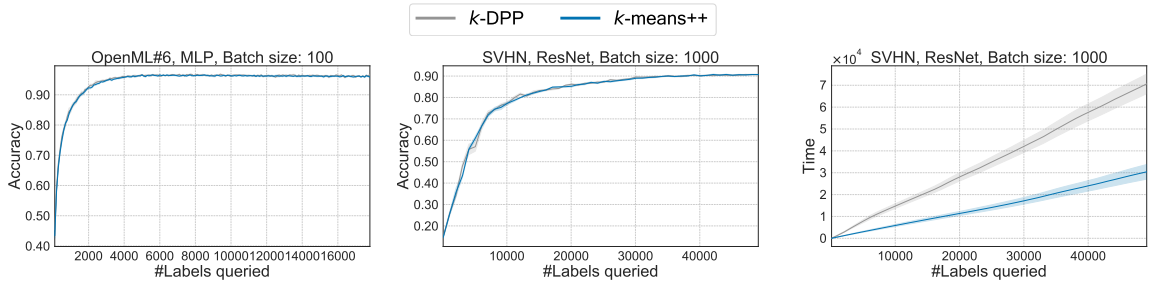


Figure 3.1: **Left and center**: Learning curves for k-MEANS++ and k-DPP sampling with gradient embeddings for different scenarios. The performance of the two sampling approaches nearly perfectly overlaps. **Right:** A run time comparison (seconds) corresponding to the middle scenario. Each line is the average over five independent experiments. Standard errors are shown by shaded regions.

have a small norm, and vice versa for samples where the model is uncertain (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 newly-acquired labeled samples to induce large and diverse changes to the model. To this end, in our gradient space, the selection procedure should 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 label could alleviate our uncertainty on all remaining $k-1$ samples.

A natural way of making this selection without introducing additional hyperparameters is to sample from a k-Determinantal Point Process (k-DPP; [50]). That is, to select a batch of k points with probability proportional to the determinant of their Gram matrix. In this process, when the batch size is very low, and linear

independence is easily achieved, the selection will naturally favor points with large norm, which corresponds to uncertainty in our space. When the batch size is large, the sampler focuses more on diversity because linear independence, which is more difficult to achieve for large k, is required to make the Gram determinant non-zero.

Unfortunately, sampling from a k-DPP is not trivial. Sampling algorithms typically rely on MCMC, where mixing time poses a significant computational hurdle [51, 52]. To overcome this, we suggest instead sampling using the k-MEANS++ seeding algorithm [49], originally made to produce a good initialization for k-means clustering. k-MEANS++ seeding selects centroids by iteratively sampling points in proportion to their squared distances from the nearest centroid that has already been chosen, which, like a k-DPP, tends to select a diverse batch of high-magnitude samples. For completeness, we give a formal description of the k-MEANS++ seeding algorithm in Appendix A.1.

Example: multiclass classification with softmax activations. Consider a neural network f where the last nonlinearity is a softmax, i.e. $\sigma(z)_i = {}^{e^{z_i}}/_{\sum_{j=1}^{K} e^{z_j}}$. Specifically, f is parametrized by $\theta = (W, V)$, where $\theta_{\text{out}} = W = (W_1, \ldots, W_K)^\top \in \mathbb{R}^{K \times d}$ are the weights of the last layer, and V consists of weights of all previous layers. This means that $f(x; \theta) = \sigma(W \cdot z(x; V))$, where z is the nonlinear function mapping 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

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

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 (i.e. the last-layer gradients corresponding to label i) is

$$(g_x)_i = \frac{\partial}{\partial W_i} \ell_{\text{CE}}(f(x; \theta), \hat{y}) = (p_i - I(\hat{y} = i)) z(x; V). \tag{3.1}$$

Based on this expression, we can make the following observations:

1. Each block of g_x is a scaling of $z(x; V)$, which is the output of the penultimate layer of the network. In this respect, g_x captures x's information similar to that of strategies that operate in this space [53].

2. Proposition 1 below 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. p is skewed towards a standard basis vector $e_{\hat{y}}$, then the vector $(p_i - I(\hat{y} = i))_{i=1}^K$ has small length. Therefore, g_x has small length as well. Such high-confidence examples tend to have gradient embeddings of small magnitude, which are unlikely to be repeatedly selected by k-MEANS++.

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

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

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

Proof. Observe that by Equation (3.1),

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

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

The k-MEANS++ sampler tends to produce diverse batches similar to a k-DPP. As shown in Figure 3.1, switching between the two samplers does not affect the active learner's statistical performance but greatly improves its computational performance. Appendix A.6 compares run time and test accuracy for both k-MEANS++ and k-DPP based sampling based on the gradient embeddings of unlabeled examples.

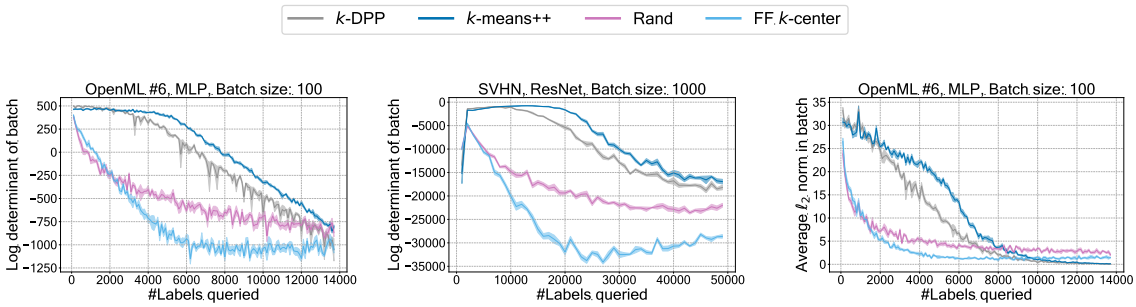


Figure 3.2: A comparison of batch selection algorithms using our gradient embedding. **Left and center:** Plots showing the log determinant of the Gram matrix of the selected batch of gradient embeddings as learning progresses. **Right:** The average embedding magnitude (a measurement of predictive uncertainty) in the selected batch. The FF-k-CENTER sampler finds points that are not as diverse or high-magnitude as other samplers. Notice also that k-MEANS++ tends to actually select samples that are both more diverse and higher-magnitude than a k-DPP, a potential pathology of the k-DPP's degree of stochastisity. Standard errors are shown by shaded regions.

Figure 3.2 illustrates the batch diversity and average gradient magnitude per selected batch for a variety of sampling strategies. As expected, both k-DPPs and k-MEANS++ tend to select samples that are diverse (as measured by the magnitude of their Gram determinant) and high magnitude. Other samplers, such as furthest-first traversal for k-Center clustering (FF-k-CENTER), do not seem to have this property. The FF-k-CENTER algorithm is the sampling choice of the CORESET approach to active learning, which we describe in the proceeding section [53].

3.4 Experiments

We evaluate the performance of BADGE against several algorithms from 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 performs representative sampling, the next three are uncertainty based, the fifth is a hybrid of representative and uncertainty-based approaches, and the last is traditional supervised learning.

1. CORESET: A diversity-based approach using coreset selection. The embedding of each example is computed by the network's penultimate layer and the samples at each round are selected using a greedy furthest-first traversal conditioned on all labeled examples [53].

2. CONF (Confidence Sampling): An uncertainty-based active learning algorithm that selects B examples with smallest predicted class probability, $\max_{i=1}^{K} f(x;\theta)_i$ [54].

3. MARG (Margin Sampling): An uncertainty-based active learning algorithm that selects the bottom B examples sorted according to their 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 $f(x;\theta)$ [55].

4. ENTROPY: An 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 $H(p) = \sum_{i=1}^K p_i \ln 1/p_i$ [54].

5. ALBL (Active Learning by Learning): A bandit-style meta-active learning algorithm that selects between CORESET and CONF at every round [56].

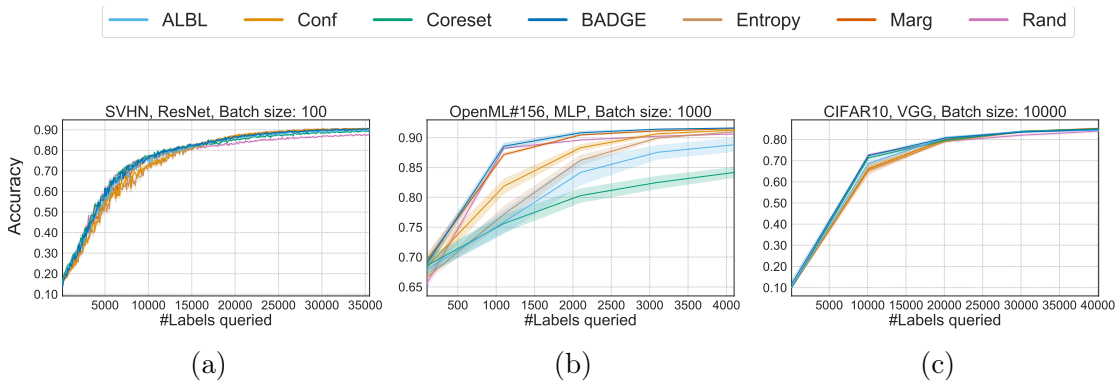
6. RAND: The naive baseline of randomly selecting k examples to query at each round.

We consider three neural network architectures: a two-layer Perceptron with ReLU activations (MLP), an 18-layer convolutional ResNet [57], and an 11-layer VGG network [58]. We evaluate our algorithms using three image datasets, SVHN [59], CIFAR-10 [60] and MNIST [61] [1], and four non-image datasets from the OpenML repository (#6, #155, #156, and #184). [2] We study each situation with 7 active learning algorithms, including BADGE, making for 231 total experiments.

For image datasets, the embedding dimensionality in the MLP is 256, while for OpenML datasets, the embedding dimensionality of the MLP is 1024, as more capacity helps the model fit training data. We train models using cross-entropy loss and the Adam variant of SGD until training accuracy exceeds 99%. We use a learning rate of 0.001 for image data and of 0.0001 for non-image data. We avoid warm starting and retrain models from scratch every time new samples are queried [62]. All

[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.



(a) (b) (c)

Figure 3.3: Active learning test accuracy versus the number of total labeled samples for a range of conditions. Standard errors are shown by shaded regions.

experiments are repeated five times. No learning rate schedules or data augmentation are used. Baselines use implementations from the libact library [63]. All models are trained in PyTorch [64].

Learning curves. Here we show 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 Figure 3.3a, which shows CORESET outperforming confidence-based methods at first, but then doing worse than these methods later on.

In this figure, BADGE performs as well as diversity 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 architectural 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 3.3b). That is, the diversity induced by unconditional random sampling can often yield a batch that better represents the data. 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 3.3c). Comprehensive plots of this kind, spanning architecture, dataset, and batch size are in Appendix A.2.

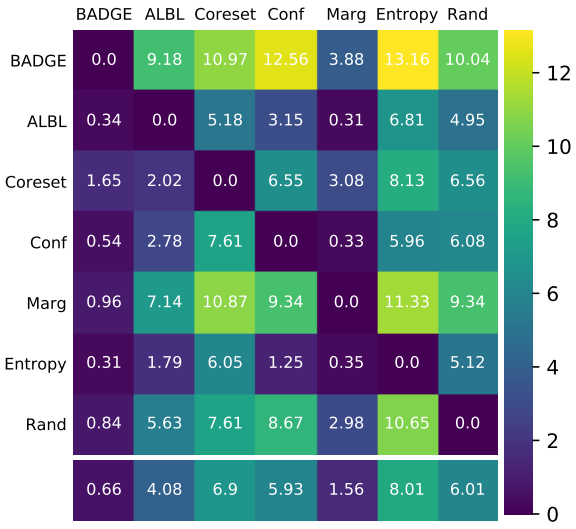


Figure 3.4: A pairwise penalty matrix over all experiments. Element $P_{i,j}$ corresponds roughly to the number of times algorithm i outperforms algorithm j. Column-wise averages at the bottom show overall performance (lower is better).

Pairwise comparisons. We next show a comprehensive pairwise comparison 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 label budgets are large enough, all algorithms eventually reach similar performance, making the comparison between them uninteresting in the large sample limit. For this

reason, for each combination of (D, B, A), we select a set of labeling budgets L where learning is still progressing. We experimented with three different batch sizes and eleven dataset-architecture pairs, making the total number of (D, B, A) combinations $3 \times 11 = 33$. 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 $\{M + 2^{m-1}B : m \in [\lfloor \log((n_0 - M)/B) \rfloor]\}$. The calculation of scores in the penalty matrix P follows the following protocol: For each (D, B, A, L) combination and each pair of algorithms (i, j), we have 5 test errors (one for each repeated run), $\{e_i^1, \ldots, e_i^5\}$ and $\{e_j^1, \ldots, e_j^5\}$ respectively. We compute the t-score as $t = \sqrt{5}\hat{\mu}/\hat{\sigma}$, where

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

We use the two-sided t-test to compare pairs of algorithms: algorithm i is said to *beat* algorithm j in this setting if $t > 2.776$ (the critical point of p-value being 0.05), and similarly algorithm j beats algorithm i if $t < -2.776$. 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 is assigned equal influence

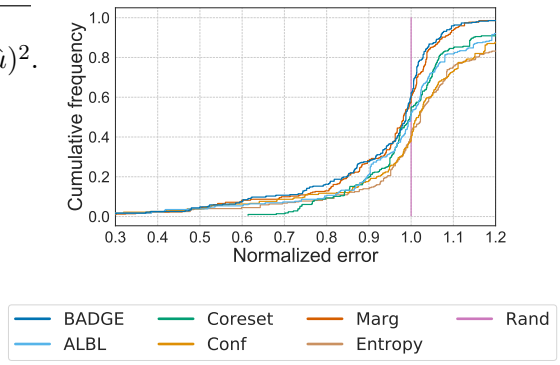


Figure 3.5: The cumulative distribution function of normalized errors for all acquisition functions.

in the aggregated matrix. Therefore, the largest entry of P is at most 33, the total number of (D, B, A) combinations. Intuitively, each row i indicates the number of settings in which algorithm i beats other algorithms and each column j indicates the number of settings in which algorithm j is beaten by another algorithm.

The penalty matrix in Figure 3.4 summarizes all experiments, showing that BADGE generally outperforms baselines. Matrices grouped by batch size and architecture in Appendix A.3 show a similar trend.

Cumulative distribution functions of normalized errors. For each (D, B, A, L) combination, we compute the average error for each algorithm i as $\bar{e}_i = \frac{1}{5}\sum_{l=1}^{5} e_i^l$. 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 $\text{ne}_i = \bar{e}_i/\bar{e}_r$, where r is the index of the RAND algorithm. By definition, the normalized errors of the RAND algorithm are identically 1 in all settings. Like with penalty matrices, for each (D, B, A) combination, we only consider a subset of L values from the set $\{M + 2^{m-1}B : m \in [\lfloor \log((n_0 - M)/B) \rfloor]\}$. 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.

We plot the generated CDFs in Figures 3.5, A.17 and A.18. We can see from Figure 3.5 that BADGE has the best overall performance. In addition, from Figures A.17 and A.18 in Appendix A.4, we can conclude that when batch size is small

(100 or 1000) or when an MLP is used, both BADGE and MARG perform best. 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.

3.5 Discussion

We have established that BADGE is empirically an effective deep active learning algorithm across different architectures, datasets, and batch sizes, generally performing similar to or better than other active learning algorithms.

Even though we designed BADGE with efficiency in mind (i.e. substituting k-DPP sampling with k-MEANS++ sampling), we find that active learning experiments, like those shown in this chapter, can take an extremely long time to execute. For example, a single active learning experiment, involving iteratively querying batches of 100 samples from the CIFAR-10 dataset until it has been entirely labeled, can take more than six days, regardless of acquisition function choice, even with state-of-the-art hardware and well-written software.

This is because virtually all deep active learning work retrains the network from scratch every time new labeled data are appended to the training set [15, 16, 17]. The more efficient approach, which is to initialize network parameters to those found in the previous round of active learning, tends to cause significantly worse generalization performance in comparison to randomly-initialized models. This property is especially troubling because training performance is similar regardless of the network's initialization scheme.

In the next chapter we explore this phenomenon, which we call the warm-start problem, in detail. We will show that this is not a problem with just active learning, but with any situation in which data arrive incrementally. Standard techniques, such as regularizing model optimization or varying hyperparameters, are unable to both close this generalization gap and decrease convergence time in comparison to training from scratch. We will introduce a simple initialization scheme that corrects this pathology.

Chapter 4

On Warm-Starting Neural Network Optimization

In many real-world deployments of machine learning systems, data arrive piecemeal. These learning scenarios may be passive, where data arrive incrementally due to structural properties of the problem (e.g., daily financial data) or active, where samples are selected according to a measure of their quality (e.g., experimental design). In both of these cases, we are building a sequence of models that incorporate an increasing amount of data. We would like each of these models in the sequence to be performant and take advantage of all the data that are available to that point. Conventional intuition suggests that when solving a sequence of related optimization problems of this form, it should be possible to initialize using the solution of the previous iterate—to "warm start" the optimization rather than initialize from scratch—and see reductions in wall-clock time. However, in practice this warm-starting seems

to yield poorer generalization performance than models that have fresh random initializations, even though the final training losses are similar. While it appears that some hyperparameter settings allow a practitioner to close this generalization gap, they seem to only do so in regimes that damage the wall-clock gains of the warm start. Nevertheless, it is highly desirable to be able to warm-start neural network training, as it would dramatically reduce the resource usage associated with the construction of performant deep learning systems. In this work, we take a closer look at this empirical phenomenon and try to understand when and how it occurs. We also provide a surprisingly simple trick that overcomes this pathology in several important situations, and present experiments that elucidate some of its properties.

4.1 Introduction

Although machine learning research generally assumes a fixed set of training data, real life is more complicated. One common scenario is where a production ML system must be constantly updated with new data. This situation occurs in finance, online advertising, recommendation systems, fraud detection, and many other domains where machine learning systems are used for prediction and decision making in the real world [65, 66, 67]. When the new data arrive, the model needs to be updated so that it can be as accurate as possible and to also account for any domain shift that is occurring.

As a concrete example, consider a large-scale social media website, to which users are constantly uploading images and text. The company requires up-to-the-minute

predictive models in order to recommend content, filter out inappropriate media, and select advertisements. There might be millions of new data arriving every day, which need to be rapidly incorporated into the production ML pipelines.

It is natural in this scenario to imagine maintaining a single model that is updated with the latest data at a regular cadence. Every day, for example, new training might be performed on the model with the updated, larger data set. Ideally, this new training procedure is initialized from the parameters of yesterday's model, i.e., it is "warm-started" from those parameters rather than a fresh initialization. Such an initialization makes intuitive sense: the data used yesterday are mostly the same as the data today, and it seems wasteful to throw away all the previous computation. For convex optimization problems, such warm starting is widely used and highly successful, e.g., [65]; the theoretical properties of online learning are well understood. However, warm-starting seems to hurt generalization in deep neural networks. This is particularly troubling, because warm-starting *does not* damage training accuracy.

