Exploratory model analysis for machine learning

Biye Jiang

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-34
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-34.html

May 9, 2018

Exploratory model analysis for machine learning

by

Biye Jiang

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Canny, Chair
Professor Marti Hearst
Professor Joshua Blumenstock

Spring 2018

Exploratory model analysis for machine learning

Abstract

Exploratory model analysis for machine learning

by

Biye Jiang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor John Canny, Chair

Machine learning is growing in importance in many different fields. However, it is still very hard for users to tune hyper-parameters when optimizing their models, or perform a comprehensive and interpretable diagnosis for complex models like deep neural nets. Existing developer tool like TensorBoard only provides limited functionality which usually visualizes model statistics based on metrics predefined before the training starts. Almost nothing can be adjusted during the training. But the real model optimization and diagnosis procedures actually involve lots of interaction and continuous experiments. To tackle those challenges, we developed a framework which allows users to perform exploratory model analysis based on the hardware accelerated machine learning toolkit BIDMach. Our system is unique that we allow users to interactively add visualizations and adjust hyper-parameters during training. Also, we use Monte Carlo style algorithms to allow users to explore the entire model space under different user preferences rather than only reaching the local optimums.

We demonstrate the usage of our system in several real-world applications. For problems like advertisement optimization or clustering where multiple optimization objectives exist, users can incorporate secondary criteria into the model-generation process and make trade-offs in an interactive way. For deep convolution neural net diagnosis, users can use our LDAM (Langevin Dynamic Activation Maximization) algorithm to systematically explore the images that stimulate a given neuron. We conduct experiments and user studies on several public datasets to show how users from different background can benefit from our tools.

To my parents and the star brightening my way

Contents

List of Figures

List of Tables

Acknowledgments

The way towards PhD is really a long journey. I am glad that I have received so much help and support along the way. First of all, I would like to thank my advisor Professor John Canny for mentoring and guiding me these years. I first met John in March 2013, during the PhD visit day at Berkeley. I didn't recognize that he was the author of Canny edge detection at that time. However, I was amazed by the BIDMach project, which is a large-scale machine learning framework leveraging GPU acceleration and the beautiful Scala language. Also, BIDMach is designed for interactive purpose which is a perfect fit for me as my research interest is interactive visualization for machine learning. From our first conversation, I already learned that John has a very clear understanding about almost every single detail of the existing systems and algorithms. Therefore, over the years, I have received a tremendous amount of comments and advice at a very detailed level. John and I always sat side by side to diagnose and fix the bugs together. Focusing on the details turns out to be very important for machine learning nowadays, as more and more phenomenon can not be explained by the general theory. And the tools we will describe in this dissertation are indeed designed to accelerate such process: Use empirical observation to diagnose and refine the complicated machine learning models. On the other hand, conducting interdisciplinary research in designing machine learning tools requires knowledge from both AI, system and HCI. John's wide range of interests and experiences in all those fields make our discussions very helpful and constructive. John also encouraged me to take more classes and seminars to expand my knowledge. We also organized two workshops together. I am really inspired by John's passion to learn and accept new things. Life is long and I will keep pushing myself forward in the future.

During these years, I also learned a lot from many Berkeley professors. I would like to thank Maneesh Agrawala, who provided the research funding for my first year study, and gave me a lot of guidances on visualization research. I would like to thank Marti Hearst who is on my dissertation committee and served as the chair of my qualification exam committee. Marti is always good at pointing out the usability problems and questioning whether or not we can bring value to the users. I would like to thank Joshua Blumenstock who is also on my dissertation committee and introduced me several interesting machine learning applications for social good. And thanks to Anca Dragan for being on my qualification exam committee. Thanks to Joey Gonzalez, Deborah Nolan, Fernando Perez for teaching the Data Science 100 class together. Thanks to Bin Yu, Björn Hartmann, Eric Paulos, Armando Fox, Michael Franklin for the helpful discussions. Thanks to Paul Hilfinger for organizing the Berkeley programming contest so that I was able to participate the ACM/ICPC World Finals. Thanks to Ion Stoica and the AMPLab for providing me the research funding.

I would also like to thank the collaborators at Berkeley that conducted research with me. Thanks Huasha Zhao for helping me a lot during my first year here. Thanks Pablo Paredes for providing me so many suggestions from the HCI perspective. Thanks Yuansi Chen for having so many discussions with me about statistical properties of deep learning. And thanks David Chan for the pioneering work on the diagnostic visualization project.

And thanks to all the master and undergraduate students that have been working on the BIDMach visualization tools.

Also, I would like to thank all the internship mentors that I have worked with during the summers. Thanks to Zhicheng Liu, Mira Dontcheva from Adobe Research; Saleema Amershi, Ran Gilad-Bachrach from Microsoft Research; Jaehyung Lee from Gracenote and Kat Ellis, Leo Dirac from Amazon. Those exciting internship projects about visualization, machine learning have lots of impact on the research I have done so far.

And thanks to my undergraduate research advisor at Tsinghua University: Kun Xu, Shimin Hu, Kang Chen. From there I started the journey of research. Also thanks to Jeffrey Heer for introducing me to the area of visualization during the Stanford summer research program in 2012. This is the turning point of my research career.

And thanks to all the friends in the research community that have helped me in the past. Thanks to Polo Chau, Aditya Khosla, Xiangmin Fan for organizing the visualization for deep learning workshop with us. Thanks to Junyan Zhu for inspiring me a lot on the interactive machine learning and many helpful comments for our paper. Thanks Chang Lan for working on so many class projects and reading group meetings together. Thanks Chi Jin and Yubei Chen for teaching me so many ideas from theoretical machine learning and statistics. In addition, thanks to Lingqi Yan, Tinghui Zhou, Hang Su, Yijing Li, Yanpei Cao, Bin Liu, Tian Tian, Yang Gao, Jinkyu Kim, Yichao Zhou, Yan Duan, Haoran Tang, Xuaner Zhang, Zhuang Liu, Weixin Cai for helpful discussions in machine learning and computer graphics; thanks to Liang Gong, Jiannan Wang, Zongheng Yang, Xin Wang, Qijing Huang, Ben Zhang, Reynold Xin for helpful discussions in computer systems; thanks to Yeshuang Zhu, Haijun Xia, Andrew Head, Yifan Wu, Cesar Torres, Hezheng Yin, An Ju for helpful discussions in human computer interaction. Thanks to Yi Wu, Luhang Lai, Yan Duan and Lewin Gan for being on the Berkeley ACM/ICPC programming team together. Thanks Sen Wu, Zhihao Jia, Shike Mei for hosting all kinds of events in the bay area. Also thanks to Daniel Seita who setup a standard for me about what a good PhD student should be.

At last, I would like to thank my parents. Without you, none of those could ever happen. And sorry that I didn't have enough time to be with you these years. Also, special thanks to Shibei Deng. You make me understand that there are words that one should keep inside and there are things that one should dedicate to.

Thanks everyone for accompanying me on this wonderful journey!

Chapter 1

Introduction

1.1 Motivation

Machine learning has been widely used in many different fields and lots of machine learning algorithms have been proposed. People are using machine learning to recognize objects from images, find clusters or topics from millions of documents, recommend movie or product for customers, and even learn to play Go against the best human players.

As defined in Wikipedia, machine learning is a field of computer science that uses statistical techniques to give computer the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. This is kind of revealing the magic behind machine learning, that the algorithm can learn by itself as long as we feed it with data. As we are entering the era of big data, machine learning community benefits a lot from the massive amount of data that we can collect and process nowadays. Also, the recent success of deep learning introduces the concept of end-to-end learning and saves lots of human labor which is previously used for feature engineering.

However, we should notice that, the triumph of machine learning or deep learning is also due to the advancement of developer tools [1, 9, 46]. The modern tools now enable users to easily prototype machine learning models using high-level programming languages like Python, Scala, Lua while providing high-performance backend implementations leveraging hardwares like GPU. But those developer tools are far from perfect. The real workflow for machine learning can still be very tedious and painful for both experts and beginners. Users need to conduct many experiments to choose between different loss functions, model structures and hyper-parameters. As models become more and more complex, debugging and diagnosis also become very difficult.

As the typical training time for a machine learning model usually takes minutes or hours, and even days, people usually analyze the training process by collecting statistics during training and conduct the analysis after training. Such workflow also affects the design and research for machine learning tools. As analysis is usually done after training ends, most tools seem to assume no information is being gained during training, so that they don't allow

users to modify what kind of statistics should be collected after the training starts.

Another assumption people usually made is we should not manually adjust hyper-parameters during training, otherwise the results may not be reproducible. This is because most machine learning algorithms use gradient descent style optimization on a non-convex landscape. Adjusting the hyper-parameters could make the algorithm converge to a different local optimum even if you switch back to the original hyper-parameters. However, we can take the view from Bayesian machine learning [96], that we should not obtain just one single model after training, but should obtain model samples from the posterior distribution. Recent research also shows that, using methods like snapshot ensemble [43] even get better performance by combining those model samples.

Therefore, we break those assumptions and introduce exploratory model analysis which allows users to plug in new visualizations, and adjust hyper-parameters during training. We build our system using the open source machine learning framework BIDMach [14]. In this dissertation, we will discuss the implementation of our system and several real-world applications.

1.2 BIDMach: high-performance, customized machine learning toolkit

The first key to interactive and customized machine learning is an architecture which supports it. BIDMach [14] is a machine learning toolkit which has demonstrated extremely high performance with modest hardware (single computer with GPUs), and which has the modular design shown in Fig 1.1. BIDMach uses minibatch updates, typically many per second, so that models are being updated continuously. This is a good match to interactive modeling, since the effects of analysts actions will be seen quickly. Rather than a single model class, the system comprises first a primary model (which typically outputs the model loss on a minibatch and a derivative or other update for it). Next an optimizer is responsible for updating the model using the gradients. Several are available including simple SGD, AdaGrad [26] etc. Finally, mixins represent secondary constraints or likelihoods. Gradient-based primary models and mixins are combined with a weighted sum. In our interactive context, these weights are set interactively.

BIDMach uses *roofline design* [97] to optimize computational kernels toward hardware limits. On a large collection of benchmarks it has proved to be typically two orders of magnitude faster than other single-machine toolkits (when BIDMach is used with a GPU), and one to two orders of magnitude faster than cluster toolkits running on 10-100 nodes. Part of the difference is due to BIDMach's complete suite of GPU primitives. Almost all computation is done on the GPU, CPU/GPU transfers are minimized, and custom kernels give close to theoretically optimal GPU performance. GPUs typically achieve an order-of-magnitude speedup in dense matrix operations vs. mid-range CPUs.

Less well known is their advantage in main memory speed, which (at 300 GB/s) is

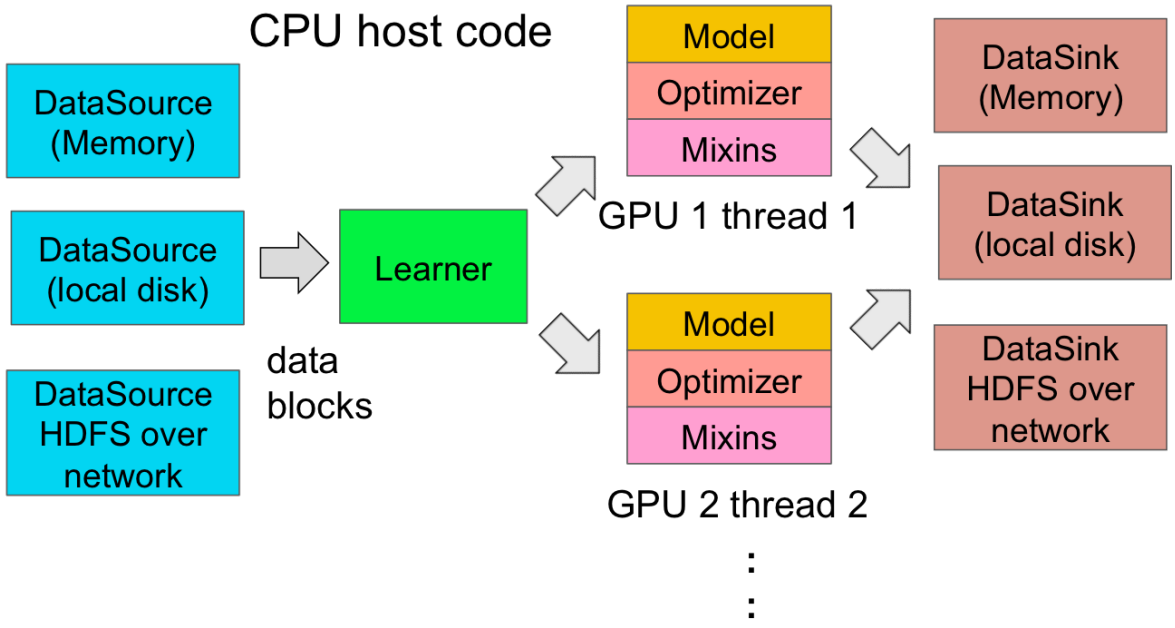


Figure 1.1: BIDMach's Architecture

nearly an order-of-magnitude faster than recent quad-channel CPUs (at around 40 GB/s). This memory speed gap also gives GPUs a similar advantage for *sparse* matrix operations which are central to most real-world ML applications. These differences explain one order of magnitude of the performance gap that we observe with BIDMach. The balance is due to the fact that most other systems are not close to their (CPU) rooflines. Because of this BIDMach has a significant performance edge for most ML algorithms even when run on one CPU.

High performance is very important for interactivity. BIDMach has reduced the running time of many non-trivial ML tasks from hours to minutes. And even for models that take minutes to train fully, the effects of parameter changes are typically visible in seconds due to mini-batch online learning algorithm.

BIDMach provides the building blocks for our interactive toolkit. In later chapters, we will discuss in details about how to to implement systems for exploratory model analysis using BIDMach.

Chapter 2

Interactive customized optimiztion

2.1 Introduction

Machine learning has evolved around the optimization of a single, usually narrowly-defined criterion. However, in many areas, users really want solutions with good performance in multiple dimensions. e.g. in computational advertising, the exchanges' primary goal is to maximize revenue. But they must also satisfy marketers by providing sufficient high-quality ad impressions, and they must placate end-users by not deluging them with ads. More generally, unsupervised models often under-specify users goals. Basic clustering methods (e.g. K-Means) minimize only sample-to-centroid distance, and there is no control over cluster size. But users often expect different clusters to be well-separated and similar-sized. Topic models such as Latent Dirichlet Allocation (LDA) produces topics that best predict the data, but users often want those topics to be well-separated which LDA does not model. Finally, many unsupervised methods including K-Means and LDA, optimize non-convex criteria with many local modes. Simulated annealing is one approach to improving model quality, but it requires careful tuning and scheduling. Even for supervised learning, we need to tune L1/L2 regularization to trade-off training accuracy and validation accuracy, or even AIC (Akaike information criterion)/ BIC(Bayesian information criterion) which are used to measure generalization ability. Therefore, including human control can steer the solution more rapidly toward an optimum.

Because ML models today are not flexible enough to incorporate all these criteria, secondary constraints are often applied *after* model training (by overriding the model's choices) in a way that is inevitably sub-optimal. By contrast, *interactive customization and optimization* allows the analysts to incorporate secondary constraints into the model-generation process in an interactive way. There are several benefits to this:

- Models can be fully optimized given a suitable mixture of the criteria.
- Families of models can be trained to deal with variability in the application context.

- Analysts can explore the effects of particular trade-offs instantly, without waiting for a live test.
- Through this exploration, an analyst can gain intuition for the effects of various criteria, and make better trade-offs in the long run.

In order to do so, we use a machine learning architecture which is modular and supports primary and secondary loss functions. We use an weighted additive loss function to represent different criteria and turn multi-objective optimization problem into a parameter tuning problem, (i.e, tuning the weights for each individual loss function)

Even though people can trade-off different criteria by changing the weights of their cost functions, hyper-parameter tuning remains to be a difficult task even for single-objective machine learning problems. Lots of research [5, 8, 23, 55, 62, 89] have been done just for optimizing the training loss. In our case, however, tuning hyper-parameters will change the loss function itself. The value of the loss functions are just performance indicators rather than ultimate goals of the users. Users are supposed to discover the loss function combination that best match their model preferences during the tuning process. Therefore, instead of using those auto tuning algorithms, we allow the users to directly manipulate parameters during training, and observe the feedback in real-time. Direct parameter manipulation such as Tensorflow Playground [86] turns out to be very intuitive for people to tune and understand their models. But that works only for small scale dataset. Our tool instead leverages GPU hardware and mini-batch training strategy to deliver real-time user feedback on real-world dataset for a variety of models. Users can dynamically create highly-interactive visualizations and controls to match various optimization criteria. We will discuss the both system design and the interface design in later chapters.

2.2 Related work

Interactive model refinement

In the context of supervised learning, Fail and Olsen [29] describes partially-supervised learning with a user supplying some (sparse) labelled data to help an ML algorithm label the rest. A number of other works have followed this route, by focusing on manipulation of the training data rather than internals of a particular algorithm. Other work focused on human-assisted feature selection (rather than algorithm training) [81]. The Prospect system [77] automates model selection and tuning allowing users to focus on data manipulation. EnsembleMatrix [94] uses custom visualizations to aid in the design of ensemble classifiers. Amershi et al. [3] provides a detailed summary of the work in this area. Much of these work attempt to improve only the accuracy of a machine learning problem by adding a human in the loop, which is quite different from our work. Perhaps the closest to ours is [49] which integrates a human-assisted optimization strategy with the design of multi-class classifiers. But our framework is not limited to classification problem and we focus on

different families of optimization algorithms. Another interactive hyper-parameter tuning tool called Tensorflow playground [86] has been a good educational tool for people who want to understand how neural network works and how hyper-parameters may affect the model. But it runs on toy dataset while our system can provide real-time feedback when training on real-world datasets.

Interactive clustering

Interactive clustering is another active sub-area. Since clustering is widely-used to simplify the interpretation of large datasets, and since the natural metrics for a new domain may be difficult to articulate, interactive exploration [27,83,105] is a natural and powerful approach. In [27, 83], the authors used visualizations to rapidly explore the results of a clustering algorithm, and these approaches have become important tools in computational biology [27]. AverageExplorer [105] allow users to explore and summarize large collection of images by interactively posing constraints.

Recently, there has been much interest in using visualization to support the refinement of topic models [18, 42, 99]. Since the latent topics extracted by the algorithm are not always semantically meaningful [15,70], different constrained topic models [64,69] have been developed. Systems like [42, 99] also allow users to iteratively refine the model based on their preference. However, those models always require solving a complicated optimization problem with some very specific constraint. In contrast, our framework remains flexible and could adapt to different applications.

Hyper-parameter tuning

Hyper-parameter tuning has been an important yet difficult problem for machine learning. Besides hand tuning, researchers have developed auto tuning algorithms including random search [8], gradient based methods [5, 62], reinforcement learning [23] or bayesian optimization [89]. However, those auto tuning algorithms are compute-intensive and turn out to be impractical for many problems. On the other hand, all those algorithms try to search the parameter space based on the final training loss or validation loss of the current parameter setting as well as some prior information of model structure. This is mimicking human tuning strategy and human may actually do much better if machines can only conduct limited number of experiments. Recently, methods using bandit optimization such as [55] began to explore early stopping strategy which saves a lot of computation. This also provides intuition that interactive parameter tuning could leverage information during live training. Also, all these algorithms above require one fixed quantitative goal, while in our case such goal doesn't exist. Users often need to trade-off different criteria by observing their corresponding performance indicators at the same time.

Hyper-parameters can be very different. There are structural hyper-parameters such as the type of the activation function and number of layers in a neural network. Such hyper-parameter can not be changed after the training is started. Also, there are hyper-parameters

whose behavior are hard to interpret, such as the learning rate, momentum rate. Our toolkit is not focusing on those hyper-parameters. Instead, we tune hyper-parameters which can have interpretable visual feedback. As we are tuning the weights of the loss function which therefore change the weights of their gradient directions, the value of the corresponding loss functions will go up and down. Also, sparseness, correlation, cluster-size distribution are human-interpretable concepts that can be seen directly from the model visualization of Topic modeling or K-Means.

2.3 System design

Secondary criteria as Mixins

Model customization is useful for both supervised and unsupervised problems. Unsupervised learning involves a certain amount of arbitrariness in the criteria for the "best" latent state. Therefore regularization is widely used as a secondary constraint on the primary objective [64, 69, 99].

In a bit more detail, clustering algorithms like K-Means usually use the measure of sample/centroid similarity, and may use inter-cluster distance or cluster size as measures. Indeed, unsupervised learning models are often *evaluated* using a variety of criteria that are much more complex than the criteria used to derive the learning algorithms [15, 70]. The same holds true for topic models such as LDA, NMF and Word2Vec, and for collaborative filtering.

This is a paradox. Clearly one should get better scores for these criteria if they were directly optimized as part of training. Beyond these standard criteria, there are many others that are commonly used in the applications of machine learning. Historically it has probably been too difficult to optimize these criteria (the criteria may be expensive to evaluate, or non-locally computable). Also simply scoring a model is not enough for optimization — one needs to know how to *change* the model to improve it, e.g. through a likelihood derivative.

On the other hand, computing power is abundantly available now, especially in graphics processors. The bottleneck is often *moving* data rather than computing on it. Thus it is often practical to evaluate multiple, relatively complex criteria as part of optimization.

Combining these approaches, we can deal with a variety of secondary or "mixin" criteria as part of the learning process. In our present implementation, we use a linear combination of cost functions for primary and secondary criteria:

$$arg\min_x f_0(x, d) + \sum_i \lambda_i * f_i(x) \tag{2.1}$$

Where x is the model parameters, d is data, f_0 is the primary cost function and f_i are the user-defined *Mixin* functions. The weights λ_i are "controls" that are dynamically adjusted by the analyst as part of training. For the primary criterion f and each secondary criteria f_i there should be at least one dynamic graphic that captures changes in that criterion in

an intuitive way. The analyst watches these as each of the controls are adjusted to monitor the tradeoff between them.

Client-server visualization architecture

With all the pieces above, we are able to use a client-server architecture with 3 components as shown in Fig 4.2: a computing (BIDMach) engine, a web server, and a web based front end. BIDMach is implemented in the Scala language which supports concurrency with high level "actor" primitives. Another thread runs in the same Scala runtime, communicating with the web server. This thread receives parameter updates from the web server, and updates the corresponding model training parameters. As the model is trained, primary and mixin cost functions are evaluated on minibatches, providing regular updates which are passed to the web server.

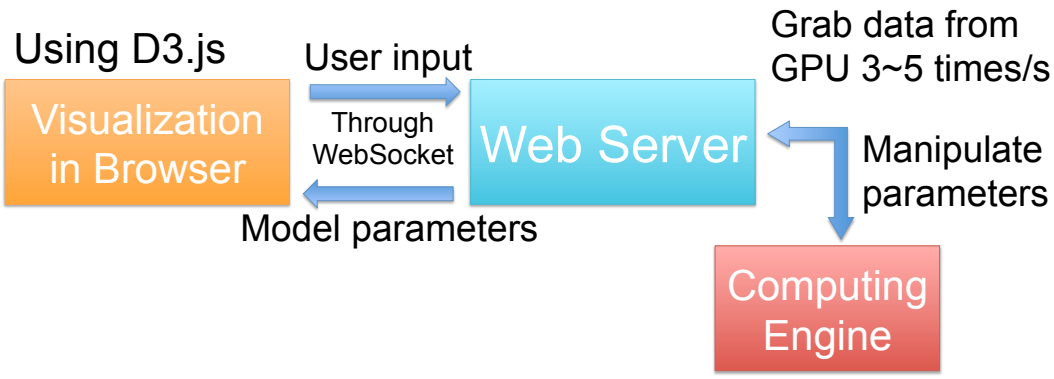


Figure 2.1: Visualization Architecture

In the client side, we implement a web based interface which uses D3.js [12] for data visualization. D3 is widely used, has very powerful graphics elements and good support for animation. As a browser-based system, it runs transparently with a local or remote server. We will discuss the interface design in detail in the next section.

The communication between the client and server is bi-directional, with both client and server initiating transfers. We therefore use WebSockets instead of e.g. a one-way RESTful web service. For simplicity and extensibility, we use JSON as the over-the-wire exchange format.

2.4 Interface Design

In this section, we describe the visual interface of our interactive machine learning system. Our approach relies on a dynamic visual interface that provides streaming feedback to the user.

Visual dashboard

We use a dashboard approach where user can customize their own visualizations. As shown in Fig 2.2, the left side of the interface contains the menus and control sliders. From the menus, a user can select the metrics and controls for the modeling task. A corresponding control or metric visualization is then added to the dashboard, which can then be dragged, dropped and resized. There is at least one corresponding performance indictor for each control parameter, and more than one can be added to the dashboard. The details of each visualization component will be described next.

