# GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation

Minsuk Kahng, Nikhil Thorat, Duen Horng (Polo) Chau, Fernanda B. Viégas, and Martin Wattenberg

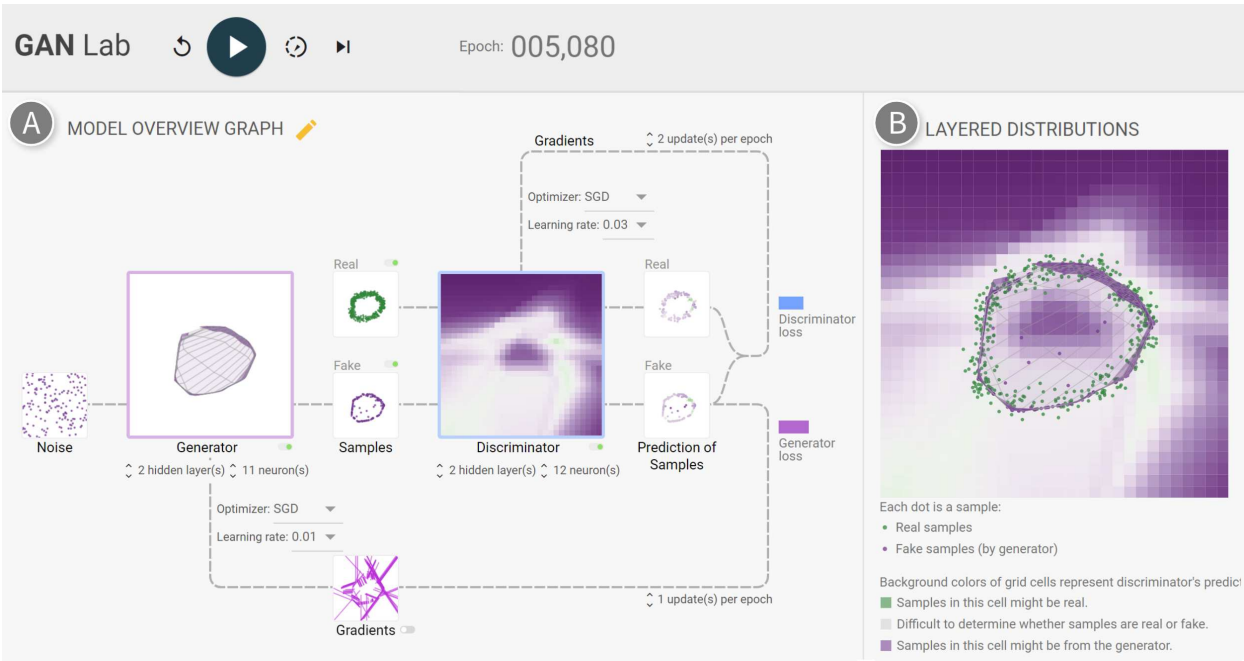



Fig. 1. With GAN Lab, users can interactively train Generative Adversarial Networks (GANs), and visually examine the model training process. In this example, a user has successfully used GAN Lab to train a GAN that generates 2D data points whose challenging distribution resembles a ring. **A.** The *model overview graph* summarizes a GAN model's structure as a graph, with nodes representing the generator and discriminator submodels, and the data that flow through the graph (e.g., fake samples produced by the generator). **B.** The *layered distributions* view helps users interpret the interplay between submodels through user-selected layers, such as the discriminator's classification heatmap, real samples, and fake samples produced by the generator.

**Abstract**—Recent success in deep learning has generated immense interest among practitioners and students, inspiring many to learn about this new technology. While visual and interactive approaches have been successfully developed to help people more easily learn deep learning, most existing tools focus on simpler models. In this work, we present GAN Lab, the first interactive visualization tool designed for non-experts to learn and experiment with *Generative Adversarial Networks (GANs)*, a popular class of complex deep learning models. With GAN Lab, users can interactively train generative models and visualize the dynamic training process's intermediate results. GAN Lab tightly integrates an *model overview graph* that summarizes GAN's structure, and a *layered distributions* view that helps users interpret the interplay between submodels. GAN Lab introduces new interactive experimentation features for learning complex deep learning models, such as *step-by-step* training at multiple levels of abstraction for understanding intricate training dynamics. Implemented using *TensorFlow.js*, GAN Lab is accessible to anyone via modern web browsers, without the need for installation or specialized hardware, overcoming a major practical challenge in deploying interactive tools for deep learning.

**Index Terms**—Deep learning, information visualization, visual analytics, generative adversarial networks, machine learning, interactive experimentation, explorable explanations

## 1 Introduction

Recent success in deep learning has generated a huge amount of interest from practitioners and students, inspiring many to learn about this technology. Visual and interactive approaches have successfully been used to describe concepts and underlying mechanisms in deep learning [17, 28, 36, 44]. For example, Karpathy's popular interactive demo [17] enables users to run convolutional neural nets and visualize neuron activations, inspiring researchers to develop more interactive tools for deep learning. Another notable example is Google's *TensorFlow Playground* [36], an interactive tool that visually represents a neural network model and allows users to interactively experiment with the model through direct manipulation; Google now uses it to educate their employees about deep learning [31].

**The rise of GANs and their compelling uses.** Most existing interactive tools, however, have been designed for simpler models. Meanwhile, modern deep learning models are becoming more complex. For ex-


- *Minsuk Kahng and Duen Horng (Polo) Chau are with Georgia Institute of Technology. E-mail: {kahng | polo}@gatech.edu.*
- *Nikhil Thorat, Fernanda B. Viégas, and Martin Wattenberg are with Google Brain. E-mail: {nsthorat | viegas | wattenberg}@google.com.*


*Manuscript received 31 Mar. 2018; accepted 1 Aug. 2018.*
*Date of publication 16 Aug. 2018; date of current version 21 Oct. 2018.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TVCG.2018.2864500*

ample, *Generated Adversarial Networks (GANs)* [9], a class of deep learning models known for their remarkable ability to generate synthetic images that look like natural images, are difficult to train and for people to understand, even for experts. Since the first GAN publication by Goodfellow et al. [9] in 2014, GANs have become one of the most popular machine learning research topics [12, 18]. GANs have achieved state-of-the-art performance in a variety of previously difficult tasks, such as synthesizing super-resolution images based on low-resolution copies, and performing image-to-image translation (e.g., converting sketches to realistic images) [8].