Figure 4.1 illustrates this phenomenon. Three 18-layer ResNets have been trained on the CIFAR-10 natural image classification task to create these figures. One was trained on 100% of the data, one was trained on 50% of the data, and a third warm-started model was trained on 100% of the data but initialized from the parameters found from the 50% training. All three achieve the upper bound on training accuracy. However, the warm-started network performs worse on test samples than the network trained on the same data but with a good random initialization. Problematically, this phenomenon incentivizes performance-focused researchers and engineers to constantly retrain models from scratch, at potentially enormous financial and environmental

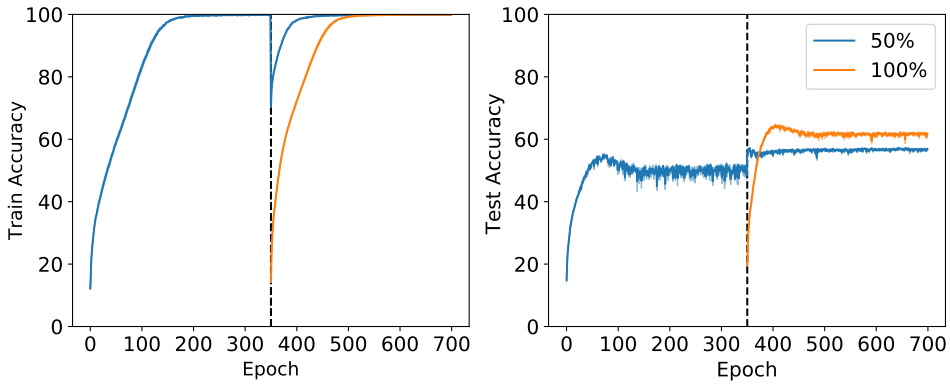


Figure 4.1: Comparison between ResNets trained using a warm start and a random initialization on CIFAR-10. Blue lines are models trained on 50% of CIFAR-10 for 350 epochs then trained on 100% of the data for a further 350 epochs. Orange lines are models trained on 100% of the data from the start. The two procedures produce similar training performance but differing test performance.

cost [68]; this is an example of "Red AI" [69], disregarding resource consumption in the pursuit of raw predictive performance.

The warm-start phenomenon has implications for other situations as well. In active learning, for example, unlabeled samples are abundant but labels are expensive: the goal is to identify maximally-informative data and have those labeled by an oracle. Ideally, these decisions would use all the samples that had been seen so far [70], and it would be time efficient to simply update the model with each new selection. However, this seems to damage generalization. Although this phenomenon has not received much direct attention from the research community, it is common in deep active learning to retrain at every step [15, 71]. Popular deep active learning repositories on Github [16, 17] also retrain models from scratch.

The ineffectiveness of warm-starting has been observed anecdotally in the community, but this paper seeks to examine its properties more closely in controlled settings. Note that the findings in this paper are not inconsistent with extensive work on unsupervised pre-training [72, 73] and transfer learning in the small-data and "few shot" regimes [74, 75, 76, 77]. Rather here we are examining how to accelerate training in the large-data supervised regime in a way that is consistent with expectations from convex problems.

This chapter is structured as follows. Section 4.2 examines the generalization gap induced by warm-starting neural networks. Section 4.3 surveys approaches for improving generalization in deep learning, and shows that these techniques do not resolve the problem. In Section 4.4, we describe a simple trick that overcomes this pathology, and report on experiments that give insights into its behavior. We defer discussion of related work to Section 5.2.

4.2 Warm Starting Damages Generalization

In this section we provide empirical evidence that warm starting consistently damages generalization performance in neural networks. We conduct a series of experiments across a several different architectures, optimizers, and image datasets. Our goal is to create simple, reproducible settings in which the warm-starting phenomenon is observed.

Table 4.1: Validation percent accuracies for various optimizers and models for both warm-started and randomly initialized models on various indicated datasets. We consider an 18-layer ResNet, three-layer multilayer perceptron (MLP), and logistic regression (LR) as our classifiers. Validation sets are a randomly-chosen third of the training data. Standard deviations are indicated parenthetically. Accuracies for the first-round of training warm-started models are in Apendix Table B.1.

	ResNet	ResNet	MLP	MLP	LR	LR
CIFAR-10	SGD	Adam	SGD	Adam	SGD	Adam
Random Init	56.2 (1.0)	78.0 (0.6)	39.0 (0.2)	39.4 (0.1)	40.5 (0.6)	33.8 (0.6)
Warm Start	51.7 (0.9)	74.4 (0.9)	37.4 (0.2)	36.1 (0.3)	39.6 (0.2)	33.3 (0.2)
SVHN						
Random Init	89.4 (0.1)	93.6 (0.2)	76.5 (0.3)	76.7 (0.4)	28.0 (0.2)	22.4 (1.3)
Warm Start	87.5 (0.7)	93.5 (0.4)	75.4 (0.1)	69.4 (0.6)	28.0 (0.3)	22.2 (0.9)
CIFAR-100						
Random Init	18.2 (0.3)	41.4 (0.2)	10.3 (0.2)	11.6 (0.2)	16.9 (0.18)	10.2 (0.4)
Warm Start	15.5 (0.3)	35.0 (1.2)	9.4 (0.0)	9.9 (0.1)	16.3 (0.28)	9.9 (0.3)

4.2.1 Basic Batch Updating

Here we consider the simplest case of warm-starting, in which a single training dataset is partitioned into two subsets that are presented sequentially. In each series of experiments, we randomly segment the training data into two batches. The model is trained to convergence on the first half, then is trained on the union of the two batches, i.e., 100% of the data. This is repeated for three classifiers: ResNet-18 [57], a multilayer perceptron (MLP) with three layers and tanh activations, and logistic regression. Models are optimized using either stochastic gradient descent (SGD) or the Adam variant of SGD [78], and are fitted to the CIFAR-10, CIFAR-100, and SVHN image data. All models are trained using a mini-batch size of 128 and a

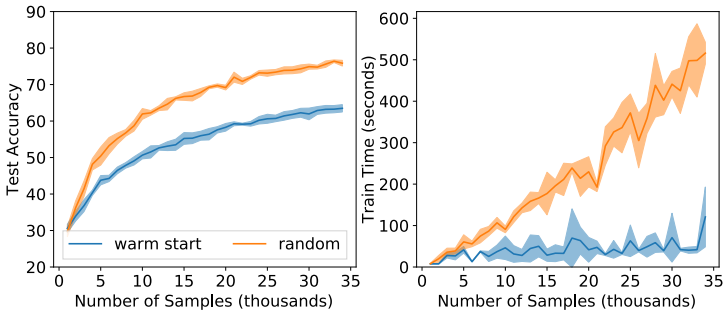


Figure 4.2: A passive online learning experiment for CIFAR-10 data using a ResNet. The horizontal axis shows the total number of samples in the training set available to the learner. Notice the significant generalization gap between warm-started and randomly-initialized models.

learning rate of 0.001, the smallest learning rate used in the learning schedule for fitting state-of-the-art ResNet models [57]. We further investigate the effect of these parameters in Section 4.3.

Our results (Table 4.1) indicate that generalization performance is damaged consistently and significantly for both ResNets and MLPs. This effect is more dramatic for CIFAR-10, which is considered relatively challenging to model (requiring, e.g., data augmentation), than for SVHN, which is considered easier. Logistic regression, which enjoys a convex loss surface, is not significantly damaged by warm starting for any of the datasets. Figure 4.3 extends these results, showing that the gap is inversely proportional to the fraction of data available in the first round of training.

This result is surprising. Even though MLP and ResNet optimization is non-convex, conventional intuition suggests that the warm-started solution should be close to the full-data solution and therefore a good initialization. One view on pre-training is that the initialization is a "prior" on weights; we often view prior distributions as

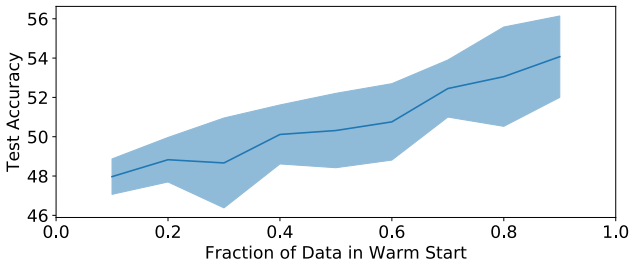


Figure 4.3: Warm-started ResNet-18 generalization as a function of the fraction of total data available in the first round of training. Models are trained on the indicated fraction of CIFAR-10 training data until convergence, then trained again on 100% of CIFAR-10 training data to produce this figure. When the initial data used to warm-start training more overlaps with the second round of training data, the generalization gap is less severe.

arising from inference on old (or hypothetical) data and so this sort of pre-training should always be helpful. The generalization gap shown here creates a computational burden for real-life machine learning systems that must be retrained from scratch to perform well, rather than initialized from previous models.

4.2.2 Online Learning

A common real-world setting involves data that are being provided to the machine learning system in a stream. At every step, the learner is given k new samples to append to its training data, and it updates its hypothesis to reflect the larger dataset. Financial data, social media, and recommendations systems are common examples of scenarios where new data are constantly arriving. This paradigm is simulated in Figure 4.2, where we supply CIFAR-10 data, selected randomly without replacement, in groups of 1,000 to an 18-layer ResNet. We examine two cases: 1) where the model

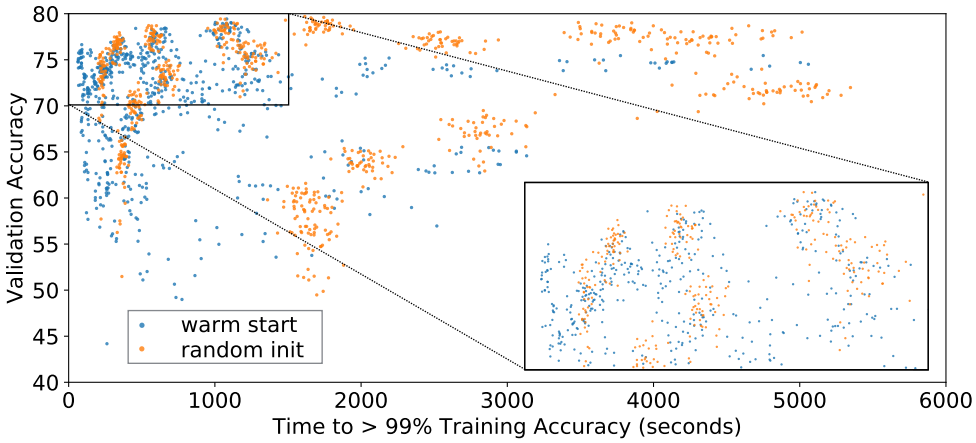


Figure 4.4: A comparison between ResNets trained from both a warm start and random initialization on CIFAR-10 for various hyperparameters. Orange dots are randomly-initialized models and blue dots are warm-started models. Warm-started models that perform roughly as well as randomly-initialized models offer no benefit in terms of training time.

is retrained from scratch after each batch, starting from a random initialization, and 2) where the model is trained to convergence starting from the parameters learned in the previous iteration. In both cases, the models are optimized with Adam, using an initial learning rate of 0.001. Each was run five times with different random seeds and validation sets composed of a random third of the training data, reinitializing Adam's parameters at each step of learning.

Figure 4.2 shows the trade-off between the two approaches. The mean and standard deviations across the five runs are shown. On the right are the training times: clearly, starting from the previous model is preferable and has the potential to vastly reduce computational costs and wall-clock time. However, as can be seen on the left, generalization performance is worse in the warm-started situation. As more data arrive, the gap in validation accuracy increases substantially.

4.3 Conventional Approaches

The design space for initializing and training deep neural network models is very large, and so it is important to evaluate whether there is some trick that could be used to help warm-started training find good solutions. Put another way, a reasonable response to this problem is "Did you see whether X helped?" where X might be anything from batch normalization [79] to increasing the mini-batch size [80]. This section tries to answer some of these questions and further empirically probe the warm-start phenomenon. Unless otherwise stated, experiments in this section use a ResNet-18 model trained using SGD with a learning rate of 0.001 on CIFAR-10 data. All experiments were run five times to report means and standard deviations. No experiments in this paper use data augmentation or learning rate schedules, and all validation sets are a randomly-chosen third of the training data.

4.3.1 Is this an effect of batch size or learning rate?

One might reasonably ask whether or not there exist *any* hyperparameters that close the generalization gap between warm started and randomly initialized models. In particular, can setting a larger learning rate at either the first or second round of learning help the model escape to regions that generalize better? Can shrinking the batch size inject stochasticity that might improve generalization [81, 82]?

Here we again consider a warm-started experiment of training on 50% of CIFAR-10 until convergence, then training on 100% of CIFAR-10, using the initial round of training as an initialization. We explore all combinations of batch sizes $\{16, 32, 64, 128\}$, and learning rates $\{0.001, 0.01, 0.1\}$, varying them across the three rounds of training.

This allows for the possibility that there exist different hyperparameters for the first stage of training that are better when used with a different set after warm-starting. Each of these combinations is run with three random initializations.

Figure 4.4 visualizes these results. Every resulting 100% model is shown from all three initializations and all combinations, with color indicating whether it was a random initialization or a warm-start. The horizontal axis shows the time to completion, excluding the pre-training time, and the vertical axis shows the resulting validation performance.

Interestingly, we do find warm-started models that perform as well as randomly-initialized models, but they are unable to do so while benefiting from their warm-started initialization. The training time for warm-started ResNet models that generalize as well as randomly-initialized models is roughly the same as those randomly-initialized models. That is, there is no computational benefit to using these warm-started initializations. It is worth noting that this plot does not capture the time or energy required to identify hyperparameters that close the generalization gap; such hyperparameter searches are often the culprit in the resource footprint of deep learning [69]. Wall-clock time is measured by assigning every model identical resources, which consists of 50GB of RAM and an NVIDIA Tesla P100 GPU.

This increased fitting time occurs because warm-started models, when using hyperparameters that generalize relatively well, seem to "forget" what was learned in the first round of training. Figure 4.5 demonstrates this phenomenon by computing the Pearson correlation between the weights of converged warm-started models and their initialization weights, again across various choices for learning rate and batch

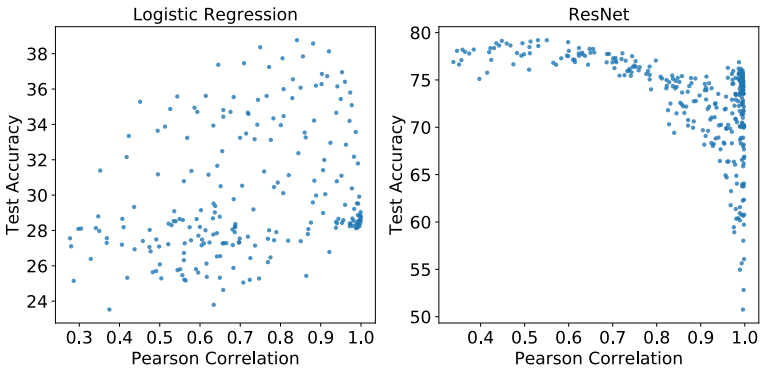


Figure 4.5: Validation accuracy as a function of the correlation between the warm-start initialization and the solution found after training for a large number of hyperparameter settings. **Left**: Warm-started logistic regressors often remember their initialization. **Right**: Warm-started ResNets that perform well do not retain much information from the initial round of training.

size, and comparing it to validation accuracy. Models that generalize well have little correlation with their initialization—there is a trend downward in accuracy with increasing correlation—suggesting that they have forgotten what was learned in the first round of training. Conversely, a similar plot for logistic regression shows no such relationship, and some of the best models have large correlations.

4.3.2 How quickly is generalization damaged?

One of the surprising results of our investigation is how little training is necessary to damage the validation performance of the warm-started model. Our hope was that warm-starting success might be achieved by switching from the 50% to 100% phase *before the first phase of training was completed.* We did a search over switching times to try to identify whether there might be a "sweet spot" in which a partially-trained

Table 4.2: Average validation percent accuracies for various regularizers and regularization penalties with both warm-started (WS) and randomly-initialized (RI) models on CIFAR-10 data.

L2	1×10^{-1}	1×10^{-2}	1×10^{-3}	1×10^{-4}
RI	72.7 (4.2)	55.4 (2.7)	54.6 (2.4)	55.1 (3.4)
WS	63.9 (6.4)	51.2 (2.7)	50.5 (1.8)	50.4 (1.3)
Adversarial				
RI	54.8 (1.3)	55.1 (1.5)	55.3 (1.4)	55.6 (0.9)
WS	52.4 (1.0)	52.6 (1.5)	52.7 (1.2)	50.4 (1.4)
Confidence				
RI	53.1 (1.9)	55.8 (1.3)	55.4 (1.2)	55.9 (1.4)
WS	50.3 (0.7)	50.0 (3.8)	51.2 (1.2)	49.3 (1.2)

checkpoint might provide a good initialization. We fit a ResNet-18 model on 50% of the training data, as before, and checkpointed its parameters every five epochs. We then took each of these checkpointed models and used them as an initialization for training on 100% of those data. As shown in Figure 4.6, generalization is damaged even when initializing from parameters obtained by training on incomplete data for only a few epochs.

4.3.3 Is regularization helpful?

A common approach for improving generalization is to include a regularization penalty. Here we investigate three different approaches to regularization: 1) basic L_2 weight penalties [83], 2) confidence-penalized training [84], and 3) adversarial training [85]. We again take a ResNet fitted to 50% of available training data and use its parameters to warm-start learning on 100% of those data. Regularization is applied in both

rounds of training. Table 4.2 shows the result of these experiments. Regularization often helps, but it does not resolve the generalization gap created by warm-starting. We regularize both stages using the regularization scheme and penalty size indicated in Table 4.2. Still, applying the same regularization to randomly-initialized models always produces a better-generalizing classifier.

4.3.4 Can we warm-start some layers but not others?

A common practice in deep learning is to train on one task, then continue training only the last network layer when new data become available [86, 87]. This subsection investigates how performance is affected when we train a model on 50% of data, then use that initialization to retrain only the last layer of the network. As the gradient of the last layer affects all earlier layers during training, it is one possible culprit for the vast gradient magnitude differences in Section 4.4.1.

We examine this hypothesis in two ways, as shown in Table B.3 in the Appendix. First we ran experiments that fixed all parameters but the last layer to their pre-trained values and then only trained the last layer in the second phase (LL). Second we extend this experiment to a third phase in which the rest of the network was trained after allowing the last layer to converge to something reasonable (LL+WS). While training only the last layer from the warm-started initialization is typically worse than training all parameters (WS), some gains can be had by training the entire network after having only trained the last layer. That is, the last layer alone does not seem to be sufficient to explain this generalization gap.

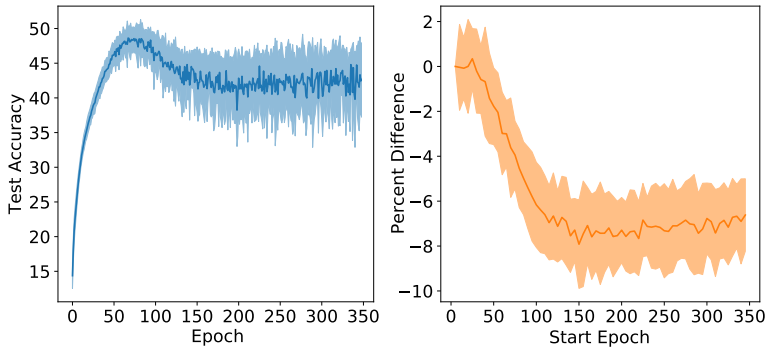


Figure 4.6: **Left:** CIFAR-10 validation accuracy of a ResNet as training progresses on 50% of the dataset. **Right:** The amount of validation accuracy damage, in terms of percentage difference from random initialization, after training on 100% of the data. Each warm-started model was initialized by training on 50% of CIFAR data for the indicated number of epochs.

4.3.5 Is this a case of catastrophic forgetting?

Warm starting is conceptually similar to continual learning. In that framework, an agent is given learning tasks sequentially, and the goal is to become good at the current task while avoiding "catastrophic forgetting," i.e., losing performance on previous tasks. One hypothesis to consider is whether the warm-start phenomenon is simply a case of such catastrophic forgetting. We can examine this hypothesis by trying to fix the warm-start problem using a technique commonly used to prevent catastrophic forgetting. One such approach is Elastic Weight Consolidation (EWC) [88], which adds a regularization penalty that encourages avoiding updating weights that are important for previously-learned tasks.

However, in the warm-start problem, each round of training adds data to what was available in the previous round. As described here, these data are often from the same distribution as before. Because a model that works well on the second task

Table 4.3: CIFAR-10 Validation percent accuracies for warm-started ResNets using different degrees of EWC. Standard deviations are indicated parenthetically.

1×10^{-1}	1×10^{-2}	1×10^{-3}	1×10^{-4}
48.8 (2.16)	50.8 (1.0)	51.2 (1.3)	51.9 (0.4)

of the warm-start problem also works well on the first, there is no reason to avoid updating important parameters. Accordingly, as shown in Table 4.3, including the EWC penalty in warm-started network training actually damages performance more than not including it at all.

