Interactive machine learning via a GPU-accelerated Toolkit

Biye Jiang
Computer Science Division
UC Berkeley
Berkeley, CA 94720
bjiang@cs.berkeley.edu

John Canny
Computer Science Division
UC Berkeley
Berkeley, CA 94720
jfc@cs.berkeley.edu

ABSTRACT

Machine learning is growing in importance in industry, sciences, and many other fields. In many and perhaps most of these applications, users need to trade off competing goals. Machine learning, however, has evolved around the optimization of a single, usually narrowly-defined criterion. In most cases, an expert makes (or should be making) trade-offs between these criteria which requires high-level (human) intelligence. With *interactive customization and optimization* the expert can incorporate secondary criteria into the model-generation process in an interactive way.

In this paper we develop the techniques to perform customized and interactive model optimization, and demonstrate the approach on several examples. The keys to our approach are (i) a machine learning architecture which is modular and supports primary and secondary loss functions, while users can directly manipulate its parameters during training (ii) high-performance training so that non-trivial models can be trained in real-time (using roofline design and GPU hardware), and (iii) highly-interactive visualization tools that support dynamic creation of visualizations and controls to match various optimization criteria.

Author Keywords
Machine learning, hyper-parameters tuning, interactive, multiple objective optimization, GPU

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation (e.g. HCI): User Interfaces

INTRODUCTION

Machine Learning is now at the center of data analysis in many fields across the sciences, business, health care and other realms. In many areas, users really want solutions with good performance in multiple dimensions while most ML methods optimize a single narrow criterion. 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.

IUI 2017, March 13–16, 2017, Limassol, Cyprus.
Copyright © 2017 ACM ISBN 978-1-4503-4348-0/17/03 ...$15.00.
http://dx.doi.org/10.1145/3025171.3025172

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 [3, 34, 26, 2, 24, 10] 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 [33] 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 minibatch 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.

RELATED WORK

Interactive model refinement

In the context of supervised learning, Fail and Olsen [15] 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) [31]. The Prospect system [30] automates model selection and tuning allowing users to focus on data manipulation. EnsembleMatrix [35] uses custom visualizations to aid in the design of ensemble classifiers. Amershi et al. [1] 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 [20] 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 [33] 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 [14, 32, 40] is a natural and powerful approach. In [14, 32], the authors used visualizations to rapidly explore the results of a clustering algorithm, and these approaches have become important tools in computational biology [14]. AverageExplorer [40] 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 [37, 18, 8]. Since the latent topics extracted by the algorithm are not always semantically meaningful [29, 7], different constrained topic models [27, 28] have been developed. Systems like [37, 18] 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 [3], gradient based methods [26, 2], reinforcement learning [10] or bayesian optimization [34]. 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 [24] 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.

SYSTEM DESIGN

BIDMach: high-performance, customized machine learning toolkit

The first key to interactive and customized machine learning is an architecture which supports it. BIDMach [6] 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. 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 [13] 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.

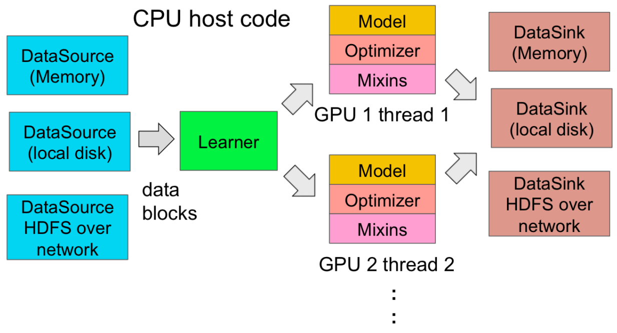


Figure 1: BIDMach's Architecture

BIDMach uses *roofline design* [36] 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 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. We will see this in the examples later.

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 [27, 28, 37].

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 [29, 7]. 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:

$$arg\min_x f_0(x,d) + \sum_i \lambda_i * f_i(x) \tag{1}$$

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 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.