**Key challenges in designing learning tools for GANs.** At the high level, a GAN internally combines two neural networks, called *generator* and *discriminator*, to play a game where the generator creates "fake" data and the discriminator guesses whether that data is real or fake (both types of data are mixed together). A perfect GAN is one that generates fake data that is virtually indistinguishable from real data. A user who wishes to learn about GANs needs to develop a mental model of not only what the two submodels do, but also how they affect each other in its training process. The crux in learning about GANs, therefore, originates from the iterative, dynamic, intricate interplay between these two submodels. Such complex interaction is challenging for novices to recognize, and sometimes even for experts to fully understand [32]. Typical architecture diagrams for GANs (e.g., Fig. 2, commonly shown in learning materials) do not effectively help people develop the crucial mental models needed for understanding GANs.

**Contributions.** In this work, we contribute:

- **GAN Lab, the first interactive tool designed for non-experts** to learn and experiment with GAN models, a popular class of complex deep learning models, that overcomes multiple unique challenges for developing interactive tools for GANs (Sect. 4).
- **Novel interactive visualization design** of GAN Lab (Fig. 1), which tightly integrates a *model overview graph* that summarizes GAN's structure (Fig. 1A) as a graph, selectively visualizing components that are crucial to the training process; and a *layered distributions* view (Fig. 1B) that helps users interpret the interplay between submodels through user-selected layers (Sect. 6). GAN Lab's visualization techniques work in tandem to help crystalize complex concepts in GANs. For example, GAN Lab visualizes the generator's data transformation, which turns input noise into fake samples, as a manifold (Fig. 1, big box with purple border). When the user hovers over it, GAN Lab animates the input-to-output transformation (Fig. 3) to visualize how the input 2D space is folded and twisted by the generator to create the desired ring-like data distribution, helping users more easily understand the complex behavior of the generator.
- **New interactive experimentation features** for learning complex deep learning models, such as *step-by-step* training at multiple levels of abstraction for understanding intricate training dynamics (Sect. 7). The user can also interact with the training process by directly manipulating GAN's hyperparameters.
- **A browser-based, open-sourced implementation** that helps *broaden public's education access to modern deep learning technologies* (Sect. 7.3). Training deep learning models conventionally requires significant computing resources. For example, deep learning frameworks, like TensorFlow [1], typically run on dedicated servers. They are not designed to support low-latency computation needed for real-time interactive tools, or large number of concurrent user sessions through the web. We overcome such practical challenges in deploying interactive visualization for deep learning by using *TensorFlow.js Core*,[1] an in-browser GPU-accelerated deep learning library recently developed by Google; the second author is a lead developer of TensorFlow.js Core. Anyone can access GAN Lab using their web browsers without the need for installation or specialized backend. GAN Lab runs locally on the user's web browser, allowing us to easily scale up deployment for our tool to the public, significantly broadening people's access to tools for learning about GANs. The source code is available in `https://github.com/poloclub/ganlab/`.
- Usage scenarios that demonstrate how GAN Lab can help beginners learn key concepts and training workflow in GANs, and assist practitioners to interactively attain optimal hyperparameters for reaching challenging equilibrium between submodels (Sect. 8).

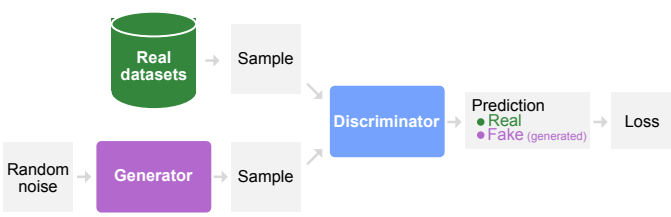



Fig. 2. A graphical schematic representation of a GAN's architecture commonly used.

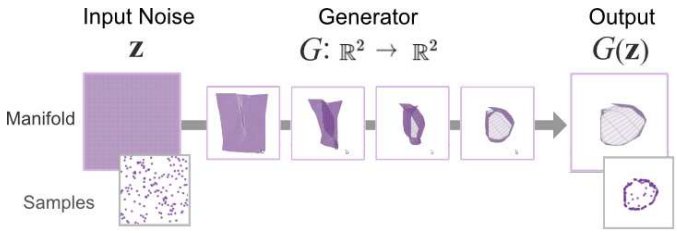



Fig. 3. In GAN Lab, the *generator*'s non-trivial data transformation is visualized as a manifold, which turns *input noise* (leftmost) into fake samples (rightmost). GAN Lab animates the input-to-output transformation to help users more easily understand this complex behavior.

**VIS's central role in AI.** We believe in-browser interactive tools developed by our VIS community, like GAN Lab, will play critical roles in promoting people's understanding of deep learning, and raising their awareness of this exciting new technology. To the best of our knowledge, our work is the first tool designed for non-experts to learn and experiment with complex GAN models, different from recent work in visualization for deep learning [16, 20, 21, 30, 37, 43] which primarily targets machine learning experts. Our work joins a growing body of research that aims to use interactive visualization to explain complex inner workings of modern machine learning techniques. Distill, a new interactive form of journal, is dedicated to achieving this exact goal [29]. We hope our work will help inspire even more research and development of visualization tools that help people better understanding artificial intelligence technologies.

## 2 Background: Generative Adversarial Networks

This section presents a brief introduction of Generated Adversarial Networks, which will help ground our discussion in this paper.

Generative Adversarial Networks (GANs) [9] are a new class of unsupervised generative deep learning models that model data distributions. It can be used for generating multi-dimensional data distributions (e.g., an image is a multi-dimensional data point, where each pixel is a dimension). The model takes *real samples* and random vectors (i.e., *random noise*) as inputs and transforms the random vectors into *fake samples* that mimic the real samples. Ideally, the distribution of the fake samples will be indistinguishable from the real samples. The architecture of GANs is composed of two neural networks, called *generator* and *discriminator*, and is often represented as an abstracted data-flow graph as in Fig. 2. The generator, $G$, takes a random noise vector, $\mathbf{z}$, as input and transforms it into a fake sample, $G(\mathbf{z})$ (i.e., a multi-dimensional vector); the discriminator, $D$, which is a binary classifier, takes either a real or fake sample, and determines whether it is real or fake ($D(\mathbf{x})$ represents the probability that $\mathbf{x}$ is real rather than fake).