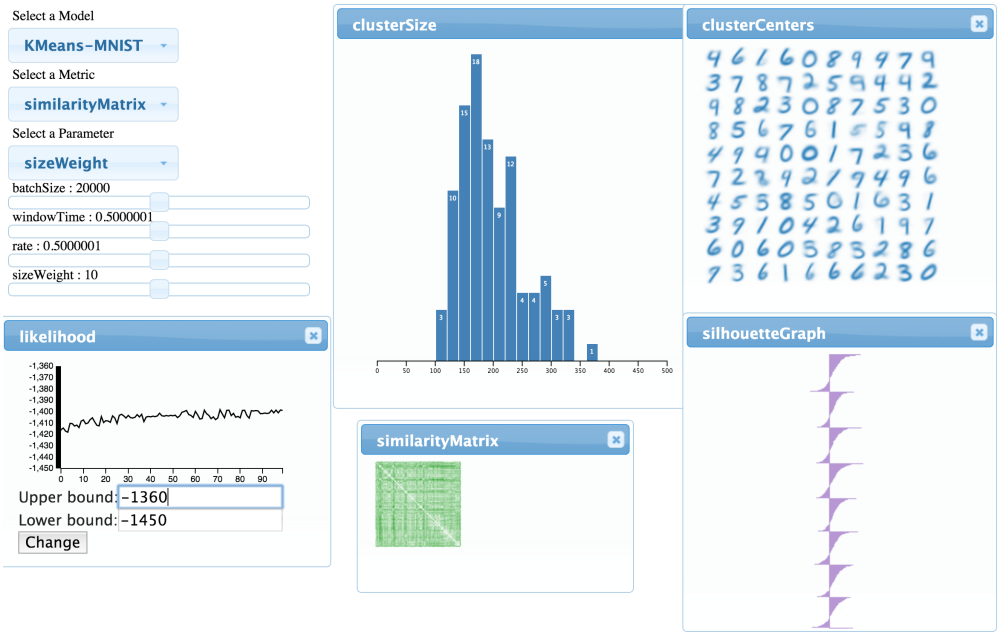


Figure 2.2: Dashboard for KMeans using MNIST dataset

Visualizing the model

As mentioned earlier, machine learning algorithms can be formalized as finding the best model parameters x given an objective function $f(x, d)$. For a variety of models, directly visualizing the model parameters provides a nice summarization for the current training and users can gain a general understanding about the behavior of the algorithms. It can also

help identify obvious errors and verify assumptions or intuitions. While there are different types of data and algorithms, the visualizations are necessarily model-specific, and should provide a natural interpretation of the model directly.

For image data, clusters can be visualized directly. The images we tend to visualize can either be the cluster centers as in Fig 2.3a, or learned image dictionary using dictionary learning algorithms like NMF [53], as in Fig 2.3b.

For more general matrix data, a simple direct visualization of element weights can usually work well. This was the approach taken in the Termite system [18] and we use it for our topic model, which will be described in details in the use cases section. As shown in Fig 2.3c and 2.3d, the area of each pink circle in row i and column j encodes the corresponding value from the matrix. In the topic model case, that encode the weight (likelihood) for word i to be appeared in topic j. We also display the word label on the left side of each row to help people interpret the results.

One common challenge for visualizing topic model comes from the huge size of the model. Typically there are hundreds or thousands of topics and tens of thousands of different words. Limited screen size and limited human perception power require us to filter out information according to some saliency metric [18]. The metric will provide an ordering to show only the most important words and topics. Also, such an approach can significantly reduce the amount of data that need to be transferred from the server to the client.

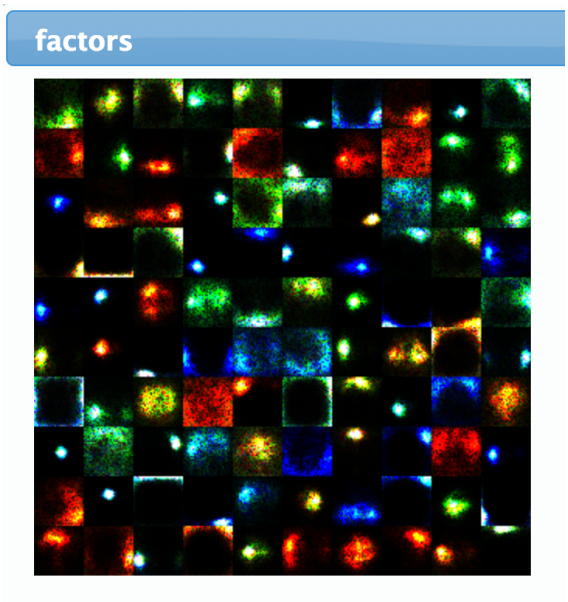
However, as we are displaying in real-time an evolving model, having a simple, consistent metric for ordering can actually help users interpret the visualization and avoid confusion. We therefore only use the total weights across all topics to rank the words. This approach will show more redundant words comparing to the original Termite design. But the main goal here is to analyze the dynamics of the algorithm, rather than the model itself. In order to support a detailed zoom in for each topic, we also support ranking words using weight from a particular topic alone. This feature will be triggered when users mouse over the topic title, as shown in Fig 2.3d. It also helps the users to compare different topics.

Continuous visualization of model quality

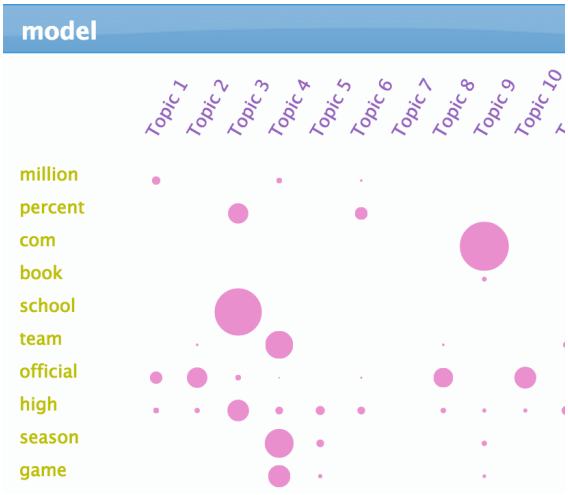
As described in Equation 3.1, we are optimizing an additive function which consists of a main loss term as well as several *Mixin* terms. The value of the cost function can reflect how the model behaves under each criteria. As the engine computes the update, we visualize those metrics as streaming data, as shown in Fig 2.4. Visualizing the main loss function is very important when we change the control parameters. It will reflect how the algorithm responds to the user control and whether the tradeoff for *Mixin* functions may affect the general model performance.

As discussed earlier, the main loss function is computed on each minibatch, and is a single scalar. We use a simple, dynamic curve plot to display its state. This plot style is easy to read and understand. With the parameters fixed, one normally sees a rapid initial increase in likelihood, followed by a slow increase on a plateau as in Fig 2.4.

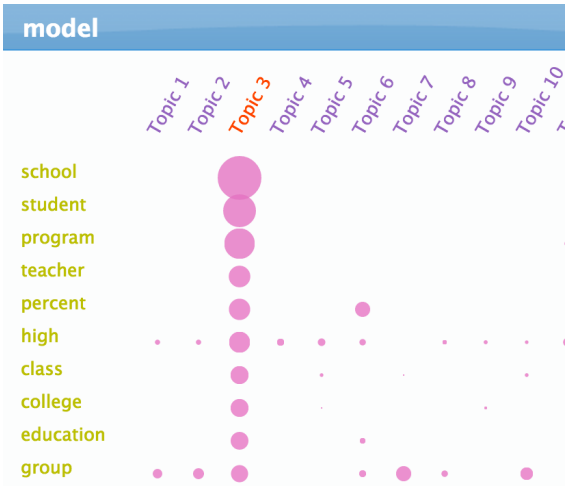
(a) MNIST clustering centers



(b) Dictionary learned by NMF on CIFAR dataset



(c) Ranking words by sum of weights



(d) Ranking words using weight from topic 3

Figure 2.3: Model visualization

Visualizing other performance indictors for clustering

As mentioned above, the evaluation of unsupervised clustering algorithm is hard in general. Therefore showing multiple aspects of the clustering results at the same time would help users better interpret the model.

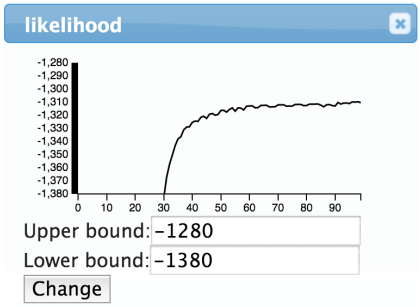


Figure 2.4: Continuous visualization of the likelihood function

Cluster size distribution

To examine the cluster size balance, we are interested in the *distribution* of cluster sizes. The natural visualization for this kind of data is a histogram or kernel density plot. Fig 2.5a shows the size distribution for the clusters of digits on the MNIST dataset.

Silhouette graph

Another useful metric is the widely used silhouette graph (Fig 2.5b) for evaluating clustering results. The silhouette score is calculated for each data point x_j as:

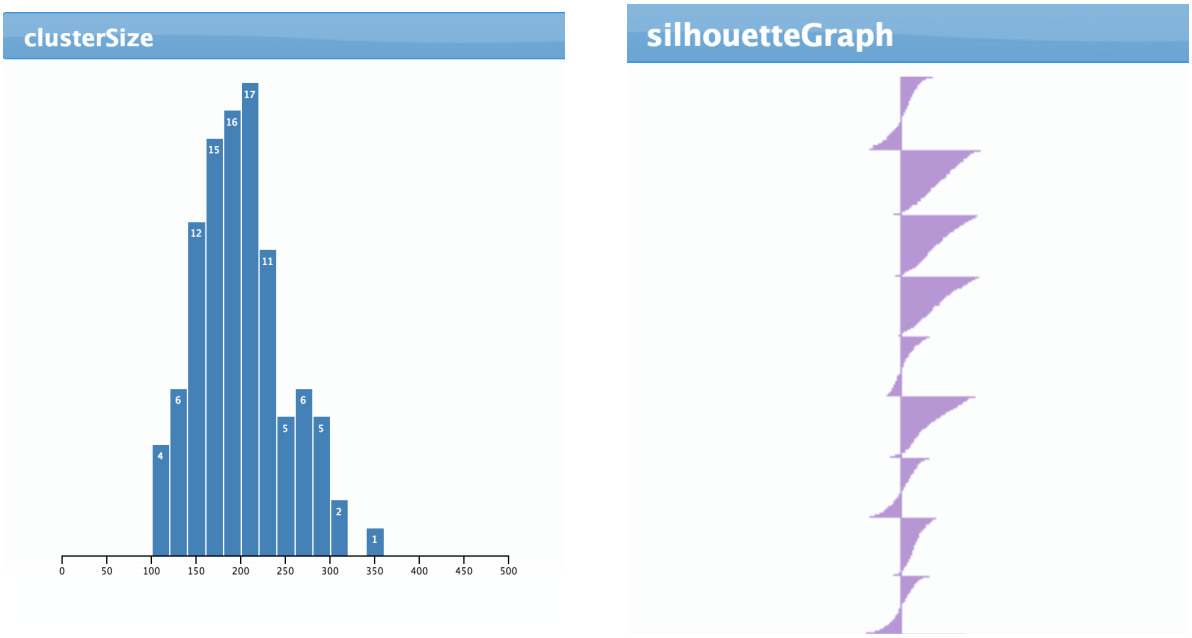
$$s_j = \frac{b_j - a_j}{max(a_j, b_j)} \tag{2.2}$$

Where a_j is the average distance between x_j and all other data points in the same cluster, b_j is the lowest average distance of x_j to any other cluster.

It could be seen that $-1 \leq s_j \leq 1$ and larger values indicate better clustering results. Silhouette graph then shows the silhouette score for each data point as horizontal line. Data points are grouped by cluster and are sorted by their silhouette score vertically. The silhouette graph can intuitively show how tight it is for each cluster by showing the silhouette score distribution within that cluster. A low silhouette score typically indicates small clusters are at the periphery of larger ones.

Slider controls

Along with the visual interface, we also provide several kinds of control including weights for *Mixin*, learning rate, temperature etc. So far these are all continuous scalars, and are all implemented as slider widgets. When the user selects one of the controls from the menu, a labeled slider widget is created on the dashboard.



(a) Distribution of Cluster Size

(b) Silhouette graph

Figure 2.5: Performance indictors for clustering

2.5 Use cases

In this section, we will demonstrate how to use our system for interactive model customization using KMeans and topic model algorithms on some real-world datasets.

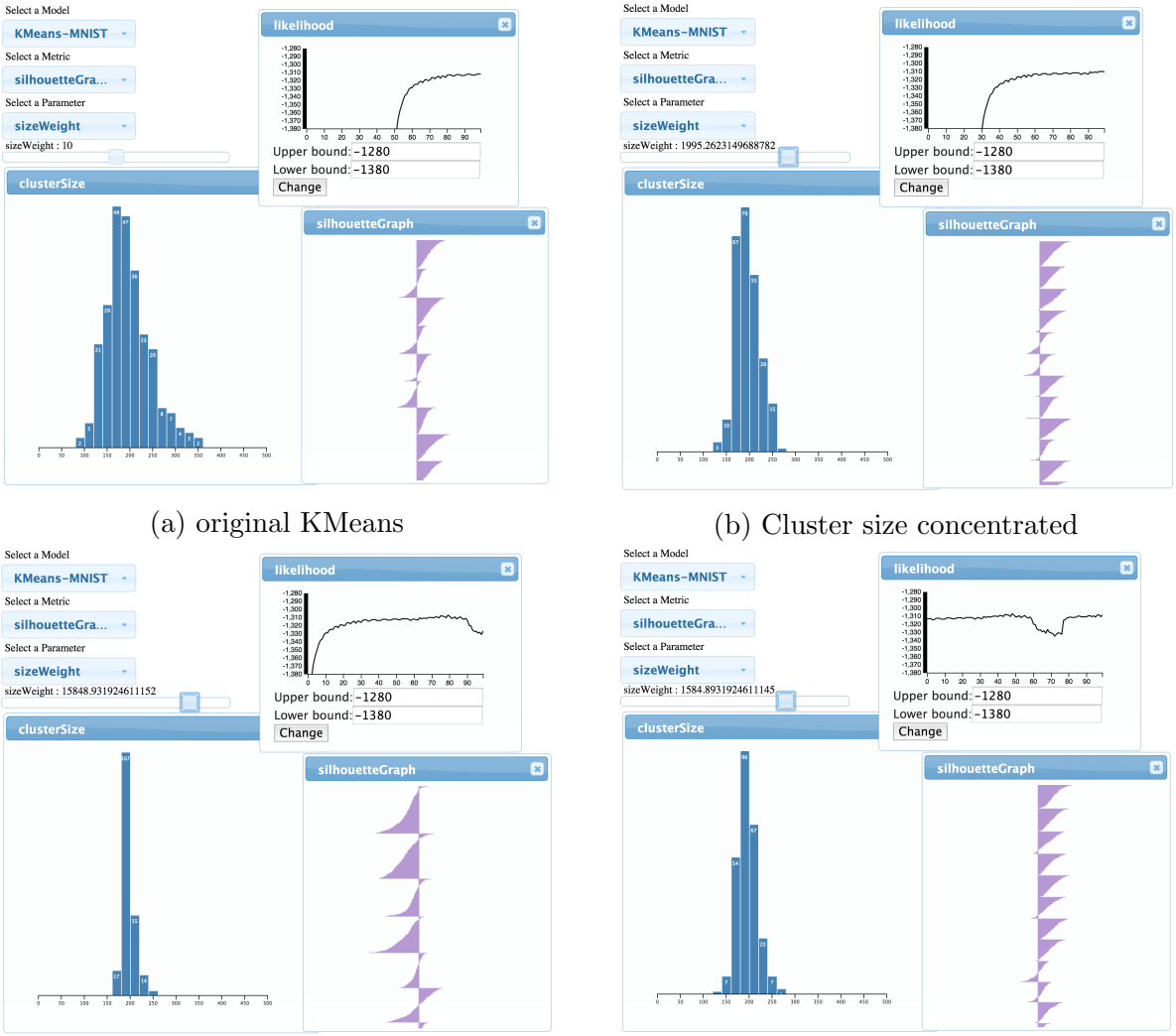
KMeans

Our first application is the KMeans algorithm, which is a widely used clustering algorithm. Though the algorithm is simple, it can find very useful feature representations if used properly [21]. Also, it is well known that KMeans is very sensitive to initialization and distance metrics. It can easily go into local optimum and hard to escape during the training. In this section, we will discuss how to address those problems using our system.

Implementation and evaluation for KMeans

For KMeans, the primary loss function is inertia: the sum of squared distances from points to their centroid. The KMeans algorithm starts off by randomly initializing k cluster centers. Then, during each iteration, it assigns data points to their closest cluster, and for each cluster, re-computes its center using the average of the elements in that cluster.

However, as mentioned previously, the evaluation and tuning of the KMeans algorithm turn out to be hard. We therefore use multiple criteria to examine different aspects of the



(a) original KMeans

(b) Cluster size concentrated

(c) Begin to loss performance

(d) Recover the model

Figure 2.6: Interactive tuning for KMeans

clustering results. Aside from the main loss, we use silhouette graph to measure how tight it is for each cluster. A low silhouette score typically indicates small clusters at the periphery of larger ones.

Besides, we also use cluster size balance as a criteria, which adds penalty to clusters that have bigger size. The optimization problem is then to partition the dataset X into k disjoint sets s by minimizing the following loss:

$$\arg\min_{s} \sum_{i} (\sum_{j \in s_i} ||x_j - \mu_i||_2^2 + \lambda |s_i|^2) \tag{2.3}$$

Where $|s_i|$ represents size of the i-th cluster, and μ_i represents the center of the i-th

cluster.

The size balance criteria could be naturally visualized with a histogram. This criteria can also be approximately optimized within the KMeans algorithm. The overall minimization algorithm then becomes:

$$id(x) = \arg\min_i ||\mu_i - x||_2^2 + \lambda |s_i| \tag{2.4}$$

Where $id(x)$ is the assigned cluster id for data point x.

The loss term $\lambda * |s_i|$ penalizes clusters with a larger size. The algorithm would prefer to assign new data points to a smaller cluster, which will tend to balance cluster sizes over time. Size homogeneity matches most users' intuition about clustering, and it may also be important for accurate estimation of cluster statistics. The λ also becomes a parameter that we can interactively tune.

Incremental update

In order to take the advantage of mini-batch processing which can return early feedback during the training, we follow a similar approach to [103] for incremental KMeans updating. And we also need an averaging update for $size_i$ to maintain its scale and make the visualization consistent. For each batch $\{x_j\}$, we compute the update as:

$$\begin{aligned} average_i &= \frac{\sum_j x_j * \mathbb{1}(id(x_j) = i)}{\sum_j \mathbb{1}(id(x_j) = i)} \\ \mu_i &= (1 - \eta) * \mu_i + \eta * average_i \\ size_i &= (1 - \alpha) * size_i + \alpha * \sum_j \mathbb{1}(id(x_j) = i) \end{aligned} \tag{2.5}$$

Normally η and α are set to $0.1 \sim 0.2$.

Experiment on MNIST dataset

We ran an experiment on the MNIST dataset [52], which contains 8 million $28 \times 28 \times 3$ RGB images of hand written digits. We train the model using NVIDIA Titan X GPU, which could process roughly 500MB raw data (16.7k images) per second. As we set the mini-batch size to be 50000, every second the system could perform 3 batch updates, which is enough for real-time visualization.

In our dashboard shown in Fig 2.2, we choose to visualize the main loss, cluster size distribution, as well as the silhouette graph. The main loss is the averaging distance between the data points and their assigned cluster centers.

The parameter that we choose to tune is the weight for size balance λ in eq(2.4), which we refer as $sizeWeight$ in the interface. Initially, we set the number of clusters K as 256 and we assigned a small value to $sizeWeight$, so that the algorithm will behave as the original KMeans algorithm. As shown in Fig 2.6a, the likelihood is gradually improving and it quickly converges to local minimum. The cluster size distribution is quite diverse.

We then slightly increase the $sizeWeight$, which gives us a more concentrated cluster size distribution, while the main loss and silhouette score are not affected, as shown in Fig 2.6b. This implies that the algorithm now moves to another local optimal by assigning some data points to a suboptimal but smaller cluster. Notice that this has almost no effect on the primary likelihood.

However, as we continue to increase $sizeWeight$, the loss will start to increase, and the silhouette graph also shows more defects (more negative area) (Fig 2.6c). From here, we decrease the $sizeWeight$. Since the KMeans algorithm is incremental, this change brings us back to the loss that it has before, as shown in Fig 2.6d. This example illustrates the tradeoffs that can be made, and the speed of recognizing poor parameter choices.

Experiment on ImageNet dataset

ImageNet [24] is a widely used image-classification dataset which contains millions of labeled images. The recent success of applying convolutional neural network [51] not only achieves human-level performance in image-classification task, but also provides powerful feature embedding methods that could be used for semantic image clustering [25] and neural style transfer [32] which decomposes each image into different semantic features. For examples, image clustering can have many different criteria such as clustering by shape of the objects, by scene of the background, or by semantic meaning of the objects. Those criteria can be expressed as different distance metrics in KMeans algorithm, and can be merged into one metric using tunable weights. Therefore, users can tune those weights via our interface to try different combinations of the distance metric in real-time.

To conduct the experiment, we use the caffe [46] toolkit to generate feature embedding by feeding each image into a pre-trained neural network for a forward pass. The activation output from each network layer then becomes the feature for that image. We use the reference caffenet model (AlexNet), and to reduce memory footprint, we only use output from pool1, pool2, pool5 layer. The pool1 and pool2 layers capture low-level visual features while the pool5 layer has higher-level features which are invariant to scale and location. One $227 \times 227 \times 3$ RGB image will then generate a $27 \times 27 \times 96$ pool1 feature, a $13 \times 13 \times 256$ pool2 feature and a $6 \times 6 \times 256$ pool5 feature. We concatenate these 3 features into one 122464 dimensional vector. The distance function we used is straightforward: just apply the L2 distance on each of these 3 features and then compute the weighted sum:

$$\begin{aligned} d(x,y) = s_1 * \alpha_1 * ||x_{pool1} - y_{pool1}||^2 \\ +s_2 * \alpha_2 * ||x_{pool2} - y_{pool2}||^2 \\ +s_3 * \alpha_3 * ||x_{pool5} - y_{pool5}||^2 \end{aligned} \tag{2.6}$$

Where $\alpha_1, \alpha_2, \alpha_3$ are normalization constants to make those 3 metrics have the same scale. During training, users can tune s_1, s_2, s_3 via the interface to change the distance metric.

We use 8 different classes of images from the ImageNet dataset, which contains about 13000 images. Since each image contains a huge dense feature vector, our system can process

about 1200 images per second when K is 128. We therefore set mini-batch size to be 512. The interface of the system is shown in Fig 2.7. We sort the clusters by their size which are shown in the titles. For each cluster, we show 20 randomly selected images to represent that cluster.

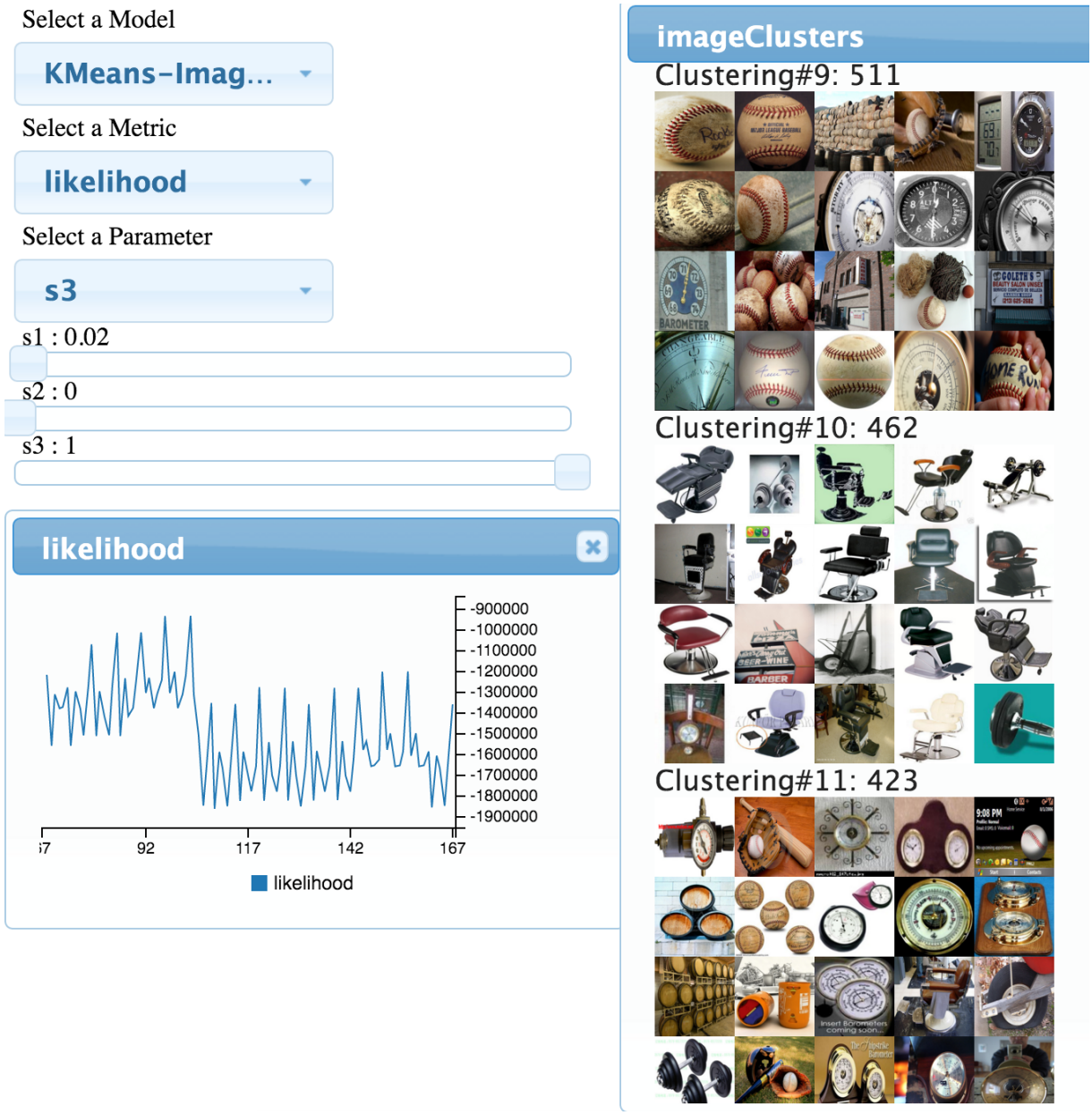


Figure 2.7: Clustering on ImageNet data

By tuning the weights for the distance metric, we can get different clustering results as shown in Fig 2.8 and Fig 2.9. Clusters generated by pool5 feature usually have consistent

semantic meaning, that most images come from the same category. While clusters generated by pool1 feature usually contain images with similar object shape or color.