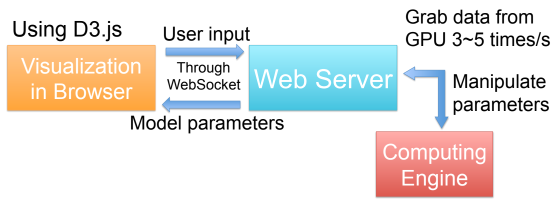


Figure 2: Visualization Architecture

In the client side, we implement a web based interface which uses D3.js[5] 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.

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 3, 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.

Visualizing the model

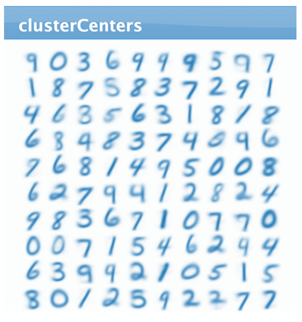
As mentioned earlier, machine learning algorithms can be formalized as finding the best model parameters x given an objective function $f(x, d)$. For a variety of models, directly visualizing the model parameters provides a nice summarization for the current training and users can gain a general understanding 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 4a, or learned image dictionary using dictionary learning algorithms like NMF[23], as in Fig 4b.

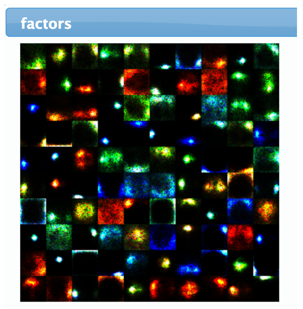
For more general matrix data, a simple direct visualization of element weights can usually work well. This was the approach taken in the Termite system [8] and we use it for our topic model, which will be described in details in the use cases section. As shown in Fig 4c and 4d, the area of each pink 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[8]. 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.

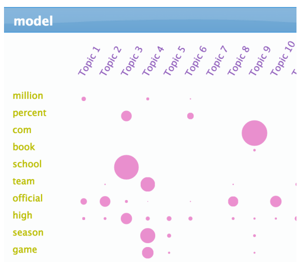
However, as we are displaying in real-time an evolving model, having a simple, consistent metric for ordering can actually help users interpret the visualization and avoid confusion. We therefore only use the total weights across all topics to rank the words. This approach will show more redundant words 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 4d. It also helps the users to compare different topics.