A GAN model is iteratively trained through a game between the discriminator and generator. In GAN, two cost functions exist: the one for the discriminator measures the probability of assigning the

[1] TensorFlow.js (`https://js.tensorflow.org`) was formerly deeplearn.js.

correct labels to both real and fake samples (i.e., the sum of $D(\mathbf{x})$ and $1-D(G(\mathbf{z}))$); the other for the generator measures that for fake samples only (i.e., $1-D(G(\mathbf{z}))$). The goal of the discriminator is to maximize its cost, but the goal of the generator is to minimize its cost, which introduces conflicts (i.e., zero-sum). Therefore, it has to play a mini-max game to find the optimum. Goodfellow et al. [9] used an interesting analogy to explain how it works, where we can view the generator as a *counterfeiter* who makes fake dollar bills, and the discriminator as the *police*. If the police can spot the fake bills, that means the counterfeiter is not "good enough," so the counterfeiter carefully revises the bills to make them more realistic. As the discriminator (police) differentiates between real and fake samples, the generator (counterfeiter) can glean useful information from the discriminator to revise its generation process so that it will generate more realistic samples in the next iteration. And to continue to receive such helpful information, the generator keeps providing its updated samples to the discriminator. This iterative interplay between the two players leads to generating realistic samples.

## 3 Related Work

### 3.1 Visualization for Understanding Deep Learning

Researchers and practitioners have written articles and deployed explorable web-based demos to help people learn about concepts in deep learning. One of the popular examples is Chris Olah's series of essays,[2] explaining mathematical concepts behind deep learning using visualizations. One article explains how neural networks transform and manipulate manifolds [28]. Another popular example is Andrej Karpathy's collection of web-based demos developed using ConvNetJS,[3] a lightweight JavaScript library for deep learning. His MNIST demo [17] dynamically visualizes intermediate results, such as neuron activation.

Olah's articles and Karpathy's demos have inspired many researchers to develop interactive visualizations for novices to easily understand deep learning techniques [11, 36]. A notable example is *TensorFlow Playground* [36], an interactive visualization tool for non-experts to train simple neural net models. Google has integrated it into its internal machine learning course for educating its employees; the course is now available to the public [31]. Distill, a new online interactive journal, has recently been created and it is dedicated to interactive explanation of machine learning [29]. The journal features a growing number of articles with interactive visualization [5, 7, 42]. However, most existing visualizations focus on simpler models. Modern deep learning models are much more complex, and we will present and discuss unique design challenges that stem from such complexity (Sect. 4).

### 3.2 Algorithm Visualization & Explorable Explanations

Even before the surge of interest in deep learning techniques, researchers had studied how to design interactive visualization to help learners better understand the dynamic behavior of algorithms [14, 15, 33, 35]. These tools often graphically represent data structures and allow students to execute programs in a step-by-step fashion [10, 35]. While many of these tools target algorithms covered in undergraduate computer science curricula, some specialized tools exist for artificial intelligence [2]. As deep learning models are a category of specialized algorithms, when we design GAN Lab, we draw inspiration from the principles and guidelines proposed in the aforementioned related domains [34].

As web has become a central medium for sharing ideas and documents, many interactive experimentation tools implemented in JavaScript have been viewed as "explorable explanations," an umbrella term coined by Bret Victor in 2011 [39]. He advocated the use of interactive explanations with examples to help people better understand complex concepts by actively engaging in the learning process. Many interactive tools instantiate this idea, including the ones showcased on the popular website with the same name (Explorable Explanations[4]). These tools aim to help people actively learn through playing and interactive experimentation. GAN Lab aligns with this research theme.

[2]Colah's blog, `http://colah.github.io`

[3]ConvNetJS, `https://cs.stanford.edu/people/karpathy/convnetjs/`

[4]Explorable Explanations, `http://explorabl.es/`

### 3.3 Visual Analytics for Deep Learning Models & Results

Over the past few years, many visual analytics tools for deep learning have been developed [4, 16, 20, 21, 27, 37, 41], as surveyed in [13, 23]. Most were designed for experts to analyze models and their results. For instance, TensorFlow Graph Visualizer [43] visualizes model structures, to help researchers and engineers build mental models about them. Many other tools focus to visually summarize model results for interpreting how models respond to their datasets. For example, CNNVis [21] was designed for inspecting CNN model results; LSTMVis [37] and RNNVis [27] were for RNN models. A few other tools allow users to diagnose models during training. For example, DeepEyes [30] does so through t-SNE visualizations. Two visual analytics tools have been developed for GANs [20, 41]. DGMTracker [20] allows experts to diagnose and monitor the training process of generative models through visualization of time-series data on data-flow graphs. GANViz [41] helps experts evaluate and interpret trained results through multiple views, including one showing the distributions of real and fake image samples, for a selected epoch, using t-SNE. Different from all existing tools designed to help experts analyze models and results that we summarized above, we focus on non-experts and learners, helping them build intuition of the internal mechanisms of models, through interactive experimentation.

## 4 Design Challenges for Complex Deep Learning Models

Our goal is to build an interactive, visual experimentation tool for users to better understand GANs, a complex deep learning model. To design GAN Lab, we identified four key design challenges unique to GANs.

C1. [Model] **Complex model structures with submodels.** The structures of modern deep learning models (including GANs) are complex; they often incorporate multiple base neural networks or deep learning models as submodels. For example, a GAN combines two neural nets: generator and discriminator; an image captioning model often consists of both CNNs and RNNs for translation between images and text [40]. Effective visualization of such models calls for new strategies different from those designed for conventional models. For example, it is crucial to find the appropriate levels of visual abstraction for the models, as visualizing all low-level details will overwhelm users. Special visual design may be needed to help users interpret the intricate interplay between submodels (e.g., discriminator and generator).

C2. [Data] **High-dimensional datasets.** As deep learning models often work with large, high-dimensional datasets, visualizing their distributions would quickly create many traditional challenges well-studied in information visualization research [22]. While we may use techniques like *dimensionality reduction* to partially address such issues, this could introduce additional complexities to the systems, potentially distracting users from their main goal of understanding how deep learning models work.

C3. [Training Process] **Many training iterations until convergence.** Deep learning models are trained through many iterations (i.e., at least thousands), introducing nontrivial challenges for developing interactive tools. First of all, as it takes time to converge, the tools need to keep providing users with information during training (e.g., progress), and users may also want to provide feedback to models (e.g., by changing hyperparameters). In addition, one popular feature used in many experimentation tools is a step-by-step execution of systems [10, 33], however, the definition of *steps* becomes different in training of complex models, because the training process consists of many iterations and each iteration also consists of the training of multiple submodels.