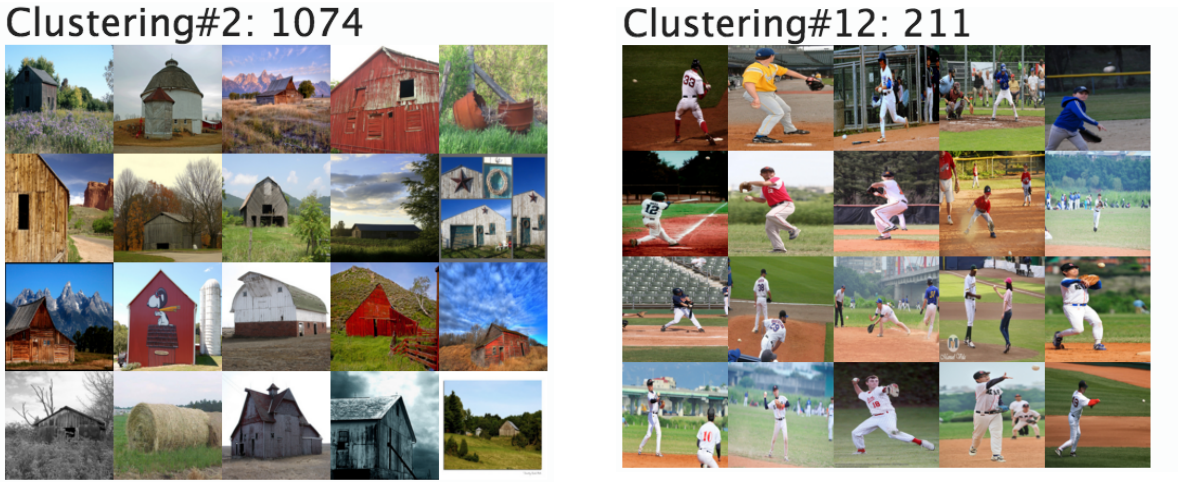


Figure 2.8: Clustering using pool5

Figure 2.9: Clustering using pool1

Besides using only one kind of feature, we can use a mixture of them. We can even smoothly transit from one distance metric to the other during the training. Such transit is equivalent to using the previous clustering results as the initial cluster centers. Comparing to the random initialization, this gives us more flexibility and can yield more interesting results, as shown in Fig 2.10. On the other hand, changing the distance metric during training can help the algorithm escape from local optimal.

(a) Images that have something in red

(b) Images with a lot of barrels

Figure 2.10: Transit from pool1 to pool5

Topic Model

We then apply our system to topic modeling using Latent dirichlet allocation(LDA) [11], one of the most widely used topic model algorithm. Topic model has many applications since it can find compact representation of a large document set. However, as discussed in [15], results generated by maximizing the likelihood objective may actually infer less semantically meaningful topics. This raises the question of how to do model selection. On the other hand, automatic evaluation of topic quality [70] is complicated and users may have different criteria at the same time. Our system allows users to make judgement directly from the model results, and to fine-tune the model by adjusting weights of the mixin functions.

Implementation

LDA is a generative process to model the documents. For each document d, it proceeds as follows (K is the number of latent topics):

- Draw a topic distribution for the document d as $\theta_d \sim Dirichlet(\alpha)$, a K-dimensional Dirichlet prior.
- For each word position i (across all docs), draw a topic index $z_{d,i} \in \{1, ..., K\}$ from $z_{d,i} \sim \theta_d$
- Draw the word $w_{d,i}$ from the multinomial distribution $w_{d,i} \sim \varphi_{z_{d,i}}$, which also has a prior: $\varphi_{z_{d,i}} \sim Dirichlet(\beta)$

Where α, and β are hyper-parameters specifying the Dirichlet prior. The other two parameters of the model are θ, which can be represented as a document-topic matrix, and

φ, the word-topic matrix. The algorithm therefore is to use Gibbs sampler to draw samples for hidden states z_i:

$$P_X(z_{d,i} = k|z_{-d,i}, \alpha, \beta, x_{d,i} = w) \sim \theta_{d,k} * \varphi_{k,w} \tag{2.7}$$

After drawing the samples, model updates for θ' and φ' can be computed via Maximum Likelihood Estimation. In addition, the strength of the model is measured by the likelihood. In order to apply annealing to the optimization procedure, we use SAME sampling [102] to draw m independent samples instead of just one from Z each time. This results in a cooled Gibbs sampler and the parameter m can be used to control the temperature. A low value of m gives a higher-variance random-walk while increasing m can cause parameters converge to a nearby optimum. By default m is set to 100, and it can be tuned during the training. We refer to m as *nsamps* in the interface.

Incremental update for LDA as described in [39] is used. Model will be updated after each mini-batch data is processed.

Mixins

Mixins are functions that capture users' intention for model customization. Here we use two kinds of mixins: A L1-norm function to approximately measure sparseness:

$$f_1(\varphi) = |\varphi| \tag{2.8}$$

and a pairwise cosine-similarity function to measure the similarity of the topics:

$$f_2(\varphi) = \sum_{i \neq j} (\sum_w \varphi_{i,w} \varphi_{j,w}) \tag{2.9}$$

As describe in the system design section, the implementation is very straightforward. For each mini-batch, after computing the model update φ', we also compute the sub-gradient update for the mixins:

$$\begin{aligned} g_1(\varphi_{k,w}) &= sign(\varphi_{k,w}) \\ g_2(\varphi_{k,w}) &= (\sum_i \varphi_{i,w}) - \varphi_{k,w} \end{aligned} \tag{2.10}$$

We then add them into the model using the weighted averaging approach we discussed above:

$$\varphi_{t+1} = (1 - \alpha)\varphi_t + \alpha\varphi' - \lambda_1 g_1(\varphi_t) - \lambda_2 g_2(\varphi_t) \tag{2.11}$$

We refer the weight λ_1 as $L1 - reg$ since it is equivalent to the common L1-regularization technique, and λ_2 as $CosineSim$ in the interface.

Experiment on NYTimes dataset

We run an experiment on the NYTimes dataset [56], which contains about 300K documents, 102K different words and a total of 100 million word tokens. The size of the sparse matrix format data file is about 522MB. We train our model using Titan X GPU.

To demonstrate the usage of our system, we set the topic number K to be 32, since smaller topic number makes topics more likely to overlap with each other. We also set the initial weights for both mixins as 0. Our system can process about 15000-20000 documents per second. We therefore set mini-batch size to be 5000 in order to receive 3-4 update per second. The algorithm's behavior is shown in Fig 2.11. The likelihood quickly converges to a local optimal, but the topic results are still very noisy, and the topics are overlapping. We then adjust the $L1 - reg$ slider away from zero. However, tuning $L1 - reg$ alone may not always yield changes because the model may be trapped in a local optima. Since SAME sampling is used in our LDA implementation, we can decrease *nsamps* to increase the variance of the random-walk, which makes it easier to jump between possible solutions.

After we increase the temperature, as shown in Fig 2.12, the likelihood drops significantly but we get a very sparse model. Afterward, we set the $L1 - reg$ back to a small value and use a large *nsamps* which prevents large changes in model state. This is equivalent to only allowing the model to make very small movement around that local optimal. From Fig 2.13, we can see that the likelihood returns to a normal value while the sparsity is maintained.

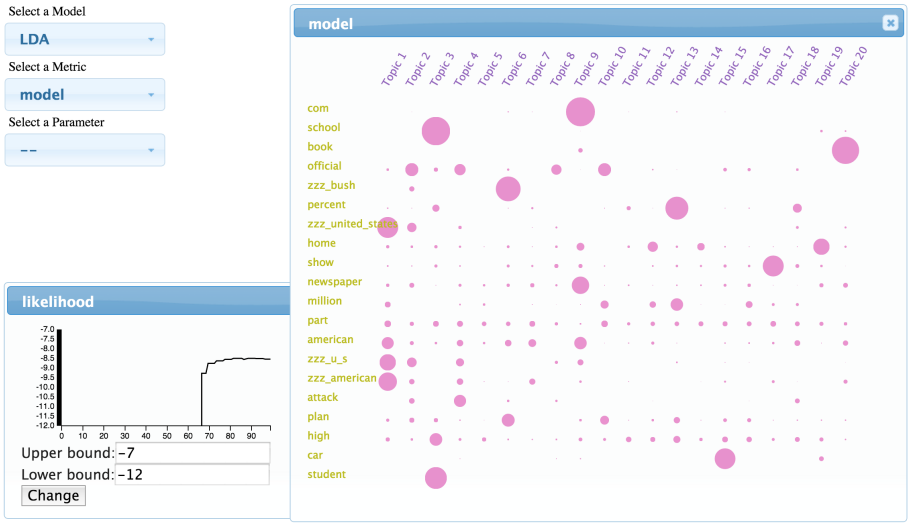


Figure 2.11: Overlapping topics, K=32

Such hyper-parameters scheduling is hard to obtain without interaction. The model likelihood and sparseness seems to be conflicting goals, but we are able to optimize both when using a particular scheduling.

With the same topic number K, after the algorithm converges to a local minimum, we instead increase parameter *cosineSim* to a reasonable scale. As we can see in Fig 2.14, the

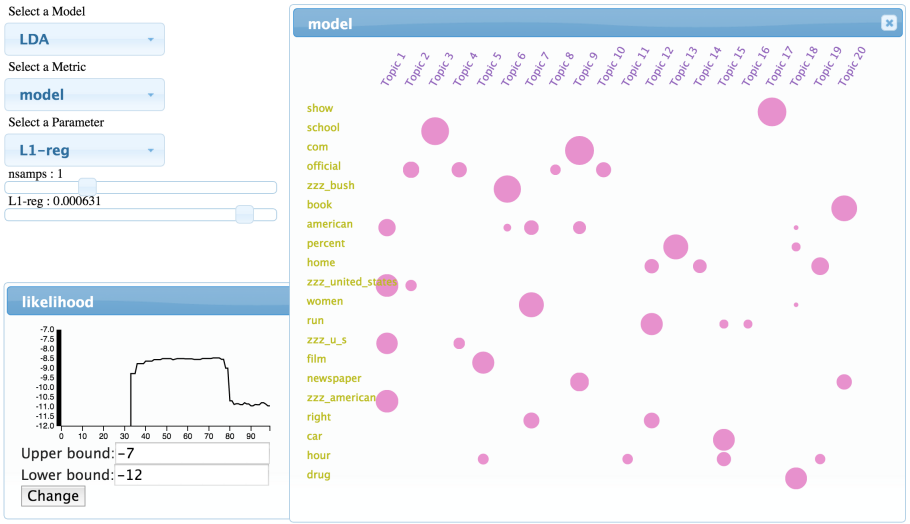


Figure 2.12: High temperature with high L1-reg

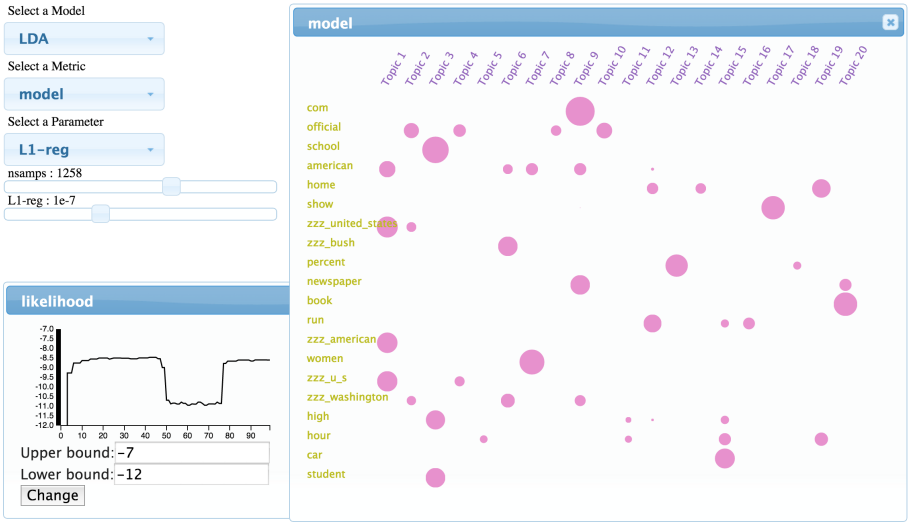


Figure 2.13: Likelihood back to normal, sparsity preserved

cosineSim mixin function value starts to decrease and converges to 0 at last. While at the same time, the likelihood function maintains at the original level. Besides the original goal to reduce pair-wise topics similarity, we can observe that the model becomes sparse and the noise is removed as well.

This implies that two mixin functions can both improve sparseness, and that they have very different effects on the model likelihood. If we compare the two mixin gradient functions carefully, the $L1 - reg$ update is actually performing a hard thresholding that each entry in the matrix will be subtracted by a constant value. On the other hand, the *cosineSim* gradient subtracts a value that is relative to the total weight in other topics: $(\sum_i \varphi_{i,w}) - \varphi_{k,w}$.

Figure 2.14: Cosine similarity decreased

Therefore, $L1 - reg$ will force many long-tail words to have 0 weight in any topics which results in worse model likelihood. But for the *cosineSim* gradient update, at least the dominant weight for each word will be kept. And the noisy weights, which are relatively smaller than the dominant ones will receive a higher penalty, and quickly approach 0.

This example demonstrates that even a simple mixin function can have multiple effects on the model quality. And it would be hard to predict the final outcome when multiple optimization objectives are combined together. In our system, this problem is expressed as a hyper-parameter tuning task and we allow the users to do model selection based on the full spectrum of information.

2.6 Conclusion

In this chapter, we have demonstrated how to perform interactive optimization on customized models using our system. Using mini-batch based online algorithms with the GPU accelerated toolkit, we are able to get real-time feedback which makes the interaction possible. The use of *Mixin* function or secondary loss function provides a convenient and useful way to capture user's intuition and create customized model. To solve those multi-objective optimization problems, we allow users to fine-tune hyper-parameters based on the feedback from visual dashboard during training time. In the examples of KMeans and topic model algorithm, we show that interactive optimization has several benefits including trading off multiple criteria at the same time, setting adaptive temperature schedule, helping algorithm to escape from local optimal etc. In the next chapter, we will describe a case study which allow users to fine-tune topic modeling and KMeans algorithms on several real-world dataset.

Chapter 3

A case study of interactive machine learning

3.1 Introduction

In the previous chapter, we describe the implementation of our system which allows users to perform interactive customized optimization. As we mentioned earlier, more and more people from different background start to use machine learning algorithms to process and analyze their data. With the help of open source toolkits like Scikit-learn [78], Tensorflow [2], users can reuse algorithms and models for their own data with least amount of work. Though using those toolkits saved a lot of time for coding, people now spend more and more time on tuning and improving their models by running empirical experiments again and again. The tuning process typically involves exploring the parameters space and understanding the outcome of algorithms along the way. Similar to explorative data analysis, this should be a iterative process that users keep trying and gaining insights, in order to make a better decision. However, traditional parameter tuning workflow usually has huge latency between user action and system feedback, since user can only see results after the model is fully trained.

We therefore developed the toolkit that supports interactive visual analysis for parameter tuning. As it could provide real-time visual feedback during training and to allow users to interactively manipulate model parameters, we are wondering, is our toolkit really useful for the users? In this chapter, we will describe a user study we conducted to analyze how interactive parameter tuning could help users improve the model and gain insights.

While automatic parameter tuning is a hot research topic in machine learning community, we are particular interested in one kind of parameter that usually requires human domain knowledge. That is the parameter to control the trade-offs between multiple objectives. This may sound a bit uncommon since most machine learning algorithms optimize a single narrow criterion. For example, supervised learning algorithms usually optimize training accuracy/loss, unsupervised learning algorithms optimize model likelihood, reinforcement

learning algorithms optimize expected rewards. However, real-world applications usually involves multiple evaluation criteria and sometimes we need to make trade-offs between those competing goals. For example, in computational advertising, while maximizing click through rate and revenue is the primary goal, user satisfaction and advertiser satisfaction are also important. For classification methods, users need to trade off between training accuracy and validation accuracy to obtain a model that has nice generalization property. While machines can automatically optimize single criterion quite well, optimizing and trading off multiple objective remains challenging and requires human intelligence. Besides, we hypothesize that this kind of parameter is relatively easier for human to tune, since they can observe the changes of the values in evaluation functions to manage their progress. By monitoring the effects of their control, users should be able to learn to anticipate the outcome of the parameters, and to make better trade-offs.

To test our hypothesis, we recruited 28 subjects to conduct the user study. The visual interface we used is shown in Figure 4.3. It contains control sliders as well as streaming graphs that showing the real-time changes of the measurements. Subjects were asked to adjust the parameters to improve the quality of model. We used topic modeling and KMeans clustering algorithms for the study.

In this paper, we will describe the setup of the study in details and summarize both quantitative and qualitative findings. The major findings include:

- Most subjects can quickly learn to use the toolkit to tune parameters and gain insights about the model
- Streaming graphs showing the changes of numerical measurements are the most effective for tuning parameters, while other visualizations could provide more context and information to help subjects understand the meaning of those numbers
- The existence of latency, the non-linear the non-reversible behavior of machine learning algorithms can lead to confusion and misunderstanding
- Multiple objective optimization is hard for human as well. Subjects need to have either a predefined goal, or a deep understanding about each objective in order to make trade offs
- Displaying more than 3 graphs would be difficult for subjects to track at the beginning, but as their familiarity increases, less effort is required

3.2 Related work

Interactive machine learning

Interactive machine learning has been an active research direction in recent years. Its goal is to plugin in different kinds of user interaction to improve either the performance or usability

of machine learning algorithms. Amershi et al. [3] provides a detailed summary of the work in this area. Since the workflow of using machine learning algorithms typically involves multiple steps such as data labeling, data modeling, parameter tuning and model selection etc. researchers therefore have applied interaction techniques in different ways to solve different problems.

In the context of supervised learning, label-and-learn [4, 92] is a well-studied and useful technique that human can interactively label the dataset while seeing how accuracy is being improved as more labels are provided. EnsembleMatrix [94] allows users to interactively ensemble multiple classifiers into one while using confusion matrices to provide visual feedback about the performance.

For unsupervised learning, interactive clustering is an active sub-area. AverageExplorer [105] allows users to explore and summarize large collection of images by interactively posing constraints. iVisClustering [54] is a interactive document clustering system which allow users to iteratively refine the model, by looking at different views of the current model. Scatter/Gather Clustering [41] is another system which could incorporate user-supplied constraints into the optimization process. However, it is typically hard to evaluate the results of clustering algorithms, which makes it hard for users to know whether or not they're making progress.

On the other hand, interactive manipulation requires fast system response. Otherwise, it has been found in [59] that latency can reduce the rate at which users perform explorative analysis. However, as it is often takes hours or days to train machine learning algorithms on large scale dataset, many existing interactive machine learning systems either use pretrained model, or can only be used on small scale dataset. Recently years, by using mini-batch algorithms and GPU acceleration, systems like Tensorflow [2], BIDMach [14] can greatly speed up training process and can even provide real-time feedback during training. Therefore, interactively manipulate training parameters has became possible and has been implemented as interactive machine learning toolkit [47].

Nonetheless, to our best knowledge, no research has been done to analyze how interactive parameter manipulation would benefit the users. One relevant research may be the Tensorflow playground [85], which allows users to interactively manipulate the parameters of a small scale neural network and gain insights by comparing different configurations. However, it only works for toy dataset and it is not clear what kind of insights users would really gain.

Parameter tuning for machine learning

Hyper-parameter tuning has been an important yet difficult problem for machine learning. Besides hand tuning, researchers have developed auto tuning algorithms including random search [8], gradient based methods [5, 62], reinforcement learning [23] or bayesian optimization [89]. However, those auto tuning algorithms are compute-intensive and turn out to be impractical for many problems. On the other hand, all those algorithms try to search the parameter space by using a single objective such as training loss or validation loss, and they

usually need to fully train the model. Recently, methods using bandit optimization such as [55] began to explore early stopping strategy which saves a lot of computation. This also provides intuition that interactive parameter tuning could leverage information during live training.

On the other hand, to our best knowledge, no research has been done to analyze how human would tune parameters for machine learning algorithms. Researchers in machine learning community have summarized practical ways for parameter tuning [7]. But it remains unclear how novice can gain those knowledge by exploring the parameter space.

3.3 Experimental method

The goal of this study is to understand how users would use interactive parameter tuning for machine learning, and whether or not the visual interface can help subjects finish the tasks and gain insights. However, the intrinsic property of the machine learning algorithms, the design of the visual interface, the background knowledge of the subjects, the communication during the study will all have effects on the results. Therefore, we try out best to design the study to confront those issues. In this section, we will describe the details about how the study is designed and conducted.

Experimental System

We use the open source machine learning toolkit called BIDMach [14, 47] to conduct the experiments. BIDMach [14] has demonstrated extremely high performance with GPU acceleration. Instead of using batch-update methods where each update is performed after scanning the whole dataset, BIDMach uses mini-batch updates that only a small fraction of data will be used for each update. Models are being updated continuously, usually many per second. Therefore the effects of subjects actions will be seen quickly. BIDMach has reduced the running time of many non-trivial ML tasks from hours to minutes. And even for models that take minutes to train fully, the effects of parameter changes are typically visible in seconds due to the mini-batch online learning algorithms like stochastic gradient descent.

Using BIDMach, it is also flexible and easy to incorporate secondary criteria into the optimization process. This is implemented by using a linear combination of cost functions for primary and secondary criteria:

$$arg\min_x f_0(x, d) + \sum_i \lambda_i * f_i(x) \tag{3.1}$$

Where x is the model parameter, d is data, f_0 is the primary cost function and f_i are the user-defined secondary cost functions. This way to construct the cost function is similar to what computational graph can achieve in other machine learning systems like Tensorflow [2]

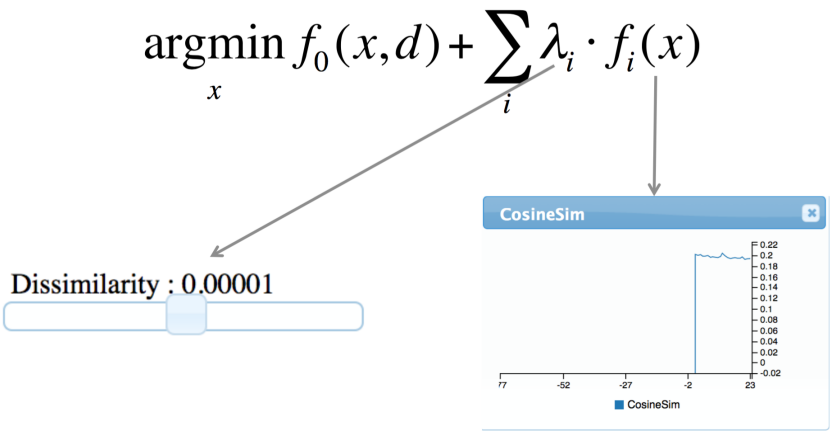


Figure 3.1: Each control is associated with a measurement

Figure 3.2: Visualization Dashboard for topic modeling

and Theano [9]. Subjects can flexibly express their preference on the model by adjusting weight λ_i of each cost function.