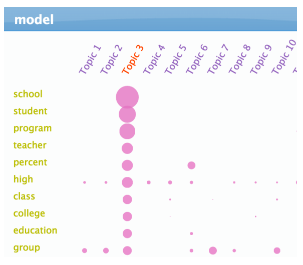
(a) MNIST clustering centers



(b) Dictionary learned by NMF on CIFAR dataset

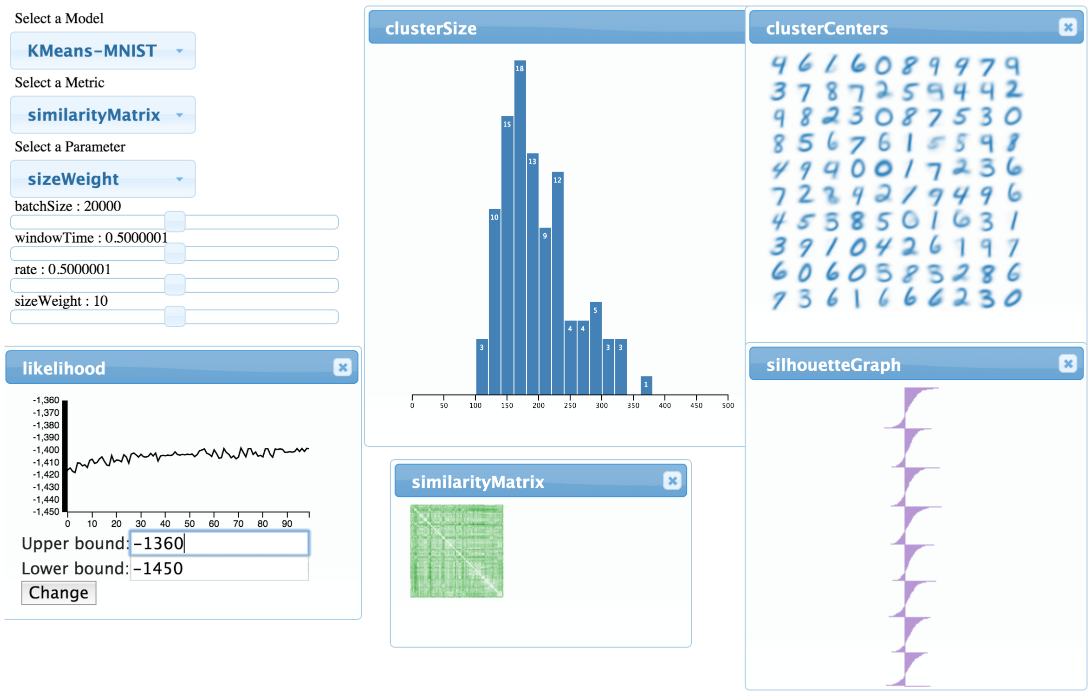


(c) Ranking words by sum of weights



(d) Ranking words using weight from topic 3

Figure 4: Model visualization


Select a Model
KMeans–MNIST
Select a Metric
similarityMatrix
Select a Parameter
sizeWeight
batchSize : 20000
windowTime : 0.5000001
rate : 0.5000001
sizeWeight : 10

Upper bound: -1360
Lower bound: -1450

Figure 3: Dashboard for KMeans using MNIST dataset

Continuous visualization of model quality

As described in Equation 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 5. 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 5.

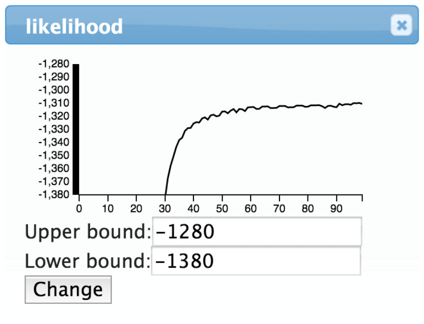

Upper bound: -1280
Lower bound: -1380

Figure 5: Continuous visualization of the likelihood function

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.

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 6a shows the size distribution for the clusters of digits on the MNIST dataset.

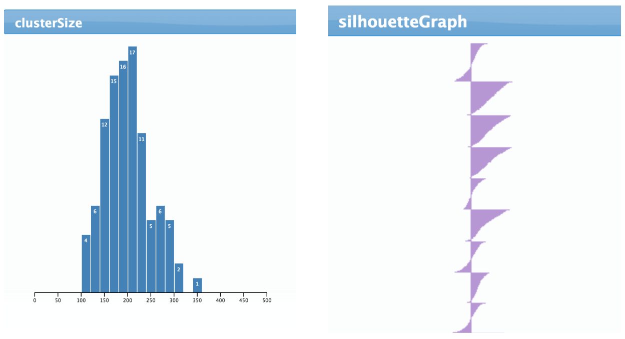
Silhouette graph

Another useful metric is the widely used silhouette graph (Fig 6b) for evaluating clustering results. The silhouette score is calculated for each data point x_j as:

$$s_j = \frac{b_j - a_j}{max(a_j, b_j)} \tag{2}$$

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.



(a) Distribution of Cluster Size

(b) Silhouette graph

Figure 6: Performance indictors for clustering

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.

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 [9]. 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 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:

$$\arg\min_{s} \sum_{i} \Big(\sum_{j \in s_i} ||x_j - \mu_i||_2^2 + \lambda |s_i|^2 \Big) \qquad (3)$$

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:

$$id(x) = \arg\min_{i} ||\mu_i - x||_2^2 + \lambda |s_i| \qquad (4)$$

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 [39] 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:

$$\begin{aligned} average_i &= \frac{\sum_j x_j * \mathbb{1}(id(x_j) = i)}{\sum_j \mathbb{1}(id(x_j) = i)} \\ \mu_i &= (1 - \eta) * \mu_i + \eta * average_i \\ size_i &= (1 - \alpha) * size_i + \alpha * \sum_j \mathbb{1}(id(x_j) = i) \end{aligned} \qquad (5)$$