4.4 Shrink, Perturb, Repeat

While the presented conventional approaches do not remedy the warm-start problem, we have identified a remarkably simple trick that efficiently closes the generalization gap. At each round of training t, when new samples are appended to the training set, we propose initializing the network's parameters by shrinking the weights found in the previous round of optimization towards zero, then adding a small amount of parameter noise. Specifically, we initialize each learnable parameter θ_i^t at training round t as $\theta_i^t \leftarrow \lambda\theta_i^{t-1} + p^t$, where $p^t \sim \mathcal{N}(0, \sigma^2)$ and $0 \leq \lambda \leq 1$.

Shrinking weights preserves hypotheses. For model layers that use ReLU nonlinearities, shrinking parameters preserves the relative activation at each layer. If bias terms and batch normalization are not used, the output of every layer is a scaled version of its non-shrunken counterpart. In the last layer, which usually consists

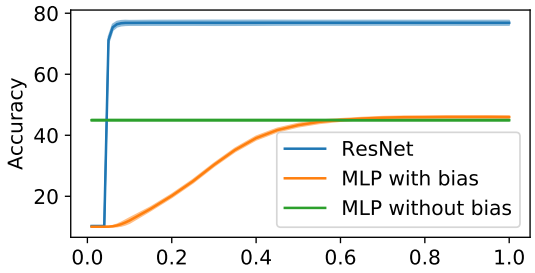


Figure 4.7: We fit a ResNet and MLP (with and without bias nodes) to CIFAR-10 and measure performance as we shrink weights.

of a linear transformation followed by a softmax nonlinearity, shrinking parameters can be interpreted as increasing the entropy of the output distribution, effectively diminishing the model's confidence. For no-bias, no-batchnorm ReLU models, while shrinking weights does not necessarily preserve the output $f_\theta(x)$ they parametrize, it does preserve the learned hypothesis, i.e. $\operatorname{argmax} f_\theta(x)$. A simple derivation is provided for completeness as Proposition 2 in the Appendix.

For more sophisticated architectures, this property largely still holds: Figure 4.7 shows that for a ResNet, which includes batch normalization, only extreme amounts of shrinking are able to damage classifier performance. This is because batch normalization's internal estimates of mean and variance can compensate for the rescaling caused by weight shrinking. Even for a ReLU MLP that includes bias nodes, classifier damage is done only for $\lambda < 0.5$. Separately, note that when internal network layers instead use sigmoidal activations, shrinking parameters moves them further from saturating regions, allowing the model to more easily learn from new data.

Shrink-perturb balances gradients. Figure 4.8 shows a visualization of average gradients during the second of a two-phase training procedure for a ResNet on CIFAR-

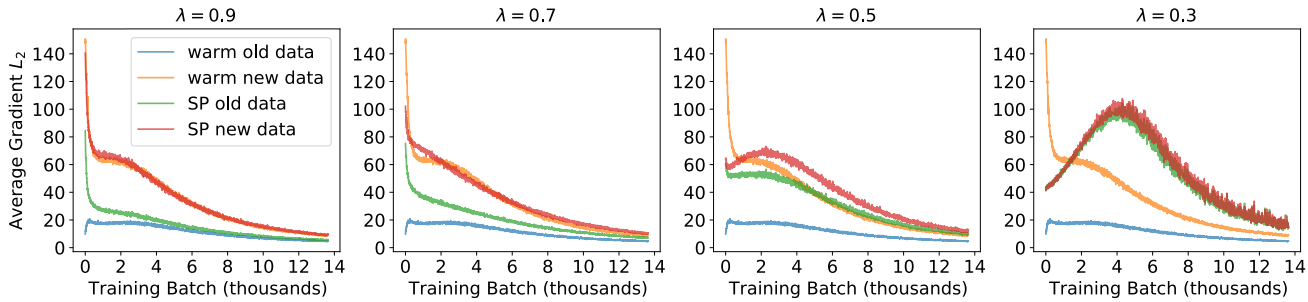


Figure 4.8: A two-phase experiment like those in Sections 4.2 and 4.3, where a ResNet is trained on 50% of CIFAR-10 and is then given the remainder in the second round of training. Here we examine the average gradient norms separately corresponding to the initial 50% of data and the second 50% for models that are either warm-started or initialized with the shrink and perturb (SP) trick. Notice that in warm-started models, there is a drastic gap between these gradient norms. Our proposed trick balances these respective magnitudes while still allowing models to benefit from their first round of training; i.e they fit training data much quicker than random initializations.

10, like those discussed in Sections 4.2 and 4.3. We plot the second phase of training, where magnitudes are shown separately for the two halves of the data set. For this experiment models are optimized with SGD, using a small learning rate to zoom in on this effect. Outside of this plot, experiments in this section use the Adam optimizer.

For warm-started models, gradients from new, unseen data tend to be much larger magnitude than those from data the model has seen before. These imbalanced gradient contributions are known to be problematic for optimization in mutli-task learning scenarios [89], and we believe they are problematic in the warm-start regime as well. Resolving this imbalance without damaging what the model has already learned is key to efficiently resolving the generalization gap studied in this article.

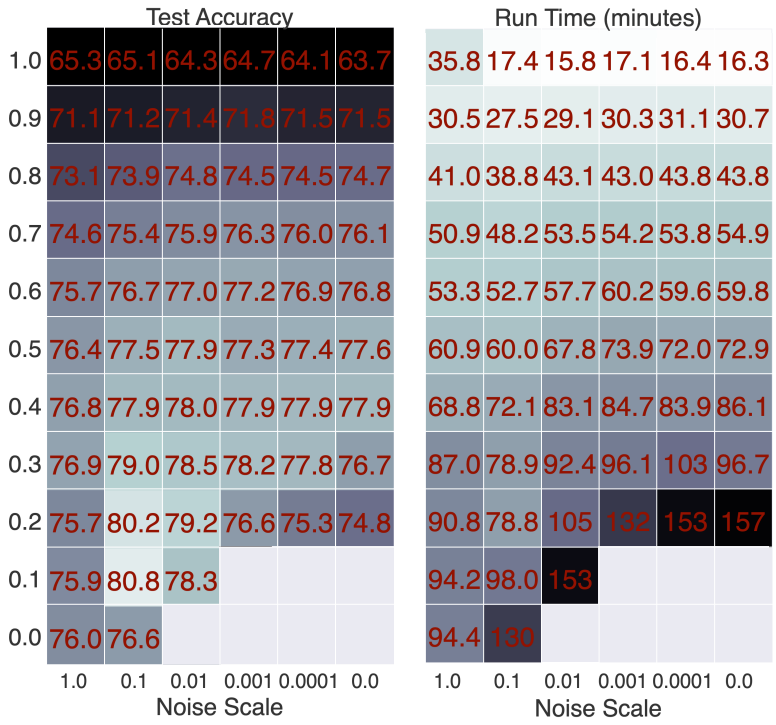


Figure 4.9: Model performance as a function of λ (indicated on the vertical axis) and σ. Numbers indicate the average final performance and total train time for online learning experiments where ResNets are provided CIFAR-10 samples in sequence, 1,000 per round, and trained to convergence at each round. Note that the bottom left of this plot corresponds to pure random initializing while the top right corresponds to pure warm starting. **Left**: Validation accuracy tends to improve with more aggressive shrinking. When $\lambda < 0.5$, some amount of noise improves generalization. **Right**: Model train times increase with decreasing values of λ. This is expected, as decreasing λ widens the gap between shrink-perturb parameters and warm-started parameters. Adding noise helps models train more quickly. Unlabeled boxes correspond to initialization values that were too small for the model to reliably learn. Networks in this plot were optimized with Adam.

Shrinking the model's weights increases its loss, and correspondingly increases the magnitude of the gradient induced even by samples that have already been seen. Preposition 2 shows that in an L-layer ReLU network without bias nodes or batchnorm, shrinking weights by λ shrinks softmax inputs by λ^L, rapidly increasing the entropy of the softmax distribution and the cross-entropy loss. As shown in Figure 4.8, the loss increase caused by shrink perturb trick is able to balance gradient contributions between previously unseen samples and data on which the model has already been trained.

We believe the success of the shrink and perturb trick lies in its ability to standardize gradients while preserving learned hypotheses. We could instead normalize gradient contributions by, for example, adding a significant amount of parameter noise, but this also damages the learned function. Consequently, this strategy drastically increases training time without fully closing the warm-start generalization gap (Appendix Table B.2). As an alternative to shrinking all weights, we could try to increase the entropy of the output distribution by shrinking only parameters in the last layer (results in Appendix Figure B.1), or by regularizing the model's confidence while training (results in Table 4.2), but these are unable to resolve the warm-start problem. For sophisticated architectures especially, it is important to holistically modify parameters before training on new data.

Figure 4.10 demonstrates the surprising effectiveness of this trick. Like before, we present a passive online learning experiment where 1,000 CIFAR-10 samples are supplied to a ResNet in sequence. At each round we can either reinitialize network parameters from scratch or warm start, initializing them to those found in

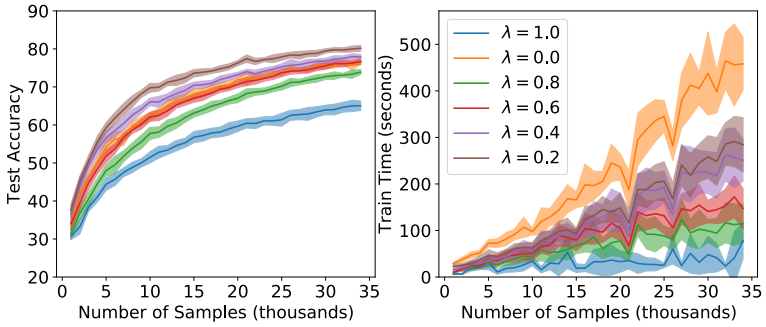


Figure 4.10: An online learning experiment where we vary the value of λ and keep the noise scale fixed at 0.01. Note that $\lambda = 1$ corresponds to fully-warm-started initializations and $\lambda = 0$ corresponds to fully-random initializations. The proposed trick with $\lambda = 0.6$ performs identically to randomly initializing in terms of validation accuracy, but also trains much more quickly. Interestingly, smaller values of λ are even able to outperform random initialization while still training faster.

the previous round of optimization. As expected, we see that warm-started models train faster but generalize worse. However, if we instead initialize parameters using the shrink and perturb trick, we are able to both close this generalization gap and significantly speed up training.

4.4.1 The shrink and perturb trick normalizes gradients

We notice some interesting phenomena when examining the L_2 norm of the gradients as training progresses for a shrink-perturb-initialized model. Figure 4.8 shows a visualization of average gradients during the second of a two-phase training procedure for a ResNet on CIFAR-10, like those discussed in Sections 4.2 and 4.3. We plot the second phase of training, where magnitudes are shown separately for the two halves of the data set. For this experiment models are optimized with SGD, using a small

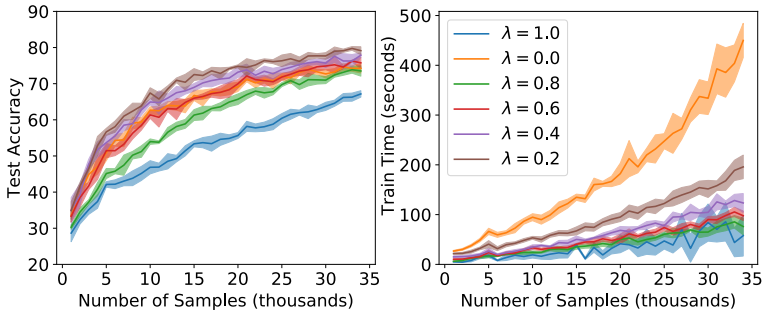


Figure 4.11: An online learning experiment where we vary the value of λ, keep the noise scale fixed at 0.01, and apply a weight decay of 0.001 in all rounds of optimization. Like in Figure 4.10, $\lambda = 0.6$ perfectly overlaps with randomly initializing ($\lambda = 0.0$), with smaller values performing slightly worse and larger values performing better. Notice that adding an L_2 penalty widens the gap between randomly-initialized and shrink-perturb-initialized models.

learning rate to zoom in on this effect. Outside of this plot, all experiments in this section use the Adam optimizer instead.

As expected, for fully-warm-started models, gradient norms corresponding to new data tend to be much larger magnitude than those for data that the model has seen before. Conversely, the shrink and perturb trick pulls these magnitudes closer together as λ decreases. Anecdotally, we notice that λ that close this gradient magnitude gap seem to perform at least as well as randomly-initialized models.

4.4.2 The shrink and perturb trick and regularization

Exercising the shrink and perturb trick at every step of SGD would be very similar to applying an aggressive, noisy L_2 regularization. It is natural to ask, then, how does this trick compare with weight decay? Figure 4.12 shows that in non-warm-started environments, where we just have a static dataset, the iterative application of the shrink and randomize results in marginally improved performance. These experiments

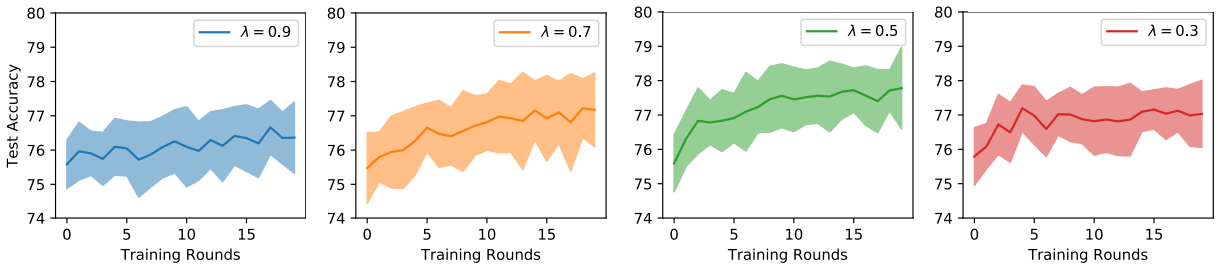


Figure 4.12: The result of fitting a ResNet on 100% of CIFAR-10 to convergence for twenty rounds and applying the shrink-perturb trick after each. Here we show four versions of that experiment for the indicated λ and a noise scaling of 0.01. Applying this trick iteratively has a slight regularization effect.

fit a ResNet to convergence on 100% of CIFAR-10 data, then shrink and perturb weights before repeating the process, resulting in a modest performance improvement. We can conclude that the shrink-perturb trick has two benefits. Most significantly, it allows us to quickly fit high-performing models in sequential environments without having to retrain from scratch. Separately, it offers a slight regularization benefit, which in tandem with the first property sometimes allows shrink-perturb models to generalize even better than randomly-initialized models.

This L_2 regularization benefit is not enough to explain the success of the shrink-perturb trick. As Table 4.2 demonstrates, L_2-regularized models are still vulnerable to the warm-start generalization gap. Figure 4.11 shows that we are able to mitigate this performance gap with the shrink and perturb trick even when models are being aggressively regularized (regularization penalties any larger prevent networks from being able to fit the training data) with weight decay. The shrink and perturb trick

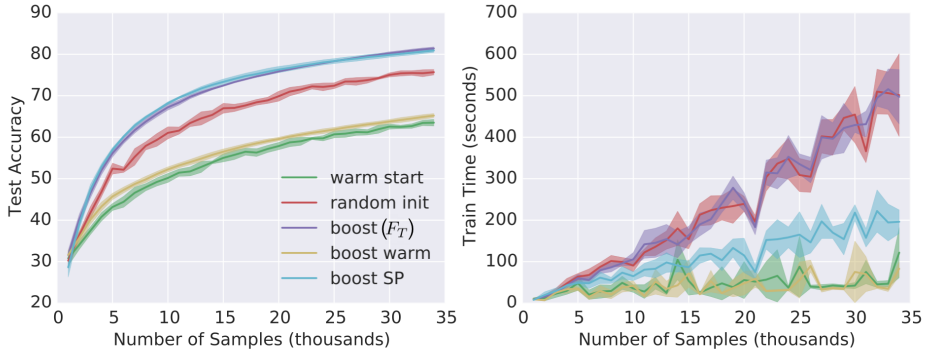


Figure 4.13: An experiment with online boosting using CIFAR-10 data and ResNets as weak learners. Warm-started weak learners train much quicker than randomly-initialized weak learners, but the resulting ensemble is very poor performing. Weak learners initialized via shrink perturb (SP) both train quicker than randomly-initialized models without sacrificing performance.

is actually *especially* appealing in this context, as it seems to allow models to train even more quickly than unregularized analogues in Figure 4.10.

4.5 Applications

In this section we hilight some interesting use cases for the shrink-perturb trick outside of the online and active learning applications discussed earlier.

4.5.1 Quick Ensembling

There are many reasons why one might want to build an ensemble of models. It is well understood that ensembles generally produce more accurate predictions than

their non-ensemble analogues, and that the predictive variance across ensemble items can be a useful measurement of uncertainty.

In the experiment presented here, like before, data are arrive iid in batches of 1,000, all from the CIFAR-10 dataset. Each time a fresh batch of data arrive, we train a new weak learner f_T on the residuals of the current state of the ensemble,

$$F_{t-1}(x) = \frac{1}{T-1} \sum_{t=1}^{T-1} f_t(x).$$

This is effectively batch-mode online gradient boosting. If we randomly initialize each weak learner, which is shown by the purple line in Figure 4.13, we can improve performance over a single model.

However, since we would expect each of the constituent models in the ensemble to learn somewhat overlapping things, like edge and color detectors, we should be able to warm start the parameters of each new weak learner–initializing its weights to those of its predecessor. If we do that, which is shown by the yellow line, we get a big improvement in terms of train time but suffer a drastic loss in terms of generalization performance. The alternative, which is to initialize each new weak learner's parameters by shrinking and perturbing the weights of the preceding weak learner (shown in cyan), allows us to match the performance of the randomly-initialized ensemble while affording much faster train times than simply randomly initializing.

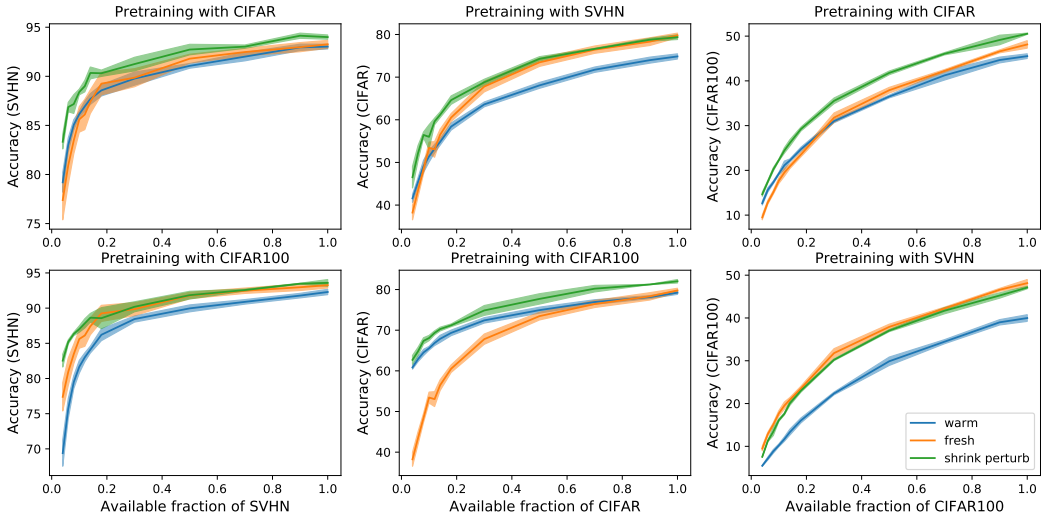


Figure 4.14: When target data are abundant, or significantly different from source data, it is often better to train models from scratch instead of pre-train them. Each line shows the performance of the indicated initialization protocol as a function of the fraction of target data available for training. Shrink-perturb initialization offers performance that is at least as high as that offered by either of the two afformentioned approaches.