C4. [Deployment] **Conventional deep learning frameworks ill-fitted for multi-user, web-based deployment.** Training deep learning models conventionally requires significant computing resources. Most deep learning frameworks written in Python or C++, like TensorFlow [1], typically run on dedicated servers that

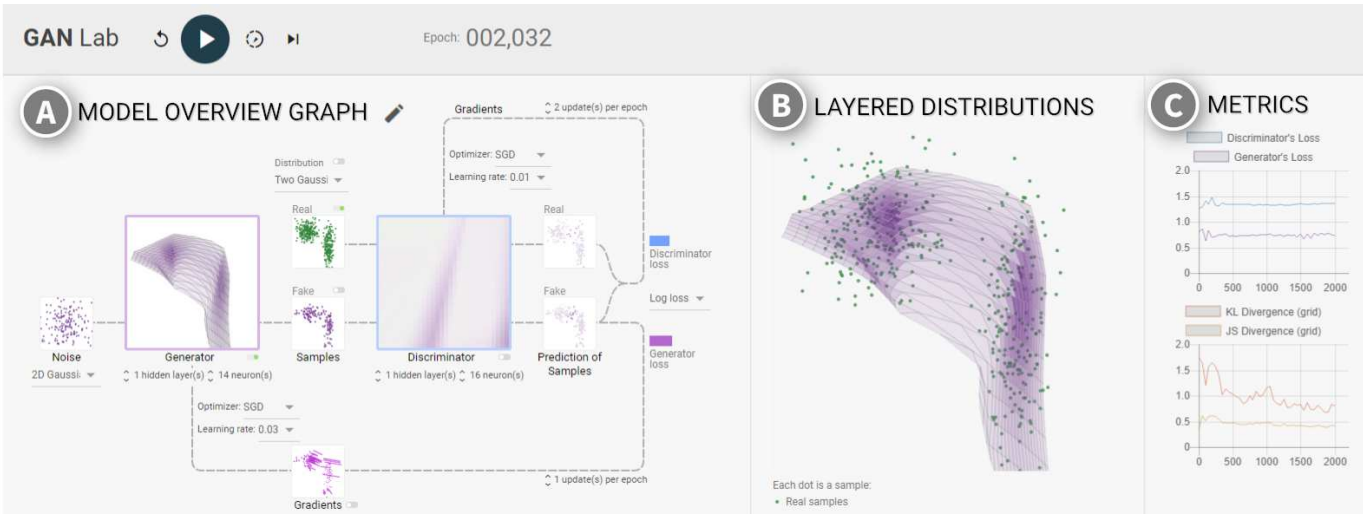



Fig. 4. The GAN Lab interface integrates multiple views: **A.** The *model overview graph* summarizes a GAN model's structure as a graph, with nodes representing the submodels, and the data that flow through the graph; **B.** The *layered distributions* view overlays magnified versions of the graph's component visualizations, to help users more easily compare and understand their relationships; **C.** The *metrics* view presents line charts that track metric values over the training process. Users start the model training by clicking the *play* button on menu bar. The three views are dynamically updated, as training progresses. In this example, real samples are drawn from two Gaussian distributions, and the generator, consisting of a single hidden layer with 14 neurons, has created samples whose distribution is quite similar to that of the real samples.

utilize powerful hardware with GPU, to speed up the training process. However, even with a powerful backend, they cannot easily support a large number of concurrent user sessions through the web, because each session requires significant computation resources. When combined, even a small number of concurrent sessions can bog down a powerful server. Off-loading computation to the end user is a possible solution, but conventional deep learning frameworks are not designed to support low-latency computation needed for real-time interactive tools.

## 5 Design Goals

Based on the identified design challenges in the previous section, we distill the following main design goals for GAN Lab, a novel interactive visualization tool for learning and experimenting with GANs.

G1. **Visual abstraction of models and data flow.** To give an overview of the structure of complex models, we aim to create a visual representation of a model by selectively choosing and grouping low-level operations (and intermediate data) into high-level components (C1). It helps users visually track how input data are transformed throughout the models. For users to clearly examine the internal model training process and data flow, we would use low-dimensional datasets (C2). (Sect. 6.1)

G2. **Visual analysis of interplay between discriminator and generator.** As GANs internally use two different neural nets, it is important for users to understand how they work together, to get a holistic picture of the overall training process (C1). In response, we would like to enable users to examine and compare the visualizations of the model components to understand they affect each other to accomplish the generation tasks. (Sect. 6.2)

G3. **Dynamic experimentations through direct manipulation of hyperparameters.** We aim to let users dynamically play and experiment with models. To help users quickly understand the roles of many hyperparameters and control them (C3), we would like to design interactive interfaces which users can easily locate and manipulate the options. The users' actions are directly applied to the model training process. (Sect. 7.1)

G4. **Supporting step-by-step execution for learning the training process in detail.** Since the training process of deep learning models consists of many iterations and each iteration also consists of several steps, the step-by-step execution of models can greatly help novices to understand the training process (C3). To address this needs, we aim to design multiple ways to execute models in a step-by-step fashion by decomposing the training process into steps at multiple levels of abstraction. (Sect. 7.2)

G5. **Deployment using cross-platform lightweight web technologies.** To develop a tool that is accessible from multiple users without a need to use specialized powerful backend (C4), we would like to use web browsers both for training models and visualizing results. (Sect. 7.3)

## 6 Visualization Interface of GAN Lab

This section describes GAN Lab's interface and visualization design. Fig. 4 shows GAN Lab's interface, consisting of multiple views. Using the control panel on top, users can run models and control the speed of training, which we describe in detail in the next section (Sect. 7). This section primarily describes the other three views that visualize models and trained results: (A) **model overview graph** view on the left (Sect. 6.1); (B) **layered distributions** view in the middle (Sect. 6.2); (C) **metrics** view on the right (Sect. 6.3). In the figure, 2D real samples are drawn from two Gaussian distributions. The user's goal is to train the model so that it will generate a similar distribution, by transforming 2D Gaussian noise using a neural net with a single hidden layer.

**Color scheme.** In our visualization, we color real data green and fake data purple. We do not use a more traditional green-red color scheme, as we do not want to associate fake data with a negative value. For visualizing the discriminator, we use blue, a color unrelated to the color scheme chosen for coloring data. For visualizing the generator, we again use the color purple because the generated points are the fake points the model sees.

### 6.1 Model Overview Graph: Visualizing Model Structure and Data Flow

The *model overview graph* view (Fig. 4 at A) visually represents a GAN model as a graph, by selectively grouping low-level operations into high-level components and presenting data flow among them.