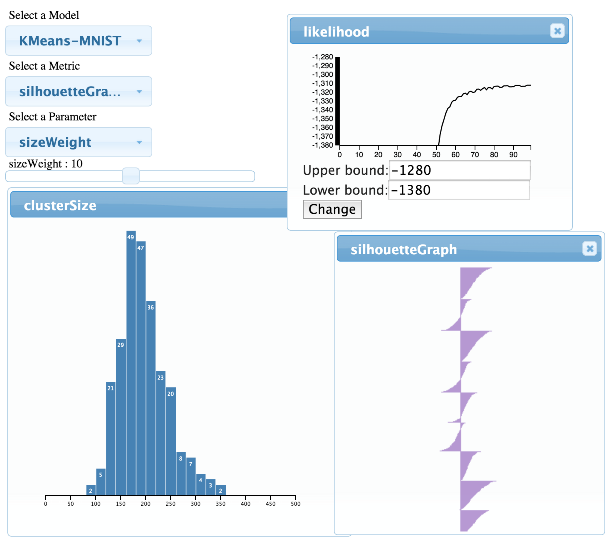
Normally η and α are set to $0.1 \sim 0.2$.

Experiment on MNIST dataset

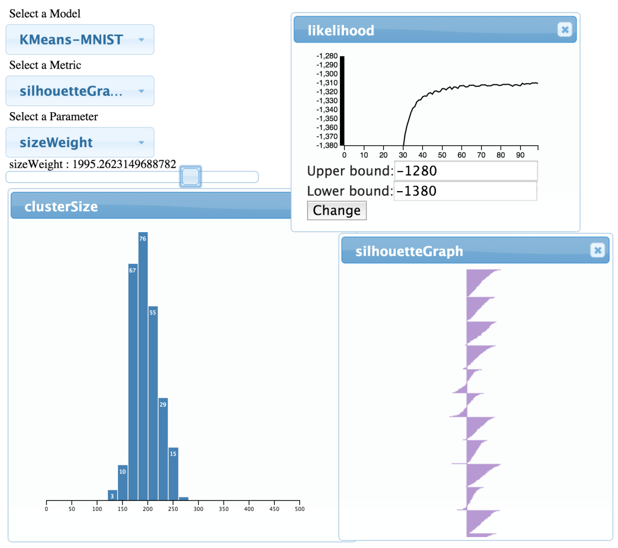
We ran an experiment on the MNIST dataset [22], which contains 8 million $28 \times 28 \times 3$ RGB images of hand written digits. We train the model using NVIDIA Titan X GPU, which could process roughly 500MB raw data (16.7k images) per second. As we set the mini-batch size to be 50000, every second the system could perform 3 batch updates, which is enough for real-time visualization.

In our dashboard shown in Fig 3, 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.

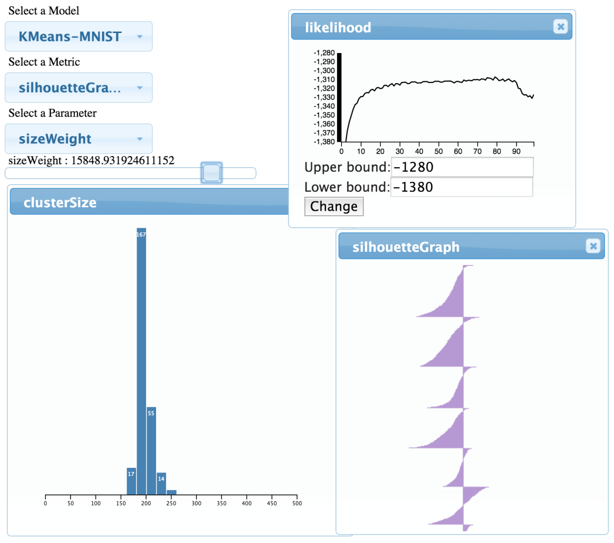
The parameter that we choose to tune is the weight for size balance λ in eq(4), which we refer as $sizeWeight$ in the interface. Initially, we set the number of clusters K as 256 and we assigned a small value to $sizeWeight$, so that the algorithm will behave as the original KMeans algorithm. As shown in Fig 7a, the likelihood is gradually improving and