4.5.2 Pre-Training

Despite successes on a variety of machine learning tasks, deep neural networks are still data hungry and generally require large training sets to generalize well. For problems where only limited data are available, it has become popular to warm-start learning using the parameters from training on a different but related problem [76, 90]. Transfer and "few-shot" learning in this form has seen success in computer vision and NLP [86].

The experiments we perform in this chapter, however, imply that when the second problem is not data-limited, this transfer learning approach deteriorates

model quality. That is, at some point, the pre-training transfer learning approach is essentially the same as warm-starting under domain shift, and generalization performance should suffer.

We demonstrate this phenomenon by first training a ResNet-18 to convergence on one dataset, then using that solution to warm start a model trained on a varying fraction of another dataset. When only a small portion of target data is used, this is essentially the same as the pre-training transfer learning approach. As the proportion increases, the problem turns into what we have described here as warm starting. Figure 4.14 shows the result of this experiment, and it appears to support our intuition. Often, when the second dataset is small, warm starting is helpful, but there is a crossover point where better generalization would be achieved from training from scratch on that fraction of the target data.

4.6 Discussion

This chapter presented the challenges of warm-starting neural network training and proposed a simple and powerful solution. While warm-starting is a problem that the community seems somewhat aware of anecdotally, it does not seem to have been directly studied. We believe that this is a major problem in important real-life tasks for which neural networks are used, and it speaks directly to the resources consumed by training such models.

We further showed that the shrink and perturb trick is a simple and effective initializing that can remedy the warm-start problem, allowing models to train significantly quicker than random initializations without sacrificing performance.

the enormous computational expense of retraining models from scratch disproportionately burdens research groups without access to abundant computational resources. The shrink and perturb trick lowers this barrier, democratizing participation in online learning, active learning, and pre-training research with neural networks.

Chapter 5

Related Work

5.1 Batch Active Learning with Neural Networks

Active learning is a very well studied problem [91, 92, 93]. In this thesis, we address the batch-mode variant of active learning, where unlabeled samples are selected in batches of size k instead of one at a time [94, 95, 96, 97, 98]. There are two major strategies for batch active learning—representative sampling and uncertainty sampling.

Representative sampling algorithms query labels for batches of unlabeled examples that are representative of the unlabeled set. This strategy is based on the intuition that a set of representative examples, once labeled, can act as a surrogate for the full dataset. The hope is that performing loss minimization on this surrogate suffices to ensure a low error with respect to the full dataset. In the context of deep learning,

popular diversity-based methods select representative examples based on core-set construction, a fundamental problem in computational geometry [53, 99].

Uncertainty sampling is based on a different principle—to select new samples that maximally reduce the predictive uncertainty of the learner on target data. For linear classification, multiple articles have proposed uncertainty sampling methods that query examples that lie closest to the current decision boundary [100, 101, 102]. Some uncertainty sampling approaches have theoretical guarantees on statistical consistency [93, 46]. Such methods have also been somewhat generalized to deep learning, for example, by measuring predictive uncertainty via sampling from a model with Dropout [103]. These sorts of algorithms can be traced back to more foundational work on uncertainty-based active learning strategies [104, 105, 106, 107]. It has further been shown that an ensemble of classifiers could also be used to effectively estimate uncertainty [108].

There are several existing approaches that support a hybrid of representative sampling and uncertainty sampling, often relying on meta-active learning objectives that incorporate multiple active learning algorithms [109, 56]. One method in this class is Active Learning by Learning, which uses a sequential decision process to choose either a diversity-based algorithm or an uncertainty-based algorithm at each round of unlabeled sample selection [56]. Inspired by expected loss minimization, other work suggests query criteria that balance representativeness and informativeness of examples [110].

One query criterion, which is related to what will be proposed in Chapter 3, selects samples based on expected gradient length (EGL) [111]. Recent work shows that the

EGL criterion is related to the T-optimality criterion in experimental design [112]. They further demonstrate that samples selected by EGL are very different from those selected by a predictive entropy-based uncertainty criterion. The EGL criterion in active sentence and document classification with CNNs [113] . These approaches differ most substantially from our algorithm (discussed in Chapter 3) in that they do not consider batch diversity.

There is a wide array of theoretical articles that focus on the related problem of adaptive subsampling for fully-labeled datasets in regression settings [114, 115, 116]. Empirical studies of batch stochastic gradient descent also employ adaptive sampling to "emphasize" hard or representative examples [117, 118]. These works aim at reducing computation costs or finding a better local optimal solution, as opposed to reducing labeling cost. Nevertheless, our work is inspired by their sampling criteria in that it also emphasizes samples that induce large model updates.

5.2 Warm-Starting Neural Networks

Chapter 4 studies what we called the *warm-start problem*, which concerns itself with situations in which data arrive sequentially in large batches. To have the highest-performing learner possible at all times, we need to train it on all data that has been encountered at each round. Each time the network is trained, we can either initialize its parameters to those found in the previous round of optimization (warm start) or we can use a fresh random initialization. We show that while warm-started

models train much quicker than their randomly-initialized peers, they also generalize significantly worse.

Warm-starting and online learning are well understood for convex models like linear classifiers [119] and SVMs [120, 121]. However, it does not appear that generally applicable techniques exist for deep neural networks that do not damage generalization, and so models are typically retrained from scratch, e.g., [15, 122].

There has been a variety of work in closely related areas, however. For example, in analyzing "critical learning periods," researchers show that a network initially trained on blurry images then on sharp images is unable to perform as well as one trained from scratch on sharp images, drawing a parallel between human vision and computer vision [123]. In this article, we will show that this phenomenon occurs more generally and that test performance is damaged even when first and second datasets are drawn from identical distributions.

The problem of warm starting is closely related to the rich literature on initialization of neural network training "from scratch". Indeed, new insights into what makes an effective initialization have been critical to the revival of neural networks as machine learning models. While there have been several proposed methods for initialization [124, 125, 72, 126, 127], this body of literature primarily concerns itself with initializations that are high-quality in the sense that they allow for quick and reliable model training. That is, these methods are typically built with training performance in mind rather than generalization performance.

Work relating initialization to generalization suggests that networks whose weights have moved far from their initialization are less likely to generalize well compared

with ones that have remained relatively nearby [128]. Chapter 4 however shows with experimental results that warm-started networks that have *less* in common with their initializations seem to generalize better than those that have more. So while it is not surprising that there exist initializations that generalize poorly, it is surprising that warm starts are in that class. Still, before retraining, our proposed solution brings parameters closer their initial values than they would be if just warm starting, suggesting some relationship between generalization and distance from initialization.

The warm start problem is fundamentally about generalization performance, which has been extensively studied both theoretically and empirically within the context of deep learning. These articles have investigated generalization by studying classifier margin [129, 130], loss geometry [131, 80, 132], and measurements of complexity [133, 134], sensitivity [135], or compressiblity [136].

These approaches can be seen as attempting to measure the intricacy of the hypothesis learned by the network. If two models are both consistent for the same training data, the one with the less complicated concept is more likely to generalize well. We know that networks trained with SGD are implicitly regularized [81, 82], suggesting that standard training of neural networks incidentally finds low-complexity solutions. It's possible, then, that the initial round of training disqualifies solutions that would most naturally explain the general problem of interest. If so, the trick we propose seems to make these solutions accessible again.

The warm-start problem is very similar to the idea of unsupervised and supervised pre-training [137, 73, 72, 138]. Under that paradigm, learning where limited labeled data are available is aided by first training on related data. The warm start problem,

however, is not about limited labeled data in the second round of training. Instead, the goal of warm starting is to hasten the time required to fit a neural network by initializing using a similar supervised problem without damaging generalization. Our results suggest that while warm-starting is beneficial when labeled data are limited, it actually damages generalization to warm-start in data-rich situations.

Chapter 6

Conclusion

This thesis studies problems related to sample complexity. That is, the question of how we can find the highest-performing model given a fixed labeling budget, and how we can do so efficiently. When using neural networks, good answers to these questions need to consider the way in which models are initialized, what kinds of priors they encompass, the type of data being used, and the amount of samples being received.

In deep active learning, we observed that sample selection strategies are highly sensitive to these aforementioned variables, and that performant acquisition functions need to consider these details to successfully trade-off between predictive uncertainty and batch diversity. Because cross-validation is costly in active learning, it is crucial that our acquisition function is hyperparameter free.

We also elucidated a separate trade-off, between accuracy and efficiency, for even non-active sequential learning problems. We explored this problem in detail, and discussed how the shrink-perturb trick is able to balance gradient contributions while

simultaneously preserving the previously learned hypothesis–effectively allowing us to mitigate the deleterious effect of warm starting.

One might notice that these problems are orthogonal properties of reinforcement learning. In the exploration phase of reinforcement learning, we want to actively select actions that, once executed, will be maximally informative to the learner. For efficiency, this is sometimes done in parallel, raising questions about how to best incorporate diversity across simultaneously unrolled state-action trajectories. BADGE offers a promising solution for problems of this variety, but currently leverages a simple measurement of uncertainty that assumes a classification setting. Most reinforcement learning models rely on predicting continuous terms, like Q values or future states, so BADGE will need to be adapted in order to be effective in these environments. This adaptation to regression problems, and experimentation in reinforcement learning settings, is an interesting direction for future work.

As the agent explores, new samples that capture transition and reward dynamics are integrated into the training set on which models are being fitted. Does this sequential optimization damage generalization? If so, the shrink-perturb trick, in its current form, is unlikely to remedy the problem. New state trajectories are being added to the training set very frequently, rather than in large, sparse batches. This would suggest we need to apply the shrink-perturb trick at almost every step of optimization, which as mentioned in Chapter 4, devolves to an aggressive, noisy L_2 regularization. Identifying more flexible ways of solving this problem could be a step towards fixing some of the reproducibility issues plaguing deep reinforcement learning research.

Resolving problems of sample complexity, and remedying the reliance of neural networks on specific hyperparameters, is paramount to making useful, general purpose machine learning algorithms. Truly automated learning equipment should be flexible and reliable, and should not depend on practitioner experience. This thesis aims to take a step in that direction, highlighting and remedying nuanced problems in the use of deep learning sequential problems that affect their usability in real-life scenarios.

Appendix A

Batch Active Learning

A.1 The k-MEANS++ seeding algorithm

Here we briefly review the k-MEANS++ seeding algorithm by [49]. 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 [49] that the set of centers returned is guaranteed to approximate the k-means objective function in expectation, thus ensuring diversity.

A.2 All learning curves

We plot all learning curves (test accuracy as a function of the number of labeled example queried) in Figures A.1 to A.7. In addition, we zoom into regions of the learning curves that discriminates the performance of all algorithms in Figures A.8 to A.14.

Algorithm 2 The k-MEANS++ seeding algorithm [49]

Require: Ground set $G \subset \mathbb{R}^d$, target 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.

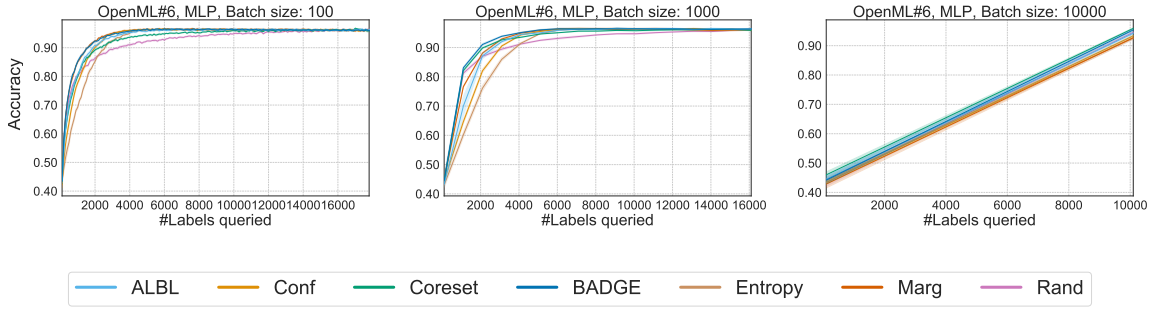


Figure A.1: Full learning curves for OpenML #6 with MLP.

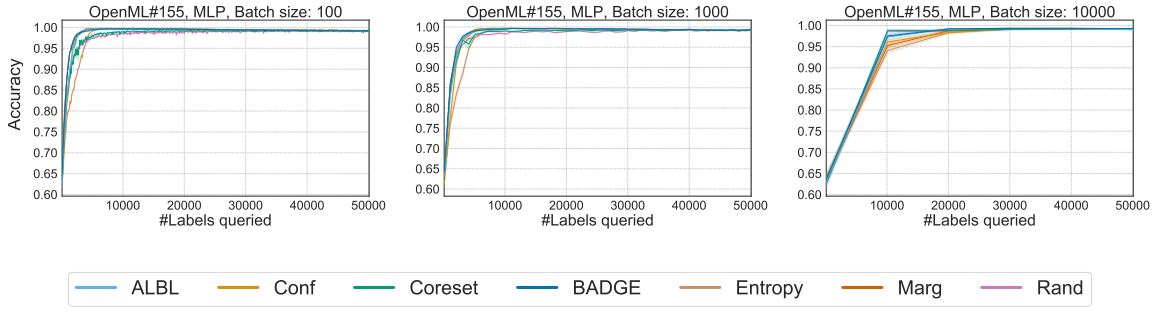


Figure A.2: Full learning curves for OpenML #155 with MLP.

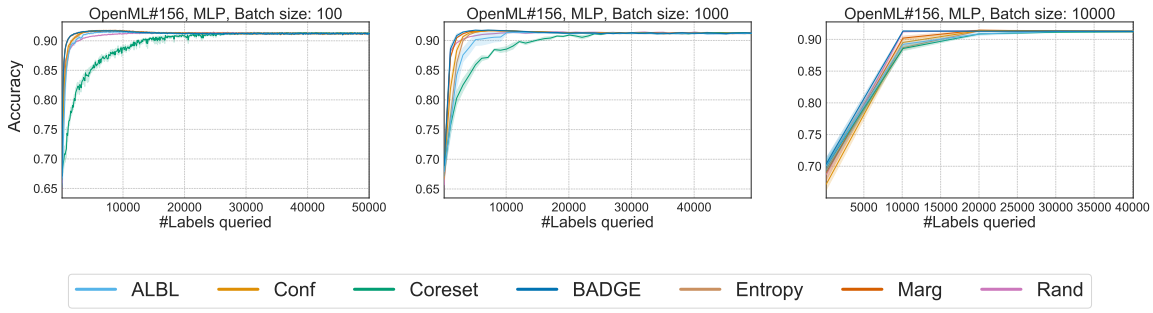


Figure A.3: Full learning curves for OpenML #156 with MLP.

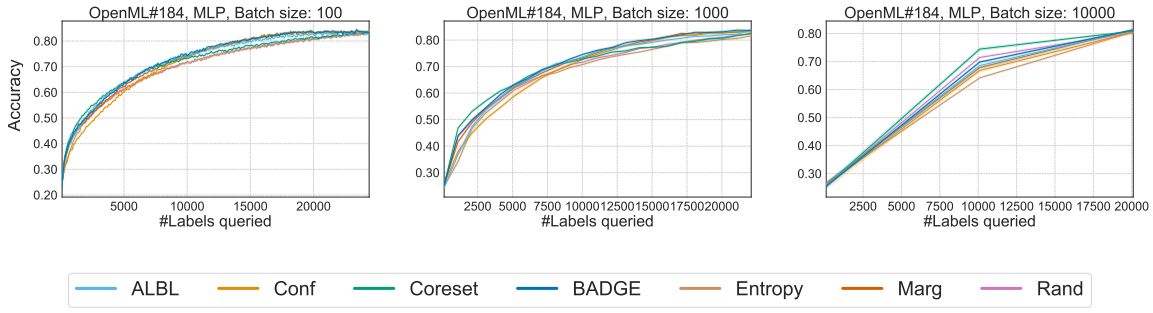


Figure A.4: Full learning curves for OpenML #184 with MLP.

A.3 Pairwise comparisons of algorithms

In addition to Figure 3.4 in the main text, we also provide penalty matrices (Figures A.15 and A.16), 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). For each penalty matrix, the parenthesized number in its title is the total number of (D, B, A) combinations aggregated; as discussed in Section 3.4, this is also an upper bound on all its entries. It can be seen that uncertainty-based methods (e.g. MARG) perform well only in small batch size regimes (100) or when using

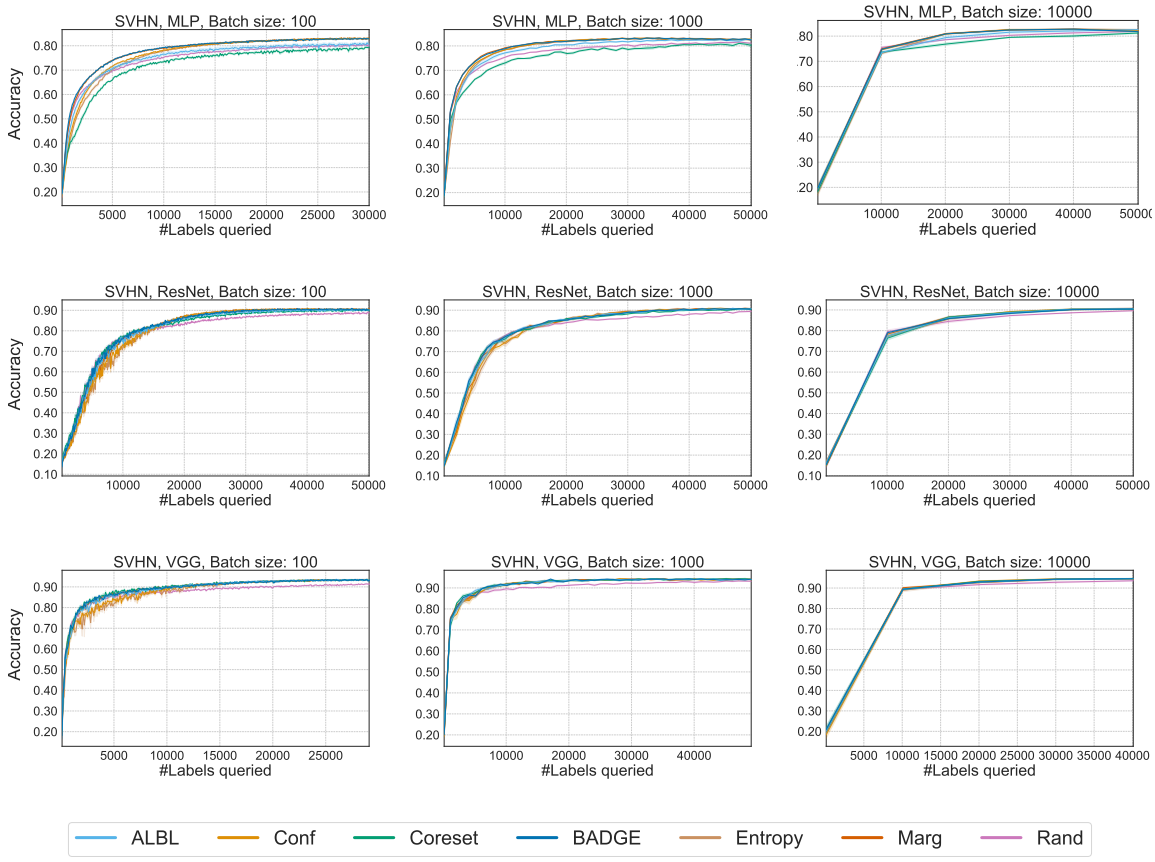


Figure A.5: Full learning curves for SVHN with MLP, ResNet and VGG.