To perform interactive parameter tuning, we use a similar but modified interface from [47], as shown in Figure 4.3 and Figure 3.3. For the primary criterion f_0 and each secondary criteria f_i, there is a slider bar which is used for adjusting the corresponding weight λ_i, as well as one streaming graph showing changes in that criterion. Their relationship between them is shown in Figure 3.1. The subjects can first adjust the slider bar and then watch the changes in its corresponding graph.

Datasets and algorithms

BIDMach [13] supports a variety of models such as logistic regression, K-Means clustering, Topic modeling (Latent Dirichlet Allocation [11]), and Deep neural nets. We choose topic modeling and KMeans clustering for the study since it has been widely used by people from different domains. We will describe them in details in this section.

Topic modeling

Topic modeling is a widely used algorithm to find out topics within a large document set. It has many useful applications in science [34] and industry. The input of the algorithm is a document-word matrix representing the word distribution in each document, and the algorithm is to estimate the model parameters of the data generative process. The most useful model parameters is the word-topic matrix $\varphi_{w,i}$ describing the distribution of each individual word w within topic i. By visualizing this word-topic matrix using a visualization similar to the Termite system [18], subjects without machine learning background can also intuitively understand the outcome of the algorithm. Subjects can also hover the topic title to inspect individual topic, as shown in Figure 3.5.

Another reason to choose Latent Dirichlet Allocation is because we use Collapsed Gibbs Sampling [80], a Markov Chain Monte Carlo algorithm to maximize the likelihood of the words within each document. Comparing to other gradient descent based algorithms such as neural nets, Gibbs sampling algorithms can easily escape local minimum smoothly by increasing the variance of the sampling process [102]. This is a very important and useful feature since analyst can frequently get stunk into some local minimums. For algorithms like neural nets or KMeans which we will describe later, restarting the training process is usually required. Topic modeling is therefore easier for us to conduct the user study.

On the other hand, though Gibbs sampling algorithm is optimizing the likelihood of observed data, it has been found in [15] that models with higher likelihood may actually infer less semantically meaningful topics. Thus likelihood is not the best or at least not the only way to measure model quality. Many research have been done to develop new ways to evaluate topic modeling results [70, 95], and to improve the model quality by incorporating regularization [69] or domain knowledge [6] into the model.

To perform the study, we choose two relatively simple evaluation criteria and their controls:

- *L1-reg* is controlling the density of the weights in the model
- *Dissimilarity* is controlling the pair-wise cosine similarity between topics

To be specific, we add the following two cost functions into the model:
A L1-regularization function:

$$f_1(\varphi) = \sum_{w,i} |\varphi_{w,i}| \tag{3.2}$$

and a pairwise cosine-similarity function to measure the similarity of the topics:

$$f_2(\varphi) = \sum_{i \neq j} (\sum_w \varphi_{w,i} \varphi_{w,j}) \tag{3.3}$$

Where φ is the word-topic matrix. $\varphi_{w,i}$ represents the probability of word w in topic i. $\sum_w \varphi_{w,i} = 1$.

During the optimization, the gradient of these two cost functions will be applied to the model matrix φ. Please note that there is a normalization step in LDA algorithm so that the probability of all words sum up to 1. Therefore the value of f_1 remains unchanged after the normalization, but its gradient will affect model updates.

We instead use g_1 to measure the density:

$$g_1(\varphi) = \frac{\sum_{w,i} \mathbb{1}_{x \geq 0.01}(\varphi_{w,i})}{\sum_{w,i} \mathbb{1}} \tag{3.4}$$

and use g_2 as a normalized measurement for similarity:

$$g_2(\varphi) = \frac{f_2(\varphi)}{\sum_{i \neq j} \sum_w \mathbb{1}} \tag{3.5}$$

Besides the model likelihood, the value of g_1 and g_2 are visualized via the streaming graph, named *Density* and *CosineSim*, as shown in Figure 4.3. λ_1 and λ_2 for the two cost functions can also be adjusted via the slider bar, named *L1-reg* and *Dissimilarity*. It is coherent that when λ_1/ λ_2 is increased, g_1/g_2 should decrease correspondingly. We also provide a third parameter called *temperature*, which controls the variance of the Gibbs sampling algorithms as described in [102]. When temperature is high, the algorithms will become more unstable and can be easier to escape from those local minimums. While temperature is low, the algorithm will be more likely to converge.

We use the UCI NYTimes dataset [56] which contains around 300K documents, 102K different words and a total of 100 million word tokens. The online LDA algorithm [39] is used to perform mini-batch updates. Using a Titan X GPU, training for one full pass takes around 20s.

KMeans clustering

The other algorithm we used in the study is KMeans, which is the most widely used clustering algorithm due to its simplicity. However, KMeans algorithm can easily fall into local optimum and its performance is very sensitive to initialization. The evaluation criterion we added into the model is the variance of the size for different clusters, since users often try to avoid the case that some clusters are much bigger than the others. In order to improve this criterion, we add *sizeWeight* parameter to control how much penalty that will be added to the bigger clusters when computing cluster assignment. When adjusting *sizeWeight* to a large value, the variance of the cluster size should go down. However, if *sizeWeight* is too big, it may

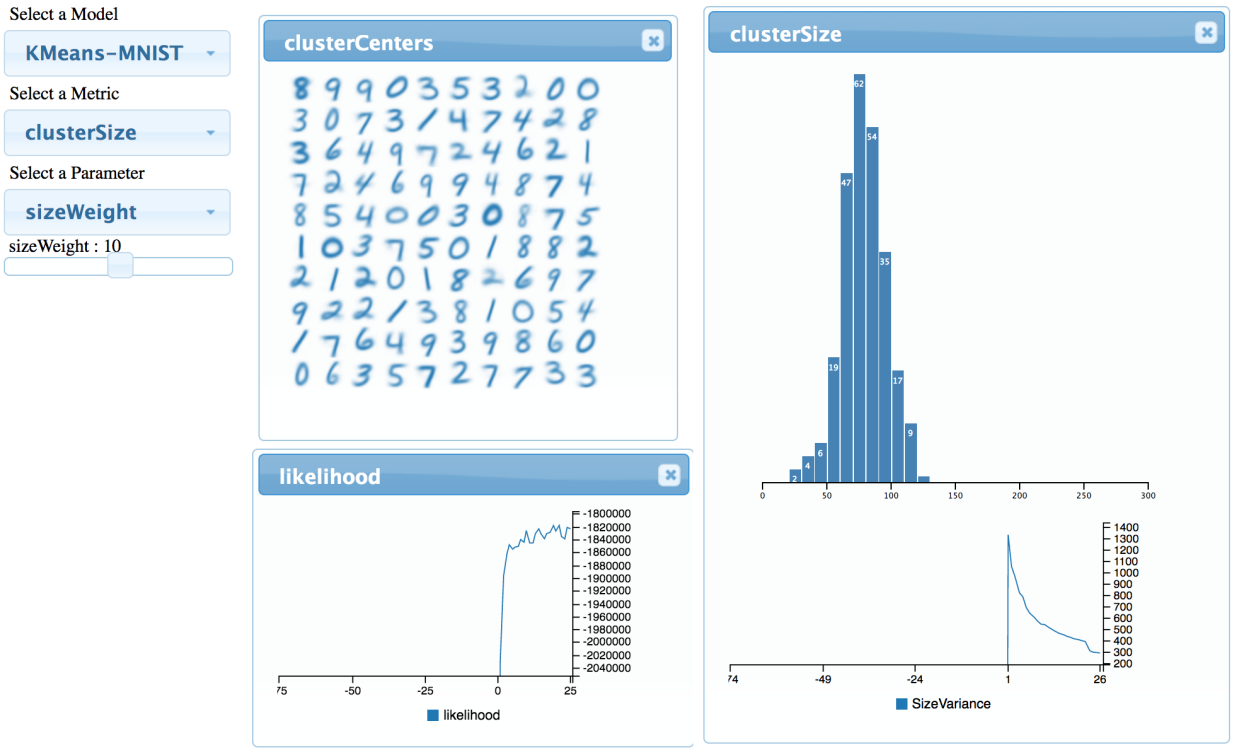


Figure 3.3: Visualization Dashboard for KMeans

make the clustering results almost like random assignment. Therefore we use the MNIST dataset [52] to conduct the study and visualize the cluster centers to help subjects avoid bad clustering results. The interface of tuning the KMeans algorithm can be seen in Figure 3.3.

The KMeans algorithm only has 1 parameter to tune, therefore it is relatively easier than the topic modeling. Also, only limited subjects have tried KMeans after they finished topic modeling experiments. In the rest of the paper, we will mainly discuss our experiments and findings for topic modeling.

Subject background

We recruited 28 subjects from the college campus via posters, emails. 17 subjects are undergraduate students, while others include 9 graduate students and 2 post-doc researchers. They come from a variety of background. The distribution of their home department is shown in Table 3.1. Majority of the subjects have few or none machine learning experience. 18 subjects claim they *almost never* used clustering or any unsupervised learning algorithms before. On the other hand, 22 subjects have ever used regression methods. 17 subjects have programming experience including R, MATLAB, Python etc.

Major	Subject number
Electrical Engineering and Computer Science	5
Natural science	7
Social Science	4
Engineering	7
Mathematics and Statistics	2
Undeclared major	3

Table 3.1: Subject background distribution

Study Procedure

The main task that subjects are supposed to finish is to explore the parameter space via the visual interface and to improve the topic modeling algorithm. However, since most subjects don't have machine learning background, it would be hard for them to learn how to make trade off between different evaluation criteria in a limited time. Therefore, we separated subjects into two groups. The first group contained 13 subjects and the second contained 15 subjects. The first group did not receive a concrete goal about what kind of improvement should be made. Instead, they were told to improve the model in terms of sparseness, topic uniqueness, which is more open-ended. In contrast, subjects in the second group are given a predefined goal which is to keep *likelihood* higher than -9, while making *density* and *cosineSim* as low as possible.

During the study, we first gave each subject an instruction sheet which includes the description of the dataset, the outcome of the algorithm and how to use the visual interface. We describe the system in such a way that no machine learning experience is required in order to use the toolkit. Subjects are told that *L1-reg* corresponds to *density*, and *dissimilarity* corresponds to *cosineSim*, *temperature* is controlling the variance and fluctuation level of the algorithm. Before the subject start to tune the model, we double checked that the subject understand the basic of the topic modeling algorithm, understand the meaning of the visualizations and the controls. Subjects are allowed to ask questions during the study, but we didn't answer questions that would give more information than the instruction sheet.

We record the computer screen for the whole tuning process. Video/audio recording is optional but preferred. During the study, we encourage subjects to think aloud when they were using the toolkit. We also record how many insights did each subject find out by observing their behaviors and confirming with them via some necessary questions. The criteria to determine if the subject has gained the insights include: (a). **The subject can describe the insight in his/her own language**. (b). **The subject can reproduce the phenomenon using the correct operations**. The full list of the insights that we are trying to measure can be seen in section 3.3

For each subject, We also measured and recorded the completion time (Only the first

time of completion will be recorded) for the following subtasks:

- 1. The subject successfully decreased *density* or *cosineSim* by any amount
- 2. The subject successfully turned *cosineSim* into 0 (See Figure 3.8c)
- 3. The subject successfully increased *density*/*likelihood* after decreasing them first (See Figure 3.8b)

The completion time is measured by counting how many values has passed in the streaming graph, as the streaming graph has a constant rate of showing 40 values per minutes. After they finished tuning, we continue to conduct a short interview to further discuss their findings and comments about the tuning process.

Expected insights

We list a checklist of those expected insights below:

- 1. There is trade off between *density* and *likelihood*. As *density* went low, *likelihood* would go low as well.
- 2. *cosineSim* can be optimize to as low as 0, while keeping *likelihood* almost unchanged.
- 3. In default temperature, first increase and then decrease *L1-reg* can make *density* fall and then rise. But decreasing *dissimilarity* after first increasing it has no effect on *cosineSim*.
- 4. Increasing *temperature* can help escaping local optimum. For example decreasing *dissimilarity* can make *cosineSim* rise again.
- 5. Decreasing *density* can shorten the list of words within each topic
- 6. Decreasing *cosineSim* can make topics independent to each other

The first 4 insights are describing the relationship between the parameters and the loss functions. They can be observed by looking at the streaming graph alone. The latter 2 insights are about the meaning of those 2 loss functions, and subjects need to look at the model visualization in order to find out such insights.

3.4 Quantitative results

As mentioned in section 3.3, for each subject we recorded which insights did he/she find, and the time to complete those subtasks. We conduct a thorough analysis on the user data and we will discuss several quantitative findings In this section.

Overall statistics

All 28 subjects successfully finished the subtasks we described in section 3.3. The averaging completion time is as follow:

	Subtask 1	Subtask 2	Subtask 3
Group 1	193.2s	464.4s	571.0s
Group 2	196.3s	454.4s	333.7s

Table 3.2: Averaging completion time

The details of the completion time are shown in Figure 3.4. Subtask 1 is relatively easy to complete since subjects only need to increase *L1-reg* or *dissimilarity* to reach a threshold value. Also, 9 subjects finished all subtasks within the first 3 minutes. However, since we didn't explicitly recommend subjects to try extreme parameter values, which can lead to fast completion time, some of the subjects used a more conservative strategy to gradually increase the parameters. Therefore it could take a while before noticeable change can be seen in the graph. Also, the averaging completion time of subtask 1 and subtask 2 is very similar between the two subject groups. For subtask 3, the difference between the two groups is not significant, either. We perform a two sample t-test on the data and result is : $t(df = 17.257) = 1.4487, p = 0.1653$.

Besides, in terms of measuring how many insights (Those described in section 3.3), we found that only 24 subjects are able to gain the first 2 insights, as shown in Table 3.3. This means at least 4 subjects finished those subtasks without really understood their cause. This is because subjects can finish the tasks by randomly moving the sliders without conducting any analysis about them.

On the other hand, the *temperature* parameter is much harder to understand and therefore is hard to use it in the right way. Even though we told subjects in the instruction sheet that *temperature* is used to control the level of variance and fluctuation of the algorithm, it is hard to perceive its effect from the graphs after adjusting temperature. Typically, increasing *temperature* will increase the level of noise in topic visualization, as shown in Figure 3.7. It may also lead to drastic changes in other streaming graphs. Usually it increases *cosineSim* while decreasing *likelihood* and *density*. Therefore it is hard for subjects to anticipate its effect without domain knowledge in machine learning and optimization. Only 3 subjects obtained insight 4 by successfully escaping from a local optimum under high temperature condition.

Insight 3 is also hard to gain, since most subjects are not familiar with such non-reversible behavior. We will discuss subjects response to it in the next section.

Having predefined goal reduces topic inspection

As described in section 3.3, the study has two subject groups. Subjects in the second group are given a predefined goal which is to make *density* and *cosineSim* as low as possible while

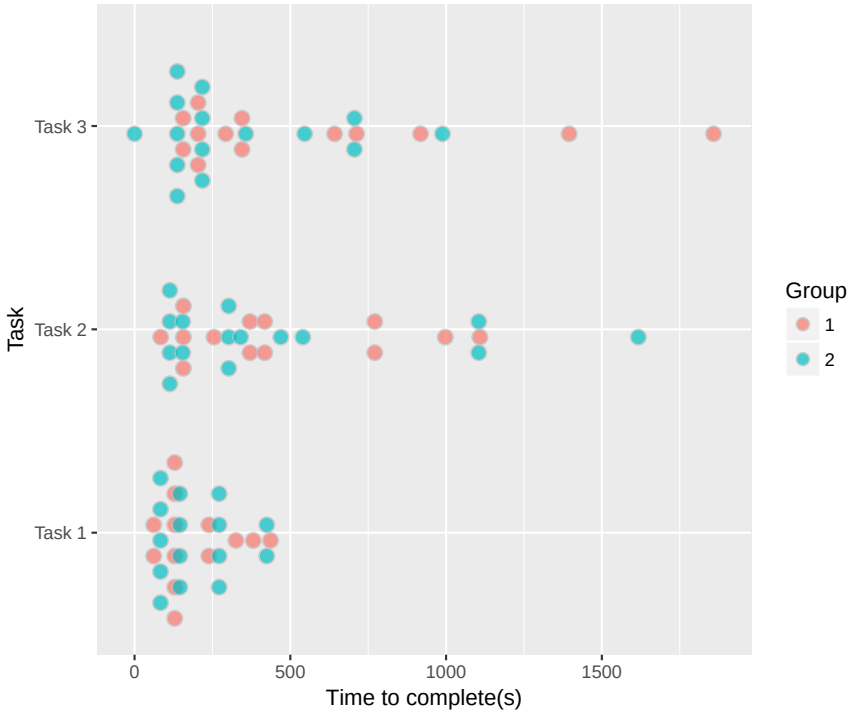


Figure 3.4: Distribution of task completion time

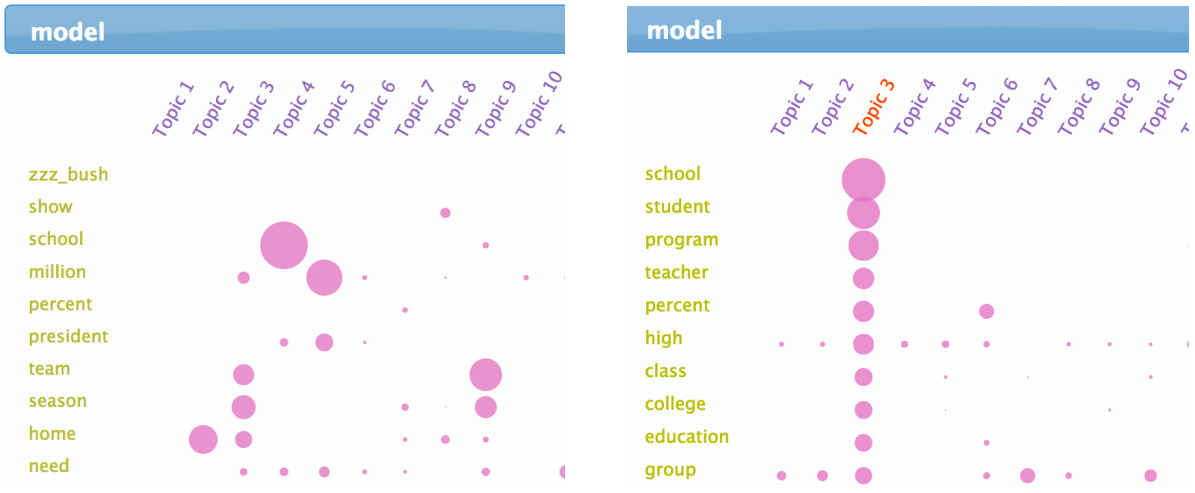
Insight	1	2	3	4	5	6
Count	25	24	4	3	5	12

Table 3.3: Number of subjects who gain the insight

maintaining *likelihood* higher than -9. Other instructions and experiment environments are the same. During the study, we found that, the 13 subjects from group 1 are more willing to spend time on the topic visualization. Subjects would dive into each topic one by one and try to understand how they are correlated with the streaming graphs.

Though we don't have equipment like eye tracking device, we instead measure how long did subjects put their mouse on the topic visualization. This is due to the fact that, in order to look at the words in individual topic, subject needs to hover the title of each topic. We plot the data in Figure 3.6, and it can be seen that there is huge difference between the two groups. We also conducted a two sample t-test, and group 1 spent significantly higher time than group 2 ($t(df = 14.491) = 4.1363, p < 0.01$).

By spending more time to inspect the topics, group 1 were on average doing better on finding insight 5, 6 as well. The success rate to find insight 5, 6 for group 1 is 3/13 and 8/13, while the rate for group 2 is 2/15 and 4/15. On the other hand, as mentioned earlier, even though group 1 spent more time on look at topic visualization, there was no significant difference for subtask completion time between the two groups. This may be partly due to



(a) Overview of the topics (b) Diagnosing the topic by hovering its title

Figure 3.5: Topic visualization

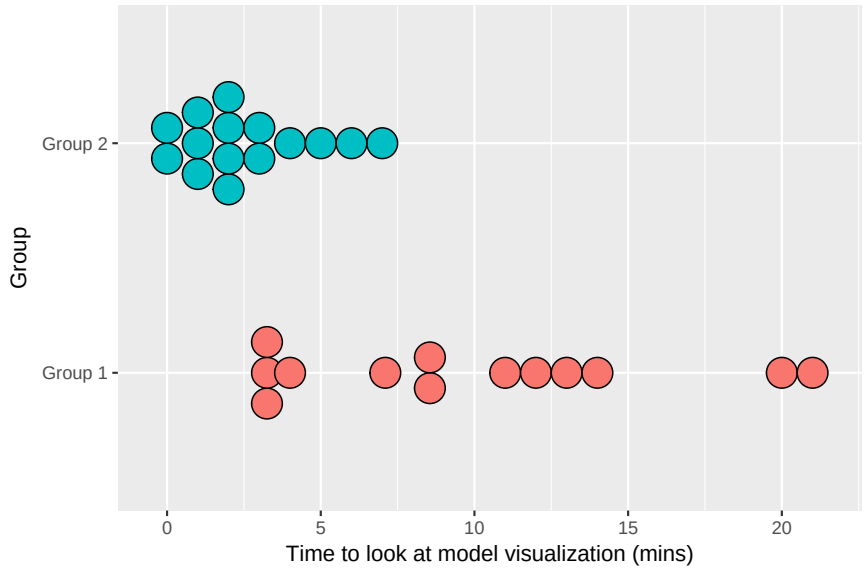
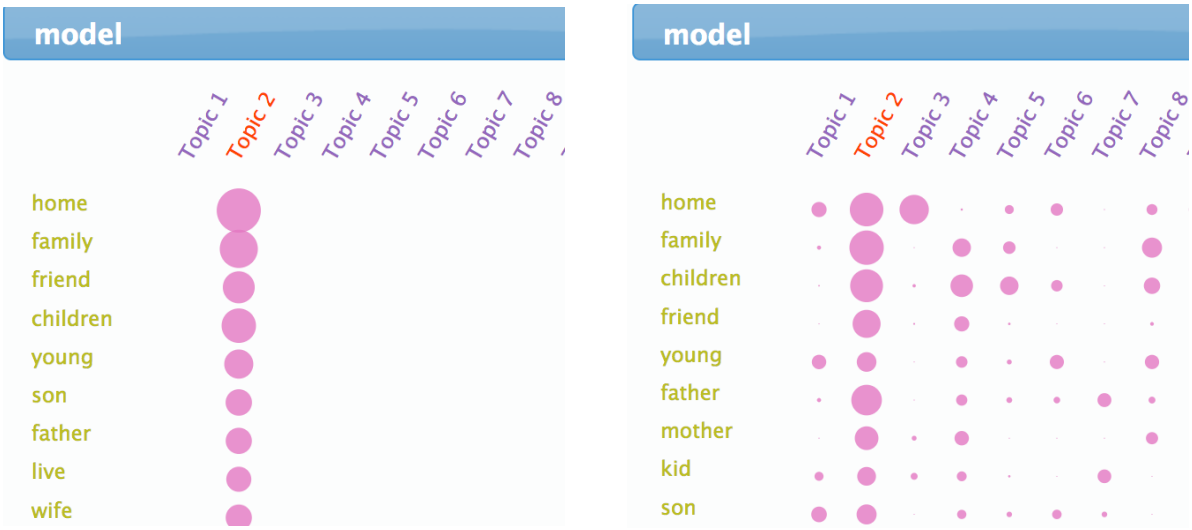


Figure 3.6: Group 1 spent more time looking at topic visualization

that inspecting the topics and tuning parameters could be done in parallel.

3.5 Subjects response

We encouraged subjects to think aloud and their verbal responses were being recorded. We therefore analyze how subjects are describing their tuning strategies as well as comments



(a) Before increasing temperature

(b) After increasing temperature

Figure 3.7: Effect of adjusting temperature

to the toolkit. We describe several qualitative findings in this section. When listing the quotations from the subjects, we include their subject id. S1 to S13 are in group 1, while S14 to S28 are in group 2. We also add necessary explanations within a parenthesis if subjects are pointing something on the screen or something that need to be inferred from the context. Such as: *"look at this one (cosineSim)"*

Strategy

It is very important to analyze how subjects plan their strategies. We therefore encouraged subjects to explain their actions during the study. Here are some of the common actions that subjects would perform.

Understanding the measurements:

The first thing that subjects usually did, especially for those in group 1, is to understand the relationship between numerical measurements and the topic visualization. Here is some quotation:

S4: "I notice that when the cosineSim was higher, there would be more similarity between the topics on a given word. Hum, and, I could go through in just, you know, kind of look at all of those (topics), or I could just, you know, trust the cosine similarity was capturing it. I mean, yep, like, this seems to reflect this petty well. So I start to just try to get this looks nice."