Select a Model: KMeans-MNIST
Select a Metric: silhouetteGra...
Select a Parameter: sizeWeight
sizeWeight : 10
Upper bound: -1280
Lower bound: -1380

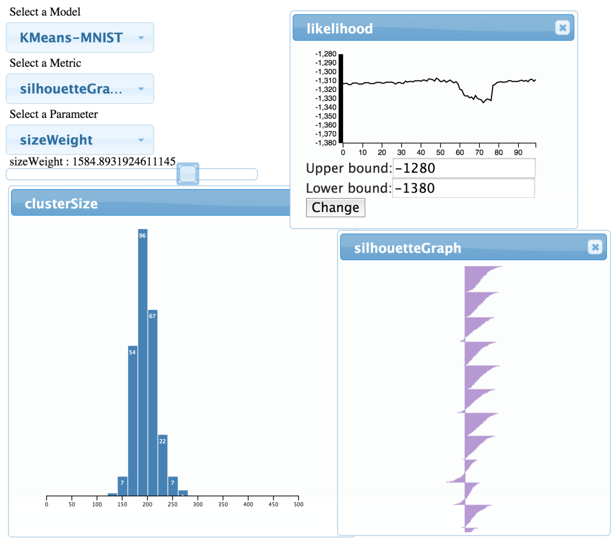
(a) original KMeans


Select a Model: KMeans-MNIST
Select a Metric: silhouetteGra...
Select a Parameter: sizeWeight
sizeWeight : 1995.2623149688782
Upper bound: -1280
Lower bound: -1380

(b) Cluster size concentrated


Select a Model: KMeans-MNIST
Select a Metric: silhouetteGra...
Select a Parameter: sizeWeight
sizeWeight : 15848.931924611152
Upper bound: -1280
Lower bound: -1380

(c) Begin to loss performance


Select a Model: KMeans-MNIST
Select a Metric: silhouetteGra...
Select a Parameter: sizeWeight
sizeWeight : 1584.8931924611145
Upper bound: -1280
Lower bound: -1380

(d) Recover the model

Figure 7: Interactive tuning for KMeans

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 7b. 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 7c). 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 7d. This example illustrates the trade-offs that can be made, and the speed of recognizing poor parameter choices.

Experiment on ImageNet dataset

ImageNet[11] is a widely used image-classification dataset which contains millions of labeled images. The recent success of applying convolutional neural network[21] 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[12] and neural style transfer [16] 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[19] 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:

$$\begin{aligned} d(x, y) = & s_1 * \alpha_1 * ||x_{pool1} - y_{pool1}||^2 \\ & + s_2 * \alpha_2 * ||x_{pool2} - y_{pool2}||^2 \\ & + s_3 * \alpha_3 * ||x_{pool5} - y_{pool5}||^2 \end{aligned} \quad (6)$$

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 8. 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.

By tuning the weights for the distance metric, we can get different clustering results as shown in Fig 9 and Fig 10. 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.

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 11. On the other hand, changing the distance metric during training can help the algorithm escape from local optimal.

Topic Model

We then apply our system to topic modeling using Latent dirichlet allocation(LDA)[4], one of the most widely used topic model algorithm. Topic model has many applications since it can find compact representation of a large document

