Analyzing the Training Processes of Deep Generative Models

Mengchen Liu, Jiaxin Shi, Kelei Cao, Jun Zhu, Shixia Liu

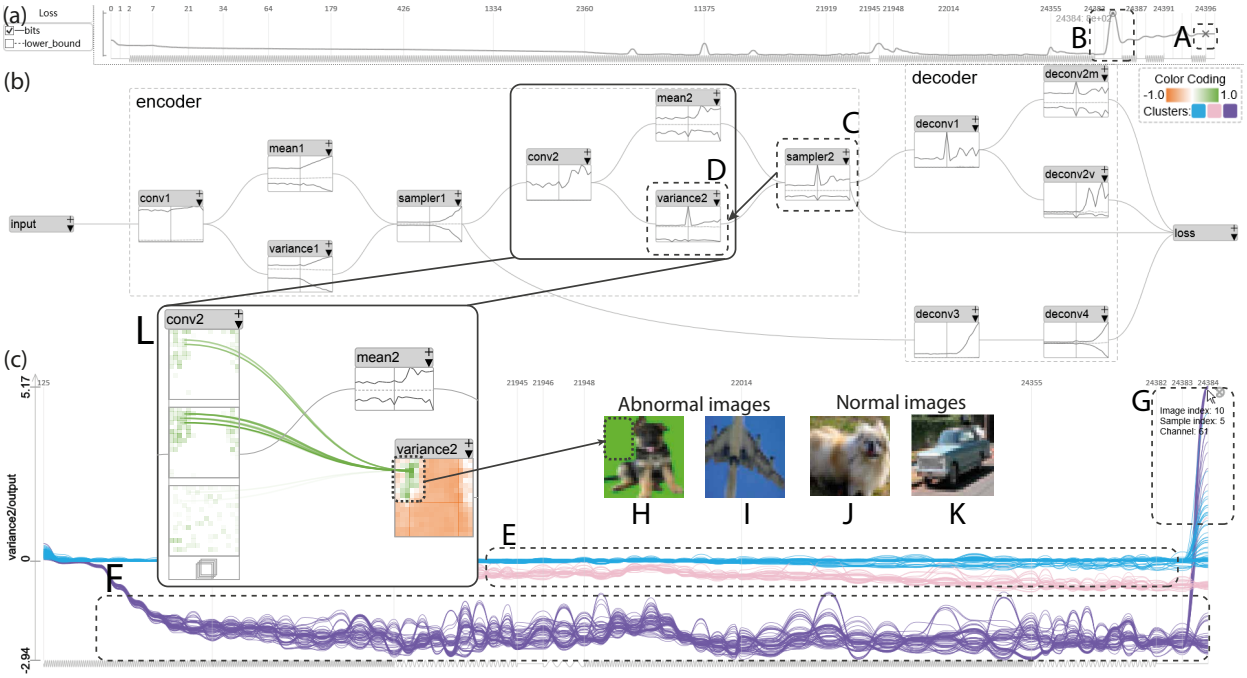

Fig. 1. DGMTracker, a visual analytics tool that helps experts understand and diagnose the training processes of deep generative models (DGMs): (a) the loss changes; (b) the data flow visualization to illustrate how data flows through a DGM and disclose how other neurons influence the output of the neuron of interest; (c) visualization of the training dynamics (e.g., activation changes).

Abstract— Among the many types of deep models, deep generative models (DGMs) provide a solution to the important problem of unsupervised and semi-supervised learning. However, training DGMs requires more skill, experience, and know-how because their training is more complex than other types of deep models such as convolutional neural networks (CNNs). We develop a visual analytics approach for better understanding and diagnosing the training process of a DGM. To help experts understand the overall training process, we first extract a large amount of time series data that represents training dynamics (e.g., activation changes over time). A blue-noise polyline sampling scheme is then introduced to select time series samples, which can both preserve outliers and reduce visual clutter. To further investigate the root cause of a failed training process, we propose a credit assignment algorithm that indicates how other neurons contribute to the output of the neuron causing the training failure. Two case studies are conducted with machine learning experts to demonstrate how our approach helps understand and diagnose the training processes of DGMs. We also show how our approach can be directly used to analyze other types of deep models, such as CNNs.

Index Terms—deep learning, deep generative models, blue noise sampling, credit assignment.

1 Introduction

Deep generative models (DGMs) provide a powerful solution to unsupervised and semi-supervised learning, where the primary focus is to discover the hidden structure of data without resorting to external labels [28] or with relatively small labeled datasets [22]. They overcome the limitations of previous deep learning models for supervised learning (e.g., CNNs), which typically require a large set of labeled data. Accordingly, DGMs have a wide range of applications, including data clustering, image denoising, 3D scene construction, scene understanding, density estimation, data compression, representation learning, and semi-supervised classification [20, 27].

However, training DGMs often requires more skill, experience, and know-how than other kinds of deep models, such as CNNs [15], due to the following reasons. First, unlike CNNs, which only involve deterministic functions (e.g., convolution), DGMs often involve both deterministic functions and random variables (e.g., Gaussian random variables), which are typically more difficult to deal with in the training process. Second, a CNN involves a bottom-up process that takes an input (e.g., image) at the bottom layer and gradually produces high-level features and outputs (e.g., categories), while a DGM typically involves a top-down generative process to describe low-level inputs (e.g., images) based on latent features. Therefore, a bottom-up Bayesian inference process is often needed in a DGM to reveal the latent features when an input (e.g., image) is provided. Though substantial progress

- *M. Liu, K. Cao, and S. Liu are with Tsinghua University and National Engineering Lab for Big Data Software. Email: {liumc13,ckl13}@mails.tsinghua.edu.cn; shixia@tsinghua.edu.cn. S. Liu is the corresponding author.*
- *J. Shi and J. Zhu are with Tsinghua University. Email: shijx15@mails.tsinghua.edu.cn; dcszj@tsinghua.edu.cn.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

has been made on large-scale Bayesian inference [55], it is still highly nontrivial to use it in practice.

For this reason, there is a growing interest in visually understanding and diagnosing the training process of a DGM, which is of theoretical and practical significance for deep learning experts. Visually analyzing the training process is technically demanding. There are two major challenges that we must address. The first challenge is to efficiently and effectively handle the large amount of time series data produced in the training process of a DGM. Typical time series data from the training process includes activation/gradient/weight changes over time (training dynamics). Since a DGM may consist of millions of activations/gradients/weights, the extracted time series dataset potentially comprises millions of time series. Directly visualizing all the time series with a line chart will induce excessive visual clutter. The second challenge is how to identify the root cause of a failed training process. In the training process of a DGM, the loss function is more likely to become NaN (not a number) or Inf (infinity), which leads to a training failure. Such errors are very hard to handle because they could arise from multiple possible sources. Potential causes include a bug or an error in the code, a lack of numerical stability in the computational environment (random variables, library versions, etc.), or an inappropriate network structure. Even when we can determine that the error is caused by the network structure, it is often difficult to locate the specific neurons because the neurons influence each other.

In an attempt to tackle these challenges, we have developed an interactive, visual analytics tool, DGMTracker, to better understand and diagnose the training process of a DGM. The key to analyzing the training process is to thoroughly examine training dynamics at different levels of granularities. To this end, we represent the change of each weight/activation/gradient as a time series and utilize a line chart to encode the time series data. A blue-noise polyline sampling algorithm is also developed to select polyline samples with blue-noise properties, which means the samples are located randomly and uniformly in the space. This algorithm can both preserve outliers and reduce visual clutter caused by a large number of time series. To help experts identify the root cause of a failed training process, we propose a credit assignment algorithm. For a neuron that leads to a training failure, the algorithm quickly discloses how other neurons contribute to the output of the neuron of interest.

The key technical contributions of this work are:

- **A visual analytics tool** that helps better understand the training process of a DGM and identify the root cause of a failed training.
- **A blue-noise polyline sampling scheme** that selects polyline samples with blue-noise properties.
- **A credit assignment algorithm** that explains how other neurons contribute to the output of the neuron causing a training problem.

Because a DGM usually contains a CNN or a multilayer perceptron (MLP) as its base component [2, 41], **DGMTracker can be directly used to analyze other types of deep models**, such as CNNs and MLPs.

2 Related Work

In the field of visual analytics and computer vision, a number of approaches have been developed to illustrate the working mechanisms of deep models [32]. They can be categorized into two groups: single-snapshot-based and multiple-snapshot-based.