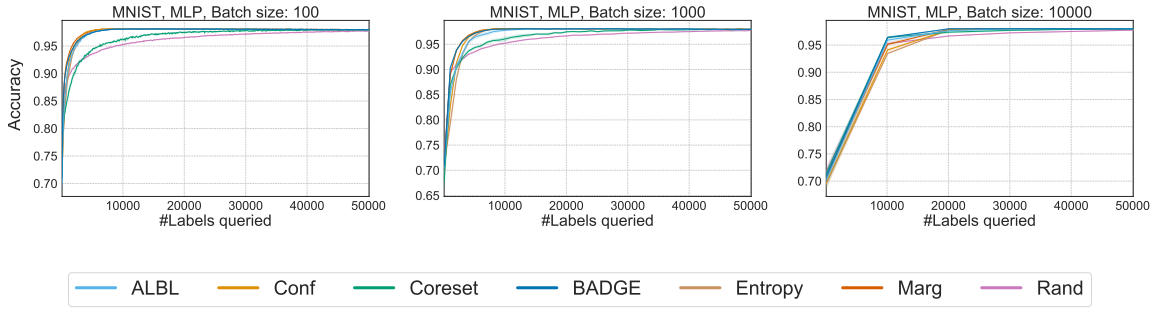


Figure A.6: Full learning curves for MNIST with MLP.

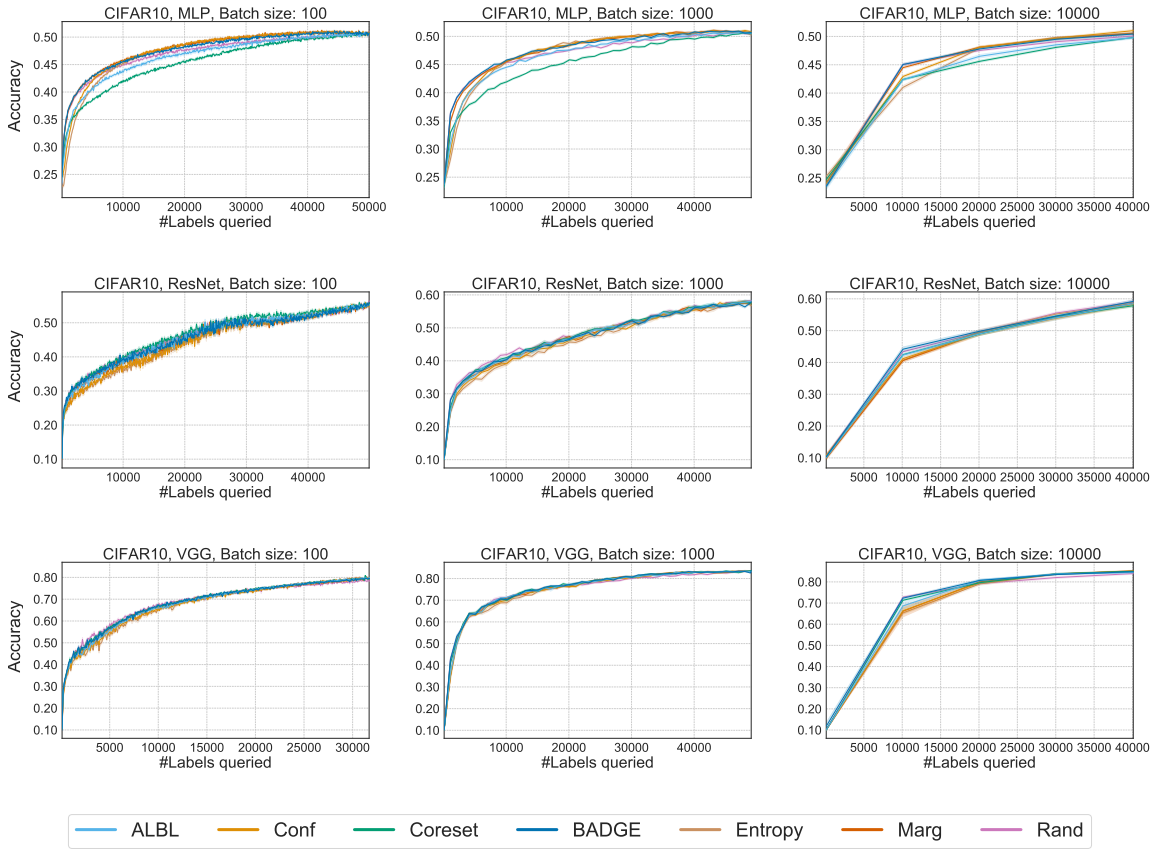


Figure A.7: Full learning curves for CIFAR10 with MLP, ResNet and VGG.

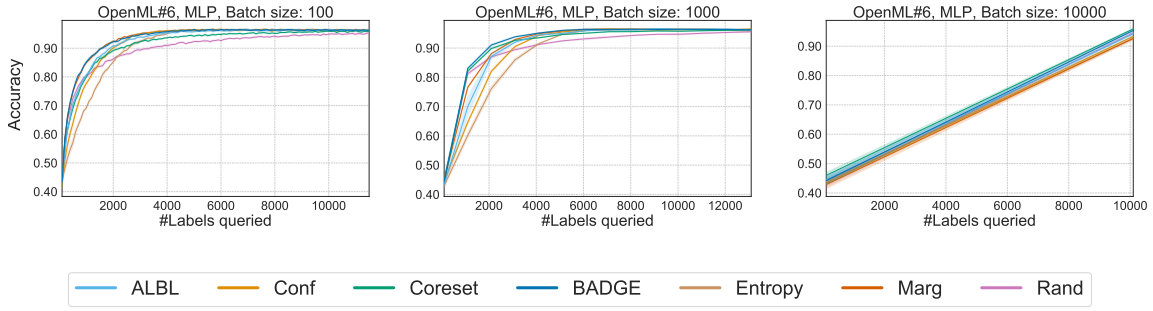


Figure A.8: Zoomed-in learning curves for OpenML #6 with MLP.

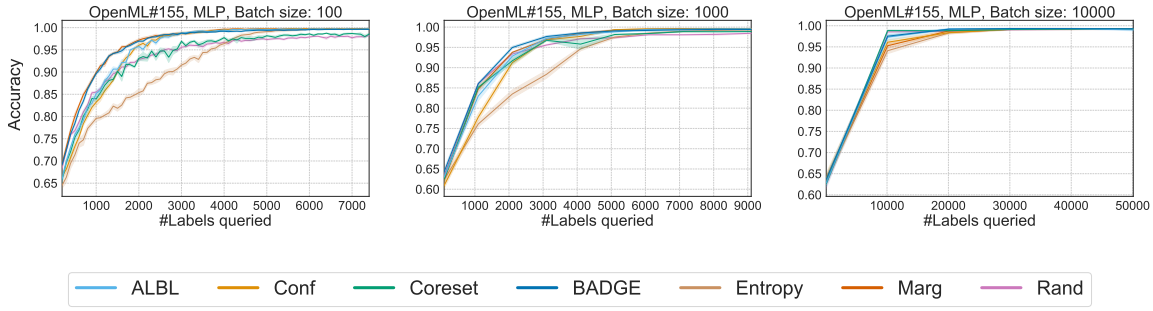


Figure A.9: Zoomed-in learning curves for OpenML #155 with MLP.

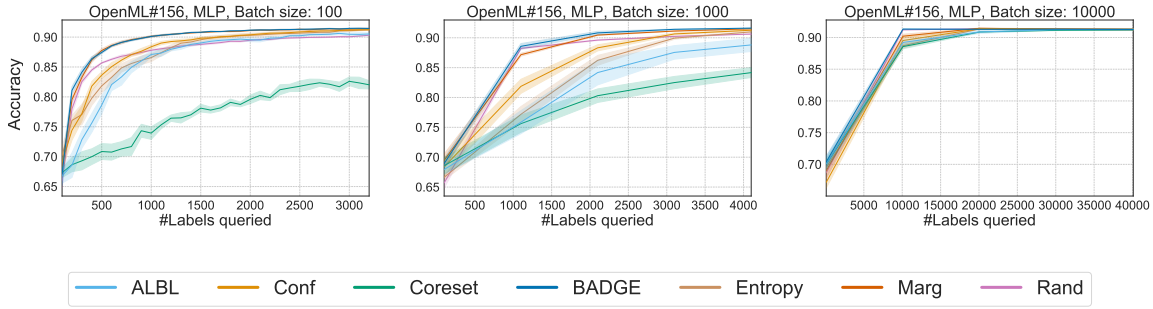


Figure A.10: Zoomed-in learning curves for OpenML #156 with MLP.

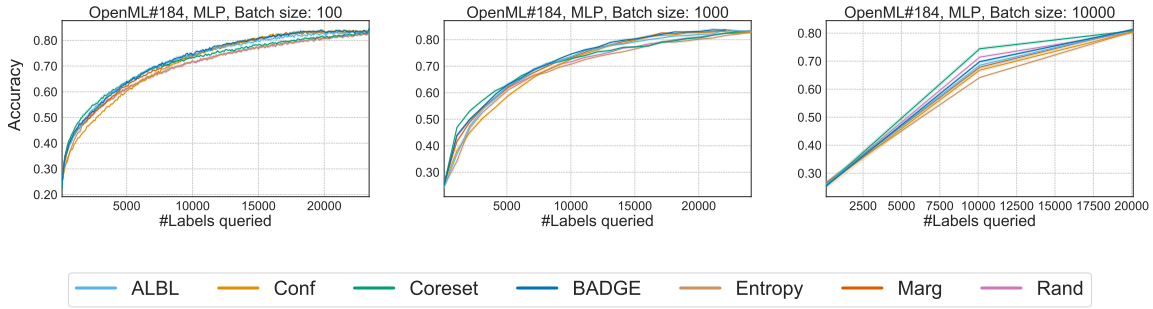


Figure A.11: Zoomed-in learning curves for OpenML #184 with MLP.

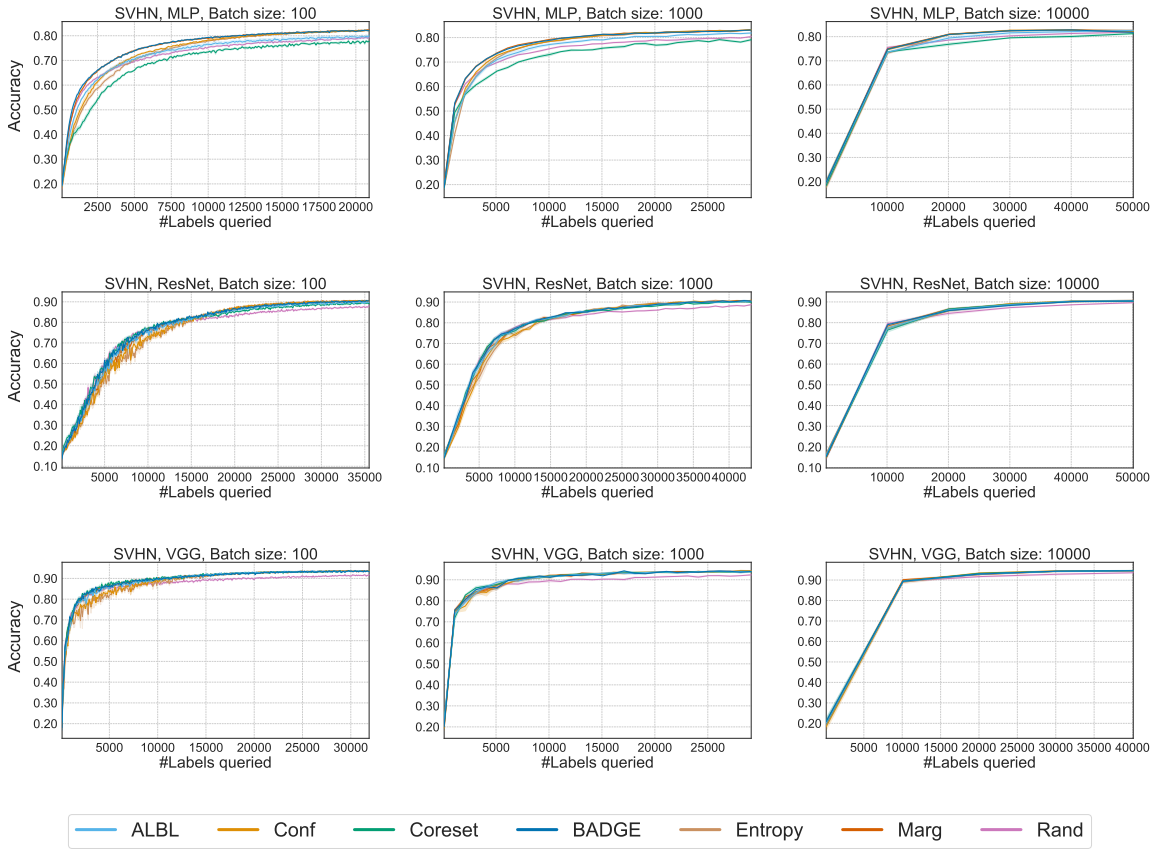


Figure A.12: Zoomed-in learning curves for SVHN with MLP, ResNet and VGG.

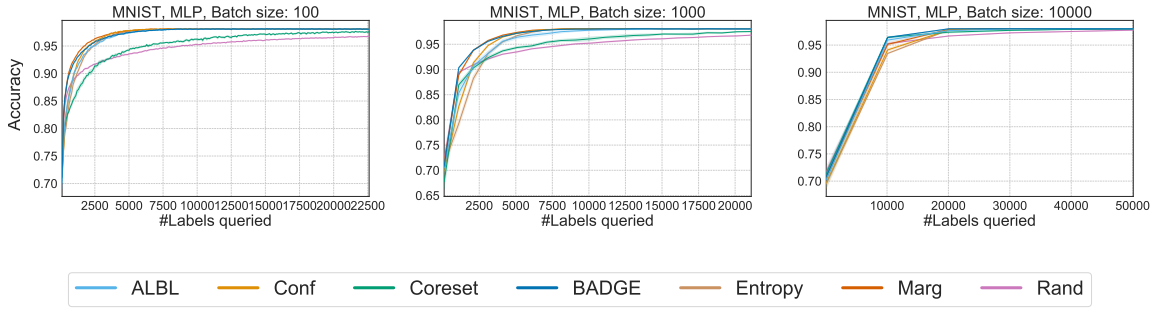


Figure A.13: Zoomed-in learning curves for MNIST with MLP.

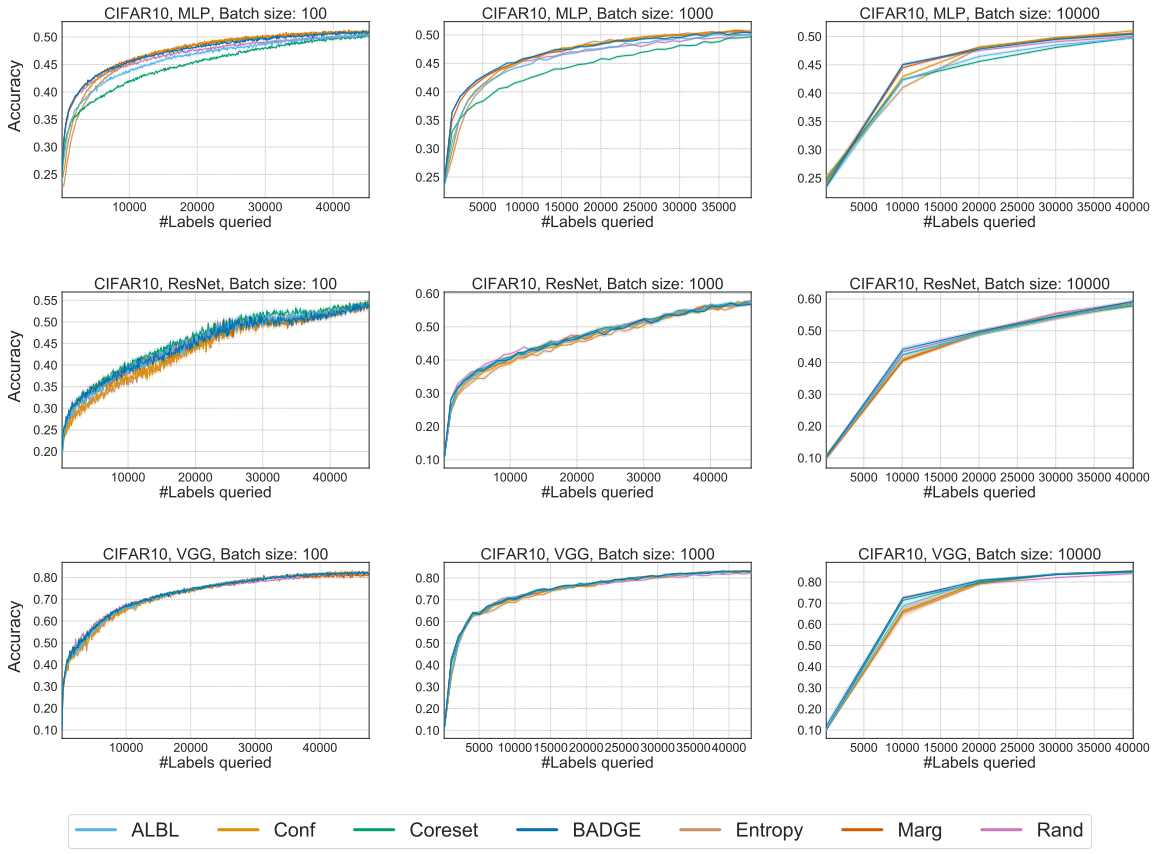


Figure A.14: Zoomed-in learning curves for CIFAR10 with MLP, ResNet and VGG.

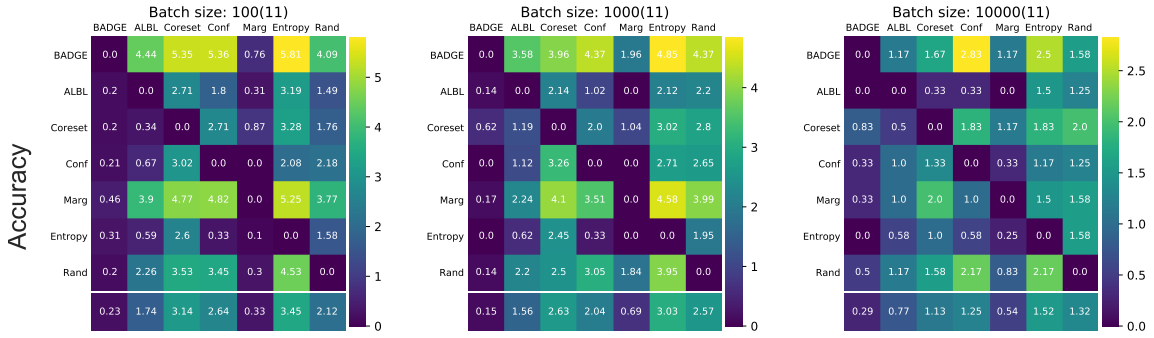


Figure A.15: Pairwise penalty matrices of the algorithms, grouped by different batch sizes. The parenthesized number in the title is the total number of (D, B, A) combinations aggregated, which is also an upper bound on all its entries. Element (i, j) corresponds roughly to the number of times algorithm i beats algorithm j. Column-wise averages at the bottom show aggregate performance (lower is better). From left to right: batch size = 100, 1000, 10000.

MLP models; representative sampling based methods (e.g. CORESET) only perform well in large batch size regimes (10000) or when using ResNet or VGG models. In contrast, BADGE's performance is competitive across all batch sizes and neural network models.

A.4 CDFs of normalized errors of different algorithms

In addition to Figure 3.5 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 A.17, and show the CDFs of normalized errors by conditioning on fixed neural network models (MLP, ResNet and VGG) in Figure A.18.