#### Abstraction of Model Architecture as Overview Graph

The model overview graph visually summarizes the architecture of a GAN model. Instead of presenting all low-level operations and intermediate data (i.e., output tensors), it selectively represents high-level

components and important intermediate data as nodes. Specificallly, nodes of the graph include two main submodels (i.e., generator and discriminator) and several intermediate data (e.g., fake samples). Each submodel, which is a neural network, is represented as a large box, and six data nodes are visualized as small boxes. This decision is based on our observation of how people draw the architecture of GANs [6] (like Fig. 2). Users are often familiar with the structure of the basic neural networks and more interested in the overall picture and interplay between the two submodels. When we determine the position of the nodes, we place input data nodes on the left side of the submodels and output nodes on the right (for forward data flow). Then we draw edges where forward data paths are drawn from left to right and backward data paths, representing *backpropagation*, are drawn as two large backward loops (one for the discriminator and the other for the generator).

### Visualization of Nodes in Overview Graph

We visualize the current states of models within the nodes in the graph for users to understand and monitor the training process.

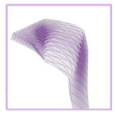
**Using 2D datasets to promote comprehension.** One challenge in visualizing this information arises from the difficulty of visualizing a large number of high-dimensional data points. To tackle this issue, we decided that we limit our GAN models to generate two-dimensional data samples, while GANs often work with high-dimensional image data. This decision is mainly for helping users easily interpret visualization and focus to understand the internal mechanisms of the models. As many researchers identified, when designing interactive tools, it is even more desirable to focus on simpler cases [34]. Visualization of two-dimensional space is easier for people to understand how data are transformed by the models than that of higher- or one-dimensional spaces: 3D or larger requires dimensionality reduction techniques that add more complexity to users and hinders their understanding.

Below we describe how we visualize each node. We show a miniaturized copy of each node's visualization from Fig. 4 for easier referencing.

**Real samples** are what a GAN would like to model. Each sample, a two-dimensional vector, is represented as a green dot, where its x and y position represents the values of its two-dimensional data point. In this example, two Gaussian distributions exist: on the upper-left, and on the right.

**Random noise**, an input to the generator, is a set of random samples. In GAN Lab, noise can be either 1D or 2D. If it is a 1D value, data points are positioned in a line; if a 2D vector (which is default), positioned in a square box, as shown in the small figure on the right.

**Fake samples** are output produced the generator by transforming the random noise. Like real samples, fake samples are also drawn as dots, but in purple. For a well-trained GAN, the generated distribution should look indistinguishable from the real samples' distribution.



**Generator**, a neural net model, is a transformation function, $G : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, that maps a 2D data point (i.e., random noise, $\mathbf{z}$) to another 2D data point (i.e., fake sample, $G(\mathbf{z})$). We visualize the transformed results as a 2D **manifold** [28], as in the figure on the right. To draw this manifold, we first create a square grid (e.g., 20x20) for the random noise (see Fig. 5, leftmost) where each cell represents a certain noise range (e.g., $\{\mathbf{z} = (z_1, z_2) \mid 0.85 \leq z_1 < 0.90 \wedge 0.10 \leq z_2 < 0.15)\}$). We color each cell in purple, encode its probability density with opacity (i.e., more opaque means more samples in the cell). The generator $G$ transforms the random noise into fake samples by placing them in new locations. To determine the transformation for the grid cells, we feed each cell's four corners into the generator, which returns their transformed positions forming a quadrangle (e.g., $G(0.85, 0.10) = (0.21, 0.75)$, $G(0.85, 0.15) = (0.24, 0.71)$, ...). Thus, the whole grid, now consisting of irregular quadrangles, would look like a warped version of the original regular grid. The density of each (warped) cell has changed. We calculate its new density by dividing the original density value (in the input noise space) by the *area* of the quadrangle. Thus, a higher opacity means more samples in smaller space. Ideally, a very fine-grained manifold will look almost the same as the visualization of the fake samples. Our visualization technique aligns with the *continuous scatterplots* idea [3] that generalizes scatterplots to continuous data by computing the density of data samples in the scatterplot space. To help users better understand the transformation, we show an animation of the square grid transitioning into the warped version (see Fig. 5), when users mouse over the generator node in the overview graph.

Fig. 5. Visualization of generator's transformation. When users mouse over the generator node, an animation of the square grid transitioning into a warped version is played.

**Discriminator** is another neural net model, which is a binary classifier, that takes a sample as input and determines whether it is real or fake by producing its prediction score (values from 0 to 1). We visualize the discriminator using a 2D **heatmap**, as in TensorFlow Playground [36]. The background colors of a grid cell encode the prediction values (darker green for higher values representing that samples in that region are likely real; darker purple for lower values indicating that samples are likely fake). As a GAN approaches the optimum, the colors become more gray (as in the above figure), indicating the discriminator cannot distinguish fake examples from the real ones.

**Predictions** are outputs from the discriminator. We place real or fake samples at their original positions, but their fill colors now represent prediction scores determined by the discriminator. Darker green indicates it is likely a real sample; darker purple likely a fake sample. In this example, most samples are predicted as fake, except for the ones on the upper left.

**Gradients for generator** are computed for each fake sample by backpropagating the generator's loss through the graph. This snapshot of gradients indicates that how each sample should move to, in order to decrease the loss value. As a gradient represents a vector, we visualize it as a line starting from the position of each sample, where length indicates strength.

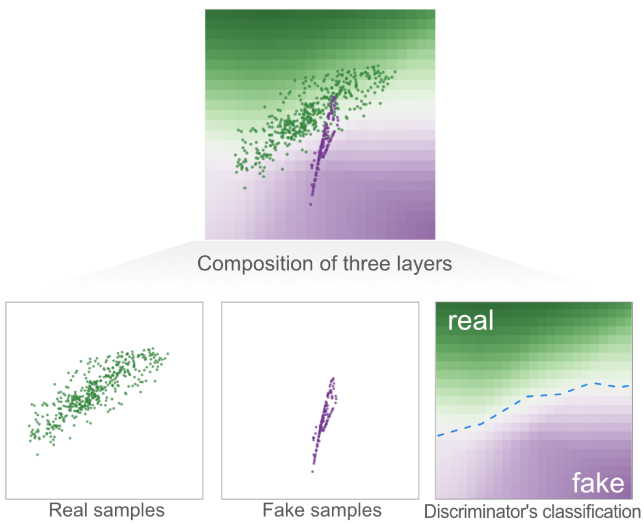



Fig. 6. The discriminator's performance can be interpreted through the *layered distributions* view, a composite visualization composed of 3 layers selected by the user: *Real samples*, *Fake samples*, and *Discriminator's classification*. Here, the discriminator is performing well, since most real samples lies on its classification surface's green region (and fake samples on purple region).