2.1 Single-Snapshot-Based Approaches

The single-snapshot-based approaches focus on visualizing one representative snapshot of the training process (e.g., the last snapshot). In the field of computer vision, researchers mostly focus on disclosing the learned feature detected by each neuron [11, 29, 35, 54]. This is achieved by finding the preferred inputs (e.g., images) that highly activate a specific neuron. Various approaches have been developed to generate such inputs. For example, Nguyen et al. [36] employed a DGM to synthesize realistic images that highly activate a neuron. In the field of visual analytics, most researchers focus on presenting the whole network structure [18, 30, 49]. Pioneering research was done by Tzeng et al. [49]. They represented a neural network as a directed acyclic graph (DAG), where each neuron was encoded by a node and each connection between neurons was encoded by an edge. Although their visualization is able to illustrate how data flows through a network, this method suffers from severe visual clutter when dealing with large networks. Recently, Liu et al. [30] developed CNNVis to effectively illustrate a large deep CNN. In particular, they cluster the layers and neurons as well as the connections between neurons, which helps to reduce the visual clutter caused by a large number of neurons and the connections between them.

These single-snapshot-based methods can help experts better understand the inner workings of deep neural networks. However, they cannot reveal the evolution from an initial randomized network to an effectively trained one. In addition, they are not enough to help experts diagnose a failed training process because experts do not know which snapshot to examine in advance. Compared with the aforementioned approaches, our approach helps experts examine the entire training process, enabling them to quickly locate the problem causing a network failure.

2.2 Multi-Snapshot-Based Approaches

There are several recent attempts that aim to visually analyze the training dynamics in multiple snapshots. Such work falls into two major categories: projection-based and non-projection-based.

Projection-based approaches use dimension reduction techniques to project high-dimensional training dynamics to lower-dimensional (usually 2D) spaces. Rauber et al. [42] proposed a compact visualization to reveal how the learned representations of training samples evolve during training. They projected the high-dimensional learned representations of each snapshot to a 2D space by t-SNE [34]. They also used 2D trails to convey the evolution of the learned representations. The t-SNE-based visualization revealed that the network was able to distinguish images from different classes better over time in the training. Although the projection-based approaches do a good job of illustrating how the relationships between learned representations change over time in a training process, they do not provide an overview of training dynamics or the individual changes of activations, gradients, or weights, over time. Examining such dynamic information is crucial for locating the neuron that leads to a training failure [38].

Thus, a more effective way to visualize the training dynamics is using non-projection-based approaches such as line charts. There are several diagnostic tools that can show high-level training dynamics using non-projection approaches [16, 37]. For example, the diagnosis tool provided in TensorFlow [16] allows users to examine the change of the overall performance statistics, such as loss and the average weight in a layer, over time. These tools are able to provide experts with an overview of the training process. However, it is not enough to locate the neuron that leads to a failed training process. Compared to these tools, our approach not only provides overall performance change but also connects the overall statistics with more detailed information, training dynamics. In particular, we first extract a large amount of time series data that represent the training dynamics of a DGM. Then a blue-noise sampling scheme is developed to select time series samples, which both preserves outliers and reduces visual clutter. The sampling scheme enables experts to locate the neurons of interest. In addition, we develop a credit assignment algorithm to help further analyze the root cause of a failed training process.

3 Background

In this section, we briefly introduce the basic principles of a DGM [14], which will be useful for subsequent discussions.

Here we take a DGM designed for generating images as an example to illustrate how it works. Suppose we have a set of images X and the goal is to generate new images similar to those in X. Mathematically, the problem can be formulated as finding the true distribution $P_t(X)$, from which these images are sampled. Finding the exact $P_t(X)$ is intractable because we only know a finite set of images from $P_t(X)$. Thus, a DGM resorts to finding an approximate distribution $P_g(X)$ that can best match $P_t(X)$. In particular, $P_g(X)$ is described by taking points $\mathbf{z}$ from a simple distribution (e.g., standard Gaussian or uniform distributions) and mapping them to the generated images $\mathbf{x}'$ through a deep neu-

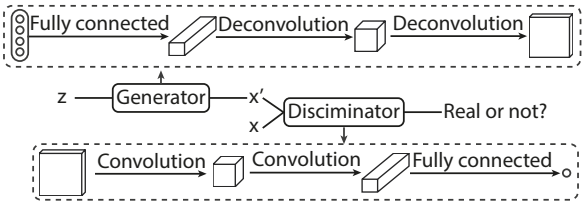

Fig. 2. An example architecture of a GAN.

ral network $f(\mathbf{z};w)$. This is based on the fact that any distribution in d dimensions can be generated by mapping d variables under a Gaussian distribution or a uniform distribution through a sufficiently complicated function (e.g., a deep neural network) [10]. The above process is similar to a decoding process, where the generated images $\mathbf{x}'$ can be seen as having been decoded from their representations $\mathbf{z}$ by a decoder $f(\mathbf{z};w)$. The most famous DGMs are variational autoencoders (VAEs) [24] and generative adversarial nets (GANs) [15], both of which have been extensively studied in unsupervised and semi-supervised learning.

VAE. The architecture of a VAE resembles that of an autoencoder, which is a traditional model in unsupervised learning. Autoencoders aim to generate a reconstruction of their input with minimum information loss [14]. An autoencoder contains two networks: an encoder network and a decoder network. The encoder maps input $\mathbf{x}$ to its representation $\mathbf{z}_a$. The decoder then maps $\mathbf{z}_a$ to a reconstructed input $\mathbf{x}'$. While an autoencoder is deterministic, a VAE can be seen as a probabilistic version of an autoencoder. In particular, in an autoencoder, the representation $\mathbf{z}_a$ is a real vector, while the representation $\mathbf{z}_v$ in a VAE is a vector of random variables (e.g., a vector of Gaussian random variables). A VAE also contains two networks (Fig. 1): a probabilistic encoder to approximate the true posterior distribution $P(\mathbf{z}_v|\mathbf{x})$, and a generative decoder to reconstruct $\mathbf{x}'$ from $\mathbf{z}_v$. Each of these networks consists of a set of standard CNN components, such as convolutional layers.

GAN. As shown in Fig. 2, a GAN contains two networks: a generator and a discriminator. The generator generates images $\mathbf{x}'$ from representations $\mathbf{z}$ and the discriminator tries to distinguish between the real images and the generated images. The discriminator is usually a CNN [2, 41] and the generator is made up of a set of standard CNN components, such as fully connected layers. The training process of a GAN can be seen as a two-player game. In the game, the generator must compete against the discriminator. The competition in this game drives both networks to improve their performance until the generated images are indistinguishable from the real images. Compared with VAEs, training a GAN is more difficult because of its unstable training dynamics [41].

4 DGMTracker

The development of DGMTracker can be divided into three stages. In the first stage, we held three workshops to gather the initial requirements from three groups of machine learning experts. In the second stage, we iteratively refined the requirements and the prototype by repeatedly consulting with the first group of experts and inviting them to try the prototype. For simplicity's sake, we denote these experts as E_i $(i = 1,2,\cdots,7)$. In the third stage, we worked with the experts to use the prototype to solve the issues encountered in their deep model training process. Because the outcomes of the third stage are reported in Sec. 7, we will briefly introduce only the first two stages here.

Workshops. To identify the initial requirements, we held three workshops, involving twenty machine learning experts and practitioners in total. We intend to invite experts who use different types of deep models such as DGMs and CNNs, so that the tool developed is more generic and can be applied to a wide range of deep models. The participants in the first workshop consist of seven machine learning experts from the Tsinghua Machine Learning Group. One of their major research interests is DGMs and CNNs in various settings, including supervised, semi-supervised, unsupervised, and reinforcement learning. In the second workshop, we invited six deep learning experts from Microsoft Research Asia, who are denoted as E_j $(j = 8,9,\cdots,13)$. The experts focus on solving computer vision tasks by using deep neural networks, such as DGMs, deep residual networks [19], and R-CNNs [13]. The third workshop involved seven experts from the Tsinghua Visual Media Group, who are denoted as E_k $(k = 14,15,\cdots,20)$. They perform pedestrian detection and image segmentation using R-CNNs. In the workshops, we mainly probed these experts about their debugging procedures and the difficulties/inconveniences of the existing diagnosis tools that they are using. Based on the aforementioned workshops, we identified a set of requirements and started to develop DGMTracker.