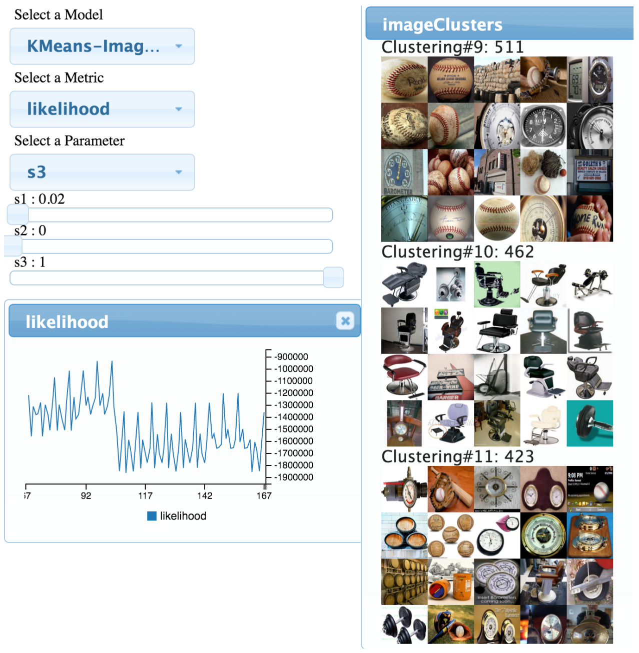

Select a Model: KMeans-Imag...
Select a Metric: likelihood
Select a Parameter: s3
s1 : 0.02
s2 : 0
s3 : 1
Clustering#9: 511
Clustering#10: 462
Clustering#11: 423

Figure 8: Clustering on ImageNet data

set. However, as discussed in [7], 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 [29] 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:

$$P_X(z_{d,i} = k | z_{-d,i}, \alpha, \beta, x_{d,i} = w) \sim \theta_{d,k} * \varphi_{k,w} \quad (7)$$

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

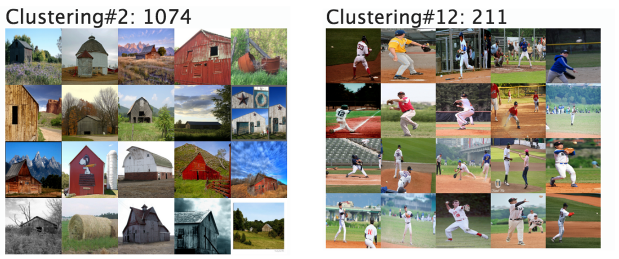


Figure 9: Clustering using pool5

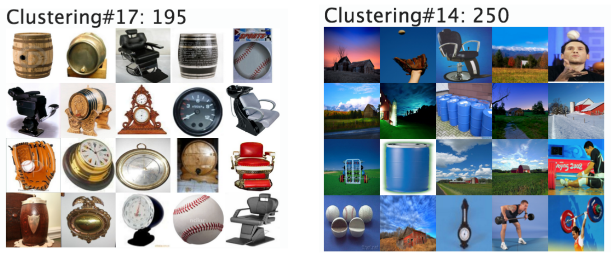
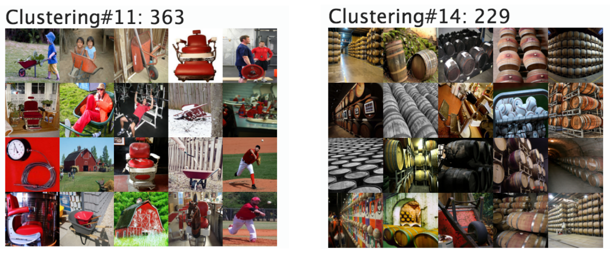


Figure 10: Clustering using pool1



(a) Images that have something in red (b) Images with a lot of barrels

Figure 11: Transit from pool1 to pool5

use SAME sampling [38] 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 [17] 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:

$$f_1(\varphi) = |\varphi| \tag{8}$$

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

$$f_2(\varphi) = \sum_{i \neq j} (\sum_w \varphi_{i,w} \varphi_{j,w}) \tag{9}$$

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:

$$\begin{aligned} g_1(\varphi_{k,w}) &= sign(\varphi_{k,w}) \\ g_2(\varphi_{k,w}) &= (\sum_i \varphi_{i,w}) - \varphi_{k,w} \end{aligned} \tag{10}$$

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

$$\varphi_{t+1} = (1-\alpha)\varphi_t + \alpha\varphi' - \lambda_1 g_1(\varphi_t) - \lambda_2 g_2(\varphi_t) \tag{11}$$

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 [25], 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 12. 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 13, 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 14, we can see that the likelihood returns to a normal value while the sparsity is maintained.