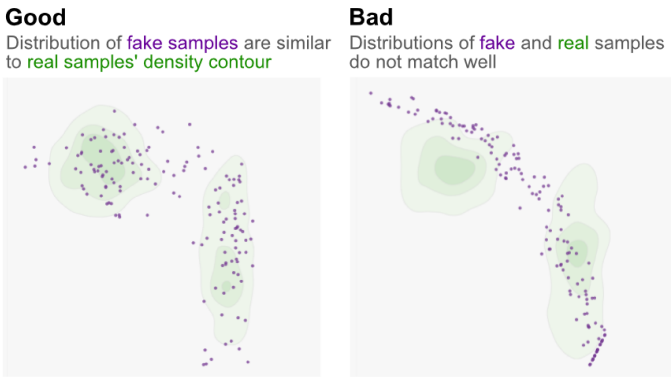



Fig. 7. Evaluating how well the distribution of fake samples matches that of real samples by turning on real samples' density contour and fake samples in the layered distributions view.

### 6.2 Layered Distributions: Visual Analysis of Interplay between Discriminator and Generator

In complex models like GANs, it is a key to understanding relationships among several elements of the models. For example, users may want to check how the distribution of fake samples are similar to those of real samples. Although users can perform a side-by-side comparison of the two different nodes on the model overview graph, this task would be greatly improved when they are overlapped in the same coordinates.

To help visually analyzing relationships among multiple components, we create a *layered distributions* view (Fig. 4 at B) that presents a large canvas showing the visual representations of the nodes in the model overview graph as multiple layers. The layers can be turned on or off using toggle switches. We do not intend to visualize all layers, as it is overwhelming to users and it is much more effective to include only the useful information for particular tasks. The view currently supports six layers. All layers, except the one for the real samples' density contour, are magnified versions of the visual representations of the graph nodes we described in the previous subsection (Sect. 6.1). The layers are:

- Real samples (green dots)
- Real samples' density contour (see Fig. 7)
- Generator transformation manifold
- Fake samples (purple dots)
- Discriminator's classification heatmap
- Generator's gradients (pink lines)

**Useful combinations of layers.** By selecting which visualizations to be included in the canvas, users can visually analyze the state of the models and the interplay between discriminator and generator, from multiple angles. We describe three example combinations that support multiple analysis tasks. First, Fig. 6 illustrates that the discriminator may be visually interpreted by comparing the samples' positions with grid's background colors. Here, the discriminator is performing well, as most real and fake samples lie on its classification's green and purple regions, respectively. The second example in Fig. 7 illustrates how users may visually evaluate how well the distribution of fake samples matches that of the real samples. It helps users to determine whether the two distributions are similar or not, which is the main goal of GANs. The last example in Fig. 8 shows how the view can help users understand the interplay between discriminator and generator. Fake samples' gradient directions point to the classification's green regions, meaning that the generator leverages information from the discriminator to make fake samples less distinguishable from the real ones.

### 6.3 Metrics: Monitoring Performances

The *metrics view* (Fig. 4 at C) shows a number of line charts that track several metric values changing as the training promises. GAN Lab currently provides two classes of metrics. The first kind is the loss values of the discriminator and generator, which are helpful for evaluating submodels and comparing their strengths. The second kind of metrics is for evaluating how similar the distributions of real and fake

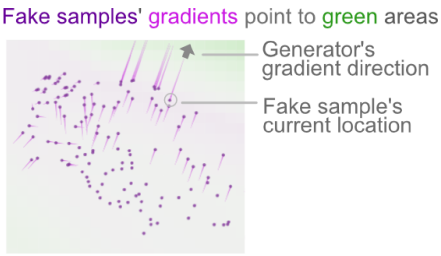



Fig. 8. Example of understanding the interplay between discriminator and generator using the layered distributions view. Fake samples' movement directions are indicated by the generator's *gradients* (pink lines), based on those samples' current locations and the discriminator's current classification surface (visualized by background colors).

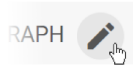
samples are. GAN Lab provides Kullback-Leibler (KL) and Jensen-Shannon (JS) divergence values [19, 38] by discretizing the 2D continuous space (via the grid). Formally, the KL divergence value is defined as $KL(P_{\text{real}}||P_{\text{fake}}) = -\sum_i P_{\text{real}}(i) \log \frac{P_{\text{fake}}(i)}{P_{\text{real}}(i)}$, where $P_{\text{real}}(i)$ is the probability density of the real samples in the $i$-th cell, calculated by dividing the number of real samples in the $i$-th cell by the total number of real samples; $P_{\text{fake}}(i)$ is similarly defined for the fake examples. We decided to use these measures, among others, because they are some of the most commonly used approaches for comparing distributions and they do not incur heavy in-browser computation overhead.

## 7 Interactive Experimentation

This section describes how users can interactively experiment with GAN models using GAN Lab.

**Basic workflow.** Clicking the play button, located on the top of the interface, starts running the training of a GAN model and dynamically updates the visualizations of intermediate results every $n$ epochs (a.k.a., iterations). This helps users keep track of the model's training and examine how they evolve. Users can pause the training by clicking the pause button (the play button changes to pause button during training).

### 7.1 Direct Manipulation of Hyperparameters




GAN Lab is designed for users to directly manipulate model's training as easy as possible. When users click the editing icon on the right side of the label for the *model overview graph* view, several up/down buttons or dropdown menus, which controls the model's hyperparameters, are shown (see Fig. 4). Each item is located near its relevant submodel or data node for users to easily locate it. Users can directly change the values using the buttons or dropdown menus, and the user's actions (e.g., increasing learning rate) are immediately applied to the model training process, except for some of the submodel-specific options (e.g., number of hidden layers), and the effects of this change will be visualized, as the training further progresses. This would greatly help users understand how these hyperparameters affect the model training process. The current available hyperparameters in GAN Lab include:

- Number of layers for generator and discriminator
- Number of neurons in each layer for generator and discriminator
- Optimizer type (e.g., Stochastic Gradient Descent, Adam) for updating the generator and discriminator
- Learning rates for updating the generator and discriminator
- Loss function (e.g., log loss [9], least square loss (LS-GAN [25]))
- Number of training runs for discriminator (and generator) for every epoch[5]
- Noise dimension (e.g., 1D, 2D) and distribution type (e.g., uniform, Gaussian)

GAN Lab also allows users to pick a distribution of real samples using the drop-down menu that currently implements five examples (e.g.,

[5]In training of GANs, for every epoch, the discriminator and generator are trained by turns. Goodfellow et al. [9] suggested that the discriminator can be updated $k$ more times in practice, and GAN Lab enables to adjust this $k$ value.