S6: "The cosineSim is good for, like making sure you're actually, like, similar, so this is going to get rid of, eh, like cross identification"

S12: "When the cosineSim is higher, then, hum, it will have like more dots in other topics. Density is just, huh, the higher is, the more like, the more dots there are in general. Likelihood is just, the higher likelihood, the bigger the circle is. "

S17: "One point is that, when likelihood went down, the performance of the model didn?t go down, in terms of human judgment. "

It can be seen that subjects have variety of ways to describe the meaning of the measurements. And subjects can actually gain wrong insights such as the meaning of the *likelihood*.

Trying extreme values:

Another useful strategy is to move sliders to the extreme and to observe what happen. This can quickly show an overview about what does the parameter space look like. Here are some quotations about how people apply this strategy:

S4: "When I move the parameters to the extreme, It does kind of gave me better understanding about what the parameters are doing, to begin with."

S23: "Just flip around, to maybe, kind of doing an extreme, and then kind of doing the other extreme, and then maybe less extreme on the other side, until you find where it actually drops."

S18: "Maybe I try some extreme cases first and then bound it to like, to make the region smaller about when it will get to the place we want."

Making tradeoff:

Subjects are supposed to use the toolkit to make tradeoff between those competing goals, which in this case are *density* and *likelihood*. However, it turned out that without knowing the meaning of the numbers, trade off are hard to make. Here are two quotations:

S5:"I am OK with black box as long as I know what the payoff function is for myself. Right now I am not 100% *clear on when I make changes, whether the result is better or worse..... I don't feel like I have done anything that requires domain knowledge yet. So far it seems like this should all be automatic."*

S6: "I think, using this plot (topic visualization) was more meaningful, because I can see, you know, I can put like what my goals were to like have less overlapped column and shorter column. What numbers are exactly correlated with those two? I think of a more like a guidance. I do not think you want them (Measurements) to be like way off, but I do not think I would use this to exactly get as optimal as possible. I think I would use like looking at the data, as it would guide me in choosing the optimal values. "

Streaming graph

Streaming graph is effective for parameter tuning:

Streaming graph is good at showing changes in time-series data. But it is not clear if it is also useful in interactive environment. As many subjects claimed they only used static

visualization before, we therefore asked them if they think the dynamic graph is hard to read and track. Here is one quotation from the subjects:

S21: "No, I don't think so. I feel like that is actually kind of helpful because that's your way of seeing the change happen, that's your way of being able to monitor whether the parameters that you're changing are actually impacting the model."

The response is quite representative, and the changes in streaming graph are the main source of insights when subjects are describing their findings. Here are quotations from two different subjects stating its usefulness in finding insights.

S5: "There is no better way to do it with interactive visual system, otherwise is all abstract. And I would never, I would probably never be able to get this level of intuitive graphs of that process, without being to able to do just like this. Very useful learning tool. "
S6: "Something like this where like, looks at it visually, and looks at the interplay between different choices, would be helpful."

Also, subjects claimed that looking at the topic visualization (Figure 3.5b) and inspecting the topics one by one is very time-consuming and is hard to make comparison. Instead, They would rather look at the numerical statistics. Here are two quotations:

S4: "Changing a parameter and then checking through every single topic and looking for, you know, its immediate effect, is, is kind of, you know, makes me more incline to look at the abstraction of it, in cosineSim and likelihood. "
S6: "Because right now this looks like they are plausible topics, so could be right. If they change, I am not sure if I can detect there is improvement or not. So then what I have to do is go topic by topic, which is kind of a lot of work, to do this. "

Also, streaming graph makes it much easier to compare current state and the history. Being able to make meaningful comparison is very important, if the main goal is to understand the effects of parameter manipulation. Here are quotations from two subjects stating the difficulties in reading topic visualization.

S6: "Those plots (Streaming graph) help a lot, cause just looking at this (topic visualization) is more difficult to absorb like how the shape is changing"
S17: "I can't do this. I can't remember what those topics are in the last iteration. "

Tracking 3 graphs could be hard:
During the study, we did notice that sometimes subjects would ignore the changes in one streaming graph while they're looking at the others. And subjects confirmed with us that 3 graph is hard for them to track. Here are quotations from two different subjects:

S18: "I think I can focus on 2 at a time, to compare the difference. But 3 maybe, maybe take time."

S21: "I think 3 graph isn't too much. It just takes time to find the pattern ..... Because when you try to make a goal considering 3 parameters and 3 graphs here, you have to keep in mind the two other goals that you want for these, and try to relate them so that you get a perfect state. I feel like it depends, but I think that 1 or 2 is just simpler"

As pointed out by the subjects, it is normal to look at 1 graph, or comparing 2 graphs at the same time. Having 3 graphs introduces complexity and subjects need to allocate their attention wisely. Since our system didn't support pausing the training, subjects therefore need to simultaneously decide where to look at as the graphs are changing all the time. This is quite challenging at the beginning when subjects are not yet familiar with the system. But as they became familiar, they could kind of anticipate what would happen. Here is one quotation:

S6: "I think the second time around, it is like when you read something before coming to the class, even if you don't really read it. You still like are familiar with the vocab, and you are familiar with kind of like with how things fit it, that you know the general behavior of it."

Gaining such experience, subjects could therefore focus on different graphs at different time. The most common strategy is to ignore the third graph unless something drastic happened. Here are quotations from two different subjects:

S6: "I don't think it is too overwhelming. I think like, just these 3 is OK. Cause I can kind of see these out of the core of my eye. Yep, if I am expecting this (L1-reg) only impact this one (likelihood/density), then I can just focus on this one. I don't have to look at this one (cosineSim). This is kind of the experience you gain like doing it multiple times. Like which one impact which one."

S18: "I just see the changes, and then generally see the other one the third one, if it does not change a lot, then I will just keep going."

This also informed us that, being able to display the history of changes is very important. Since subjects may frequently ignore some graph at first, but would come back to see it later. If the history data is not being shown, then the subjects would have no way to tell what have changed. The streaming graph is therefore doing a better job than other visualizations in such scenario.

Reversibility

Since machine learning algorithms can fall into local optimum, changing the parameter back to its original value does not necessarily bring the model back to its previous state. During the study, subjects can figure this out, even though they might not realize this at the begin-

ning. However, only 4 subjects can clearly describe the non-reversible behavior of *cosineSim*. Here are two quotations:

S10: "It seems that cosineSim is still low even if I adjust dissimilarity back to 0 ..... So this model, it seems like it has two properties, not reproducible and not reversible. "
S18: "Sometimes dramatic change would make cosineSim stay 0, and even though I change it back, it will stay 0"

On the other hand, realizing the non-reversible property seems to make subjects more conservative about their tuning strategy. Here is one quotation:

S5: "I would not have to if it behaves like a real parameters tweaker that I used to. Which is that if I go back, the state returns. This one though, is more like going through, exploring a maze. And if you take several steps, it is not easy getting back. I am cautious about just moving the sliders. "

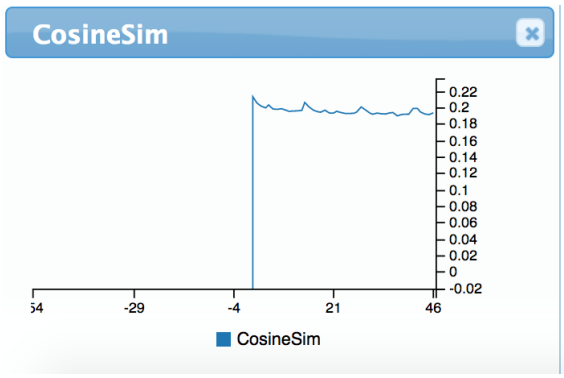
Conservative strategy is also more important in KMeans experiments, since large $sizeWeight$ can break the model and the only way is to restart the algorithm.

Latency

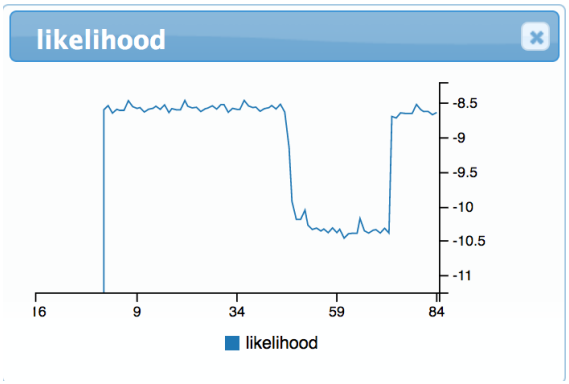
In any explorative data analysis, the existence of latency can affect how users interact with the system. As pointed out in [59], increased latency reduces the rate at which users make observations, draw generalizations and generate hypotheses. In our case, we observe that the latency in machine learning algorithm not only slow down the analytic process, but can also give subjects wrong intuitions.

Comparing to the usually fixed-time system latency in explorative data visualization, latency in machine learning algorithms comes from several different sources. There is system latency which includes data loading time, data processing time, communication and rendering time. Since BIDMach uses mini-batch algorithms, system latency actually means how long does it take to process one small batch of data so that subjects can see visual feedback. Besides, there is algorithm latency, or so called convergence rate. Empirically it means how many mini-batch data is needed for algorithm to converge and show noticeable changes on the screen. In practice, for the topic modeling algorithm in our study, system latency is around 300ms, algorithm latency is around 5 to 10s.

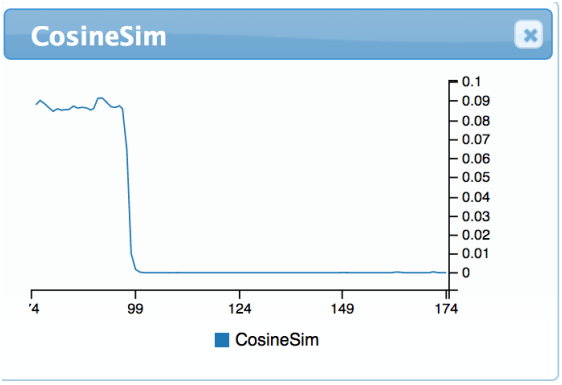
After subjects changed the parameters, usually there are three types of system response, as shown in Fig 3.8. Fig 3.8a shows the case that the weight of loss function is too small to cause any noticeable changes. This usually happened at the beginning of the tuning process that default parameters are all very small. Fig 3.8b shows the case that significant changes can be seen in the graph. Fig 3.8c is similar to Fig 3.8b, with the difference that the value of *cosineSim* can not reverse back in the default temperature. Therefore no visual changes can be seen no matter how the subjects change the parameter. We found that, before getting



(a) Flat changes when parameter is too small



(b) Sharp changes



(c) No changes after falling into local optimum

Figure 3.8: Different types of system response

familiar with the system, subjects may not be able to distinguish the situation between Fig 3.8a and Fig 3.8c, hence spent lots of unnecessary time waiting for the results. Here are two quotations:

S10: "It seems that cosineSim is still low even if I adjust dissimilarity back to 0
S23: "It is kind of interesting that when you sliding this (L1-reg), there is hum? they will be not changed for a long time, then they will just all of a sudden reach 0.1"

However, some subjects would use a much better way to handle the latency:

S5: "And it takes about 10 seconds to scan the whole things. So it shouldn't take more than 10 second before it get stable. "
S6: "I think what I was doing was that, I change it, and like if I didn't see anything change within 5 seconds, then I will just change it again. I think it was relatively easy to find the behavior, once I play with it around long enough."

On the other hand, most subjects didn't realize tuning the weight of loss functions not only changed the converged value, but also changed the convergence rate. Therefore the expected algorithm latency should be changing depending on the scale of the parameter. This can lead to some unexpected non-linear behavior that confused the subjects. But subjects could also adapt their strategies accordingly. Here are two quotations:

S4: "This is definitely a strategy that more gradual change which would allow me to kind of control specific outcome, without having wider, more drastic effect."

S23: "Like I said, there will be times when you increase it, as it feel exponential or something, and you will increase for a long time and nothing will happen to the likelihood or density, and then you will reach a point, and it will just, likelihood will drop."

3.6 Conclusion

In this chapter, we discussed a user study about how visual interface would help subjects perform interactive parameter tuning. 28 subjects participated in our study. We had 2 major quantitative findings that on average users can learn to use the toolkit in less than 10 minutes, while giving subjects a predefined goal can significantly reduce the time spending on topic inspection. Based on subject responses, we also found that subjects are able to gain useful insights despite of having difficulties understanding the behavior of machine learning algorithms. Also, streaming graph turned out to be very effective in tracking progress and making comparison, while other graphs like topic visualization can help subjects understand the meaning of the measurements which are extremely important when making tradeoffs between different criteria.

During the study, we also found that, many students felt uncomfortable for multiple-objectives optimization. They kept asking for a more specific goal so that they could figure out the direction towards it. We sometimes need to use the analogy that GPA can not fully represent a person's potentiality to explain the concepts of multiple-objectives optimization. It turns out in the modern world, KPI (Keep performance indicator) driven rules have shaped the way how people think and how people make decisions. However, as we mentioned before, finding the right loss function (KPI) could be difficult in the first place. Once the goal is decided, optimization becomes straightforward. But how should we choose the goal itself? The tool we built didn't give us a thorough solution. We only make it possible for users to explore the parameter space so that they could make comparison and gain knowledge. This is somehow similar to our daily life. Balancing all those different factors is hard, but exploring the life itself gives us knowledge and memories. And it always takes some time and cost to figure out the goal that you really need.

Chapter 4

Diagnostic visualization for deep learning

4.1 Introduction

In this chapter, we will describe how to use the idea of exploratory model analysis in deep learning by providing interactive diagnostic visualizations. As deep neural networks (DNNs) have seen wide adoption, their ability to learn ad-hoc features also makes them hard to understand and diagnose. Existing tools such as TensorBorad [98] and VisualDL [75] provide low-level APIs for deep network visualization. While Tensorflow is primarily an offline visualization tool, VisualDL can produce live training visualizations when used with a define-by-run tool like PyTorch. Both tools however use log files to flow data between the deep network runtime and the visualization system and are not reconfigurable during training. Here we focus on visualization as a diagnostic tool with tighter coupling between visualization and training, and the ability to dynamically add/terminate visualizations during training runs. We explore several computational approaches that improve the quality of diagnostic visualization for image-based networks. Visualization for deep networks has a long history with approaches like guided backpropagation [88, 90], activation maximization [71–73]. These approaches have produced impressive visualizations for fully-trained networks, however their emphasis seems to be on similarity between sythesized and real image regions. Here we focus on the direct interpretability of neuron visualizations and in making the trade-off with interpretability explicit.

A naturally-interpretable approach among these methods is activation maximization (AM), which displays the behavior of a particular neuron by finding images maximally activating it. AM directly computes the gradient of the input image with respect to a target activation of the selected neuron, and follows these gradients to produce an input image visualization. This process is highly underconstrained however, so regularization must be added to produce a unique image. AM can then be posed as an optimization problem over the image space Ξ: Given $f_\alpha(x)$, the activation of a neuron α on an input image x, and $R_i(x)$

the regularization terms, compute:

$$\text{argmax}_{x|x\in\Xi} f_\alpha(x) + \sum_i \lambda_i R_i(x) \tag{4.1}$$

In contrast to image based visualization which finds patches of training images that maximally activate the neuron, AM generates images that derive solely from the neuron's properties. However, the optimization problem is non-convex, therefore there are multiple possible maximally activating images. As [72] pointed out, neurons can have multifaceted behavior, in that they fire in response to many different types of features. Therefore for diagnostic purposes, users should be able to explore the space of images that highly activate the neuron. This presence of many maxima, combined with the multi-faceted behavior of neurons makes gradient ascent algorithms unappealing. Any particular initialization leads to a single local optimum, but it can be difficult to choose these initial images to thoroughly explore the space of highly-activating images *et. al.* [28]. The typical solution revolves around picking clever initializations on the image manifold $G(z)$. For example, [72] used multiple pre-clustered images as starting points to cover the image space.

In contrast to the customized initialization approach, we propose an algorithm, **Langevin Dynamics Activation Maximization (LDAM)**, which samples images from the entire image space based on the activation score ratio. LDAM is a gradient-based MCMC (Monte Carlo Markov Chain) algorithm using Langevin Dynamics [96] which samples directly from the distribution:

$$\forall x \in \Xi \quad P(x) \propto \exp\left(\frac{f_\alpha(x) + \sum_i \lambda_i R_i(x)}{T}\right) \tag{4.2}$$

where T is a temperature parameter.

Thus, LDAM will traverse the manifold of highly-activating images directly. The difference between gradient-based optimization and gradient-based sampling algorithm is demonstrated in Fig 4.1. Using LDAM, we can visualize the manifold traversal in real time as a live animation while allowing users to directly manipulate the hyper-parameters of the traversal. As discussed in [87], such direct manipulation can be particularly beneficial for users' understanding during training.

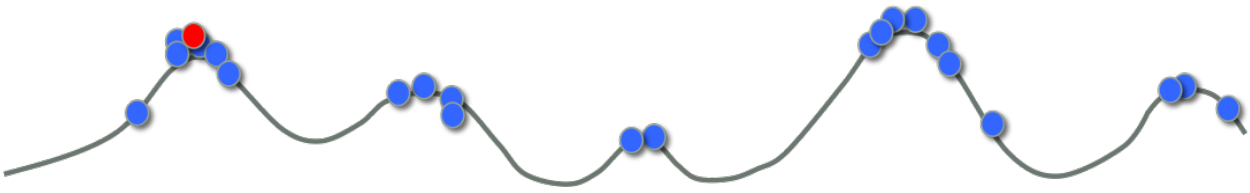


Figure 4.1: Monte carlo sampling can obtain samples (blue dots) from the entire space while optimization method usually only obtains one local optimal and is highly dependent on initialization (red dot).

Since activation-maximization is an under-constrained problem, AM algorithms often generate meaningless noise with high α neuron activation, and can be used to create "adver-

sarial" examples [93] which lead to incorrect class outputs. To help solve this problem, [73] summarized several different regularization techniques which can make results smoother [63] or more human-interpretable [71]. However the goal of most regularizers is to make AM images that more closely mimic image regions. This can help interpretability in terms of image regions, but also distorts the visualization as a representation of the underlying neuron properties (i.e. its sensitivity to particular pixels). Here we explore the trade-off between pixel-direct "mechanistic" visualization and interpretability in terms of input image regions since both can aid the understanding of neuron behavior.

Therefore, our goal is to explore the image-space while limiting our regularization to improve the diagnostic power of our system. Our major contributions are:

1. We introduce a sampling algorithm, LDAM, based on Langevin Dynamics which can explore the activating-image manifold.

2. We use L2 regularization (only) and sample averaging to obtain interpretable pixel values.

3. We use a tunable discriminator loss to control the trade-off between mechanistic (pixel-based) and image-like interpretation.

4. We evaluate LDAM on several image datasets and model architectures. We also report a novel insight about the training of deep networks with parameter averaging.

This chapter is structured as follows. Section 2 summarizes related work. Section 3 describes our algorithm in detail. Section 4 and 5 demonstrate the usage of our algorithm to diagnose networks trained with MNIST and CIFAR10 dataset. We compare models with different architecture or trained with different methods. The analysis of parameter averaging technique is discussed there. Section 6 discusses how to diagnose and explore the relationship between intermediate-layer neurons for AlexNet trained on the ImageNet dataset. Section 7 ends with conclusion and some future work.

4.2 Related work

Diagnostic deep learning visualization

There are a myriad of techniques for visualizing deep learning systems that have been presented in the last few years. Hohman *et. al.* [40] provides a comprehensive survey of existing visualization techniques for understanding and diagnosing deep neural nets - however we summarize a few of the more influential here. Many recent tools focus on visualizing the neurons during training. One of the most recent tools, ActiVis [48], is an industry scale toolkit which allow users to perform instance and subset level inspections using clustering techniques. Their toolkit allows for analyzing and viewing clusters of neuron activations, and provides activation-level information about the images. Rauber *et. al.* [82] take a different

direction by applying dimensional reduction techniques to visualize relationships between activations and between artificial neurons. Liu *et. al.* [58] uses graph visualization to show relationship between the neurons, and visualizes their activation space. Zeng *et. al.* [101] use side by side visualization to compare the activations of pairs of CNNs during training, Chung *et. al.* [19] allow users to manipulate particular neuron during training to see the training effects, and Liu *et. al.* [57] analyzes the training process of deep generative models, by providing several visualizations to identify problems during training.

While activation-level techniques can be exceptionally useful in understanding which neurons contribute to mis-classifications and understanding neural layouts (Bilal *et. al.* [10] analyzes hierarchies of similar classes to discover which neurons correspond to different levels of class hierarchy), a more general issue with many of these activation-level techniques is that they provide no way for users to understand image-level features that correspond to neuron activations. Recently, Pezzotti *et. al.* [79] provides a visualization tool which allows users to view the receptive fields of neurons composed over the entire training set. This could help users identify several diagnostic properties of a model such as stable layers, degenerated filters, undetected patterns, over-sized or unnecessary layers. However, visualizing the receptive fields using the image patches from the training set only highlights what stimulus the neuron responds to but not how it responds. In contrast, images generated by AM can give us information about the network gradient and model parameters. We will discuss the insights derived from AM in later sections as well.

Activation maximization

Central to the idea of visualizing the behavior of a neuron is the activation maximization technique. Olah *et. al.*summarizes different activation maximization methods in [73] and describe several applications in [74]. The first issue with activation-maximization is that it is underconstrained. Highly-activating images can be chosen which are visually far from the target class, as demonstrated by Szegedy *et. al.* [93]. By starting with an image of one class, and using activation maximization, [93] showed that adversarial examples could be created with very small perturbations in the pixel space.

To better constraint AM images, various regularlizers have been proposed. Mahendran *et. al.* [63] proposed two of the most popular regularizers: total-variation (TV) regularization and jitter. Total-variation regularization penalizes images which have locally non-correlated pixels and can help with de-noising the activation-maximized image. Jitter regularization often helps to control for fine detail, and can create sharper and more vivid reconstructions.

Besides using regularization, Nguyen *et. al.* [72] take the view that interpretability of the activation-maximized images is highly dependent on the initialization of the image. By using images from the training set clustered according to activation as the starting point, [72] shows that more interpretable images can be generated. In a follow-up, [71] uses a pre-trained generator to limit the optimization space, which then produces realistic images corresponding to higher-level activations. While the interpretability of generated images can be high, the creation of a single image is both computationally expensive and highly

dependent on the generator itself, which makes it unsuitable for diagnostic purpose. LDAM borrows the ideas presented in [71] by replacing the generator with an adjustable adversarial discriminator. The adjustable discriminator enables LDAM to generate interpretable image samples without introducing bias in an uncontrolled manner.

Guided back-propagation [90] is another visualization method that also uses the gradient for the input images. It can be used to visualize the saliency map for a given input image. Smilkov *et. al.* [88] improves the results from [90] by adding gaussian perturbations to gradients multiple times and average the results to achieve smoother gradients. LDAM borrows the idea from [88] by performing sample-averaging on a series of noisy samples to improve the interpretability of the results.

Generative Adversarial training

Recent advances in generating realistic images have primarily been due to the introduction of adversarial training first presented by Goodfellow *et. al.* [33] in the construction of Generative Adversarial Networks (GANs). A GAN consists of two components, a generator and a discriminator. The generator $G(z) = x$ generates an image using a random input (z), while the discriminator $D(x)$ takes in an image x and attempts to determine if x is from a real dataset or was generated. By jointly training the generator and the discriminator, the generator gets better and better at "fooling" the discriminator, leading to more realistic images produced by the generator. In addition, the formulation for a GAN also discourages the generator from copying images that are already in the dataset. The GAN training process minimizes the Jensen-Shannon divergence between the distribution of $G(z)$ and the distribution of training images. Zhu *et. al.* [104] have built an interactive toolkit which can allow users to explore such learned image manifolds in a user-controllable way. LDAM borrows a number of ideas from GANs to improve the interpretability of the generated images, and to allow users to explore and define the interpretable image space.

Langevin dynamics

One of the biggest problems with activation maximization techniques is that given an initialization, there is only a single maximum which can be achieved by gradient ascent. Not only is this the case, but Erhan *et. al.* [28] found that even with relatively distributed initialization, activation maximization produced only very few unique samples in practice. This is particularly unsatisfying for visualizing images which maximize a neuron activation, because we would like to see a large number of unique images corresponding to a neuron's activation (or a cluster of neuron's activations). Thus, we would like to sample images corresponding to high activations of a neuron.

To sample from this space LDAM borrows the optimization technique from Welling *et. al.* [96], which uses a combination of a stochastic optimization algorithm with Langevin Dynamics which injects noise into the parameter updates in such a way that the trajectory of

the parameters will converge to the full posterior distribution rather than just the maximum a posteriori mode.

A few optimizations over [96] have been studied. Feng *et. al.* [30] uses Stein Variational Gradient Descent to improve the diversity of the samples drew from a posterior distribution. Neelakantan *et. al.* [68] use gradient noise to help train deep neural nets, and Gulcehre *et. al.* [35] propose using noisy activation function to allow the optimization procedure to explore the boundary between the degenerate (saturating) and the well-behaved parts of the activation function.