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 15, the $cosineSim$ 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

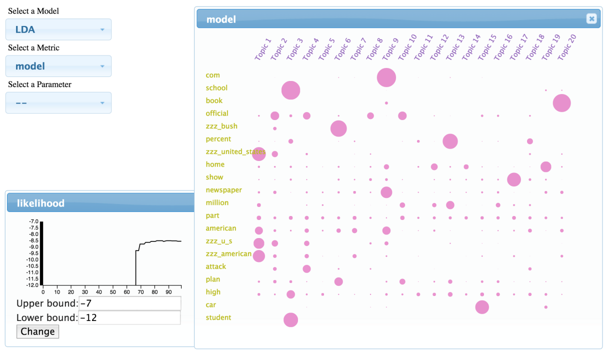

Select a Model: LDA
Select a Metric: model
Select a Parameter: --
Upper bound: -7
Lower bound: -12
Change

Figure 12: Overlapping topics, K=32

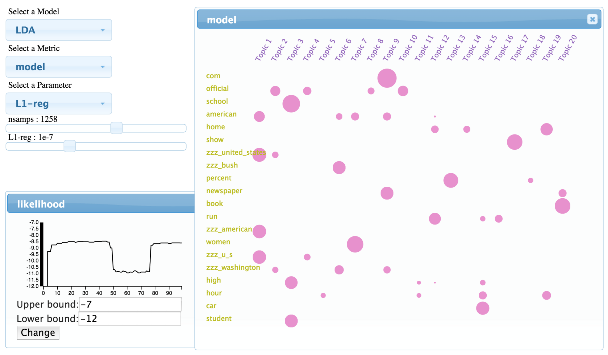

Select a Model: LDA
Select a Metric: model
Select a Parameter: L1-reg
nsamps : 1258
L1-reg : 1e-7
Upper bound: -7
Lower bound: -12
Change

Figure 14: Likelihood back to normal, sparsity preserved

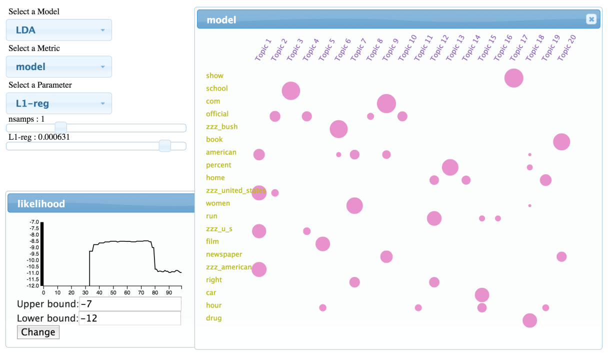

Select a Model: LDA
Select a Metric: model
Select a Parameter: L1-reg
nsamps : 1
L1-reg : 0.000631
Upper bound: -7
Lower bound: -12
Change

Figure 13: High temperature with high L1-reg

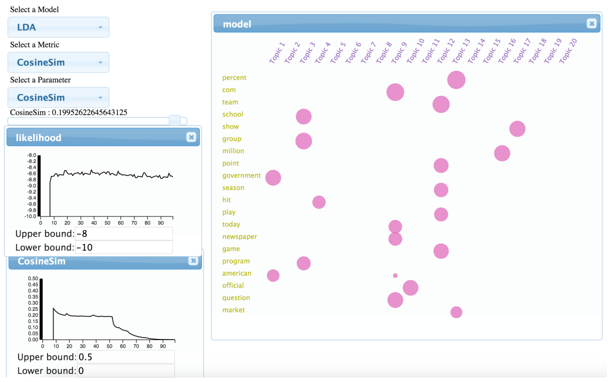

Select a Model: LDA
Select a Metric: CosineSim
Select a Parameter: CosineSim
CosineSim : 0.19952622645643125
Upper bound: -8
Lower bound: -10
Upper bound: 0.5
Lower bound: 0

Figure 15: Cosine similarity decreased

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: $(\sum_i \varphi_{i,w}) - \varphi_{k,w}$. 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.