Development. In this stage, we collaborated with the machine learning experts in the first group to develop DGMTracker over the course of six months. Two co-authors of this paper were also from this group. We held biweekly discussions with the experts, during which we demonstrated the prototype to them and gathered their feedback to iteratively refine the prototype.

4.1 Requirement Analysis

We have identified the following high-level requirements based on the analysis of the workshop discussions.

R1 - Connecting the overall statistics with detailed training dynamics. All the experts expressed the need for the overall statistics of the training, such as loss and accuracy, which serves as an entry point for analyzing a training failure. They also need to examine the overall pattern of the training dynamics, such as the activation changes, to discover the potential reason for a failed training process. Moreover, the experts said that they wanted to link the summary statistics with the detailed training dynamics in the analysis process. This linkage could enable them to efficiently locate the neurons that caused a failed training process. However, this function is poorly supported by current tools [16, 37] because visualizing all the training dynamics will result in severe visual clutter. Thus, the core problem is to bridge the gap between the overall statistics and detailed training dynamics by providing a level-of-detail visualization. For example, E_1 said, "What I really want is a multi-scale visualization, in which I can see both high-level statistics and zoom in to the details. For example, I often need to check how the activations of neurons change in the iterative training process."

R2 - Examining how data flows through the network. One major difference between a deep model and a shallow model is that a deep model is composed of many layers. These layers have different roles and are combined to approximate the target function. Thus, understanding how data flows through the layers of a network is crucial to understanding the different roles of layers [42]. In addition, a failed training process is often caused by a specific layer. For example, expert E_3 commented, "As the layers provided by the deep learning framework (e.g., TensorFlow) are usually very robust, a failed training process is usually caused by the layers constructed by me." As a result, examining the data flow among these layers, especially the layers constructed by the experts, helps them locate the exact layer that may lead to the failed training process. However, directly visualizing the data flow will result in severe visual clutter because there may be dozens or even hundreds of layers and thousands of neurons in each layer. Thus, we need an effective visualization tool to illustrate the overall pattern of how data flows through the network [42].

R3 - Facilitating the detection of outliers. Outlier (anomaly) detection aims to find data objects that behave very differently than expected [17]. Experts E_1, $E_3 - E_5$, E_{10}, and E_{15} commented that an outlier in the training process is a potential reason for a failed training process. For example, expert E_2 said, "In my experience, the hardest error to debug is the one caused by only one or a few neurons and the error is propagated to the whole network. In this scenario, I need to use the debug mode in Theano [5] or numerical checks in TensorFlow [16] to search for the neurons at fault. It is a very long, complicated searching process." As a result, detecting outliers in the training process is crucial for diagnosing a failed training process. However, it is still very challenging to automatically and accurately identify outliers in the field of machine learning [47]. Thus, the experts desired an effective way to identify outliers in training. Previous research [1, 33, 53] also indicates that visualizations can help experts better detect outliers.

R4 - Examining how neurons interact with each other. Currently there is a poor understanding of how neurons interact with each other in a DGM. As a result, even when experts can find the neuron that

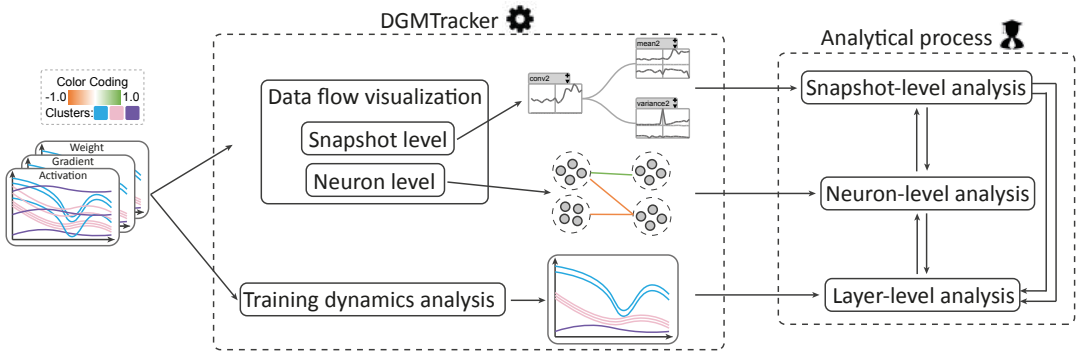

Fig. 3. DGMTracker consists of two modules: a data flow visualization and a training dynamics analysis. These modules are well aligned with the typical analytical process of an expert.

leads to a failed training process, it is hard for them to identify the root cause of a network failure. E_2 said, "Even if I find an activation (of a neuron) is abnormal, it's usually hard for me to figure out what has led to this problem." Without a comprehensive understanding of how neurons interact with each other, an exhaustive manual trial-and-error solution is infeasible. For example, E_2 commented, "I often encounter the error of infinitive weights in the trial process. I usually clip the gradients or the weights to make the model work. However, the clipping will slow down the training process or even does not work at all." Experts E_3 and E_7 also commented that sometimes an error in one layer is caused by an abnormal phenomenon in another layer and the abnormality propagates to this layer. As a result, all the experts expressed the need to understand how neurons interact with each other. In particular, they are interested in how other neurons contribute to the output of the neuron being studied. This has also been confirmed by previous research [26].

4.2 System Overview

The collected requirements motivated us to develop DGMTracker, which consists of the following modules:

- **A data flow visualization module** that visualizes how data flows through a DGM (**R2**) and discloses how other neurons influence the output of the neuron of interest (**R4**);
- **A training dynamics analysis module** that samples the time series to preserve outliers and reduce visual clutter caused by a large amount of time series data (**R1**, **R3**).

These two modules are well aligned with the tasks in an expert's typical debugging process (Fig. 4). Generally, an expert starts his or her analysis by examining the loss changes to identify the abnormal snapshots. In DGMTracker, we allow an expert to explore the loss changes with different time granularities by employing the focus+context timeline [50] (Fig. 1 (a)). The expert can click on the loss curve to select the snapshot of interest. Once the snapshot of interest has been identified, the expert usually prints out some high-level statistics for each layer (e.g., averaged activations) to identify the layer of interest (snapshot-level analysis). To support such analysis, the data flow visualization provides a hybrid visualization to illustrate how data flows through the network at the snapshot level (Fig. 1 (b)). Then, to locate the neuron that leads to the network failure, the expert usually prints out the training dynamics of the layer of interest, such as how the activations change in several snapshots (layer-level analysis). To help an expert with this task, the time series analysis module selects time series samples that can both preserve outliers and reduce visual clutter (Fig. 1 (c)). After detecting the abnormal neurons, the expert usually uses his or her prior knowledge to analyze the root cause of a failed training process (neuron-level analysis). This step heavily depends on the expertise of the expert. To ease this step, we allow the expert to interactively select a set of neurons (Fig. 1G) and explore how other neurons contribute to the output of these neurons by the data flow visualization (Fig. 1L).

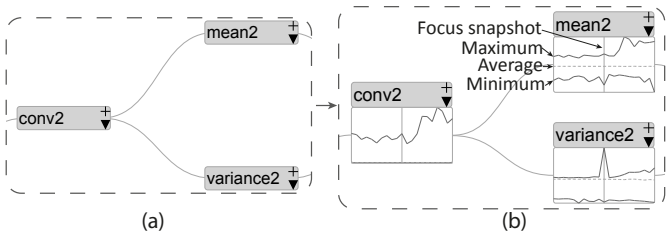

Fig. 4. A hybrid visualization to illustrate the data flow at the snapshot level: (a) a DAG layout to visualize the structure of a DGM; (b) line charts to represent the data flow.

5 Data Flow Visualization

The data flow visualization aims to illustrate how data flows through a network (snapshot-level analysis, **R2**) and how other neurons contribute to the output of the neuron of interest (neuron-level analysis, **R4**).

5.1 Snapshot Level Visualization

At the snapshot level, we focus on analyzing how data flows through a network (**R2**). Recently, Rauber et al. [42] developed a t-SNE-based method to show how the data flows through the network layers. However, this method can only handle a network with a chain structure because it utilizes a trail to illustrate the flow of each input data point through the layers. To handle networks with a more complicated structures where the layers can split and merge (e.g., VAE), we have designed a hybrid visualization that combines a directed acyclic graph (DAG) visualization (illustrating how layers are connected) with a set of line charts (presenting the data flow in each layer).