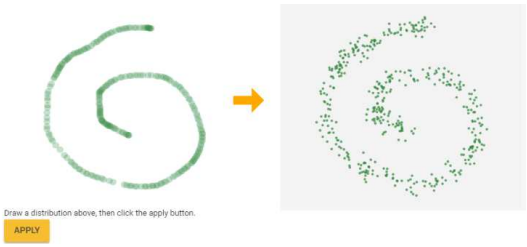



Fig. 9. Users can create real samples by drawing their distribution.

ring). Users can also specify a new distribution by drawing one on a canvas with brush, as illustrated in Fig. 9.

### 7.2 Step-by-Step Model Training at Multiple Levels

GAN Lab supports *step-by-step* training at multiple levels of abstraction for understanding intricate training dynamics. The step-by-step execution of systems is one of the useful ways for learners to understand how they work [35], however, training of GANs consists of thousands of iterations and each iteration also consists of several steps (as illustrated in Fig. 10). To address this problem, we decompose the training process into steps in multiple levels: epoch-, submodel-, and component-level.

#### 7.2.1 Manual Step Execution in Epoch-Level


Users can train a model for only one epoch, by clicking a button once. This epoch-level step execution is designed to help users track the training process to see how models update to find the optimum state through iterations. To use this feature, a user first clicks the step icon on top, which will shows three buttons. The last button ("Both") represents the training for one epoch. We describe the other two buttons' usage next.

#### 7.2.2 Manual Step Execution in Submodel-Level

A single epoch consists of training of a discriminator and generator, as illustrated in Fig. 10. GAN Lab allows users to update only the discriminator or generator. The experimentation of training only one of the two submodels is effective for users to understand how they work differently. For example, clicking the button for the discriminator changes the background grid while preserving the positions of fake samples. On the other hand, clicking the discriminator button moves the fake samples while fixing the background grid. To use this feature, users click the step icon first, then the three buttons will be shown. The first button is for training the discriminator; the second button is for the generator; and the last button is for training both submodels.

#### 7.2.3 Slow-Motion Mode in Component-Level

GAN Lab also provides the *slow-motion mode*, designed to help novices learn how each component of the model works to make updates within each epoch. It works differently from the manual step execution described in the two previous paragraphs. When users turn on this mode by clicking the icon on top during training, it slows down the speed of training. In addition, two similar lists of five steps are presented: one for updating the discriminator and the other for the generator, as depicted in Fig. 11. The five steps include (1) running the generator; (2) running the discriminator; (3) computing discriminator or generator loss; (4) computing gradients; and (5) updating the discriminator or generator. For every few seconds, it moves to the next step highlighting the corresponding model components with textual descriptions. For example, each of the five steps for the discriminator is highlighted one after another. At the same time, the whole training loop for the discriminator is also highlighted (i.e., edges colored by blue). Once the five steps are completed, it proceeds to the training of the generator, highlighting the training loop for the generator (i.e., purple edges) and executing its five steps. By following these training paths, users can learn how every component is used in training GANs.

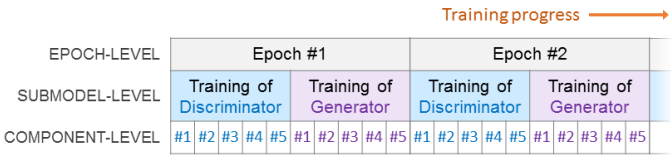



Fig. 10. Training typically involves of thousands of epochs (iterations). Each epoch includes training both discriminator and generator. GAN Lab supports step-by-step model training at different abstraction levels.

### 7.3 Browser-based Implementation for Deployment

GAN Lab is an open-source, web-based visualization tool. Anyone can access it using their modern web browsers without the need for installation or specialized backend. The demo is currently available at `https://poloclub.github.io/ganlab/`.

The tool is implemented in HTML and TypeScript (a typed version of JavaScript) with a few open-source JavaScript libraries: *TensorFlow.js*[6] is used for training and running models, which we will elaborate in detail in the next paragraph; *Polymer*[7] is used for building web applications; and *D3.js*[8] is used to visualize the model overview graph and layered distributions. The source code is available in `https://github.com/poloclub/ganlab/`.

**Using *TensorFlow.js* for model building and training.** GAN Lab runs locally on user's web browsers by using *TensorFlow.js Core* (formerly known as *deeplearn.js*), an in-browser GPU-accelerated deep learning library, developed by Google. The *TensorFlow.js* library uses WebGL to efficiently perform computation on browsers, required for training deep learning models. Not only does it enable rapid experimentation of the models, but also allows us to easily scale up deployment for the public. While most other implementations of GANs that use Python or other server-side languages would backfire when multiple users train models concurrently, our GAN models are trained in JavaScript, which means that that the models and their visualizations run locally on web browsers, enabling us to significantly broaden people's access to GAN Lab for learning about GANs.

## 8 Usage Scenarios

This section describes two example usage scenarios for GAN Lab, demonstrating how it may promote user learning of GANs. The scenarios highlight: (1) how beginners may learn key concepts for GANs by experimenting with the tool's visualizations and interactive features (Sect. 8.1); (2) how the tool may help practitioners discover advanced inner-workings of GANs, and how it can assist them to interactively attain optimal hyperparameters for reaching equilibrium between submodels (Sect. 8.2).

### 8.1 Beginners Learning Concepts and Training Procedure

Consider Alice, a data scientist at a technology company, who has basic knowledge about machine learning. Recently, she has started to learn about deep learning, and a few of the introductory articles she has been reading mention GANs. Excited about their potential, she wishes to use GAN Lab to interactively learn GANs.

**Becoming familiar with basic workflow.** When Alice launches GAN Lab in her web browser, she sees the *model overview graph*, which looks like a GAN architecture diagram that she has seen in her articles. By default, *real samples* are drawn from a 2D distribution that resembles a line. She clicks the *play* button on the tool bar. During the training, the movement of the fake samples in the *layered distribution* view attracts her attention. They keep moving towards the real samples.