4.3 Proposed Algorithm

Preliminaries & Notation

Let $f_\alpha(x;\theta)$ be the activation of a neuron α in a neural network f with parameters θ given the image x from some image space Ξ. We also let Π be the pixel space.

The goal of our algorithm is to sample images $x \in \Xi$ with high values $f_\alpha(x)$. To do so, we first define the random variable X, and the probability mass function $P(X = x|f_\alpha, \theta)$ as:

$$\forall x \in \Xi \quad P(X = x|f_\alpha, \theta) \propto \exp\left(\frac{f_\alpha(x)}{T}\right) \tag{4.3}$$

As we can see, the probability of sampling any image from the manifold is proportional to the activation of that image by the classifier $f_\alpha(x;\theta)$. The only issue with the formulation in equation is that it depends on an image manifold Ξ, which may not be available for direct sampling, or may be difficult to sample from. To resolve this issue, we can sample directly from pixel space Π, giving rise to the PMF:

$$\forall x \in \Pi \quad P(X = x|f_\alpha, \theta) \propto \exp\left(\frac{f_\alpha(x)}{T}\right) \tag{4.4}$$

The issue with equation 4.4 is that we are now no longer constrained to the image manifold Ξ, but a much more general space Π. To rectify this we use a regularization function $R(x)$ to define a second PMF over the images of Π:

$$\forall x \in \Pi \quad P(x \in \Xi) \propto \exp\left(\frac{R(x)}{T}\right) \tag{4.5}$$

We make the assumption that $P(x \in \Xi)$ and the probability that x activates $f_\alpha(x)$ are independent to give us our final PMF, $h_X(x)$ for X from which we can sample:

$$\begin{aligned} \forall x \in \Pi \quad h_X(x) &= P(X = x) \\ &= P(x|f_\alpha, \theta)P(x \in \Xi) \\ &\propto \exp\left(\frac{f_\alpha(x) + \lambda R(x)}{T}\right) \end{aligned} \tag{4.6}$$

Note that the regularization (x) could be viewed as a prior distribution on the image manifold. This prior could be provided by a discriminator $D(x)$, or could be a simple regularization, such as the $L2$ norm. Each of these different regularization functions provides for differing characteristics in the final images, which is easily seen by this factorization. In addition, this factorization into activation and regularization parts means that at the time of diagnosis, the user can smoothly vary the regularization function to view the effects that different image-manifold assumptions have on the sampling process (something which is impossible for traditional activation maximization techniques). The parameter λ in equation 4.6 is also important, as it represents a trade-off between sampling images that lie on Ξ, and images that are highly activating the selected neuron.

Monte Carlo Sampling Using Langevin Dynamics (LDAM)

In order to sample from the distribution proposed in the previous section, we propose LDAM, an algorithm which uses Monte Carlo sampling rather than pure gradient ascent. While pure gradient ascent could be used to optimize the function

$$\underset{x \in \Pi}{\operatorname{argmax}} f_\alpha(x) + \sum_i \lambda_i R_i(x)$$

we find gradient ascent unsatisfying, as it cannot sample from the PMF given in equation 4.6, and only finds a local maximum of that function, so it may miss a large number of highly probable activating images.

In Bayesian learning, Langevin Dynamics is traditionally used to generate samples from the posterior distribution $P(\theta|D)$ where θ is the model parameters, and D is the training data. We flip the traditional sense, and sample from Π according to the PMF given in equation 4.6.

To sample the random variable X, LDAM uses technique proposed in [96] which is a gradient based Monte Carlo method. LDAM generates a sample x_t at time t from the distribution of X, $h_X(x)$ by injecting suitably scaled gaussian noise in the gradient direction:

$$x_t = x_{t-1} + \beta_t \Delta x_{t-1} \tag{4.7}$$

$$\Delta x = \nabla_x f_\alpha(x) + \sum_i \lambda_i \nabla_x R_i(x) + \eta \quad \eta \sim N(0, \sigma) \tag{4.8}$$

In this case λ, β and σ are hyper-parameters which control the sampling process. In LDAM we sample our initial x_0 using isotropic Gaussian noise. Typically step sizes are $\beta_t = a(b+t)^{-\mu}$ decaying polynomially with $\mu \in (0.5, 1]$. In our implementation, we leave the step size up to the user as a trade-off between local and global exploration.

It's worth noting that for some regularizers (for example the discriminator), the scale of $\nabla_x f_\alpha(x)$ and $\nabla_x R(x)$ could be extremely different. Thus instead of directly using the gradient, we use a normalized gradient provided by the RMSprop algorithm for those regularizers. The detailed algorithm can be seen in Algorithm 1.

Algorithm 1 Langevin Dynamics Activation Maximization (LDAM)

Input: Network $f_\alpha(x)$, Regularization functions $R_i(x)$
Initialize: Random Image x_0, $v_0 = 0$
for t = 1,2,3... **do**
  Forward pass: Compute $f_\alpha(x_t), R_i(x_t)$.
  Backward pass: Compute $\nabla_x f(x_t), \nabla_x R_i(x_t)$.
  Get normalized gradient $\overline{\nabla_x f_\alpha(x_t)}$ and $\overline{\nabla_x R_i(x_t)}$ if needed
  Draw η_t from $N(0, \sigma_t)$
  $g_t = \overline{\nabla_x f_\alpha(x_t)} + \sum_i \lambda_i \overline{\nabla_x R_i(x_t)} + \eta_t$
  $v_{t+1} = \gamma v_t + \beta_t g_t$
  $x_{t+1} = x_t + v_{t+1}$
  Process new sample x_{t+1}
end for

After obtaining each new sample x_{t+1}, we could either directly visualize it as a live-animation in the interface, or use sample averaging method to compute the moving average of all the samples received so far. In later section, we will show that sample averaging method can greatly reduce noise and improve the interpretability of the generated images.

L2 regularization

Choosing the right regularization function (equation 4.5) is extremely important. The first regularization function that we consider is the $L2$ norm of the generated image, i.e $R(x) = -\frac{1}{2}||x||^2$. While L2 has frequently used in previous work to "clean up" generated activation maps, its role in pixel interpretability does not seem to have been noted. Namely that at a local optimum of activation, the *activation equals the gradient.*

$$x^* = \underset{x \in \Pi}{\operatorname{argmax}} f_\alpha(x) - \frac{1}{2}||x||^2 \implies \tag{4.9}$$

$$\nabla_x f_\alpha(x^*) - x^* = 0 \implies \tag{4.10}$$

$$\nabla_x f_\alpha(x^*) = x^* \tag{4.11}$$

An important corollary of this observation is that for neurons whose output is linear in the image values *the L2-regularized activation equals the filter weights.* i.e. the L2-regularized AM reproduces the filter weights in the first convolutional layer, and generalizes in a natural way to gradients in other layers.

Note that the solutions to $\nabla_x f_\alpha(x^*) = x^*$ may not necessarily be unique. Therefore using LDAM, we could make transitions between those solutions. Also, in LDAM, we don't need to normalize the gradient coming from L2 regularization.

In addition, adding $L2$ regularization gives us a good way to verify the correctness of the implementation. AM applied to neurons in the first convolutional layer should match filter weight images.

Adversarial Discriminator

Our goal is to maximize interpretability of AM maps from both pixel and image perspectives. A variety of subjective regularizers have been used in prior work, but each distorts the pixel interpretability of the basic L2-regularized model. Secondly, we want to use visualization during training, and the artifacts that may hinder image-perspective interpretability will change over time. So we seek to apply a regularizer that maximizes image interpretability at each stage in training, with the minimum pixel-level distortion. The solution is to train a discriminator as used in Generative Adversarial Networks, and use discriminator gradient to improve AM interpretability. By training a network to distinguish between real images $x_R \in \Xi$ and fake images $x_F \in \overline{\Xi}$ we are implicitly constructing a probability distribution $P(x \in \Xi) \propto D(x; \theta')$. Thus, we can take for our regularization function $R(x) = D(x; \theta')$. By periodically re-training the discriminator, we ensure that it tracks and minimizes the image artifacts that AM produces at various stages of training.

Nguyen *et. al.* [71] used a pre-trained generator to limit the image space that their activation-maximization process would explore. While this approach could shape the image manifold, our goal is not to generate images that emulate a dataset, but to explore and understand the image space which activates the neurons of the original network. Therefore we would like to carefully understand and regulate the impacts of our manifold shaping. Thus, we use only the discriminative portion of the network, along with the weight λ in equation 4.8 to allow users precisely control the blending ratio of the discriminator and the original network.

We present an example of LDAM using a discriminative regularization in Fig 4.2. As shown, the discriminator is a separate network from the original classifier $f_\alpha x; \theta$. While the discriminator could take many shapes and forms, for simplicity in our experiments we attempt to use a structure which is as similar to the original classification networks as possible - they all have the same input size and similar hidden layer structure.

To borrow additionally from the GAN literature, we can view LDAM as a generator attempting to fool the discriminator, thus if the goal is to generate natural images, we can train the discriminator with real images as well as fake images to create a stronger prior on the image manifold Ξ. Algorithm 2 describe the training process in details. By training the discriminator online with the LDAM method, we can decrease the burn-in time of the LDAM sampler. This is because the discriminator learns to recognize burn in samples as fake and it will provide strong gradient information to drive the sampler away from them.

It's worth noting that in Algorithm 2 we take steps using LDAM until $D(x) > 0.9$ Doing this prevents the discriminator from overwhelming the activation maximization directions.

Algorithm 2 LDAM with a Discriminator

Input: Network $f_\alpha(x;\theta)$, discriminator $D(x;\theta')$
repeat
  Initialize: Random image x_0, $t = 0$
  repeat
    Generate next sample x_{t+1} using LDAM with $f_\alpha(x;\theta)$ and $R(x) = D(x;\theta')$
    $t = t + 1$
  until $D(x_t) > 0.9$
  Collect half batch of generated samples with label 0, and half batch of real images with label 1
  Update the discriminator weights θ' using this batch of data.
until Stop by user

System Design

Here we describe the tool that we created for real-time visualization of the LDAM sampling process. As our system is designed for diagnostic purposes, we keep simplicity and interactivity in mind as key design principles. For simplicity, the LDAM algorithm and visualization should not change the data-flow and structure of the original network. The method should work as a plug-in component that can be added/removed by users at any time. For interactivity, we try to optimize the performance so that users can obtain results in real-time. The system architecture can be seen in Fig 4.2.

For hyper-parameters λ, β, σ as we described above, we create a slider for each of them in the interface so that users can adjust their values during the sampling procedure. Users can see instant feedback after they adjust the values. We implement the system in both BIDMach [14] and Tensorflow [1]. They have similar performance but the BIDMach version could provide better interactivity.

A snapshot of our interface can be seen in Fig 4.3. The upper panel is visualizing the image samples as live animation. As we process images in mini-batches, we can create samples for different neurons at the same time. Each neuron can also have multiple independent MCMC sampling procedures at the same. Usually each column represents one neuron, and each row represents different MCMC instances with different initialization. The bottom panel contains the control sliders for the hyper-parameters.

4.4 Case study: MNIST dataset

To show the applicability of our method to the diagnosis of neural architectures, we perform case studies on the MNIST, CIFAR-10, and ImageNet datasets. Through these experiments we explore how LDAM functions in practice, and how we can vary parameters in the model to achieve clearly understandable results. In addition, we explore the choice of regularization

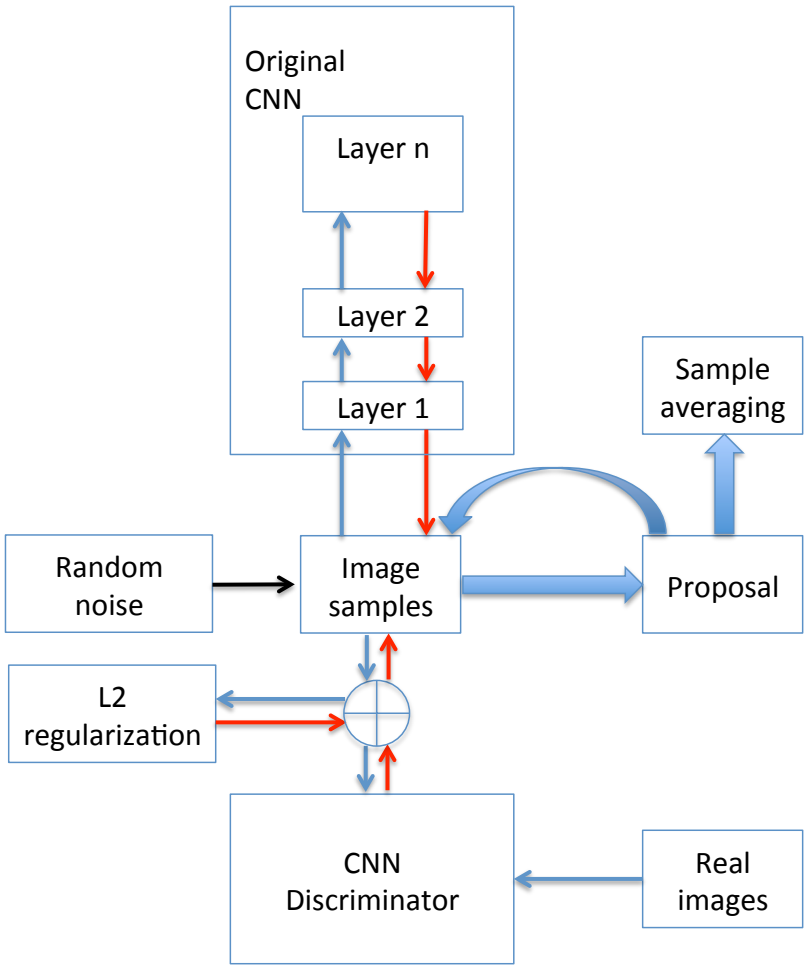


Figure 4.2: System architecture. Red arrows indicate gradient flow (backward pass), blue arrows indicate forward pass

function $R(x)$, and attempt to understand how influencing the image manifold can provide interesting insights into the actions of neurons in a classifier.

We first apply our system to the classic LeNet model trained on the classic MNIST handwritten digits dataset [52]. The MNIST dataset contains 60000 training images and 10000 testing images. The images are in gray scale and the size is 28*28. The network we use is similar to the original LeNet-5 [52] architecture. It contains 2 convolution layers with 5*5 kernel, followed by 3 fully connected layers. We train the network until it converged using RMSProp with momentum. We set learning rate as 1e-4 and momentum term as 0.9.

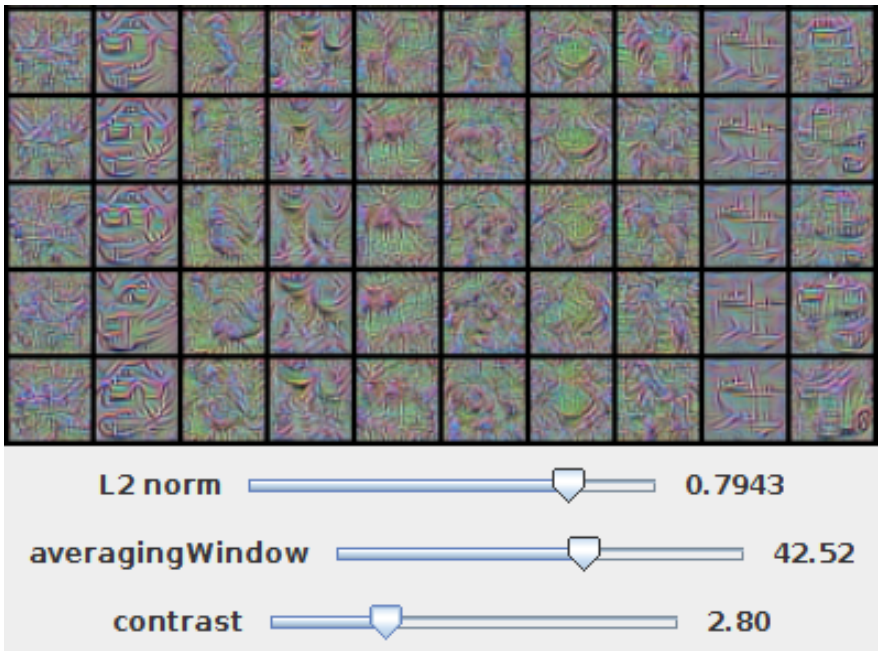


Figure 4.3: The interface of our system. Upper panel is visualizing the image samples. Each column represents one class, and each row represents different MCMC sampling procedures. Bottom panel contains the control sliders for the hyper-parameters.

Output Neurons

We begin by exploring the output-layer neurons. By running LDAM on these neurons, we should get images which correspond to human-labeled classes. Thus, these activation inputs should be easily interpretable. In this portion of the diagnosis, we are examining neurons which lie before the final softmax layer as the optimization targets. As mentioned in [73], using neurons in this layer can generate more human-readable images compared to neurons after the softmax, as the gradient from the softmax layer will be positive for the selected class and negative for other classes. This means that target neurons after the softmax will prefer image patches that are unique to a given class, possibly removing features that are relevant to other classes.

Following the steps in Algorithm 1, we start the sampling procedure from random images, and then use langevin dynamics to create proposals for all 10 classes. In our implementation, different classes will have independent sampling procedures. After the burn in period, we collect the samples and visualize them as live animation. While the burn-in period can be determined directly as discussed in [96], the computation is rather difficult. Thus, to approximate the exact calculation, we wait for activation to reach a certain threshold and become stable.

Though the input images in the training set contains only non-negative pixel values, the gradient from the network could be negative. Therefore, it is possible to generate images with

negative values that fall outside the traditional image space. Regions with negative values have negative correlation with the class prediction, and can be particularly interesting when diagnosing the performance of a classifier. In order to visualize the negative region, we normalize all the pixel values into [-128,127] and add an offset of 128 to create a standard grey scale image. Thus, pixels that are white have a high positive correlation with the target neuron, and pixels which are black have a high negative correlation. Grey pixels are uncorrelated.

Snapshots of the sampling procedures are shown in Fig 4.4. We compared different regularization techniques. Fig 4.4a shows the result without using any regularization, and as is traditional for activation maximization techniques it is very noisy.

Even though one single sample contains little information, if we compute the average of the samples for each sampling procedure, patterns start to emerge, as shown in Fig 4.4b. This is again, expected behavior because the uncorrelated pixels should average over time to a gray value, while correlated pixels should average to their correlation value. Unfortunately using sample averaging can only control for so much of the noise in an image. We can also tell that many of the pixels are saturated, since the optimization problem that we are solving is unbounded.

To solve the unbounded optimization problem, we enforce that the images correspond to the gradient of the neurons by adding the $L2$ regularization function. Even thought these samples are noisy, while sampling from this distribution we can clearly see the shape for different digit classes. By computing the sample average to reduce sample noise we can get images which clearly show human-interpretable positive and negative regions influencing the output of the classifier (Fig 4.4d).

Parameter averaging

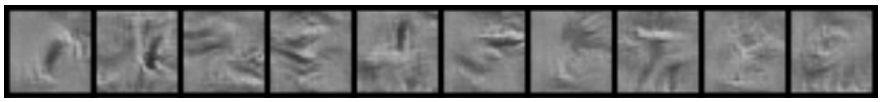
As shown in Fig 4.4, by using LDAM with L2 regularization and sample averaging, we are able to find interpretable images showing how input images can influence the classification layer. While interesting on its own, the true power of the technique can be shown by exploring how the training process of the classifier can influence how the network responds to different stimuli.

Parameter averaging techniques make up a common set of techniques used to improve classification performance. It is a well-known empirical trick and has been studied by [44] recently. Parameter averaging works by taking snapshots of the model parameters θ during the training process, and then averaging them to compute the overall model parameters. Mathematically, we can consider each θ_t of a model trained using stochastic gradient descent as an estimate of the true model parameters, thus we can use these estimates to compute the expectation of the model parameters θ by simple averaging. We notice that parameter averaging has similar testing accuracy performance to other ensemble learning methods like snapshot ensemble [43]. We have verified this by performing a quick experiment. We train the LeNet-5 model for ten epochs, and for the last five epochs we compute a moving average

(a) Without regularizations

(b) Using sample averaging

(c) Using L2 regularization (Best view in animation mode)



(d) Using L2 regularization and sample averaging

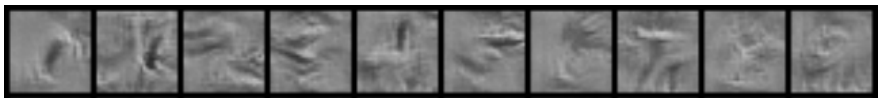
Figure 4.4: Images generated by LDAM for output neurons in LeNet trained on MNIST dataset, from left to right: 0,1,2,3,4,5,6,7,9. (Image value not clipped, an offset of 128 is added when displaying)

of the model parameters. We also computed the average prediction of those model samples in the last five epochs. We can see the performance in Table 4.1.

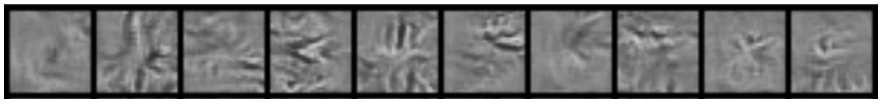
Model	Testing Accuracy
Basic model	98.7%
Parameter averaging	99.11%
Prediction averaging	99.09%

Table 4.1: Performance for different training methods on MNIST dataset

We can now use the power of LDAM with sample averaging and regularization to give us insight into why the model-average is performing better than the traditional learner. We again use LDAM with $L2$ regularization and sample averaging to compute the image samples for the 10 pre-softmax output layer neurons from a model trained using parameter averaging. The result is shown in Fig 4.5b. Compared to the base model shown in Fig 4.5a, we notice a multiple-mode effect in the images generated by the averaged model. For example, in the base model the four has only a single vertical black stripe in the center of the image, while in the averaged model there are three unique vertical black stripes. This implies that the averaged model is activated by a more diverse set of input images. We will see similar effect from model trained on CIFAR10 dataset in the next section as well.



(a) From a model trained using normal gradient descent



(b) From a model trained using parameter averaging

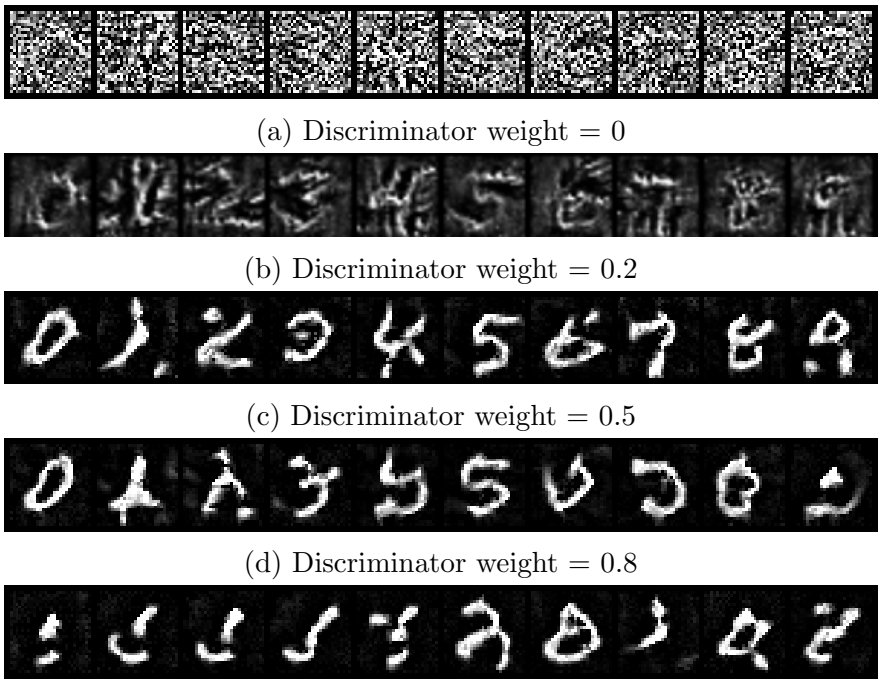
Figure 4.5: Comparing activation maximization results for LeNet models trained with different methods.

Adversarial discriminator

In the previous section, we showed that using LDAM, $L2$ regularization, and sample averaging to visualize the gradients is a powerful means of exploring the differences in classification performance between two classifiers. While $L2$ regularization alone is very useful, if we want to perform a more complex exploration of the image manifold, we would have to come up with a more complicated regularization function. Instead of hand tuning that function (which could be both difficult and tedious), we can use a discriminator to explore the image manifold as discussed in section 4.3. By giving the discriminator samples of the image manifold that we want to push our network towards (or negative samples to push it away), we can avoid inserting large amounts of tedious work and just learn from the data.

We train the discriminator following the steps in Algorithm 2. At sampling time, we provide a control for the weight of the gradient from the discriminator (the value λ in equation 4.8). If the weight is set to 0, it becomes the normal LDAM sampling process with no discriminator function. If the weight is very high, the discriminator will overpower the activation neuron, and the sampling technique will focus on sampling only from the discriminator allowed space. Thus, we can explore the boundary between the image manifold and the highly activating images by using a trade-off between discriminator loss and neuron activation loss. For the MNIST dataset, the results can be seen in Fig 4.6. In this image we can see that even a slight weight to the discriminator can quickly clean up noise in the image (as by feeding the discriminator real MNIST images, it quickly learns that those images should be sparse).