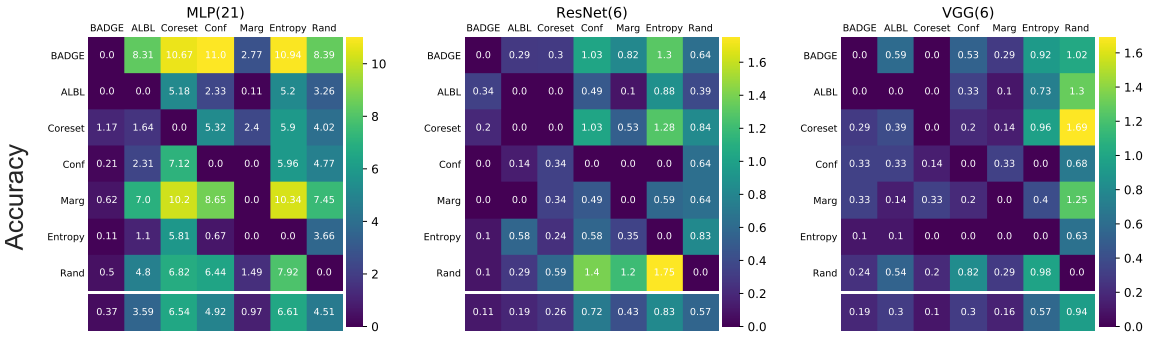


Figure A.16: Pairwise penalty matrices of the algorithms, grouped by different neural network models. The parenthesized number in the title is the total number of (D, B, A) combinations aggregated, which is also an upper bound on all its entries. Element (i, j) corresponds roughly to the number of times algorithm i beats algorithm j. Column-wise averages at the bottom show aggregate performance (lower is better). From left to right: MLP, ResNet and VGG.

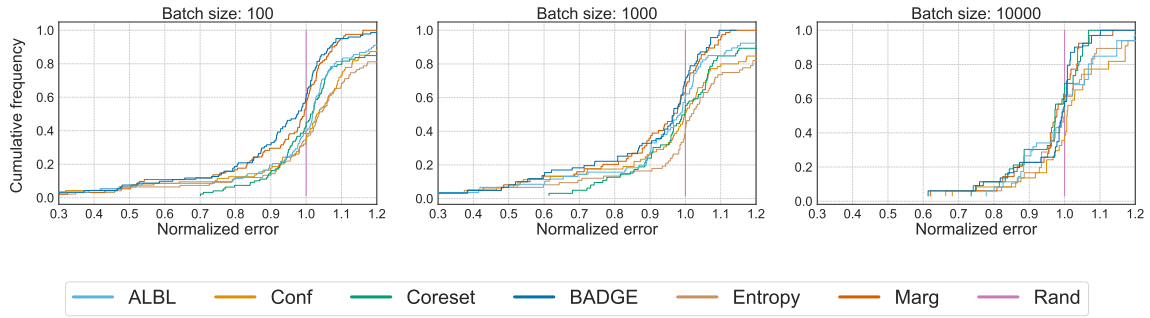


Figure A.17: CDFs of normalized errors of the algorithms, group by different batch sizes. Higher CDF indicates better performance. From left to right: batch size = 100, 1000, 10000.

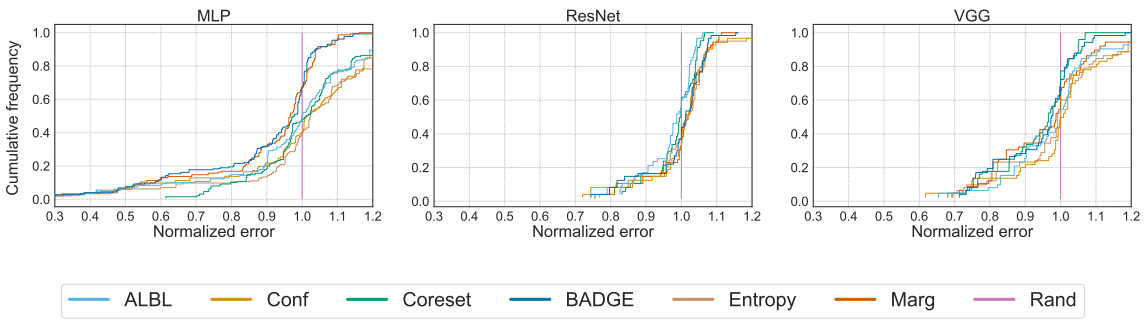


Figure A.18: CDFs of normalized errors of the algorithms, group by different neural network models. Higher CDF indicates better performance. From left to right: MLP, ResNet and VGG.

A.5 Batch uncertainty and diversity

Figure A.19 gives a comparison of sampling methods with gradient embedding in two settings (OpenML # 6, MLP, batchsize 100 and SVHN, ResNet, batchsize 1000), in terms of uncertainty and diversity of examples selected within batches. These two properties are measured by average ℓ_2 norm and determinant of the Gram matrix of gradient embedding, respectively. It can be seen that, k-MEANS++ (BADGE) induces good batch diversity in both settings. CONF generally selects examples with high uncertainty, but in some iterations of OpenML #6, the batch diversity is relatively low, as evidenced by the corresponding log Gram determinant being $-\infty$. These areas are indicated by gaps in the learning curve for CONF. Situations where there are many gaps in the CONF plot seem to correspond to situations in which CONF performs poorly in terms of accuracy (see Figure A.8 for the corresponding learning curve). Both k-DPP and FF-k-CENTER (an algorithm that approximately minimizes k-center objective) select batches that have lower diversity than k-MEANS++ (BADGE).

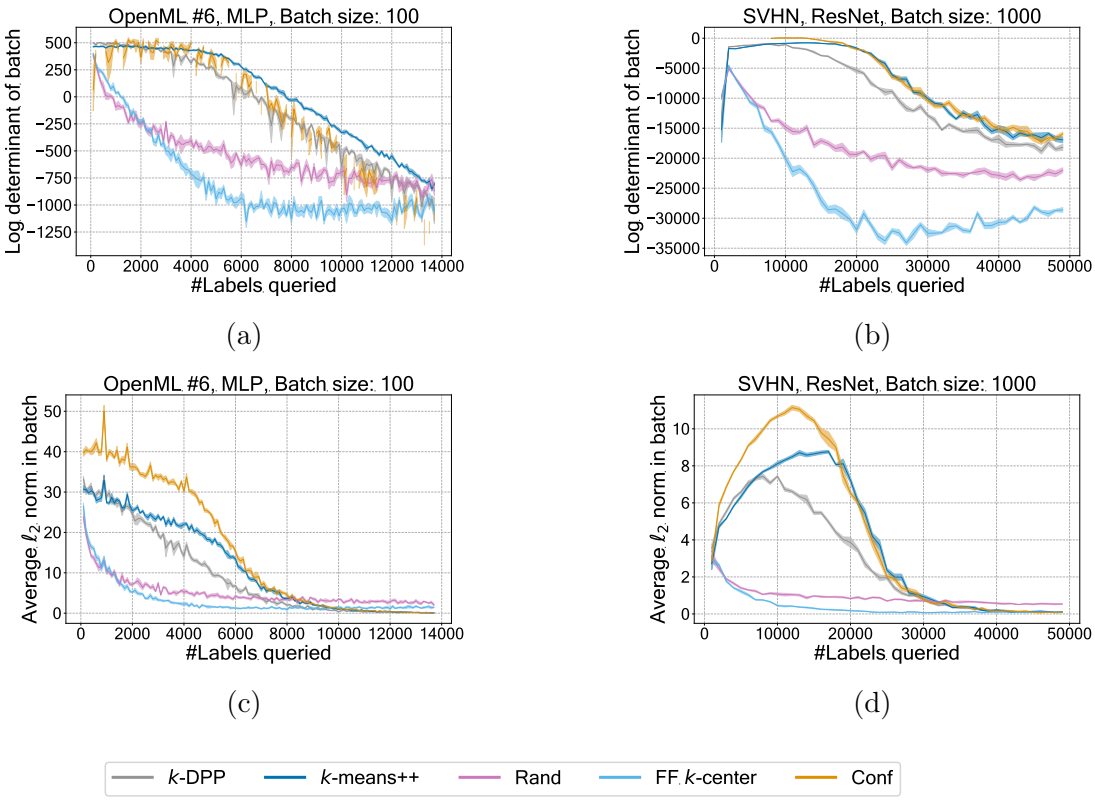


Figure A.19: A comparison of batch selection algorithms in gradient space. Plots **a and b** show the log determinants of the Gram matrices of gradient embeddings within batches as learning progresses. Plots **c and d** show the average embedding magnitude (a measurement of predictive uncertainty) in the selected batch. The k-centers sampler finds points that are not as diverse or high-magnitude as other samplers. Notice also that k-MEANS++ tends to actually select samples that are both more diverse and higher-magnitude than a k-DPP, a potential pathology of the k-DPP's degree of stochastisity. Among all algorithms, CONF has the largest average norm of gradient embeddings within a batch; however, in OpenML #6, and the first few interations of SVHN, some batches have a log Gram determinant of $-\infty$ (shown as gaps in the curve), which shows that CONF sometimes selects batches that are inferior in diversity.

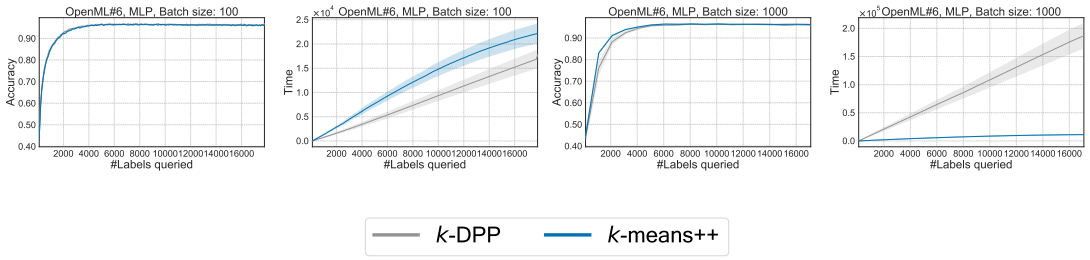


Figure A.20: Learning curves and running times for OpenML #6 with MLP.

A.6 Comparison of k-MEANS++ and k-DPP in batch selection

In Figures A.20 to A.26, 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 [51], which has a time complexity of $O(\tau \cdot (k^2 + kd))$ and space complexity of $O(k^2 + kd)$, 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++.

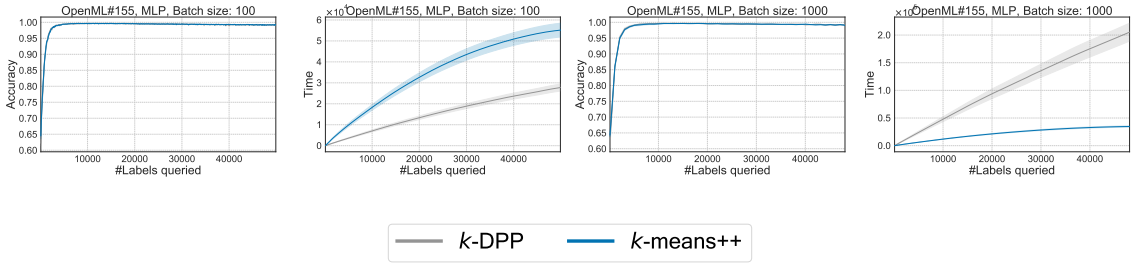


Figure A.21: Learning curves and running times for OpenML #155 with MLP.

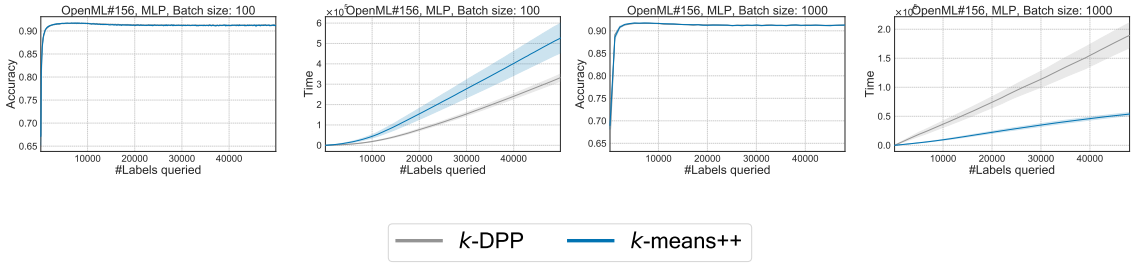


Figure A.22: Learning curves and running times for OpenML #156 with MLP.

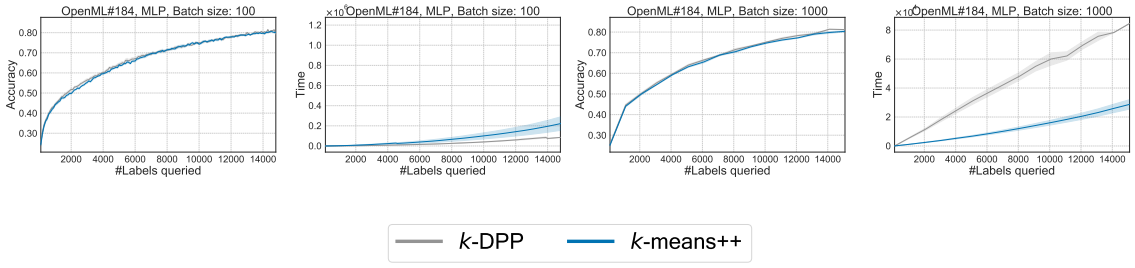


Figure A.23: Learning curves and running times for OpenML #184 with MLP.

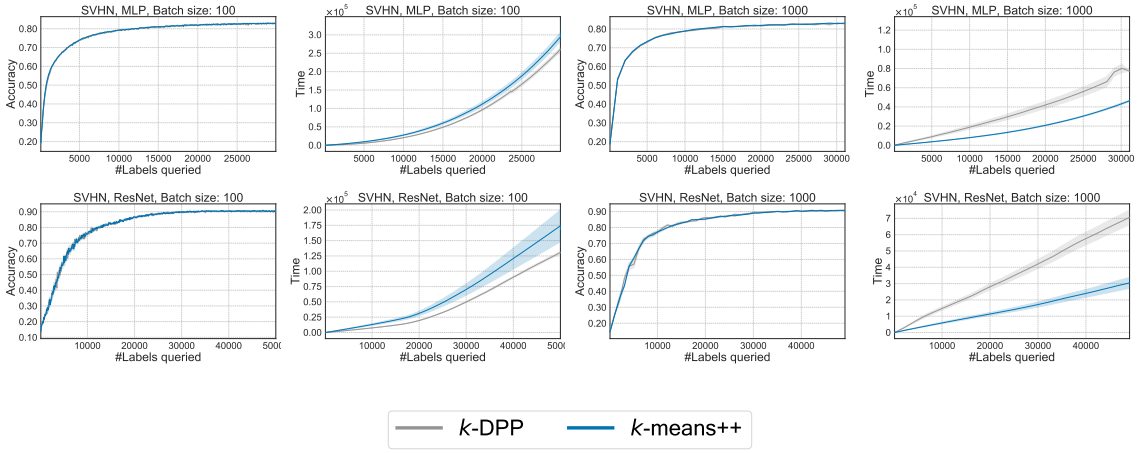


Figure A.24: Learning curves and running times for SVHN with MLP and ResNet.

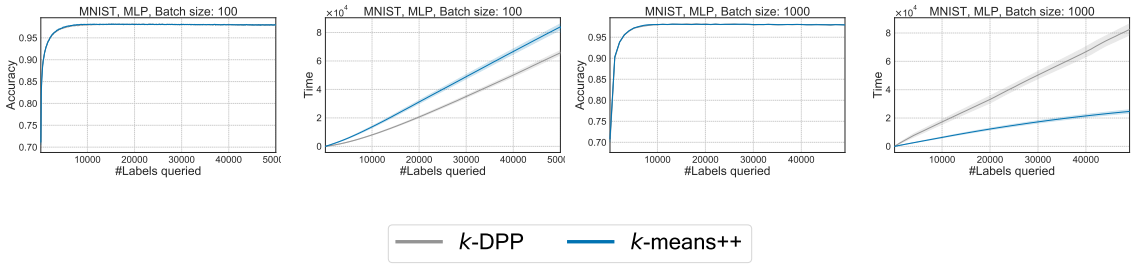


Figure A.25: Learning curves and running times for MNIST with MLP.

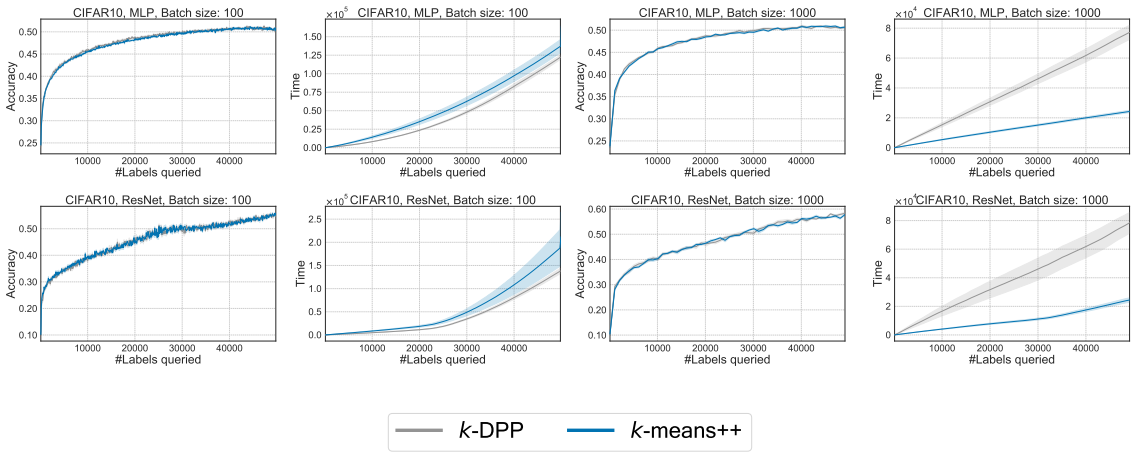


Figure A.26: Learning curves and running times for CIFAR10 with MLP and ResNet.

Appendix B

Warm Starting

Proposition 2. *Consider a neural network f_θ trained on d-dimensional samples to predict one of k classes and paramatrized by weight matrices $\theta = (W_1, W_2, .., W_L)$, such that $W_i \in \mathbb{R}^{r_i \times c_i}$, $r_i = c_{i-1}$, $c_1 = d$, and $r_L = k$. Using the ReLU nonlinearity $\sigma(z) = \max(0, z)$ and $\text{softmax}(z)_i = e^{z_i}/\sum_{j=1}^{K} e^{z_j}$, define $f_\theta(x) = \text{softmax}(W_L \cdot \sigma(W_{L-1} \cdot \ .. \ \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x))))$ for input $x \in \mathbb{R}^d$. Then, for $\lambda > 0$, $\text{argmax}\, f_\theta(x) = \text{argmax}\, f_{\lambda\theta}(x)$.*

Proof. Observe that $\sigma(\lambda z) = \lambda\sigma(z) \ \ \forall \lambda > 0$. Then,

$$\begin{aligned}
\text{argmax}\, f_{\lambda\theta}(x) &= \text{argmax}\, \text{softmax}(\lambda W_L \cdot \sigma(\lambda W_{L-1} \cdot \ .. \ \cdot \sigma(\lambda W_2 \cdot \sigma(\lambda W_1 \cdot x)))) \\
&= \text{argmax}\, \text{softmax}(\lambda^L W_L \cdot \sigma(W_{L-1} \cdot \ .. \ \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x)))) \\
&= \text{argmax}\, \text{softmax}(W_L \cdot \sigma(W_{L-1} \cdot \ .. \ \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x)))) \\
&= \text{argmax}\, f_\theta(x)
\end{aligned}$$

□

Table B.1: Validation percent accuracies for various optimizers and models for the first round of warm-started training, i.e. training on half of the training data available in Table 4.1. We consider an 18-layer ResNet, three-layer multilayer perceptron (MLP), and logistic regression (LR) as our classifiers. Validation sets are a randomly-chosen third of the training data. Standard deviations are indicated parenthetically.

	ResNet SGD	ResNet Adam	MLP SGD	MLP Adam	LR SGD	LR Adam
CIFAR-10	41.7 (7.9)	70.5 (1.6)	37.2 (0.2)	36.0 (0.2)	37.9 (0.2)	31.8 (0.7)
SVHN	85.9 (0.3)	92.3 (0.2)	72.5 (0.4)	67.5 (0.3)	27.1 (0.3)	22.2 (0.7)
CIFAR-100	10.6 (1.6)	31.5 (0.7)	10.3 (0.2)	10.5 (0.3)	15.4 (0.21)	9.3 (0.3)

Table B.2: Validation accuracies and warm-started model train times (minutes). Adding noise at the indicated standard deviations improves generalization, but not to the point of performing as well as randomly-initialized models. Better-generalizing warm-started models take even more time to train than their randomly-initialized peers, which on average achieve **55.2% accuracy in 34.0 minutes**.