DAG visualization. We represent the structure of a DGM as a DAG, where each layer is a node and their connections are edges (Fig. 4(a)). The layout algorithm in TextFlow [9] is employed to calculate the position of each node. To handle large DGMs with dozens or even hundreds of layers, we employ the method used in TensorFlow [16] to hierarchically organize the layers. In the hierarchy, each leaf node is a layer and each non-leaf node represents a layer group. For large DGMs, only the top-level nodes of this hierarchy are shown by default and experts can expand a layer group to examine the individual layers.

Line charts for representing data flow. The data flow is represented by a set of line charts. To provide the experts with the context of the training dynamics, a line chart is placed within each node (Fig. 4(b)). In a line chart, the central vertical line represents the focus snapshot. Each curve represents the training dynamics around the focus snapshot S_t, such as the averaged activations in the snapshots from S_{t-k} to S_{t+k}. We set $k = 10$ in DGMTracker and allow experts to interactively change this value.

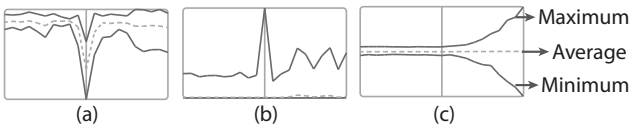

Fig. 5. Example patterns of data flow within each node: (a) abrupt changes of many activations; (b) abrupt changes of a few activations; (c) activations that become unstable after the focus snapshot.

During development, we found several interesting patterns and present several examples that correspond to activations. Fig. 5(a) indicates that many activations change abruptly at the focus snapshot. Fig. 5(b) shows that most activations remain stable but a few activations change abruptly at the focus snapshot. The above two patterns indicate that there are probably errors in the corresponding layer. These two patterns may cause other layers to behave like the one shown in Fig. 5(c), where the activations are stable before the focus snapshot and become unstable after the focus snapshot. The pattern in Fig. 5(c) implies that there are probably errors in other layers.

5.2 Neuron Level Visualization

At the neural level, we focus on computing and presenting how other neurons contribute to the output of the neuron being explored (**R4**). This helps experts analyze the root cause of a network failure (neuron-level analysis). To this end, we borrow the idea of **credit assignment** from machine learning [43]. Credit assignment determines which components (e.g., neurons) in the network are responsible for an error if the output of the network differs from the target.

5.2.1 Computation of Credit Assignment

As shown in Fig. 6, the output of a neuron n_j^l in layer l is not only influenced by the neurons in layer $l-1$ (forward contribution) but is also influenced by the neurons in layer $l+1$ (backward contribution). These two types of contributions together determine the output of n_j^l. The forward contribution has been studied in the field of machine learning [26, 54]. We adopt the state-of-the-art Layer-wise Relevance Propagation (LRP) algorithm [26], to compute the forward contribution. For the backward contribution, we leverage the backpropagation algorithm [6], which clearly discloses how the outputs of neurons in layer $l+1$ indirectly influence the outputs of the neurons in layer l. Next, we introduce how to compute the forward and backward contributions.

Forward contribution. As shown in Fig. 6, a neuron n_j^l in layer l receives the outputs of neurons in layer $l-1$. The output a_j^l of n_j^l can be computed as: $a_j^l = \sigma(\sum_i w_{ij} a_i^{l-1})$, where $\sigma()$ is the activation function and w_{ij} is the weight connecting n_j^l and n_i^{l-1}. In the LRP, the contribution $C(a_i^{l-1} \to a_j^l)$ of n_i^{l-1} on n_j^l is computed as:

$$C(a_i^{l-1} \to a_j^l) = w_{ij} a_i^{l-1} / Z, \tag{1}$$

where $Z = \sum_h w_{hj} a_h^{l-1}$ is a normalization factor.

Backward contribution. According to the backpropagation algorithm [6], the output a_k^{l+1} of the neuron n_k^{l+1} in layer $l+1$ has a backward contribution on the gradient g_{ij} of weight w_{ij}. After w_{ij} is updated according to its gradient, the weight will contributes to the output a_i^l of the neuron n_j^l. The analysis above can be summarized as:

$$a_k^{l+1} \Rightarrow g_{ij} \Rightarrow w_{ij} \Rightarrow a_i^l, \tag{2}$$

where $A \Rightarrow B$ means A has a contribution on B. In this way, the outputs of neurons in layer $l+1$ indirectly contribute to the outputs of neurons in layer l. To compute the backward contribution, we aggregate the contribution of each step in Eq. 2, and obtain:

$$C(a_k^{l+1} \to a_j^l) = w_{kj} a_k^{l+1} / Y, \tag{3}$$

where $Y = \sum_h w_{jh} a_h^{l+1}$ is a normalization factor. The detailed deduction can be found in the supplemental material.

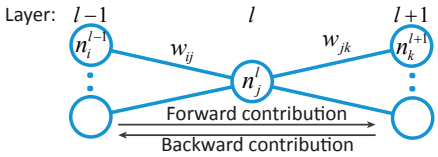

Fig. 6. Illustration of the forward and backward contribution. The forward (backward) contribution discloses how neurons are influenced by the neurons in the previous (next) layer.

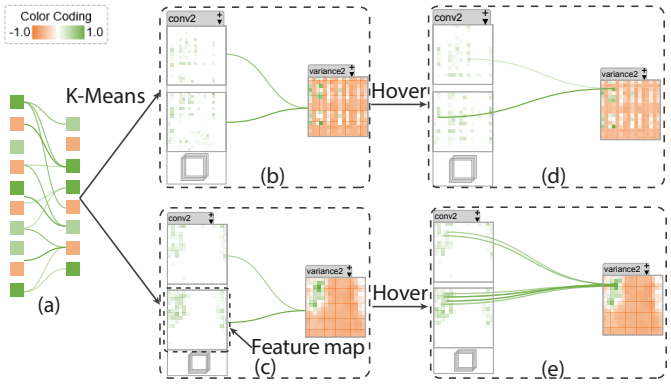

Fig. 7. Credit visualization: (a) before clustering; (b) after K-Means clustering; (c) after organizing neurons as feature maps; (d) and (e) detail contribution when mouse hovers.

5.2.2 Credit Visualization

Based on the forward and backward contribution, an expert can select a set of neurons and analyze the forward or backward contribution at a specific time point. Here, we take the forward contribution as an example to illustrate the basic idea of the visualization design. As shown in Fig. 7(a), each neuron is represented by a rectangle and colored by its activation value (red: negative activation; green: positive activation). The contribution of one neuron to another is encoded by an edge (Fig. 7(a)). We also use the same color-coding to encode the contribution value (green: a positive contribution; red: a negative contribution).

Directly visualizing all the neurons and the contributions in a layer will cause severe visual clutter. To address this issue, we first tried to cluster the neurons using the popular K-Means clustering algorithm [6] and only show the clusters whose neurons highly contribute to the output of the selected neurons (Fig. 7(b)). To save screen space, we represent the neurons in a cluster as a grid. In addition, by default, we only present the averaged contribution between two neuron clusters. The expert can hover over one neuron to examine the detailed contributions of other neurons to that neuron (Fig. 7(d)). To provide the analysis context for the expert, we combine the credit visualization with the snapshot-level visualization (Fig. 1) in a focus+context manner.

After we presented this visualization to the experts, they commented that this design was suitable for fully-connected layers but not for convolutional and deconvolutional layers. For these types of layers, they want to examine the relative position of the image patch that each neuron is influenced by. This is very useful for identifying which part of the input image might lead to the current situation if the neuron causes an input-image-related failure. Accordingly, a better solution is to represent the neurons using feature maps [27]. In a convolutional/deconvolutional layer, a feature map consists of a set of neurons that share the same weights [27]. The position of a neuron is determined by the position of the image patch that influences the output of this neuron. Each neuron in a feature map of layer l is connected to a local patch (a subset of neurons) in the feature maps of layer $l-1$. As a result, organizing neurons as feature maps can disclose which patch in the feature maps of layer $l-1$ contributes to the output of a neuron in layer l. By tracing back to layer 0 (input image), we then connect the neuron with the corresponding image patch.