CONCLUSION & FUTURE WORK

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.

More mixins functions can be tried in the future. Moreover, our system is not limited to unsupervised learning algorithms. Some concrete examples of competing goals in supervised learning include computational marketing where the primary goal is to maximize revenue, but where secondary goals include user satisfaction, advertiser satisfaction, and budget constraints.

On the other hand, supporting pause and replay in the future would be really helpful for users to navigate different versions of model.

ACKNOWLEDGMENT

We thanks Daniel Seita, Junyan Zhu and Philippe Laban for their valuable comments for the paper.

REFERENCES

1. Amershi, S., Cakmak, M., Knox, W. B., and Kulesza, T. Power to the people: The role of humans in interactive machine learning. *AI Magazine 35*, 4 (2014), 105–120.
2. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474* (2016).
3. Bergstra, J., and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*, Feb (2012), 281–305.
4. Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *the Journal of machine Learning research 3* (2003), 993–1022.
5. Bostock, M., Ogievetsky, V., and Heer, J. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on 17*, 12 (2011), 2301–2309.
6. Canny, J., and Zhao, H. Big data analytics with small footprint: Squaring the cloud. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM (2013), 95–103.
7. Chang, J., Gerrish, S., Wang, C., Boyd-graber, J. L., and Blei, D. M. Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems* (2009), 288–296.
8. Chuang, J., Manning, C. D., and Heer, J. Termite: Visualization techniques for assessing textual topic models. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, ACM (2012), 74–77.
9. Coates, A., and Ng, A. Y. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*. Springer, 2012, 561–580.
10. Daniel, C., Taylor, J., and Nowozin, S. Learning step size controllers for robust neural network training. In *Thirtieth AAAI Conference on Artificial Intelligence* (2016).
11. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE (2009), 248–255.
12. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531* (2013).
13. Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research 12* (2011), 2121–2159.
14. Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences 95*, 25 (1998), 14863–14868.
15. Fails, J. A., and Olsen Jr, D. R. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, ACM (2003), 39–45.
16. Gatys, L. A., Ecker, A. S., and Bethge, M. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), 2414–2423.
17. Hoffman, M., Bach, F. R., and Blei, D. M. Online learning for latent dirichlet allocation. In *advances in neural information processing systems* (2010), 856–864.
18. Hu, Y., Boyd-Graber, J., and Satinoff, B. Interactive topic modeling. In *Association for Computational Linguistics* (2011).
19. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014).
20. Kapoor, A., Lee, B., Tan, D. S., and Horvitz, E. Performance and preferences: Interactive refinement of machine learning procedures. In *AAAI*, Citeseer (2012).
21. Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), 1097–1105.
22. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.
23. Lee, D. D., and Seung, H. S. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems* (2001), 556–562.
24. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Efficient hyperparameter optimization and infinitely many armed bandits. *arXiv preprint arXiv:1603.06560* (2016).
25. Lichman, M. UCI machine learning repository, 2013.
26. Maclaurin, D., Duvenaud, D., and Adams, R. P. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning* (2015).
27. Mei, Q., Cai, D., Zhang, D., and Zhai, C. Topic modeling with network regularization. In *Proceedings of the 17th international conference on World Wide Web*, ACM (2008), 101–110.
28. Newman, D., Bonilla, E. V., and Buntine, W. Improving topic coherence with regularized topic models. In *Advances in neural information processing systems* (2011), 496–504.

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

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

31. Raghavan, H., Madani, O., and Jones, R. Interactive feature selection. In *IJCAI*, vol. 5 (2005), 841–846.

32. Seo, J., and Shneiderman, B. Interactively exploring hierarchical clustering results [gene identification]. *Computer 35*, 7 (2002), 80–86.

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

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

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

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

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

38. Zhao, H., Jiang, B., Canny, J. F., and Jaros, B. 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*, ACM (2015), 1495–1502.

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

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