	1×10^{-2}	1×10^{-3}	1×10^{-4}	1×10^{-5}	0
Accuracy	54.4 (0.9)	53.5 (1.0)	52.9 (1.0)	49.9 (1.6)	50.8 (1.8)
Train Time	165.3 (3.9)	38.0 (1.33)	16.5 (1.3)	14.6 (91.0)	13.6 (0.4)

Table B.3: Validation percent accuracies for various datasets for last layer only warm-starting (LL), last layer warm starting followed by full network training (LL+WS), warm started (WS) and randomly initialized (RI) models on various indicated datasets.

	LL	LL+WS	WS	RI
CIFAR-10	48.8 (1.8)	50.9 (1.5)	52.5 (0.3)	56.0 (1.2)
SVHN	86.0 (0.6)	88.2 (0.2)	87.5 (0.7)	89.4 (0.1)
CIFAR-100	16.4 (0.5)	16.5 (0.6)	15.5 (0.3)	18.2 (0.3)

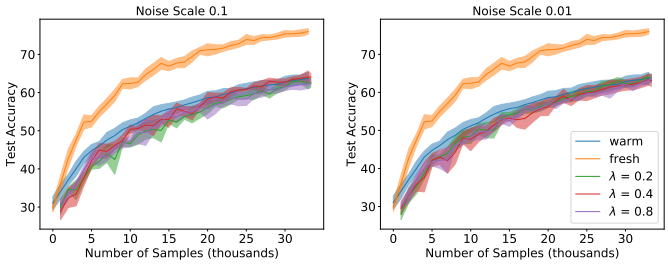


Figure B.1: An online learning experiment using a ResNet on CIFAR-10 data. Data are supplied iid in batches of 1,000. Here, instead of shrinking and perturbing every weight in the model, we modify only those in the last layer. Models modified this way, unlike the shrink-perturb trick we present, which modifies every parameter in the network, these retrained models are unable to outperform even purely warm-started models.

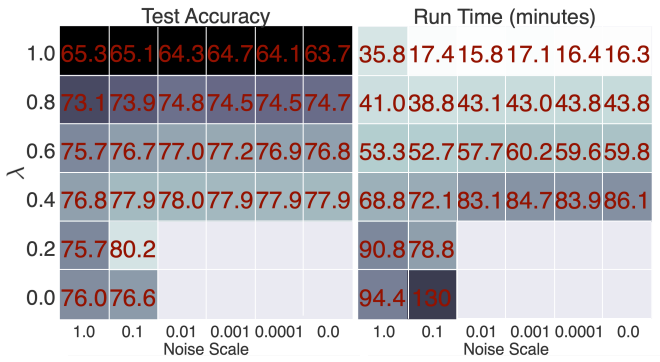
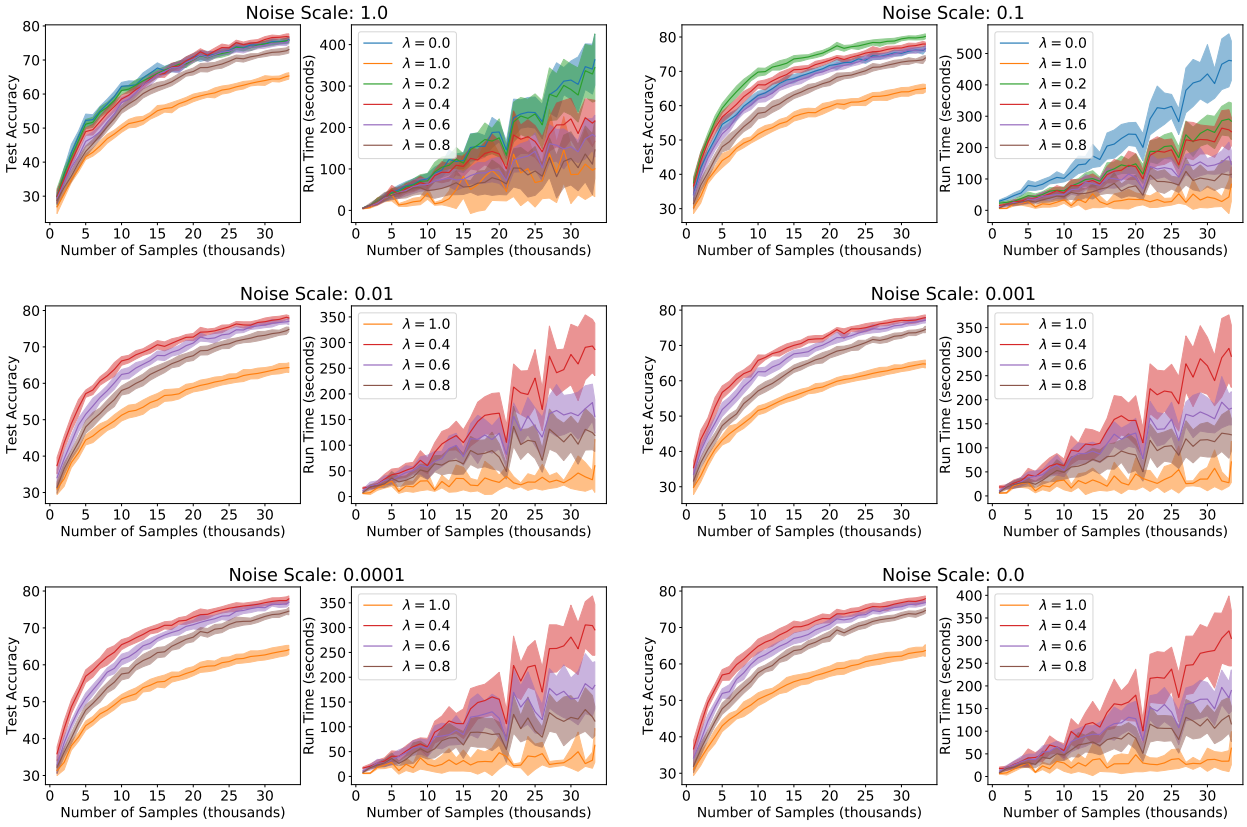


Figure B.2: An online learning experiment with a **ResNet on CIFAR-10 data**, iteratively supplying batches of 1,000 to the model. **Left:** Final accuracies and train times. **Below:** Full learning curves corresponding to each entry, where $\lambda = 0$ is warm starting and $\lambda = 1$ is randomly initializing .



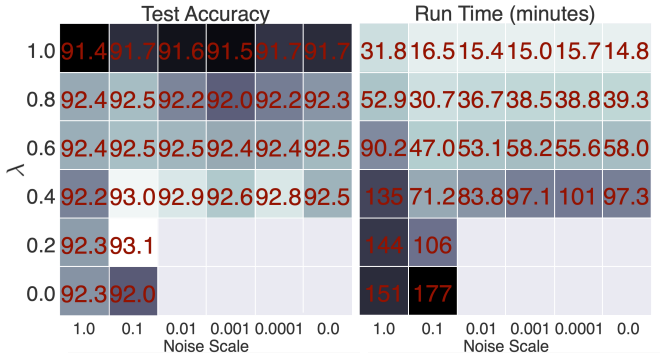
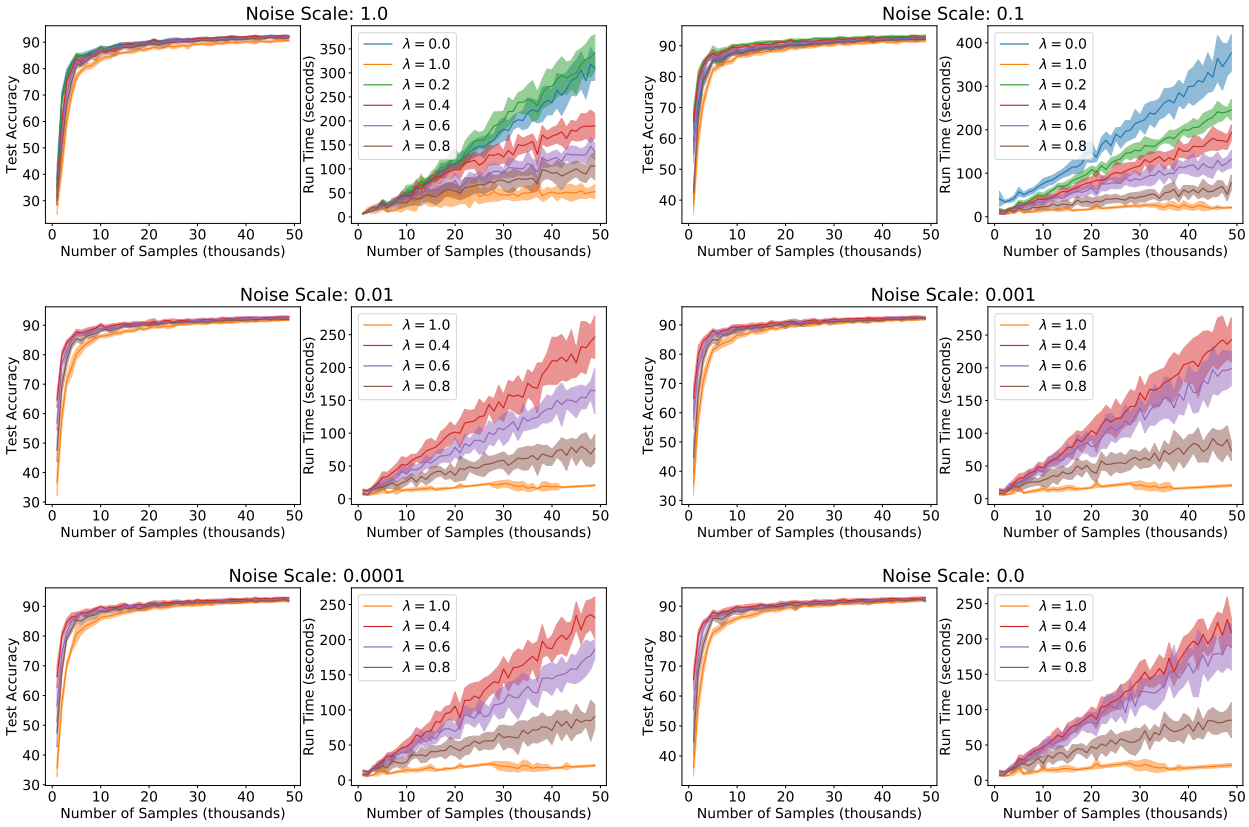


Figure B.3: An online learning experiment with a **ResNet on SVHN** data, iteratively supplying batches of 1,000 to the model. **Left:** Final accuracies and train times. **Below:** Full learning curves corresponding to each entry, where $\lambda = 0$ is warm starting and $\lambda = 1$ is randomly initializing .



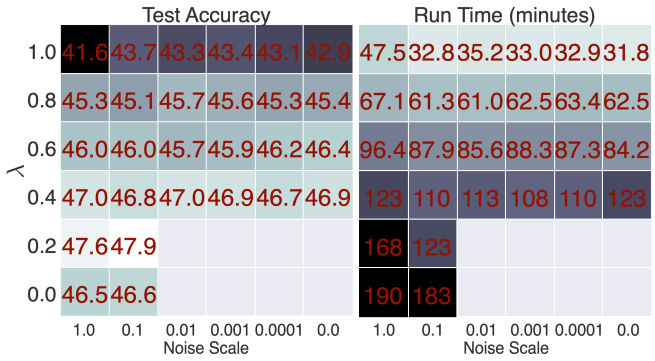


Figure B.4: An online learning experiment with an **MLP consisting of three layers, ReLU activations, and 100-dimensional hidden layers (no batch normalization) on CIFAR-10** data **Left:** Final accuracies and train times. **Below:** Full learning curves corresponding to each entry, where $\lambda = 0$ is warm starting and $\lambda = 1$ is randomly initializing.

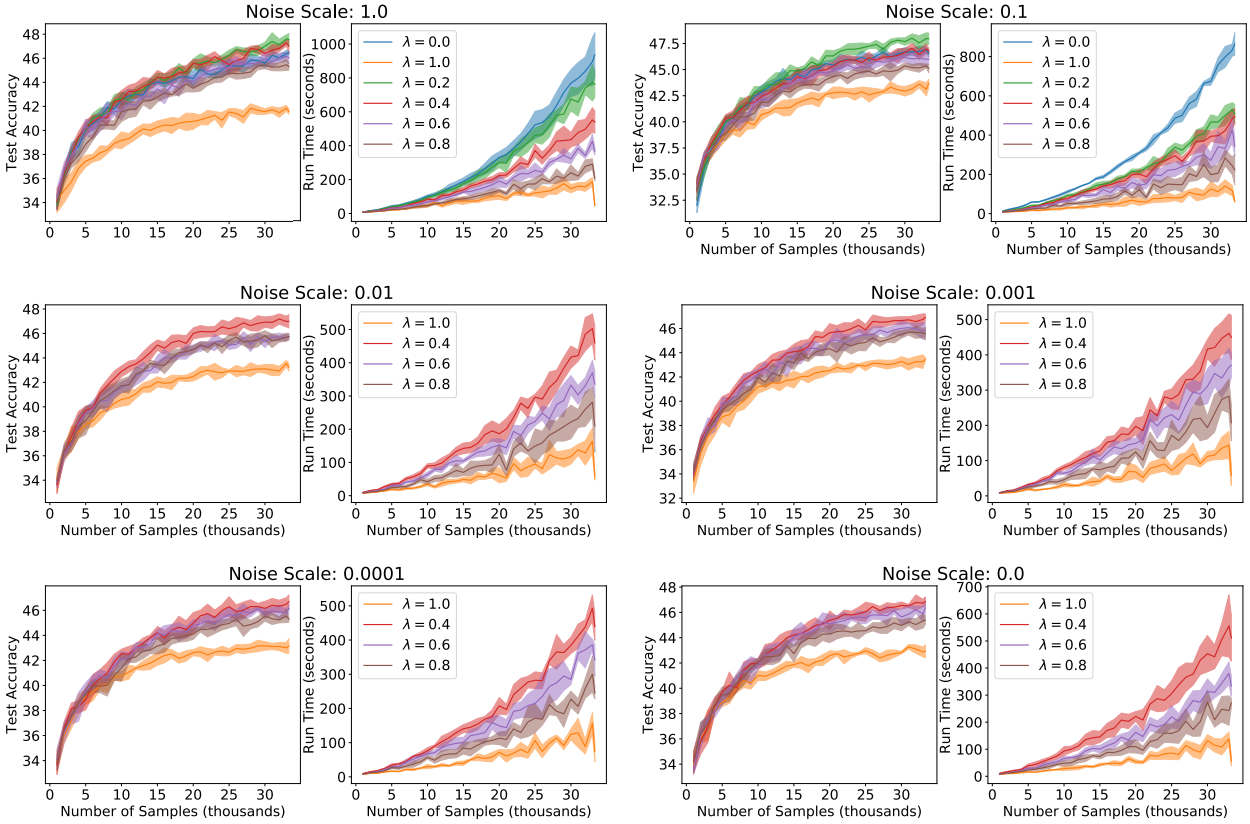


Figure B.5: An online learning experiment with an **MLP consisting of three layers, ReLU activations, and 100-dimensional hidden layers (no batch normalization) on SVHN** data **Left:** Final accuracies and train times. **Below:** Full learning curves corresponding to each entry, where $\lambda = 0$ is warm starting and $\lambda = 1$ is randomly initializing.

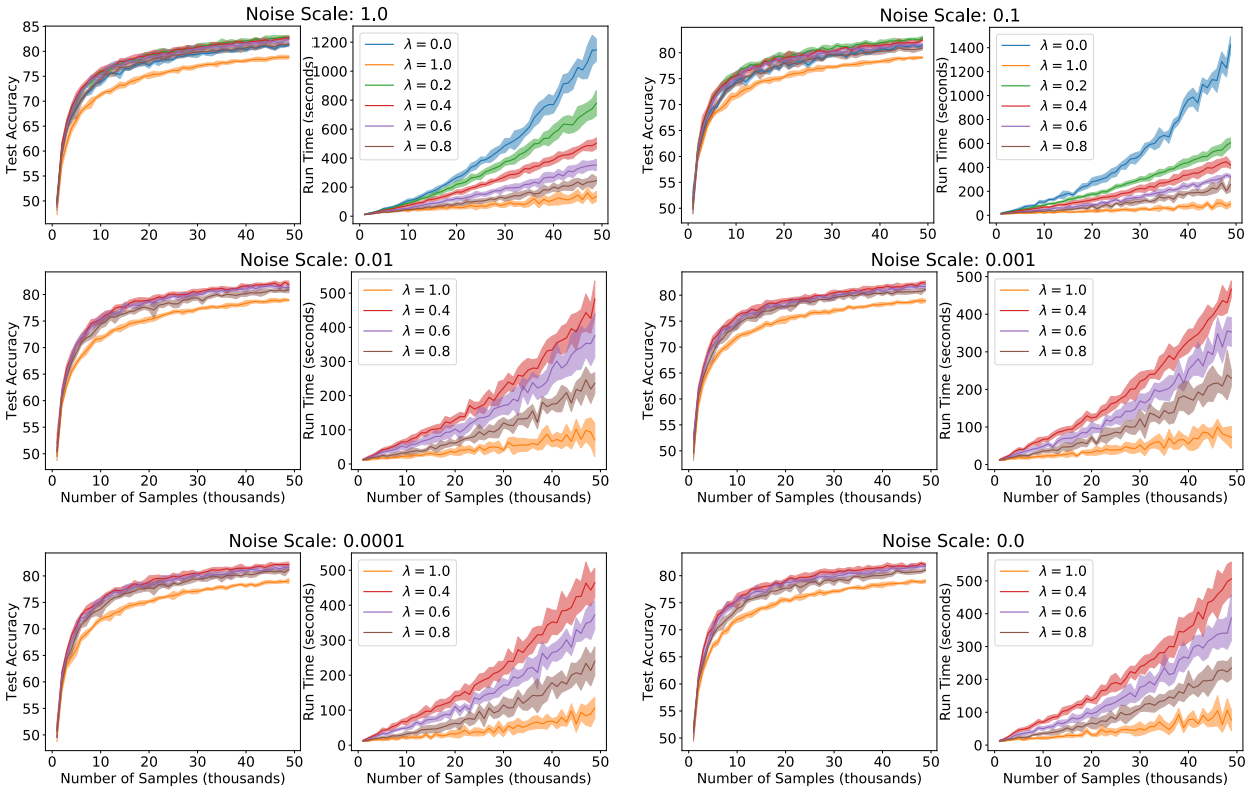
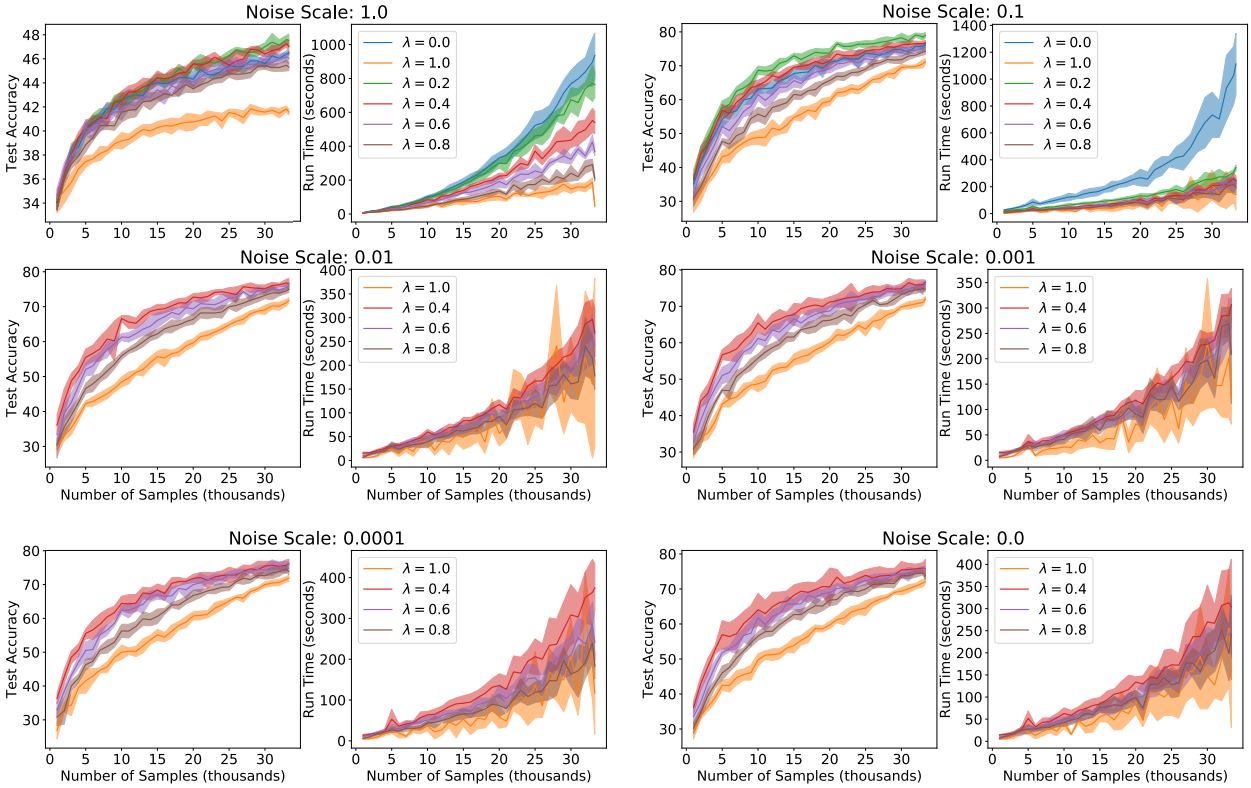


Figure B.6: An online learning experiment with a **ResNet with weight decay on CIFAR-10** data, iteratively supplying batches of 1,000 to the model. **Left:** Final accuracies and train times. **Below:** Full learning curves corresponding to each entry, where $\lambda = 0$ is warm starting and $\lambda = 1$ is randomly initializing .