Besides cleaning up the noise, we can also use the discriminator to generate human-readable adversarial examples. As describe in [93], researchers have been using activation maximization to generate images that can fool the neural network. Usually those images look like pure noise but the network will classify them into a particular class with high confidence. However, those images didn't tell us much information about the potential drawbacks of the network. In contrast, with the guidance of the discriminator, we can generate adversarial examples that are recognizable by human, and useful for determining possible failure cases of a neural network.



(a) Discriminator weight = 0

(b) Discriminator weight = 0.2

(c) Discriminator weight = 0.5

(d) Discriminator weight = 0.8

(e) Discriminator weight = 1

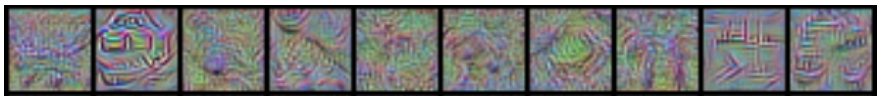
Figure 4.6: Images generated by LDAM with different discriminator weight for output neurons in LeNet trained on MNIST dataset. (Image value clipped at 0)

For example, in Fig 4.6c and Fig 4.6d, we can see symbols in weird shape but with some particular meanings. For example, a digit 1 with a horizontal stripe will still be classified as 1; a small circle on the top with two disconnected dot in the bottom will be classified as 9. These examples can give us insight into additional components or missing components that could change the eventual outcome of the network's classification. We can then use these examples to target data collection, or to evaluate other neural approaches. In this case, the reason that such adversarial examples exist is that there was a lack of training data corresponding to these examples. Because there was no training data that says these elements are not 1s or 9s, there is no reason that the network should think anything like that is amiss.

4.5 CIFAR dataset

The MNIST dataset contains only black and white images, and their background are always black. This makes the dataset simple to visualize as a starting point, however most real problems lie in much more complicated spaces. Thus, we move on to exploring models trained on the CIFAR-10 dataset [50] which contains 50000 tiny 32*32 color images. This dataset has more diverse textures and objects, and presents a much more interesting challenge

(a) Shallow architecture: 3 Convolution layers with 2 FC layer, using L2 and sample averaging



(b) Deep architecture: The VGG-16 net, using L2 and sample averaging

Figure 4.7: Comparing images generated by LDAM for different model architectures trained on CIFAR10 dataset. Each column represents a class: from left to right: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

for a visualization technique. Also, different network architectures have been proposed. Here we compare a shallow architecture with the deep VGG net [84]. The shallow architecture we used is a basic model which contains 3 convolutional layers and 2 FC layer. For deep model, we use the the VGG-16 architecture which contains 16 Convolution layers. We trained both network until they converged using RMSProp with momentum. We set learning rate as 1e-4 and momentum term as 0.9. We didn't use any data augmentation methods and the input images are not cropped. The performance of these two networks can be see in Table 4.2.

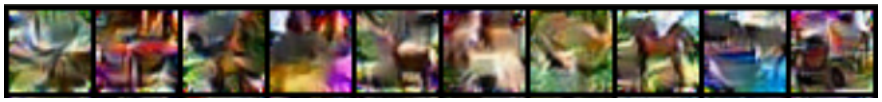
Model	Num of Conv layers	Testing Accuracy
Basic model	3	70.9%
VGG-16	16	85.3%

Table 4.2: Performance for different architectures on CIFAR dataset

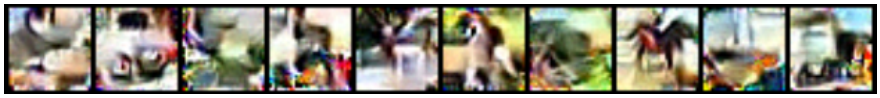
In order to diagnose and compare these two models, we use LDAM to generate image samples activating the output neurons. Again, we select neurons before the Softmax layer to get more interpretable images. We normalize the pixel values into [-128,127] and add 128 as offset when displaying. L2 regularization and sample averaging are used to improve interpretability. The results can be seen in Fig 4.7. For the shallow model (Fig 4.7a), images in the 'horse' class and 'ship' class become quite interpretable. For the deep VGG net(Fig 4.7b), the interpretability of the images are also improved. Images in 'automobile', 'deer' and 'ship' classes becomes recognizable.

Besides L2 regularization and sample averaging, we could further use the discriminator to improve image quality. Here we use a shallow discriminator with 3 convolution layers and 2 FC layers (Similar to the basic model). The basic model and the VGG net share the same discriminator architecture but the weights are trained separately. We train the discriminator according to the procedure in Algorithm 2. Also, in order to get the best quality, we found that for basic model, we should clip the pixel values between [0,255] before feed the images

(a) Discriminator weight = 0.2



(b) Discriminator weight = 0.5



(c) Discriminator weight = 0.8

Figure 4.8: Images generated by LDAM for output neurons from a network with 3 convolutional layers followed by 2 fully connected layers, under different discriminator weights. Each column represents a class: from left to right: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck (Image value clipped at 0).

into the discriminator. While for VGG-net, we could just normalize the pixel values into [128,-127].

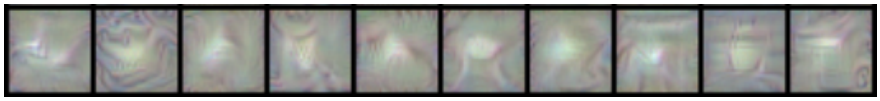
Generated samples for the basic model with different discriminator weights can be seen in Fig 4.8. We can observe that, the discriminator makes images in the 'automobile' and 'truck' classes more recognizable, comparing to the original results. Also, image samples now contains more color information. For example, images in the 'ship' class will contain regions filled with blue.

The results for VGG net, on the other hand, still contains little color information, as shown in Fig 4.9. However, the shape of object for 'bird', 'dog', 'horse' and 'truck' classes becomes more interpretable after using the discriminator. Also, if we compare the samples for these two networks carefully, we can observe that the basic model usually generate one big object at the middle, while the VGG net could generate several small objects which appear in different locations. In classes of 'car' and 'deer', such phenomenon is most obvious. For results in 'car' class from the VGG net, we can clearly observe that there is one car at the upper-left corner while there are some car components at the bottom. This demonstrates that objects activating the VGG net could have more diverse scale and could come from more diverse location. This is also aligned with the purpose to replace FC layers with convolution layers.

Again, we use LDAM to analyze the parameter averaging method. We follow the same steps described in section 4.4 to obtain a VGG-16 model trained with parameter averaging. Its testing accuracy is improved from 85.3% to 86.2%. We then use LDAM to compute the activating image samples for the 10 pre-softmax output layer neurons. The results are shown in Fig 4.10b. By comparing the results from the averaged model with the original model, we can observe that, the images in class 'bird' and class 'cat' contain multiple objects. This

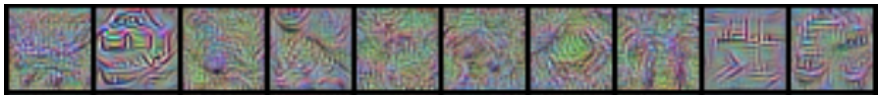
(a) Discriminator weight = 0.2

(b) Discriminator weight = 0.5

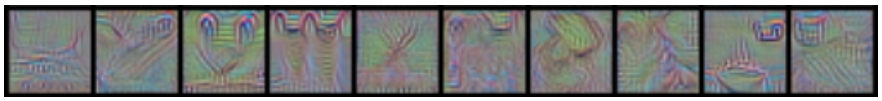


(c) Discriminator weight = 0.8

Figure 4.9: Images generated by LDAM for output neurons from the VGG-16 network, under different discriminator weights. Each column represents a class: from left to right: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.



(a) From a model trained using normal gradient descent



(b) From a model trained using parameter averaging

Figure 4.10: Comparing activation maximization results for the VGG-16 models trained with different methods.

is similar to our previous experiment using LeNet (Fig 4.5). Also, we can notice that, the image samples generated by the averaged model become sharper and cleaner. This indicates that model trained with parameter averaging can have smoother weight, same as the effect we have already seen in the sampling averaging method.

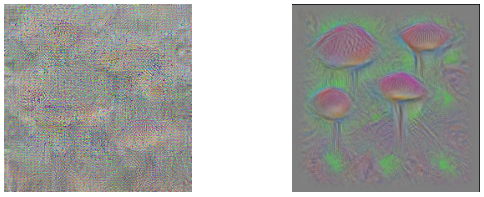
Now we could connect all the dots between ensemble learning which has been well studied, with SmoothGrad [88] and sample/parameter averaging methods. If we only obtain one solution from a non-convex optimization problem, no matter whether it is a model or a sample, it could be noisy and imperfect. However, if we introduce noise and variance in the optimization to generate a series of samples, and compute the 'average' of them appropriately, we can get better results than individual solutions. Note that, for sample averaging and parameter averaging, we are not computing the average of some arbitrary samples. Instead, the samples need to be in a chain of a monte carlo sampling procedure, or equivalently, a sequence of steps from the stochastic gradient descent [96].

4.6 ImageNet Dataset

We now move on to demonstrate the usage of our algorithm on models trained with the ImageNet dataset. The ImageNet dataset contains more than 1 million 256*256 RGB images in 1000 different classes. Many network architectures such as AlexNet [51], VGG net [84], ResNet [37] have been proposed. As bigger images allow neurons to have bigger and more diverse receptive field size, network trained with ImageNet dataset could have more meaningful and interpretable intermediate-layer neurons. Therefore in this section, we will focus on using our method to explore the relationship between intermediate-layer neurons. Such relationship could be across layers or within one layer.

Here we use the AlexNet [51] architecture as demonstration. The AlexNet contains 5 convolution layers and 3 FC layers. We train the network using RMSProp with Momentum until it converged. The learning rate is set as 1e-4.

Similar to our previous experiments, we can use LDAM to generate interpretable image samples for the output layer neurons, since those neruons contain class information. An example activating the mushroom class can be seen in Fig 4.11. However, for intermediate-layer neurons, we don't have such label information, even though some neurons can still generate images containing interpretable objects. On the other hand, there could be more than 4000 neurons in the intermediate layers. It is not realistic to examine them one by one and it is even harder to compare them.



(a) Without any regularization

(b) With L2 regularization and sample averaging

Figure 4.11: Images generated by LDAM for the output neuron of mushroom class from the AlexNet

Therefore, we use a top-down approach, to select relevant intermediate-layer neurons by using their connections with the output layer. Fig 4.12 illustrates our method. We start from the neuron representing the mushroom class in the output layer (FC8), and then we find out the most **relevant** neurons in the FC7 layer using the linear weight matrix in FC8 layer. To be specific, the FC7 layer has 4096 neurons and FC8 has 1000 neurons (each representing one class.) The linear weight matrix for FC8 layer would be a 4096 * 1000 matrix W. $W_{i,j}$ represents the linear weight between the i_{th} neuron in FC7 and the j_{th} neuron in FC8. We therefore find the top K neurons in FC7 that have the largest linear weight in the m_{th} column

of W, where m is the index of the mushroom class. For those K neurons, we use LDAM to generate samples for each of them, and then select the most interpretable ones for further exploration.

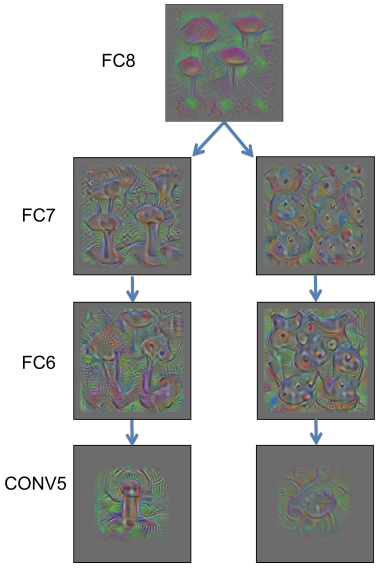


Figure 4.12: Images generated by LDAM for neurons from different layers in AlexNet. Starting from the output neuron for mushroom class in FC8 layer, we find its most relevant neurons in FC7 layer using the weight matrix.

As shown in Fig 4.12, we select two neurons in FC7 layer and use LDAM to generate activating images. One looks like the mushroom cap and the other looks like the stem. From these two neurons, we continue to use the same method to find relevant neurons in FC6, and then in Conv5. For Conv5 layer, instead of having only one output, each filter will generate a 6*6 activation map for each input image. Therefore we use the averaged weight for those 6*6 output to represent the relevance between a filter neuron in Conv5 and a neuron in FC6. Generated images for the selected neurons can be seen in Fig 4.12. Using this method, we not only find out the intermediate-layer neurons that are interpretable, but also obtain their relationship to the neurons in the next layer, all the way towards the output layer. By doing so, the class information can be propagated to intermediate layers for diagnosis purpose.

Now we have seen how to explore relationship between neurons in different layers. We can also explore relationship between neurons in the same layer by clustering them. In order to do so, we can compute the activation (or the averaging value for the activation map) for each neuron on a set of images. If we use N images for computation, each neuron will receive an activation vector with N dimension. Similar to visualizations used in [79, 82], we then use the T-SNE algorithm [61] to assign each neuron a 2-D coordinate based on their activation vector. We apply such method on the 256 neurons in Conv5 layer, and we plot their projected coordinates in Fig 4.13. In the 2-D map, it is much easier to make selection. We can select neurons within a cluster to make comparison. Some examples are shown in

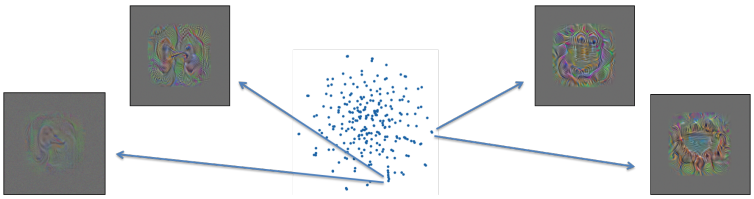


Figure 4.13: T-SNE embedding results for the 256 neurons in Conv5 layer based on their activation vector. Nearby neurons can generate similar activating images.

Fig 4.13. We can see that neurons in one cluster could respond to visually similar image patches.

4.7 Conclusion

In this chapter, we describe LDAM, a monte carlo sampling algorithm that can generate images activating selected neurons in a deep network. We propose L2 regularization, sample averaging, and an adjustable adversarial discriminator to improve the interpretability of the generated images. We demonstrate the usage of our algorithm in 3 public dataset: MNIST, CIFAR10 and ImageNet. The application of LDAM includes comparing different models (normally trained verse parameter averaging, shallow verse deep), exploring relationship between intermediate-layer neurons. Several useful insights have been found and discussed in our experiments.

Chapter 5

Designing new model structures

Our system not only supports performing exploratory model analysis for existing model structure, its flexible architecture also allows users to explore the design of new model structure based on specific problems. In this chapter, we will focus on models trained using data following power-law distribution. We design and implement a special sparse embedding matrix which can significantly save memory and computation time.

5.1 Introduction

Many real-world data follow the power-law distribution [20]. For example, natural language data follows the zipf's law that occurrence of a word is proportional to the inverse of its rank. Other human generated data on the Internet usually follow similar distributions as well. While the most frequent feature items have dominant numbers, power law also implies that data will have a long tail where each feature occurs only a few times. For models like neural network which use distributional representation [65] for discrete features, a big embedding matrix therefore must be used and its size will increase as the vocabulary size increase. Such big embedding matrix has already become bottleneck for both memory and computation in language modeling and sequence to sequence model.

For language modeling, a typical 2-layer LSTM (Long short term memory) [38] with 512 hidden units and 100K different words will have two $512 \times 100K$ embedding matrices (Input and output). In total they have more than 100 million parameters and this is much bigger than all other internal parameters whose size is $2 \times 2 \times 4 \times 512 \times 512$, around 4 million. Such gap will be even bigger if we increase either the hidden unit size or the vocabulary size.The big embedding matrix is hard to fit into the limited GPU memory and is very slow to apply gradient updates. In practice, people have come up with many ideas to address this issue, such as truncating the vocabulary, saving the input embedding matrix on CPU memory, using hierarchical softmax/negative sampling [17, 65–67] etc. However, the intrinsic dense representation remains unchanged. In addition to its inefficient, a simple argument based on number of parameters versus number of samples for the long tail features can provide

evidence what these models is very likely to over-fit.

Therefore, we propose to use power law shape matrix (as shown in Fig. 5.1) instead of dense matrix for feature embedding. The matrix shape is designed to match the distribution of the data, that frequent words/features can have more dimensions while rare words/features have only a limited dimension. We show that, such design not only reduces memory usage and improves computation efficiency, but also prevents the model from over-fitting. Most importantly, it enables us to design the shape of matrix in a flexible way. Therefore we can easily increase vocabulary size or hidden unit size based on our need.

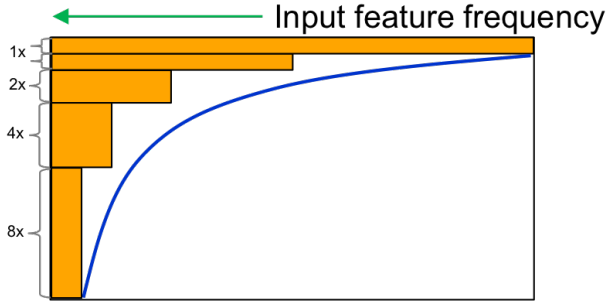


Figure 5.1: Power law shape matrix

5.2 Related work

Network compression

Song et al. [36] developed a pipeline to compress deep neural network with pruning, trained quantization and huffman coding. They achieved very high compressing rate on AlexNet [51] and VGG-16 [84], especially on those dense fully connected layers. Our work shares the idea that there exists an equivalent sparser network and our L1-regularization experiments use similar pruning strategy. However, our work differs from network compression since we train a sparser network from the beginning instead of compressing the network after the training. The pruning is only used to demonstrate that a sparser representation for embedding matrix exists.

Network regularization

Our work also has connections with network regularization methods like dropout [91] . As described in [100], applying dropout vertically in recurrent neural network can improve training. For tasks like language modeling, if the network only has 1 layer of recurrent cells, such regularization is similar to putting a mask on the input/output embedding matrix. The advantage of dropout is that the mask will be changing during the training. However, the probability to dropout each unit remains the same and the whole big embedding matrix

still needs to be stored. While in our case, we choose a fixed power law shape mask on the embedding matrix.

Training with large vocabulary size

Training deep neural networks with large vocabulary is difficult. Truncating the vocabulary size is the most common approach. But using those rare words appropriately can be quite useful in practice [60]. To train models with large vocabulary, [16] described several techniques that use parallelization and SIMD to speed up training. [17, 45] both described sampling techniques to speed up the process for updating output embedding matrix by providing a better way to compute the normalization term. [17] summarized several techniques that are useful for training language models with large vocabulary size, including Hierarchical Softmax [65, 67] and Negative Sampling [65, 66]. [17] also proposed a new method called Differentiated Softmax which assigns variable vector size for different words in the embedding matrix. However, they did not provide detailed analysis and guidance about how to choose the shape of matrix.

5.3 Sparsification of the embedding matrix

Before designing the embedding matrix, we first conduct experiments to examine whether or not these matrices can be sparsified. Even though intuitively we believe that power law shape matrix design should be used for embedding matrix modeling, it is not clear why this happens and how they should shape exactly in practice. L1-regularization based sparsification of the parameters allows a way to let the model training automatically choose the matrix weights to zero-out. This gives empirical guarantees and guidelines for our power law shape embedding matrix design.

We choose the task of language modeling using Penn Tree Bank (PTB) dataset. The training set contains 10K words and around 1M tokens. The data follows power law distribution as shown in the log-log plot Fig. 5.2.

We train the language model using the network architecture described in [100] with small settings. The implementation is based on the PTB model from Tensorflow [1]. L1-regularization for the input and/or output embedding matrix is added as an additive term to the overall training loss, which is the batch-wise word prediction loss. We compared the validation perplexity of our sparsified model to that of original setting.

The first thing we observe after applying small L1-regularization to the output embedding matrix is that the distribution of the weights could change dramatically while the prediction performance remains the same. This is illustrated in Fig. 5.3. The embedding matrix weights tend to jump back and forth around zero due to floating point issues. In this case, L1-regularization does drive a majority of the parameters in the output embedding matrix to zero without losing prediction performance. The valley in the right figure shows that

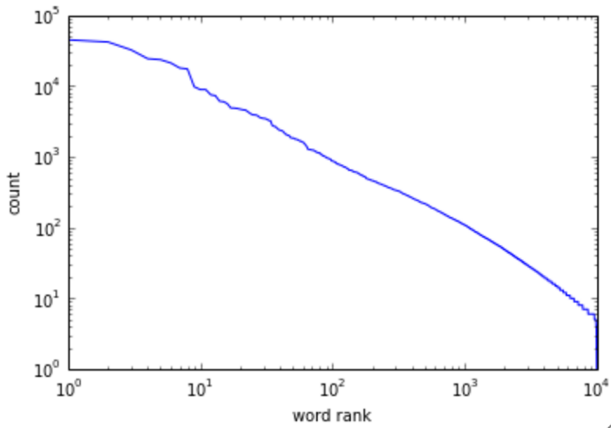
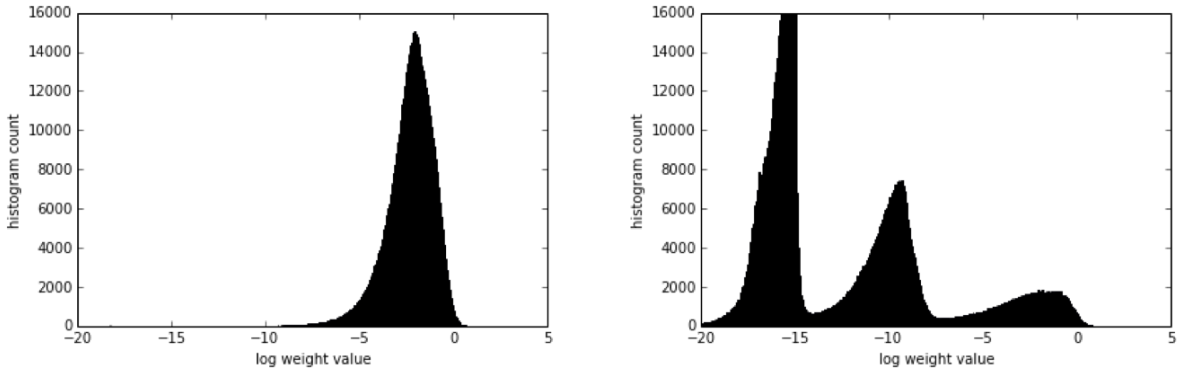


Figure 5.2: Log-Log plot of the word count for PTB dataset



(a) Original parameters distribution of the output embedding matrix, with validation perplexity 122.3

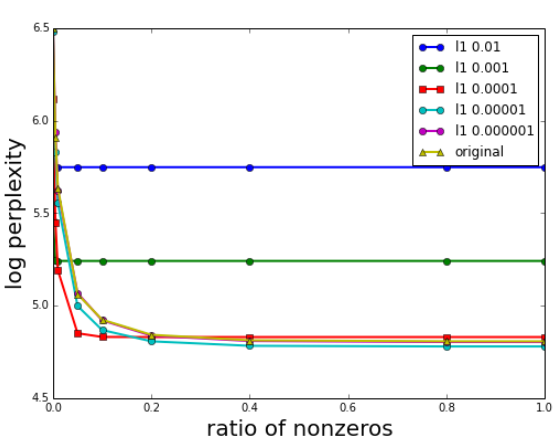
(b) Parameters distribution of the output embedding matrix when $\lambda = 1e - 4$, with validation perplexity 125.3.

Figure 5.3: Parameters distribution

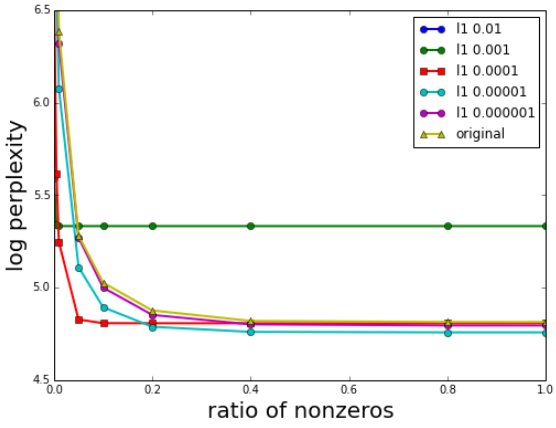
we could choose an appropriate threshold parameter to zero-out a huge amount of matrix weights that are close to the floating point limits.