Thus, for convolutional/deconvolutional layers, we organize the neurons as feature maps and use a matrix to illustrate the activation distribution of the neurons in a feature map (Fig. 7(c)). This change can help experts better identify the connection patterns between neurons in adjacent layers. For example, Fig. 7(e) indicates that the larger activation of the neuron is mostly caused by neurons in the left corner of the second feature map in the previous layer. This phenomenon cannot be identified when using K-Means clustering (Fig. 7(d)) because the neurons are placed randomly.

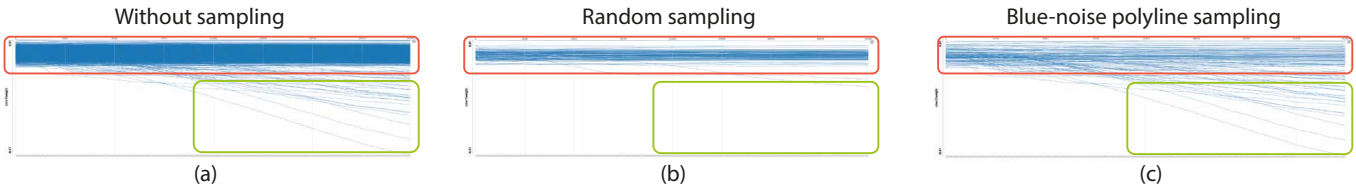

Fig. 8. Comparison of different sampling methods: (a) without sampling; (b) random sampling; (c) blue-noise polyline sampling. The blue-noise polyline sampling algorithm can better reduce visual clutter (the red rectangle) and preserve outliers (the green rectangle).

6 Training Dynamics Analysis

When an expert selects a layer, we aim to present the corresponding training dynamics to facilitate him or her in finding the neuron of interest. To enable the expert to focus on analysis, we employ a familiar visual metaphor, a line chart, to visually convey the training dynamics (a set of time series data). However, directly using a line chart to visualize a large amount of time series data will cause severe visual clutter [31]. To solve this problem, we propose a blue-noise polyline sampling algorithm to select time series samples with blue noise properties, which can both preserve outliers and reduce visual clutter (**R1**, **R3**).

6.1 Motivation

The use of blue-noise sampling is triggered by its wide usage in a variety of computer graphics applications, such as image reconstruction and color stippling [4]. Here blue-noise sampling means that the selected samples have blue-noise properties, i.e., the selected samples are located randomly and uniformly in the space [45]. This uniformity is very important in visualization [7], which is able to make the high-density regions of the candidate set less sampled and the low-density regions more sampled than random sampling. As a result, the blue-noise sampling can better reduce visual clutter and preserve outliers.

Accordingly, we propose to use blue-noise sampling to select a set of appropriate time series. The state-of-the-art method for blue-noise sampling is the line segment sampling algorithm [45]. A line segment is "a part of a line that is bounded by two distinct end points, and contains every point on the line between its end points" [52]. This algorithm first evenly groups the lines segments into N_G groups according to their angles with the x-axis. The angle of a line segment s can be computed by: $arctan(\frac{y_2-y_1}{x_2-x_1})$, where $p_1=(x_1,y_1)$ and $p_2=(x_2,y_2)$ are the end points of s. Then, a set of line segments are selected by the multi-class blue-noise sampling [51]. In particular, in each iteration, a new line segment is drawn from the most under-filled group to ensure each group of line segments are well sampled. The fill rate is defined as the number of existing samples for a group over the target number of samples for that group. The new line segment will be added to the sample set if its minimum distance from other existing line segment samples is larger than a predefined threshold [51]. The distance between two line segments are defined as the distance between their middle points. This process is repeated until the required amount of line segments are selected. Although this algorithm works well for line segments, it cannot be directly used to sample the time series data, each of which is a polyline (a connected sequence of line segments). As a result, we have developed a blue-noise polyline sampling algorithm.

6.2 Blue-Noise Polyline Sampling

As a polyline is a connected sequence of line segments, an intuitive method for sampling polyline samples with blue-noise properties is: 1) selecting line segment samples with blue-noise properties by using the blue-noise line segment sampling; 2) selecting polylines that contain the selected line segments as samples. However, this approach cannot guarantee that the selected polylines have blue-noise properties. To address this issue, we need to simultaneously select all the line segments in one polyline and maintain blue-noise properties of the selected polylines.

As stated above, the core process of blue-noise line segment sampling is **selecting** a line segment from the most under-filled group and **computing** the distance between the new samples and existing samples to determine whether to accept the new sample. Accordingly, if we want to adapt the blue-noise line segment sampling to polyline sampling, we need to solve two problems:

$\mathbf{P}_1$: how to select a polyline from the most under-filled group;

$\mathbf{P}_2$: how to compute the distance between two polylines.

Solution to $\mathbf{P}_1$. An intuitive method for solving P_1 is randomly selecting a line segment from the most under-filled group and selecting the corresponding polyline. This method is fast but may select many line segments in other over-filled groups. A better solution is to directly select the best polyline that makes the fill rates the most balanced. In particular, we compute a score s_L for each non-selected polyline L: $s_L=\sum_{i=1}^{N_G}|fr_i^{new}-1|$, where fr_i^{new} is the new fill rate of group i, if L is selected. A major issue with this solution is its expensive computational cost. To solve this problem, we employ the property that the calculation of each score is independent and use parallel computing to accelerate it.

Solution to $\mathbf{P}_2$. As the distance between two segments is computed by the distance between their middle points [45], a natural approach to computing the distance $d(L_1,L_2)$ between two polylines L_1 and L_2 is:

$$d(L_1,L_2)=\frac{1}{N_S}\sum_{i=1}^{N_S}d_C(s_1^i,s_2^i), \tag{4}$$

where N_S is the number of line segments in L_1 and L_2, $d_C(,)$ is the distance between the middle points of two line segments, and s_1^i, s_2^i are two line segments of the same snapshot belonging to L_1 and L_2. The advantage of this approach is it can preserve as many outliers as possible (Fig. 8(c)). If two segments are far apart, the distance between the corresponding polylines will mainly be determined by the distance between these two line segments.

Result. Fig. 8 compares the visualizations generated without sampling (Fig. 8(a)), with random sampling (Fig. 8(b)), and with blue-noise polyline sampling (Fig. 8(c)). The time series data we use is comprised of the changes in weights in the first layer of the VAE used in the second case study. There were originally 1,728 (3*3*3*64) time series. We sampled 5% of the time series from the data by random sampling and blue-noise polyline sampling.

Without sampling, the high-density region in Fig. 8(a) suffers severe visual clutter (the red rectangle). Compared with random sampling (Fig. 8(b)), our method better reduces visual clutter caused by a large amount of time series data (the red rectangle in Fig. 8(b) and (c)). From the visualization with no sampling, we find some time series outliers that shift away from the main trend (green rectangle in Fig. 8(a)). Comparing Fig. 8(b) and (c) indicates that, our method better preserves these time series outliers.

6.3 Interaction

We provide the following interactions to facilitate experts in examining the time series data at different time granularities. We provide a pop-up menu to show the options for analyzing the data of interest (Fig. 9).

Dimension aggregation. The activations/gradients/weights in a layer can be modeled as a tensor. For example, the activations produced by a specific image in a convolutional layer can be modeled as a three dimensional tensor $\mathbf{T}\in\mathbb{R}^{H\times W\times C}$, where H, W, and C are its height, width, and number of channels, respectively. Aggregating some dimensions can greatly reduce the number of time series to be visualized. Thus, we allow experts to interactively aggregate some dimensions of the training dynamics and decide how to aggregate these dimensions before conducting time series sampling (Fig. 9 (a)).

Focus+context timeline. As there may be hundreds of thousands of snapshots in a training process, presenting all the snapshots will cause severe visual clutter. To solve this problem, we adopt the focus+context timeline technique [50] to allow experts to zoom into the snapshots of interest. This helps experts effectively explore the training dynamics at multiple levels of time granularity. Experts can also select the time range to explore (Fig. 9 (b)).

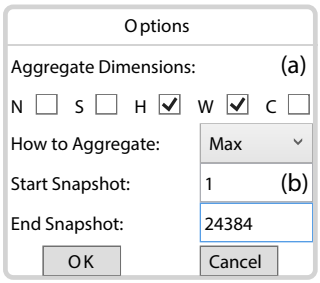

Fig. 9. The pop-up menu for the training dynamics visualization.