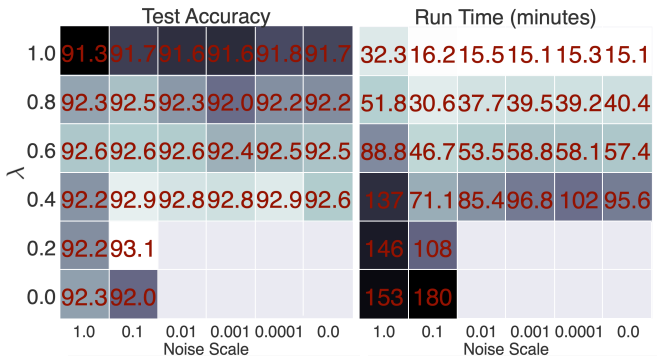
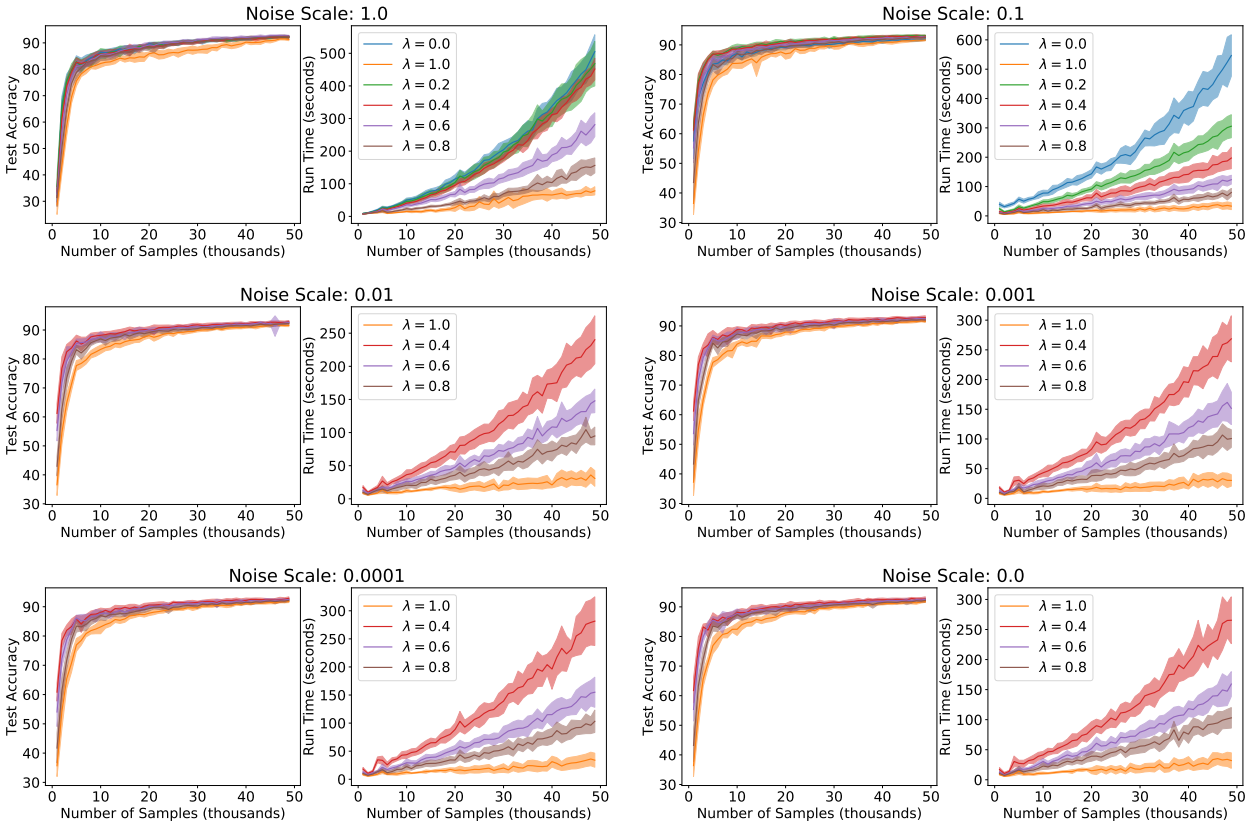


Figure B.7: An online learning experiment with a **ResNet with weight decay on SVHN** data, iteratively supplying batches of 1,000 to the model. **Left:** Final accuracies and train times. **Below:** Full learning curves corresponding to each entry, where $\lambda = 0$ is warm starting and $\lambda = 1$ is randomly initializing .



Bibliography

[1] Craig S Smith. Artificial intelligence: A special report. *The New York Times*, Apr 2020.

[2] John Seabrook. The next word. *The New Yorker*, Oct 2019.

[3] C P Gurnani. The ai revolution is here. it's up to businesses to prepare workers for it. *CNN*, May 2019.

[4] Mike Murphy. This is how much google is spending on cutting edge ai research. *Quartz*, Oct 2017.

[5] Olivia Krauth. The 10 tech companies that have invested the most money in ai. *Tech Republic*, Jan 2018.

[6] Joanna Bryson. The future of ai's impact on society. *MIT Technology Review*, Dec 2019.

[7] Bernard Marr. https://www.theguardian.com/technology/2020/jan/18/automation-jobs-universal-basic-income-daniel-susskind-interview. *Forbes*, Mar 2020.

[8] Stephen Talty. What will our society look like when artificial intelligence is everywhere. *Smithsonian Magazine*, Apr 2018.

[9] Ian Tucker. Daniel susskind: 'automation of jobs is one of the greatest questions of our time'. *The Guardian*, Jan 2020.

[10] Alekseĭ Ivakhnenko. Cybernetic predicting devices. Technical report.

[11] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Robert Geirhos, David HJ Janssen, Heiko H Schütt, Jonas Rauber, Matthias Bethge, and Felix A Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker. *arXiv preprint arXiv:1706.06969*, 2017.

[15] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[16] Rostamiz. `https://github.com/ej0cl6/deep-active-learning`, 2017–2019.

[17] Kuan-Hao Huang. `https://github.com/ej0cl6/deep-active-learning`, 2018–2019.

[18] Mihalj Bakator and Dragica Radosav. Deep learning and medical diagnosis: A review of literature. *Multimodal Technologies and Interaction*, 2(3):47, 2018.

[19] Tongxue Zhou, Su Ruan, and Stéphane Canu. A review: Deep learning for medical image segmentation using multi-modality fusion. *Array*, page 100004, 2019.

[20] June-Goo Lee, Sanghoon Jun, Young-Won Cho, Hyunna Lee, Guk Bae Kim, Joon Beom Seo, and Namkug Kim. Deep learning in medical imaging: general overview. *Korean journal of radiology*, 18(4):570–584, 2017.

[21] Shengfeng Liu, Yi Wang, Xin Yang, Baiying Lei, Li Liu, Shawn Xiang Li, Dong Ni, and Tianfu Wang. Deep learning in medical ultrasound analysis: a review. *Engineering*, 2019.

[22] Patrick L Fitzgibbons, David L Page, Donald Weaver, Ann D Thor, D Craig Allred, Gary M Clark, Stephen G Ruby, Frances O'Malley, Jean F Simpson, James L Connolly, et al. Prognostic factors in breast cancer: College of american pathologists consensus statement 1999. *Archives of Pathology & Laboratory Medicine*, 124(7):966–978, 2000.

[23] Cigdem Demir and Bülent Yener. Automated cancer diagnosis based on histopathological images: a systematic survey. *Rensselaer Polytechnic Institute, Tech. Rep*, 2005.

[24] Hojjat Seyed Mousavi, Vishal Monga, Ganesh Rao, Arvind UK Rao, et al. Automated discrimination of lower and higher grade gliomas based on histopathological image analysis. *Journal of Pathology Informatics*, 6(1):15, 2015.

[25] Mitko Veta, Josien PW Pluim, Paul J Van Diest, and Max A Viergever. Breast cancer histopathology image analysis: A review. *IEEE Transactions on Biomedical Engineering*, 61(5):1400–1411, 2014.

[26] GTEx Consortium et al. Genetic effects on gene expression across human tissues. *Nature*, 550(7675):204, 2017.

[27] Laura J Van't Veer, Hongyue Dai, Marc J Van De Vijver, Yudong D He, Augustinus AM Hart, Mao Mao, Hans L Peterse, Karin van der Kooy, Matthew J Marton, Anke T Witteveen, et al. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536, 2002.

[28] Hans Ellegren and John Parsch. The evolution of sex-biased genes and sex-biased gene expression. *Nature Reviews Genetics*, 8(9):689–698, 2007.

[29] Lara M Mangravite, Barbara E Engelhardt, Marisa W Medina, Joshua D Smith, Christopher D Brown, Daniel I Chasman, Brigham H Mecham, Bryan Howie, Heejung Shim, Devesh Naidoo, et al. A statin-dependent QTL for *GATM* expression is associated with statin-induced myopathy. *Nature*, 502(7471):377–380, 2013.

[30] Wellcome Trust Case Control Consortium et al. Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. *Nature*, 447(7145):661, 2007.

[31] Kok Hao Chen, Alistair N Boettiger, Jeffrey R Moffitt, Siyuan Wang, and Xiaowei Zhuang. Spatially resolved, highly multiplexed RNA profiling in single cells. *Science*, 348(6233):aaa6090, 2015.

[32] Sheel Shah, Eric Lubeck, Wen Zhou, and Long Cai. seqFISH accurately detects transcripts in single cells and reveals robust spatial organization in the hippocampus. *Neuron*, 94(4):752–758, 2017.

[33] Heba Z Sailem and Chris Bakal. Identification of clinically predictive metagenes that encode components of a network coupling cell shape to transcription by image-omics. *Genome Research*, pages gr–202028, 2016.

[34] Mitko Veta, Paul J van Diest, Robert Kornegoor, André Huisman, Max A Viergever, and Josien PW Pluim. Automatic nuclei segmentation in H&E stained breast cancer histopathology images. *PloS One*, 8(7):e70221, 2013.

[35] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[36] Harold Hotelling. Relations between two sets of variates. *Biometrika*, pages 321–377, 1936.

[37] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.

[38] Adrian Alexa and Jorg Rahnenfuhrer. *topGO: Enrichment Analysis for Gene Ontology*, 2016. R package version 2.30.1.

[39] Marc Carlson. *org.Hs.eg.db: Genome wide annotation for Human.* R package version 3.5.0.

[40] Marc Carlson. *GO.db: A set of annotation maps describing the entire Gene Ontology.* R package version 3.5.0.

[41] Cancer Genome Atlas Network et al. Comprehensive molecular portraits of human breast tumours. *Nature*, 490(7418):61–70, 2012.

[42] Cancer Genome Atlas Research Network et al. Comprehensive, integrative genomic analysis of diffuse lower-grade gliomas. *New England Journal of Medicine*, 2015(372):2481–2498, 2015.

[43] Qingsen Li, Abhishek Kumar, Ekta Makhija, and GV Shivashankar. The regulation of dynamic mechanical coupling between actin cytoskeleton and nucleus by matrix geometry. *Biomaterials*, 35(3):961–969, 2014.

[44] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 2015.

[45] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 1994.

[46] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *International Conference on Machine Learning*, 2006.

[47] Alina Beygelzimer, Daniel J Hsu, John Langford, and Tong Zhang. Agnostic active learning without constraints. In *Neural Information Processing Systems*, 2010.

[48] Nicolo Cesa-Bianchi, Claudio Gentile, and Francesco Orabona. Robust bounds for classification via selective sampling. In *International Conference on Machine Learning*, 2009.

[49] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *ACM-SIAM symposium on Discrete algorithms*, 2007.

[50] Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *International Conference on Machine Learning*, 2011.

[51] Byungkon Kang. Fast determinantal point process sampling with application to clustering. In *Neural Information Processing Systems*, 2013.

[52] 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*, 2016.

[53] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[54] Dan Wang and Yi Shang. A new active labeling method for deep learning. In *2014 International joint conference on neural networks*, 2014.

[55] Dan Roth and Kevin Small. Margin-based active learning for structured output spaces. In *European Conference on Machine Learning*, 2006.

[56] Wei-Ning Hsu and Hsuan-Tien Lin. Active learning by learning. In *Association for the advancement of artificial intelligence*, 2015.

[57] 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.

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

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

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

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[62] Jordan T Ash and Ryan P Adams. On warm-starting neural network training. *arXiv preprint*, 2020.

[63] 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*, 2017.

[64] 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. 2017.

[65] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at Facebook. In *Workshop on Data Mining for Online Advertising*, pages 1–9. ACM, 2014.

[66] Badrish Chandramouli, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel. StreamRec: a real-time recommender system. In *International Conference on Management of Data*, pages 1243–1246. ACM, 2011.

[67] Ludmila I Kuncheva. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In *2nd Workshop SUEMA*, volume 2008, pages 5–10, 2008.

[68] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

[69] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *arXiv preprint arXiv:1907.10597*, 2019.

[70] Liu Yang and Jaime Carbonell. Buy-in-bulk active learning. In *Advances in Neural Information Processing Systems*, pages 2229–2237, 2013.

[71] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *ICLR*, 2020.

[72] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[73] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *International Conference on Unsupervised and Transfer Learning Workshop*, pages 17–37. JMLR. org, 2011.

[74] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[75] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[76] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. JMLR. org, 2017.

[77] Jordan T Ash, Robert E Schapire, and Barbara E Engelhardt. Unsupervised domain adaptation using approximate label matching. *arXiv preprint arXiv:1602.04889*, 2016.

[78] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[79] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[80] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*, 2016.

[81] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. pages 75:2–47, 2018.

[82] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

[83] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, pages 950–957, 1992.

[84] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[85] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[86] Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in NLP applications? In *Empirical Methods in Natural Language Processing*, pages 478–489, 2016.

[87] Andrej Karpathy and Justin Johnson. Course notes: Transfer learning, CS231n convolutional neural networks for visual recognition. Course Notes, 2019.

[88] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[89] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[90] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

[91] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[92] Sanjoy Dasgupta. Two faces of active learning. *Theoretical computer science*, 2011.

[93] Steve Hanneke. Theory of disagreement-based active learning. *Foundations and Trends in Machine Learning*, 2014.

[94] Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In *Neural Information Processing Systems*, 2008.

[95] Zheng Wang and Jieping Ye. Querying discriminative and representative samples for batch mode active learning. *Transactions on Knowledge Discovery from Data*, 2015.

[96] Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In *International Conference on Machine Learning*.

[97] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*, 2015.

[98] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. In *Neural Information Processing Systems 32*, 2019.

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

[100] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2001.

[101] Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In *International Conference on Machine Learning*, 2000.

[102] Gokhan Tur, Dilek Hakkani-Tür, and Robert E Schapire. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 2005.

[103] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, 2017.

[104] David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

[105] Sethu Vijayakumar and Hidemitsu Ogawa. Improving generalization ability through active learning. *IEICE Transactions on Information and Systems*, 82(2):480–487, 1999.

[106] Sethu Vijayakumar, Masashi Sugiyama, and Hidemitsu Ogawa. Training data selection for optimal generalization with noise variance reduction in neural networks. In *Neural Nets WIRN Vietri-98*, pages 153–166. Springer, 1999.

[107] Mark Plutowski and Halbert White. Active selection of training examples for network learning in noiseless environments. *Dept. Computer Science, UCSD, TR*, pages 90–011, 1991.

[108] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[109] Yoram Baram, Ran El Yaniv, and Kobi Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 2004.

[110] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. In *Neural Information Processing Systems*, 2010.

[111] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In *Neural Information Processing Systems*, 2008.

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

[113] Ye Zhang, Matthew Lease, and Byron C Wallace. Active discriminative text representation learning. In *AAAI Conference on Artificial Intelligence*, 2017.

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

[115] HaiYing Wang, Rong Zhu, and Ping Ma. Optimal subsampling for large sample logistic regression. *Journal of the American Statistical Association*, 2018.

[116] Daniel Ting and Eric Brochu. Optimal subsampling with influence functions. In *Neural Information Processing Systems*, 2018.

[117] Cheng Zhang, Hedvig Kjellstrom, and Stephan Mandt. Determinantal point processes for mini-batch diversification. *Uncertainty in Artificial Intelligence*, 2017.

[118] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Neural Information Processing Systems*, 2017.

[119] Bo-Yu Chu, Chia-Hua Ho, Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Warm start for parameter selection of linear classifiers. In *International Conference on Knowledge Discovery and Data Mining*, pages 149–158. ACM, 2015.

[120] Dennis DeCoste and Kiri Wagstaff. Alpha seeding for support vector machines. In *International Conference on Knowledge Discovery and Data Mining*, pages 345–349. ACM, 2000.

[121] Zeyi Wen, Bin Li, Ramamohanarao Kotagiri, Jian Chen, Yawen Chen, and Rui Zhang. Improving efficiency of SVM k-fold cross-validation by alpha seeding. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[122] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788, 2018.

[123] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

[124] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[125] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385, 2015.

[126] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[127] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[128] Vaishnavh Nagarajan and J Zico Kolter. Generalization in deep networks: The role of distance from initialization. In *Neural Information Processing Systems*, 2017.

[129] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

[130] Colin Wei, Jason D Lee, Qiang Liu, and Tengyu Ma. On the margin theory of feedforward neural networks. *arXiv preprint arXiv:1810.05369*, 2018.

[131] Sepp Hochreiter and J Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[132] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[133] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[134] Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-Rao metric, geometry, and complexity of neural networks. pages 888–896, 2017.

[135] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations*, 2018.

[136] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: a PAC-Bayesian compression approach. In *International Conference on Learning Representations*, 2019.

[137] Hengduo Li, Bharat Singh, Mahyar Najibi, Zuxuan Wu, and Larry S Davis. An analysis of pre-training on object detection. *arXiv preprint arXiv:1904.05871*, 2019.

[138] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.