After we have seen that the embedding matrices could be sparsified, the next natural question to ask is how aggressively we go without losing prediction performance. We investigate the effect of different amount of L1-regularization. In this experiment, we first use different L1-regularization parameters to sparsify our output embedding matrices, then zero-out parameters with small absolute values. The effect of the pruned models on test perplexity is shown in Fig. 5.4. As we could observe from the figure, in the original setting, zeroing out about 80% of the weights in the output embedding matrix will start deteriorating the prediction performance. However, after applying L1-regularization, we could zero-out about 95% of the weights without affecting the prediction performance. The situation is very similar if apply L1-regularization on both input and output embedding matrices. Note

that the blue line does not appear on the right figure as its validation log perplexity is worse than the y-axis limit 6.5.



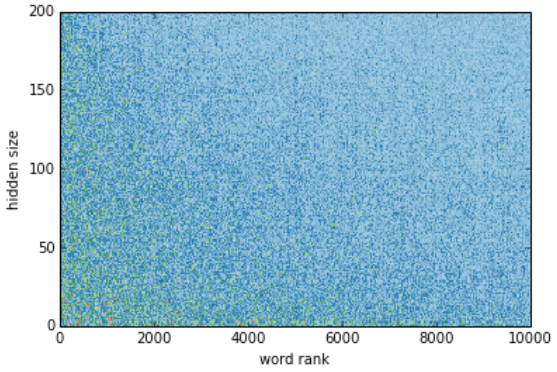
(a) Model perplexity for different L1-regularization on the output embedding matrix



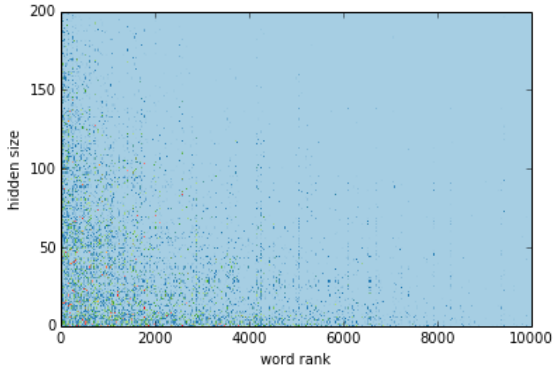
(b) Model perplexity for different L1-regularization on the both input and output embedding matrix

Figure 5.4: The effect of embedding matrix pruning for different L1-regularizations

To further investigate the effect of L1-regularization. We visualize the heatmap of the output embedding matrix without and with L1-regularization. We have reordered the rows of embedding matrix such that the rows having the most L1 norm are placed on bottom. As we can see in Fig. 5.5, the nonzero part of the L1-regularized embedding matrix tends to following the power law shape.



(a) Original output embedding matrix heatmap



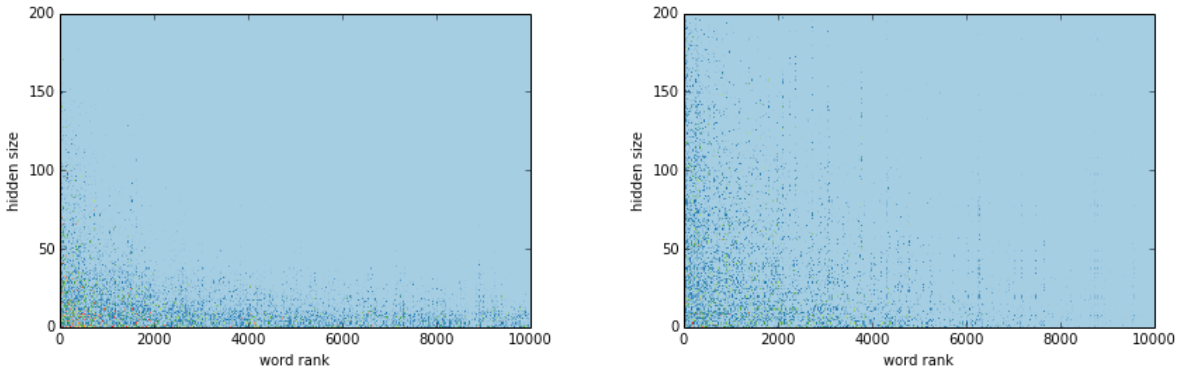
(b) Same wtih L1-regularization. $\lambda = 1e-4$.

Figure 5.5: Effect of L1-regularization on embedding matrix via heatmap

Another interesting finding about L1-regularized embedding matrix is that when the same amount of L1-regularization is put on both input and output embedding matrices, the input

embedding matrix is affected more seriously. For λ from small to large, the input embedding matrix turns sparse much earlier than the output embedding matrix. This is partly due to that output embedding matrix receives more gradient update than input embedding matrix. On the other hand, one reason why L1-regularized embedding matrix has sparse structure can also be explained by the imbalanced gradient update for frequent words and rare words. Entries with fewer and smaller gradient update will be more likely to become zero since L1-regularization will shrink parameters by a constant value in each iteration.

Further more, the modified output embedding matrix tends to ignore the tail words. In other words, the tail words representation size in the output embedding matrix is close to zero. However, for the input embedding matrix, certain dimensions are very resistant. This phenomenon is illustrated in Fig 5.6.



(a) Input heat map

(b) Output heat map

Figure 5.6: Distribution of non-zero values for embedding matrix, $\lambda = 1e - 4$.

In summary, L1-regularization experiments reveal the fact that input and output embedding matrices can be sparsified without deteriorating the prediction performance. Even though it is common practice to use dense input and output embedding matrices, we have shown that more that 90% of the matrix weights can be set to zero. The remaining matrix weights can be well approximated by a power law shape. This motivates the idea of using power law shape matrix for neural network implementation.

5.4 Power law shape matrix

Dense matrix is usually favored in most neural network models, due to its simplicity and efficiency. However, the rigid implementation also prevents people from trying new architectures, especially those with sparseness constraints. On the other hand, general sparse matrix is slow and therefore hard to use in practice. A better approach is to design special purpose sparse matrix based on the problem structure. In our case, we design the power law shape matrix for power law shape data, following the observations from previous experiments.

The design is simple and straightforward, as shown in Fig. 5.1. We ensemble a group of dense matrices with power law shape into one big matrix. Each column represents the embedding vector for one feature, and their order is arranged based on feature frequency. Features on the left is more frequent than features on the right. One strategy to compute the shape of the matrices is as follows: from top to the bottom, the width of the first matrix will be the number of features, its height is a user specified number; the second matrix will have same height and half width as the first matrix; all other matrices will have doubled height and half width comparing to the previous matrix.

We implement the new matrix operations using the BIDMat/BIDMach framework [14]. In our current implementation, the power law matrix supports multiplication like operations with normal dense matrix , as well as most point-wise operations with another power law matrix with same internal shape.

Since the new power law matrix is no more than a group of dense matrices, all operations finally break down into a series of dense matrix operations with proper offsets. This makes it quite easy to implement within the current framework and its operations can also get high computation throughput. On the other hand, the API of the new matrix remains the same as the normal dense matrix. Therefore it is very convenient to use the new matrix without modifying existing code. Additionally, it is very flexible to specify the shape of the matrix, as long as those dense matrices don't overlap to each other. Matrix shape described in [17] can also be deployed as a special case.

5.5 Experiments

Penn Tree Bank dataset

We continue to use the PTB dataset to test our power law matrix. As described previously, the PTB dataset contains 10K words and around 1M tokens. The data follows power law distribution and half of the words only occur no more than 15 times in the whole training set.

Table 5.1: Distribution of the words in PTB dataset

Rank	0∼9	10∼99	100∼999	1000∼9999
Count	284831	220678	230649	193431

Previous work [17, 100] typically use hundreds of embedding dimensions for all words. We therefore apply the power law matrix to reduce the dimension of the rare words. Two kinds of power law shape matrices are being used, as shown in Table 5.2. We train the language model using the network architecture described in [100], a 2-layer LSTM network with unrolling length of 20. We use gradient clipping method described in [76] with learning rate as 1 and max gradient norm as 5. Batch size is set to 400. Parameters are initialized

uniformly from [-0.1,0.1]. Different from [100], dropout is not used to make the comparison easier. We simply train the network until it over-fit and record the best validation results. We run all our experiments on BIDMach [14] using TITAN X GPU. The results can be seen in Table 5.3.

Table 5.2: Shape of the power law matrices for PTB dataset

Rank	0∼624	625∼1249	1250∼2499	2500∼4999	5000∼9999
Dimension	256	256	128	64	32
Dimension	512	256	128	64	32

Table 5.3: Performance for PTB dataset

Hidden unit size	Matrix type	Perplexity	Embedding parameters
200	Dense	132	4M
512	Dense	131	10.24M
256	Power law	142	1.6M
512	Power law	138	1.92M

It turns out that the PTB dataset is just too easy to over-fit without dropout, even if we reduce the number of parameters. Also, increasing the hidden unit size for frequent words is not helpful for getting better validation performance. Therefore the performance of dense matrix is slightly better than the power law shape matrix. In the next two experiments where a much larger dataset is used, we will describe how choosing a better matrix shape can lead to better performance.

Google 1 Billion dataset

The Google 1 billion language model benchmark [16] contains around 793K distinct words and 770M tokens. However, this dataset does not strictly follow power law, as shown in Table 5.4. Since this is a dataset from the Google news, its long tail contains lots of location, institution names. We therefore truncate the vocabulary size to 100K.

Table 5.4: Distribution of the words in Google 1 billion

Rank	0∼9	10∼99	100∼999	1000∼9999	10000∼99999	100K∼793K
Count	197M	165M	162M	168M	63M	11M

We also use two kinds of power law shape matrix with different dimensions. Their shape is shown in Table 5.5. We train the language model using a similar network with the one

used for the PTB dataset. A 2-layer LSTM network with unrolling length of 10. Again we use the gradient clipping method described in [76] with learning rate as 1 and max gradient norm as 5. Batch size is set to 5000. Parameters are initialized uniformly from [-0.1,0.1]. Dropout is not necessary for this dataset, since we can always get performance gain when using a larger hidden unit size. We train the models on the whole dataset for one pass and the training usually takes $1 \sim 2$ days using a single TITAN X GPU. The results can be seen in Table 5.6.

Table 5.5: Shape of the power law matrices for Google 1 billion dataset

Rank	0∼6249	6250∼12499	12500∼24999	25000∼49999	50000∼99999
Dimension	512	256	128	64	32
Dimension	1024	512	256	128	64

Table 5.6: Performance for Google 1 billion dataset

Hidden unit size	Matrix type	Perplexity	Embedding parameters	Training time
256	Dense	123	51.2M	91891s
512	Dense	106	102.4M	163262s
512	Power law	116	19.2M	68174s
1024	Power law	101	38.4M	102631s

As we can see, power law shape matrix can dramatically reduce parameter size which leads to much faster training speed. Comparing 256-dense and 1024-power_law, it is compelling that by assigning more dimensions to frequent words, we can train models with much larger hidden unit size using similar time.

Though the overall performance is improved as we use power law shape matrix with larger hidden unit size. It is not clear how words from different ranges behave since right now we assign them different dimensions. We therefore break down the results to examine what contributes to the performance gain. Table 5.7 shows the entropy for different range of words, similar from [17]. We can see performance gain after using power law shape matrix, except for the last range of words ($10K \sim 100K$). This is reasonable since the power law matrix has same or higher dimensions in all other ranges.

However, numbers in Table 5.7 do not clearly show their contributions to the final perplexity since it is the average log perplexity within each word range. Instead, to compute their real contribution, the amount of words in each range need to be considered. We therefore compute the normalized log perplexity in Table 5.8 where the sum of numbers from each range will be the log perplexity on the whole dataset. Now it is clear to make the comparison. The last range of words only contributes less than $\frac{1}{5}$ to the total log perplexity since this dataset is not strictly following power law. This implies we may even reduce the dimensions

in the long tail to match the data distribution. But if we are using dense matrix, the last range will use 90% of the memory, which is a very inefficient way to allocate resource.

Table 5.7: Entropy on validation set for different range of words

	0∼9	10∼99	100∼999	1000∼9999	10K∼99999
256 Dense	2.05	3.55	5.44	7.26	9.95
512 Power law	2.02	3.41	5.18	6.91	10.17
1024 Power law	1.93	3.33	5.04	6.70	9.83

Table 5.8: Normalized log perplexity on validation set for different range of words

	Total	0∼9	10∼99	100∼999	1000∼9999	10K∼99999
256 Dense	4.913	0.524	0.761	1.145	1.597	0.831
512 Power law	4.758	0.516	0.732	1.091	1.519	0.849
1024 Power law	4.615	0.495	0.715	1.06	1.473	0.821

Criteo challenge dataset

Criteo dataset comes from a Kaggle online competition [22]. It contains traffic logs from Criteo and the task is to develop models predicting ads click-through rate (CTR). The dataset has 37 million distinct features. And each record contains 30 $\sim$ 50 features. The distribution of the feature counts is shown in Table 5.9.

Table 5.9: Distribution of the features in Criteo challenge

Rank	0∼9	10∼99	100∼1K	1K∼10K	10K∼100K	100K∼1M	1M∼37M
Count	518M	1.398B	241M	221M	129M	41M	52M

Simple logistic regression can actually perform quite well for this problem. However, the best models in those competitions usually use ensemble methods like Gradient Boost Decision Tree [31]. An alternative to those complex feature engineering methods is to use deep neural network which also extracts high level features from the raw features.

However, using neural network requires turning each record with sparse discrete features into a dense embedding vector. Using a dense matrix as the input embedding matrix in this problem is not practical, due to memory limitation and the problem of over-fitting. Directly applying neural network with feature truncation in this problem will get even worse test performance than logistic regression.

However, we can do better than truncation by assigning those rare features at least one weight, since logistic regression is indeed turning features into embedding vector with dimension one. Then for the most frequent features, we assign each of them 200 weights. The number of weights assigned to other features will decay according to power law. Each record will then be embedded into a 200-dimension vector and passed into a 8-layer (4 linear layer and 4 activation layer) fully-connected feedforward network. We reduce the hidden unit size for higher level layers and use Sigmoid as the activation function. The best test performance we can get is shown in Table 5.10, denoted as PowerNet. Such performance is much better than the logistic regression and it actually reaches 15th place in the original Kaggle leader board.

Table 5.10: Performance of different models

Models	logistic regression	PowerNet	FFM + GBDT (First place)
Log-likelihood	-0.48396	-0.4515	-0.4446

5.6 Conclusion

Our experiments have shown that using dense embedding matrix is indeed inefficient and unnecessary. Applying L1-regularization and parameter pruning on the embedding matrix reveals a sparser solution with similar performance. We therefore design and implement the power law shape matrix, which allows user to specify dimensions of feature/words based on data distribution. Using power law shape matrix help us better utilize the resource. We can train models with larger hidden unit size using similar or even less time. This type of sparse matrix also provides us great flexibility in designing new embedding matrix shape, to solve problems that are previously constrained by the dense matrices.

Chapter 6

Discussion

As machine learning is expanding its boundary in many different fields, more and more users from different background will use machine learning algorithms to solve their own problems. Therefore designing tools to support user's workflow becomes an interesting and important research topic. As tools like Tensorflow, PyTorch provide useful functionality for users to prototype their machine learning models, this dissertation is mainly focus on exploratory model analysis which provides interactive experience when users are tuning or diagnosing their models. In contrast to other existing machine learning tools, the uniqueness of our research is that users can interactively explore the parameter space and view visual feedback in almost real-time. Such interaction is achieved by a hardware accelerated framework and a set of Monte Carlo optimization algorithms.

In this dissertation, we have demonstrated two applications for exploratory model analysis. The first is interactive customized optimization for problems with multiple objectives. We reduce the problem into a hyper-parameters tuning problem and allow users to interactively make trade-offs between different loss functions. Even though researchers usually use techniques like parameter sweeping, bayesian learning and even reinforcement learning for hyper-parameter tuning, the hyper-parameter we have is a special one, as it is used to defined the optimization objective. Therefore, we don't have a single measurable criterion to tell which hyper-parameter is better, which makes those automatic tuning techniques unsuitable in our case. This is also the reason why a human expert must explore different potential choices and figure out the criterion based on the context of the problem.

The other application is diagnosing deep neural networks. We extend the activation maximization algorithm to generate interpretable image samples that could activate the selected neurons. Our method visualizes the abstract neuron behavior as image samples so that users can compare different models or different training algorithms. User can also adjust hyper-parameters to interactively explore the image manifold and make trade-offs between local interpretability (pixel semantics) and global interpretability (shape). We also use our tool to present an empirical justification for the parameter averaging training method.

Looking forward to the future, the machine learning and the deep learning community is developing rapidly. On one hand, there are lots of research being done on automatic model

generation (autoML). Researches having been trying to automatically design neuron units, choose network structures, tune hyper-parameters etc. But on the other hand, human-in-the-loop is also becoming a wide spread research paradigm in many different fields such as machine learning and database systems. As most of the work are still done by human, improving human efficiency rather than entirely replacing human labor is also very valuable. Projects like Jupyter notebook already reshaped the way how people write python code and running data science experiments. We should expect more revolutionary tools for machine learning in the near future.

Also, interdisciplinary research becomes more important than ever before. Designing machine learning tools now requires making design considerations from both algorithm requirements (AI), system efficiency (System) and usability concern (HCI). Not to mention there are tons of ideas that we could borrow from the programming language community, database community, and visualization community. It would be more than exciting to imagine what would happen in the next few years, that everyone should be able to easily prototype and refine their machine learning models.

In this dissertation, we discussed some preliminary work on exploratory model analysis, and we wish these could inspire later research on designing better machine learning tools.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vigas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[4] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 337–346. ACM, 2015.

[5] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

[6] D. Andrzejewski, X. Zhu, M. Craven, and B. Recht. A framework for incorporating general domain knowledge into latent dirichlet allocation using first-order logic. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, p. 1171, 2011.

[7] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pp. 437–478. Springer, 2012.

[8] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[9] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 06 2010. Oral Presentation.

[10] A. Bilal, A. Jourabloo, M. Ye, X. Liu, and L. Ren. Do convolutional neural networks learn class hierarchy? *IEEE transactions on visualization and computer graphics*, 24(1):152–162, 2018.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[12] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[13] J. Canny and H. Zhao. Bidmach: Large-scale learning with zero memory allocation. In *BigLearn workshop, NIPS*, p. 117, 2013.

[14] J. Canny and H. Zhao. Big data analytics with small footprint: Squaring the cloud. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 95–103. ACM, 2013.

[15] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-graber, and D. M. Blei. Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems*, pp. 288–296, 2009.

[16] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[17] W. Chen, D. Grangier, and M. Auli. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*, 2015.

[18] J. Chuang, C. D. Manning, and J. Heer. Termite: Visualization techniques for assessing textual topic models. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 74–77. ACM, 2012.

[19] S. Chung, C. Park, S. Suh, K. Kang, J. Choo, and B. C. Kwon. Re-vacnn: Steering convolutional neural network via real-time visual analytics. In *Future of Interactive Learning Machines Workshop at the 30th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.

[20] A. Clauset, C. R. Shalizi, and M. E. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[21] A. Coates and A. Y. Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pp. 561–580. Springer, 2012.

[22] Criteo. Criteo display advertising challenge https://www.kaggle.com/c/criteo-display-ad-challenge.

[23] C. Daniel, J. Taylor, and S. Nowozin. Learning step size controllers for robust neural network training. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

[25] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[26] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[27] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[28] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[29] J. A. Fails and D. R. Olsen Jr. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pp. 39–45. ACM, 2003.

[30] Y. Feng, D. Wang, and Q. Liu. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

[31] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

[32] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[34] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[35] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio. Noisy activation functions. In M. F. Balcan and K. Q. Weinberger, eds., *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48 of *Proceedings of Machine Learning Research*, pp. 3059–3068. PMLR, 20–22 Jun 2016.

[36] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[38] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[39] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pp. 856–864, 2010.

[40] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *arXiv preprint arXiv:1801.06889*, 2018.

[41] M. S. Hossain, P. K. R. Ojili, C. Grimm, R. Mller, L. T. Watson, and N. Ramakrishnan. Scatter/gather clustering: Flexibly incorporating user feedback to steer clustering results. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2829–2838, 2012.

[42] Y. Hu, J. Boyd-Graber, and B. Satinoff. Interactive topic modeling. In *Association for Computational Linguistics*, 2011.

[43] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[44] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[45] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[46] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[47] B. Jiang and J. Canny. Interactive machine learning via a gpu-accelerated toolkit. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pp. 535–546. ACM, 2017.

[48] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2018.

[49] A. Kapoor, B. Lee, D. S. Tan, and E. Horvitz. Performance and preferences: Interactive refinement of machine learning procedures. In *AAAI*. Citeseer, 2012.

[50] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[53] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pp. 556–562, 2001.

[54] H. Lee, J. Kihm, J. Choo, J. Stasko, and H. Park. ivisclustering: An interactive visual document clustering via topic modeling. In *Computer Graphics Forum*, vol. 31, pp. 1155–1164. Wiley Online Library, 2012.

[55] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. *arXiv preprint arXiv:1603.06560*, 2016.

[56] M. Lichman. UCI machine learning repository, 2013.

[57] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE transactions on visualization and computer graphics*, 24(1):77–87, 2018.

[58] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2017.

[59] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[60] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

[61] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[62] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[63] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

[64] Q. Mei, D. Cai, D. Zhang, and C. Zhai. Topic modeling with network regularization. In *Proceedings of the 17th international conference on World Wide Web*, pp. 101–110. ACM, 2008.

[65] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[66] A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

[67] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, vol. 5, pp. 246–252. Citeseer, 2005.

[68] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[69] D. Newman, E. V. Bonilla, and W. Buntine. Improving topic coherence with regularized topic models. In *Advances in neural information processing systems*, pp. 496–504, 2011.

[70] D. Newman, J. H. Lau, K. Grieser, and T. Baldwin. Automatic evaluation of topic coherence. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 100–108. Association for Computational Linguistics, 2010.

[71] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pp. 3387–3395, 2016.

[72] A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*, 2016.

[73] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization. doi: 10.23915/distill.00007

[74] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks. doi: 10.23915/distill.00010

[75] PaddlePaddle and E. teams. A platform to visualize the deep learning process. http://visualdl.paddlepaddle.org.

[76] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1310–1318, 2013.

[77] K. Patel, S. M. Drucker, J. Fogarty, A. Kapoor, and D. S. Tan. Using multiple models to understand data. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, p. 1723. Citeseer, 2011.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[79] N. Pezzotti, T. Hllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):98–108, 2018.

[80] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 569–577. ACM, 2008.

[81] H. Raghavan, O. Madani, and R. Jones. Interactive feature selection. In *IJCAI*, vol. 5, pp. 841–846, 2005.

[82] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):101–110, 2017.

[83] J. Seo and B. Shneiderman. Interactively exploring hierarchical clustering results. In *The Craft of Information Visualization*, pp. 334–340. Elsevier, 2003.

[84] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[85] D. Smilkov, S. Carter, D. Sculley, F. B. Viegas, and M. Wattenberg. Direct manipulation visualization of deep networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[86] D. Smilkov, S. Carter, D. Sculley, F. B. Vigas, and M. Wattenberg. Direct-manipulation visualization of deep networks. *Workshop on Visualization for Deep Learning of the 33rd International Conference on Machine Learning*, 2016.

[87] D. Smilkov, S. Carter, D. Sculley, F. B. Vigas, and M. Wattenberg. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788*, 2017.

[88] D. Smilkov, N. Thorat, B. Kim, F. Vigas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[89] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[90] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[91] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[92] Y. Sun, E. Lank, and M. Terry. Label-and-learn: Visualizing the likelihood of machine learning classifier's success during data labeling. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pp. 523–534. ACM, 2017.

[93] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[94] J. Talbot, B. Lee, A. Kapoor, and D. Tan. Ensemblematrix: Interactive visualization to support machine learning with multiple classifiers. In *ACM Human Factors in Computing Systems (CHI)*, 2009.

[95] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1105–1112. ACM, 2009.

[96] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 681–688, 2011.

[97] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[98] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Man, D. Fritz, D. Krishnan, F. B. Vigas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12, 2018.

[99] Y. Yang, S. Pan, Y. Song, J. Lu, and M. Topkara. User-directed non-disruptive topic model update for effective exploration of dynamic content. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, pp. 158–168. ACM, 2015.

[100] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[101] H. Zeng, H. Haleem, X. Plantaz, N. Cao, and H. Qu. Cnncomparator: Comparative analytics of convolutional neural networks. *arXiv preprint arXiv:1710.05285*, 2017.

[102] H. Zhao, B. Jiang, J. F. Canny, and B. Jaros. Same but different: Fast and high quality gibbs parameter estimation. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1495–1502. ACM, 2015.

[103] S. Zhong. Efficient online spherical k-means clustering. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 5, pp. 3180–3185. IEEE, 2005.

[104] J.-Y. Zhu, P. Krhenbhl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pp. 597–613. Springer, 2016.

[105] J.-Y. Zhu, Y. J. Lee, and A. A. Efros. Averageexplorer: Interactive exploration and alignment of visual data collections. *ACM Transactions on Graphics (TOG)*, 33(4):160, 2014.