7 Application

We conducted two case studies to demonstrate the effectiveness of DGMTracker in helping the expert understand and diagnose DGMs. In the first case study, we collaborated with expert E_1 to achieve a better understanding of the training processes of GANs. In the second case study, we collaborated with expert E_2 to diagnose a failed training process of a VAE.

7.1 Understanding the Training Process of a GAN

This case study aims to better understand the working mechanisms of a GAN, which is one of the state-of-the-art DGMs. One major problem in training a GAN is the instability of its training process. Recently, Arjovsky et al. [2] developed the Wasserstein GAN (WGAN) to address this problem. They found that the original metric used by GAN may induce gradient vanishing. To solve this problem, they proposed a new metric, i.e., the Wasserstein distance [2], which is continuous and differentiable almost everywhere, thus provides more reliable gradients. During his investigation, E_1 got confused by two phenomena that were introduced but not fully explained in these two papers [2, 15].

Inappropriate loss function. In the original paper of GAN, Goodfellow et al. [15] claimed that a two-player-minimax-game-based loss was inappropriate in practice because it would make the training process stuck. However, E_1 did not quite understand why this loss function makes the training process stuck.

Instability of momentum-based optimizers. Momentum is a widely used technique in the optimization methods of deep learning [46]. However, in the training of WGAN, it is reported that the training process is unstable if a momentum-based optimizer is used [2]. Although the momentum is identified as a potential cause, why the momentum leads to an unstable training process is not fully explained.

As a result, the expert wanted to use DGMTracker to address these two issues.

7.1.1 Influence of an Inappropriate Loss Function

Goodfellow et al. [15] introduced a two-player minimax game [39] loss for training a GAN:

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}(x)} \log D(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D(G(z))), \quad (5)$$

where the generator is denoted by G, and the discriminator is denoted by D. Goodfellow et al. [15] claimed that using the above loss in practice was inappropriate because it would make the training process stuck, but E_1 did not understand why.

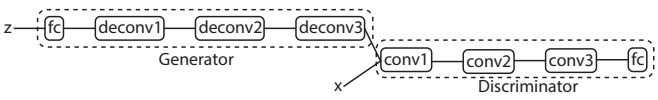

Fig. 10. The network structure of the GAN used in the case study.

To this end, he built a GAN whose structure is shown in Fig. 10. It contains 5.48 millions of weights. He trained the network using the loss on a benchmark dataset, CIFAR10 [25]. The training of this model and other models in our case studies was performed using the ZhuSuan framework [48]. From the loss curve, E_1 found the discriminator loss quickly stopped changing after a few iterations (Fig. 11). It indicated that the training process quickly got stuck.

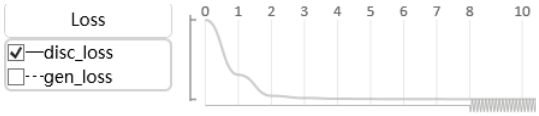

Fig. 11. The discriminator loss quickly stops changing after a few iterations which indicates the training process gets stuck.

To understand why this happened, E_1 clicked on the loss curve at iteration 8 where the training had been stuck and checked the data flow of gradients at the snapshot level. He found that the gradients were non-zero at the very beginning of the training process, but they all vanished after a few iterations (Fig. 12). To identify from which layer the gradients started vanishing, E_1 carefully examined the data flow and found the gradients vanished even in the last fully connected layer (Fig. 12A). In a deep model, the gradients are backpropagated from the last layer to the first layer. As a result, he suspected that the fully connected layer caused the training process being stuck. This triggered E_1 to check the outputs of the layer. He found an abnormal phenomenon that the outputs invoked by the generated images decreased close to 0 after a few iterations (Fig. 13A) and the outputs invoked by real images increased close to 1 after a few iterations (Fig. 13B).

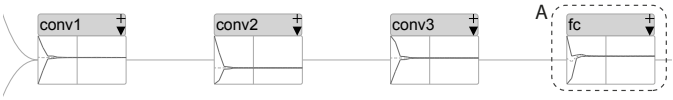

Fig. 12. The gradients vanish when using an inappropriate loss.

By looking at the generated images, E_1 understood why such phenomenon happened. At the beginning of training, the generator was not good enough to produce realistic images. In this case, the discriminator was easily trained to distinguish them from real images. Because the output of the discriminator is the probability that an image looks realistic, the outputs invoked by the generated images were close to 0 after several iterations.

After understanding why such phenomenon happened, the expert continued to analyze its influence on the training. This abnormal phenomenon drove E_1 to check the derivatives of the loss with respect to the outputs of the fully connected layer. The expert found when such phenomenon occurred, the derivatives were almost zero. According to the backpropagation algorithm, this makes the gradients of weights in all layers very small, which in turn induces the training process to be stuck.

7.1.2 Instability of Momentum-Based Optimizers

To analyze why a momentum-based optimizer made the training process of a WGAN unstable, E_1 built a WGAN whose major structure was the same as that of the GAN used above. E_1 trained the network using a momentum-based optimizer, Adam [21], on the CIFAR10 dataset.

E_1 started his analysis by examining the discriminator loss. He immediately identified two sudden increases (Fig. 14A and B). To

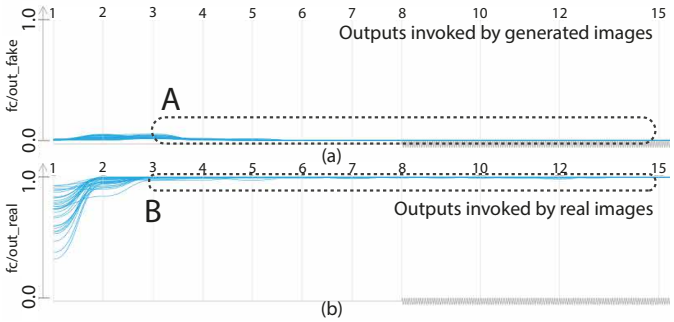

Fig. 13. The output changes of the discriminator: (a) the outputs invoked by generated images become close to 0; and (b) the outputs invoked by real images become close to 1.

identify the influence of such sudden increases, E_1 first studied how the weights are updated in Adam. Traditional stochastic gradient descent optimizer (SGD) directly updates the weight w_i by the product of its gradient g_i and the learning rate α ($\alpha > 0$):

$$w_i^{t+1} = w_i^t - \alpha g_i^t. \quad (6)$$

While Adam first adaptively estimates the mean and the variance of each gradient, and then updates the weight by the estimated mean and variance. Based on this observation, he chose to examine the means and variances of the gradients of the weights in the first convolutional layer in the discriminator because this layer is prone to training errors, such as gradient vanishing [14] (Fig. 15).

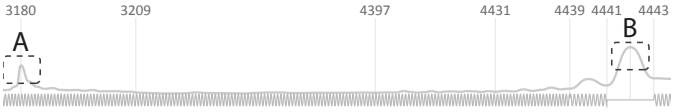

Fig. 14. Two sudden increases in the discriminator loss, which cause the training process to be unstable.

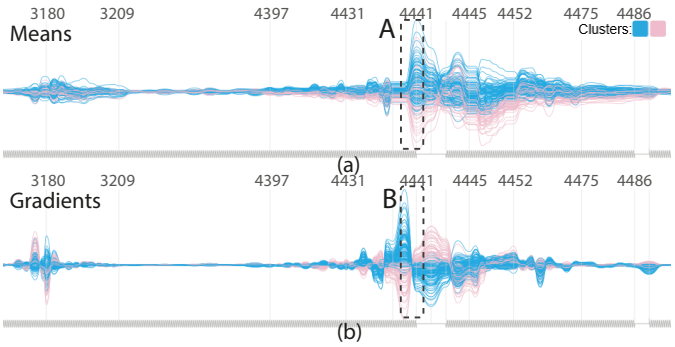

Fig. 15. The signs of the gradients have sudden changes but the signs of the means remain unchanged in a training process using Adam: (a) changes of the means; (b) changes of the gradients.

He noticed that the signs of the gradients had sudden changes at iteration 4,441 (Fig. 15B), but the signs of the means maintained unchanged (Fig. 15A). This key observation explains why momentum-based optimizers make the training process unstable. E_1 further explained, "When the signs of gradients changed, their means do not immediately reflect this change because they are determined by all the gradients before that time point (Fig. 16). As a result, the training process chooses a wrong direction and is more unstable than the one that uses a non-momentum-based optimizer (RMSprop)."

To further verify this analysis, E_1 additionally examined the changes of $(w_i^{t+1} - w_i^t)g_i^t$ for each weight w_i. This analysis is triggered by Eq. 6, which indicates that $(w_i^{t+1} - w_i^t)g_i^t = -\alpha(g_i^t)^2 \leq 0$. Thus, if this value is positive, the sign of the weight change is not consistent with its gradient; As shown in Fig. 17, there are some positive values in the training process that uses Adam (Fig. 17A). When using a non-momentum-based optimizer (RMSprop, as recommended by [2]), almost no such positive values appeared in the training process (Fig. 17B). This further verified the analysis of the expert. This phenomenon was also observed but not explained by Arjovsky et al. [2]. The expert commented, "Now

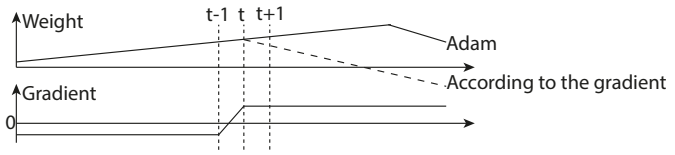

Fig. 16. Illustration of weight changes when the signs of the gradient changes in a training process using Adam.

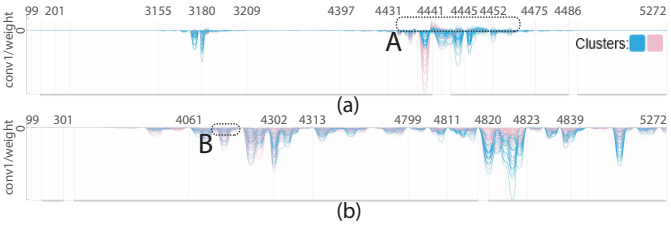

Fig. 17. The changes of $(w_i^{t+1} - w_i^t)g_i^t$ in the training processes: (a) using Adam; (b) using RMSprop.

I understand why the performance of the momentum-based optimizer is unsatisfactory in this case. The major reason is that there will be sudden changes in the gradients, which makes the momentum-based optimizer less effective. While in other types of deep models, such as CNNs, such phenomenon occurs less often. In that case, a momentum-based optimizer usually works well."

7.2 Diagnosing a Failed Training Process of a VAE

This case study demonstrates how DGMTracker helps an expert (E_2) diagnose the failed training process of a VAE. E_2 is a deep learning researcher from the first group. He has been working on VAEs for unsupervised learning, which is an important research topic in the field of deep learning [3, 23, 44]. Recently, E_2 designed a baseline network for his research, shown in Fig. 1. The network is composed of two parts: a probabilistic encoder and a probabilistic decoder. Both consist of alternating convolutional/deconvolutional layers and Gaussian sampling layers. The network contains about 0.22 millions of weights. He trained the network on the CIFAR10 dataset [25]. However, the training of this network failed. The loss became NaN in the iterations between 10,000 and 30,000 (depending on the random seed used for network initialization).

To help the expert probe the possible reason, we presented E_2 with the visualization of a failed training process. E_2 first looked at the loss of the model, which is the primary criterion for evaluating the training performance. It got to NaN at iteration 24,397 (Fig. 1A). It is notable that there was a large loss appearing at iteration 24,384 (Fig. 1B), after which quickly followed the NaN. So E_2 clicked this point on the loss curve and looked into the snapshot to examine the snapshot-level data flow. In particular, E_2 checked the maximum/average/minimum activation in each layer. E_2 quickly found that the source of this abnormal behavior was in the activation of the second Gaussian sampling layer (Fig. 1C), whose activations had a sudden increase at iteration 24,384. By tracing back the data flow, he found there was also a sudden increase of activations in the convolutional layer that outputs the logarithmic variance of the Gaussian sampling layer (Fig. 1D). This indicates that the change in this convolutional layer led to the sudden increase of the Gaussian sampling layer.

To examine why the logarithmic variance had such a sudden increase, E_2 further examined the activation changes in this convolutional layer. Because there were too many activations (about 2 million) in this layer, E_2 chose to aggregate the height and weight dimensions of the activations and got a time series for each channel in an image. As shown in Fig. 1E, most of the activations of this layer remained stable. However, some of them showed the unusual behavior of going down from the beginning of training (Fig. 1F). E_2 was drawn to these curves in purple. In addition, he found that some of the time series data had a sudden increase at iteration 24,384 (Fig. 1G). He hovered over them and found an interesting fact that all the sudden changes in activations were invoked by the 10-th image (Fig. 1G). So E_2 loaded the image to check the potential difference. This image had a very green background (Fig. 1H). As the images in CIFAR10 were RGB-formatted images, the pixel values of the green channel in the background would be very large. Having so many pixels with such extreme values in a single image is not common in a natural image dataset like CIFAR10 (Fig. 1J and K). E_2 then assumed this image led to the failed training process. To verify his assumption, E_2 selected the neurons with the largest activations and examined the forward contribution to analyze why the activations

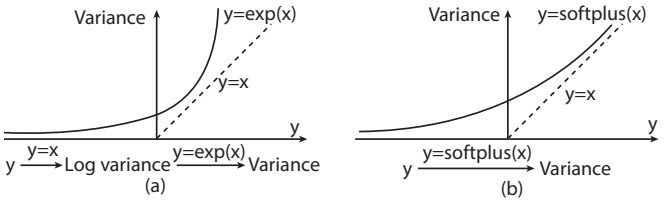

Fig. 18. Comparison between using logarithmic variance and variance in Gaussian sampling: (a) logarithmic variance; (b) variance.

were large. After hovering over the neuron with the largest activation, the expert found that some neurons from the green background had a large contribution on the output of the neuron with the largest activation (Fig. 1L). This further verified his assumption.

Having analyzed the root cause of the failure, E_2 proposed a direct solution for this. He replaced the abnormal image with another normal image (the first image) in the dataset and retrained the network. However, the network failed again with a similar behavior at a later iteration (around 300,000). This indicates that this abnormal image was only part of the reason that led to the failure. After the same analysis, he found the failure was caused by a similar image, which was a plane with a blue sky background (Fig.1I). After rethinking these discoveries and attempts, E_2 gave up solving the problem by replacing the abnormal images because there might be many other abnormal images.

Thus, E_2 decided to theoretically analyze why the network was so sensitive to the extreme values of input images. Aided by the discoveries from DGMTracker, he quickly concluded the sensitivity was caused by the transformation from logarithmic variance to variance. In particular, as shown in Fig. 18(a), if the result of the convolution operation y increased a little, the variance of the Gaussian sampler would increase a lot (when $y > 0$). Under this situation, the Gaussian sampler may generate very large samples because of the large variance. If such large samples are generated, the loss would have a sudden increase and make the training process fail. To solve this problem, E_2 proposed directly generating variance for the Gaussian sampler instead of using logarithmic variance. As the variance should be larger than zero, E_2 replaced the current identity activation function ($y = x$) with the softplus activation function $f(x) = \log(1 + e^x)$ (Fig. 18(b)).

After the replacement, the training process no longer encountered such a problem. The final loss was about 4.9, which was measured by the number of bits [14]. E_2 was quite satisfied with the result and commented, "Using a logarithmic variance is a common practice in constructing VAEs. I have noticed for a long time that the training processes of such networks are prone to the NaN error. To avoid such an error, I often try to clip the gradients or the weights to make the model work. Sometimes it works but sometimes it does not. Even if it works, the training is greatly slowed down because the clipped gradients are smaller than usual. Now I know the root cause of the error is the logarithmic variance. The result of this visual debugging process not only makes this VAE work, but also teaches me to be careful using logarithmic variance in my future research."

8 Discussion

Our case studies demonstrate the effectiveness of DGMTracker. Nevertheless, there are several opportunities for improvements.

Generalization. While the case studies focus on understanding and diagnosing DGMs, DGMTracker can be directly used to analyze a wider range of deep models, such as CNNs and MLPs. For example, we have shown in the first case study that DGM Tracker can be used to analyze CNNs. In particular, the discriminator network in the GAN is a CNN and we have analyzed why its training process is different, with different optimization approaches.

More precisely, DGMTracker can analyze the training process of a deep model in which connections between neurons do not form a cycle. Such models are called deep feedforward networks [14]. These models are the quintessential deep learning models and form the basis of many important commercial applications [14]. For example, a CNN is one kind of deep feedforward network and widely used in face recognition systems. The factor that constrains the generalization of DGMTracker is the data flow visualization. We can easily extend it to other kinds of deep models, such as recurrent neural networks (RNNs), which has cyclic connections between neurons. Specifically, we can unfold an RNN to a deep feedforward network [27] and use DGMTracker to analyze its training process.

Disk storage. A large amount of training dynamics is produced in a training process. For example, the training process of the VAE we used generates more than 5TB data (250MB per snapshot and more than 20,000 snapshots). Storing all this data to the hard disk is prohibitive for users without a powerful computer. Currently, we solve this problem by two strategies. The first strategy is saving the recent snapshots (e.g., 1,000) and a fixed number of important snapshots in the history (e.g., 2,000). We compute the importance of each snapshot by the PIP method [12] because of its efficiency and capability of giving high scores to perceptually important points. The second strategy is only saving the loss, weights, and random seeds in each saved snapshot. The activations and gradients are computed on demand using the training set, weights, and random seeds. By these two strategies, we reduced the amount of data that was saved in the VAE case study from 5TB to 6.25GB, which is not demanding for a personal computer. To further reduce the disk storage space, it is desirable to employ information theory [8] to detect more informative training dynamics.

Online analysis. Currently, in DGMTracker, all the training dynamics are collected offline and then fed into the tool for further analysis. This offline analysis already helps experts in diagnosing a failed training process to a large extent. In addition, in the back-and-forth communication with the experts, they expressed the need to analyze the online training process because training a DGM could take several days [40]. With online analysis, experts can monitor the real-time running results and stop the training process if necessary. The key to addressing this need is to design a set of visualizations that can effectively convey streaming training dynamics and to develop several data mining algorithms that can detect outliers (anomalies) from the continuously incoming training dynamics (e.g., online blue-noise polyline sampling).

9 Conclusion

In this paper, we have developed a visual analytics tool, DGMTracker, to facilitate machine learning experts in better understanding and diagnosing DGMs. DGMTracker is well aligned with the three-level analytical process of analyzing DGMs (snapshot-, layer-, and neuron-level analysis). In particular, we have designed a data flow visualization to illustrate how data flows through a DGM (snapshot level) and to disclose how other neurons contribute to the neuron of interest (neuron level). A blue-noise polyline sampling algorithm has been developed to select time series samples to preserve outliers and reduce visual clutter (layer level). We conducted two case studies to demonstrate the effectiveness and usefulness of our tool in analyzing the training process of DGMs.

Future research will focus on the following three aspects. The first task is to extend DGMTracker from offline analysis to online analysis. To this end, we plan to develop an online blue-noise polyline sampling algorithm and an online data flow visualization. Another interesting venue for future work is better easing the debugging process by employing pattern mining techniques to disclose interesting patterns in the training dynamics. The major bottleneck is the large size of the training dynamics prohibits many high-cost pattern mining techniques. Last but not least, we plan to further reduce the amount of data needed for analyzing a training process. In particular, we will leverage information theory to select the most informative training dynamics.

Acknowledgments

M. Liu, K. Cao, and S. Liu are supported by National NSF of China (No. 61672308). J. Shi and J. Zhu are supported by the National NSF of China (61620106010 and 61621136008), a grant from NVIDIA, and the Tsinghua Tiangong Intelligent Technology Institute. The authors would like to thank Prof. Kun Zhou, Chongxuan Li, Xizhou Zhu, and all the experts in our workshops for insightful discussions.

REFERENCES

[1] B. Alsallakh and L. Ren. Powerset: A comprehensive visualization of set intersections. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):361–370, 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[3] P. Bachman. An architecture for deep, hierarchical generative models. In *Advances in Neural Information Processing Systems*, pages 4826–4834, 2016.

[4] M. Balzer, T. Schlömer, and O. Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics*, 28(3):86:1–86:8, 2009.

[5] 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 *Python in Science Conference*, pages 3–10, 2010.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, and K. L. Ma. Visual abstraction and exploration of multi-class scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1683–1692, 2014.

[8] M. Chen and H. Jaenicke. An information-theoretic framework for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1206–1215, 2010.

[9] W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. J. Gao, H. Qu, and X. Tong. Textflow: towards better understanding of evolving topics in text. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2412–2421, 2011.

[10] L. Devroye. Sample-based non-uniform random variate generation. In *Conference on Winter Simulation*, pages 260–265. ACM, 1986.

[11] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4829–4837, 2016.

[12] T. Fu, F. Chung, R. Luk, and C. Ng. Representing financial time series based on data point importance. *Engineering Applications of Artificial Intelligence*, 21(2):277–300, 2008.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[15] 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*, pages 2672–2680, 2014.

[16] Google. Tensorflow. `https://www.tensorflow.org`, 2017. last accessed 2017-06-18.

[17] J. Han, J. Pei, and M. Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[18] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing*, pages 867–877, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[20] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[21] D. Kinga and J. B. Adam. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[22] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[23] D. P. Kingma, T. Salimans, and M. Welling. Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*, 2016.

[24] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[25] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Montreal, 2009.

[26] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller, and W. Samek. The LRP toolbox for artificial neural networks. *Journal of Machine Learning Research*, 17(114):1–5, 2016.

[27] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[28] C. Li, J. Zhu, and B. Zhang. Learning to generate with memory. In *International Conference on Machine Learning*, pages 1177–1186, 2016.

[29] T.-Y. Lin and S. Maji. Visualizing and understanding deep texture representations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2791–2799, 2016.

[30] 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.

[31] S. Liu, W. Cui, Y. Wu, and M. Liu. A survey on information visualization: recent advances and challenges. *The Visual Computer*, 30(12):1373–1393, 2014.

[32] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 2017.

[33] Z. Liu, Y. Wang, M. Dontcheva, M. Hoffman, S. Walker, and A. Wilson. Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):321–330, 2017.

[34] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[35] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015.

[36] 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*, pages 3387–3395, 2016.

[37] NVIDIA. NVIDIA DIGITS: Interactive deep learning GPU training system. `https://developer.nvidia.com/digits`, 2017. last accessed 2017-06-18.

[38] G. B. Orr and K.-R. Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[39] M. J. Osborne. *An Introduction to Game Theory*. New York: Oxford University Pres., 2002.

[40] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in Neural Information Processing Systems*, pages 2352–2360, 2016.

[41] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[42] P. E. Rauber, S. G. Fadel, A. X. Falco, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, 2017.

[43] D. E. Rumelhart, B. Widrow, and M. A. Lehr. The basic ideas in neural networks. *Communications of the ACM*, 37(3):87–93, 1994.

[44] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 3738–3746, 2016.

[45] X. Sun, K. Zhou, J. Guo, G. Xie, J. Pan, W. Wang, and B. Guo. Line segment sampling with blue-noise properties. *ACM Transactions on Graphics*, 32(4):127–1, 2013.

[46] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[47] G. K. L. Tam, V. Kothari, and M. Chen. An analysis of machine- and human-analytics in classification. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):71–80, 2017.

[48] Tsinghua Machine Learning Group. ZhuSuan. `https://github.com/thu-ml/zhusuan`, 2017. last accessed 2017-06-18.

[49] F. Y. Tzeng and K. L. Ma. Opening the black box - data driven visualization of neural networks. In *IEEE Visualization*, pages 383–390, 2005.

[50] X. Wang, S. Liu, Y. Chen, T.-Q. Peng, J. Su, J. Yang, and B. Guo. How ideas flow across multiple social groups. In *IEEE Visual Analytics Science and Technology*, pages 770–778, 2016.

[51] L.-Y. Wei. Multi-class blue noise sampling. *ACM Transactions on Graphics*, 29(4):79, 2010.

[52] Wikipedia. Line segment definition. `https://en.wikipedia.org/wiki/Line_segment`, 2017. last accessed 2017-06-18.

[53] P. Xu, H. Mei, L. Ren, and W. Chen. Vidx: Visual diagnostics of assembly

line performance in smart factories. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):291–300, 2017.

[54] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833, 2014.

[55] J. Zhu, J. Chen, W. Hu, and B. Zhang. Big learning with Bayesian methods. *National Science Review*, 4(3):1–25, 2017.