**Using the slow-motion mode for tracking the training procedure.** Alice is aware that discriminator and generator take turns to train, but she is unsure of what that means. To see how training progresses, Alice clicks the slow-motion icon (Sect. 7.2.3) to enter the

[6]TensorFlow.js, `https://js.tensorflow.org/`
[7]Polymer, `https://www.polymer-project.org/`
[8]D3.js, `https://d3js.org/`

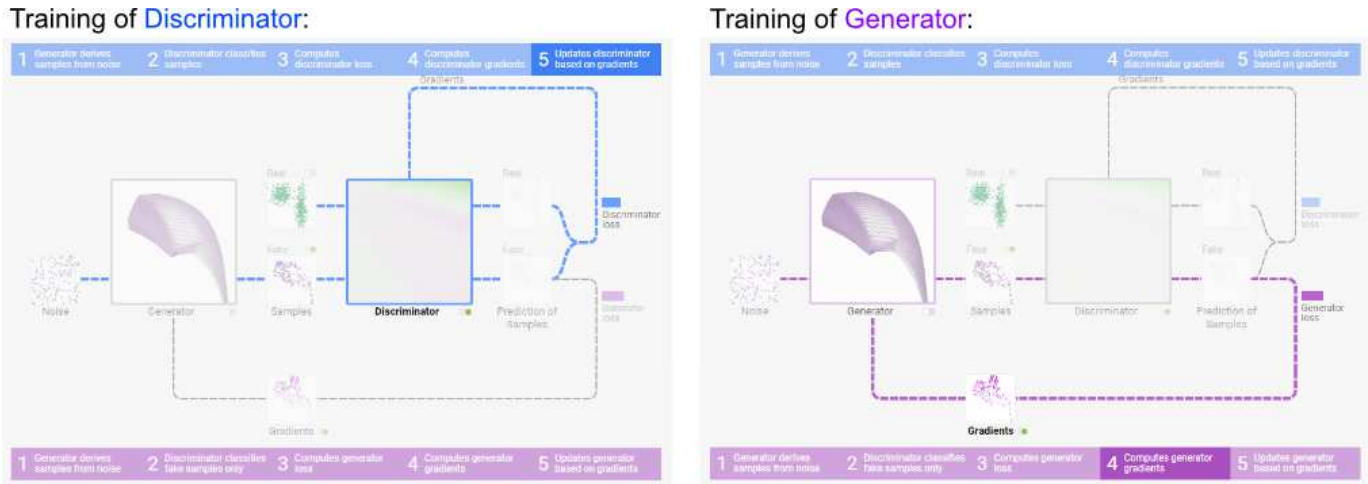



Fig. 11. The slow-motion mode slowly executes the model training process in a component level, in a step-by-step fashion. The steps are grouped into two lists, one for discriminator and the other for generator, each consisting of five steps.

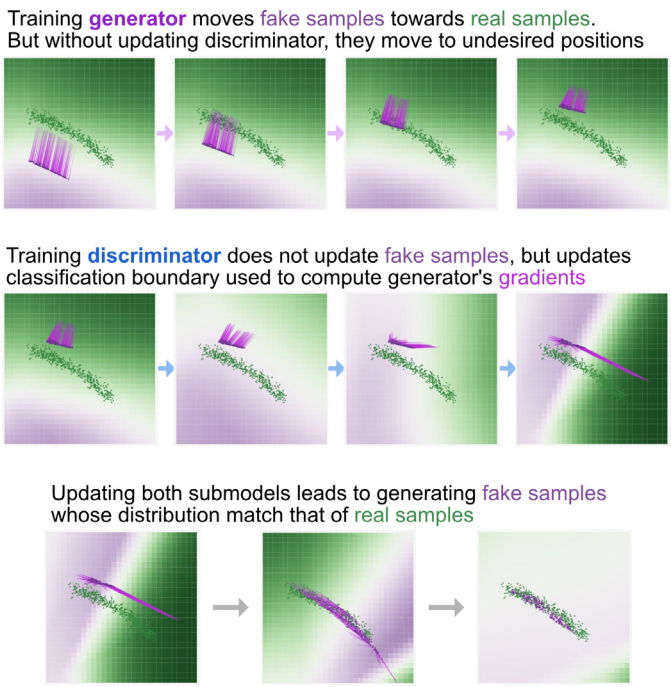



Fig. 12. Experimenting with manual step execution, to understand the interplay between discriminator and generator.

*slow-motion* training mode, which slows down the speed of training, and presents two lists of training steps, one for the discriminator, and another for the generator (see Fig. 11). She notices that in for every epoch, the discriminator is trained first, then the generator follows. The two models' training sequences seem very similar, but she discovers several key differences. For example, she is able to find that while discriminator's loss is computed by using both real and fake samples, only fake samples are used when computing the generator's loss.

**Understanding the different roles of discriminator and generator with the manual step execution.** While the slow-motion mode has helped her better understand the steps of the training process, Alice wonders how the discriminator and generator play a "game" to generate data distributions. To analyze the different effects for the discriminator and the generator, she would like to experiment with the two submodels using the manual step-by-step execution feature. She clicks the button (Sect. 7.2.1) to update the generator. Her initial clicks cause the fake samples to move towards the real samples, but as she clicks a few more times, the fake samples "overshoot," no longer matching real samples' distribution (Fig. 12, top row). She now realizes that the fake samples have moved towards regions where the colors of background grid cells are green, not directly towards the real samples. This leads Alice to hypothesize that training the discriminator is necessary for the generator to produce better fake samples. So, she switches to only training the discriminator, which does not reposition the fake samples, but the grid colors update (Fig. 12, second row) to correct a decision boundary that separates the real and fake samples. She believes that this new boundary helps guide the fake samples towards desirable regions where the real samples are located. This experiment helps her realize that updating both submodels is important for generation of better fake samples. Now she clicks the buttons for updating the discriminator and generator alternatively, which successfully creates a fake distribution that matches the real distribution. That is, the discriminator cannot distinguish between real and fake samples. (Fig. 12, last row).

### 8.2 Practitioners Experimenting with Hyperparameters

One of GAN Lab's key features is the interactive, dynamic training of GANs. Experimentation using GAN Lab could provide valuable practical experience in training GAN models even to experts. Consider Bob, a machine learning engineer at a technology company.

**Guiding models to find the optimum.** Bob launches GAN Lab and starts the training process. Fake samples quickly move towards real samples. However, as the training progresses, he notices that the fake samples oscillate around the real samples. Based on his previous experience, he believes this indicates that the *learning rates* may be set too high. He first decreases the value for the discriminator by using the dropdown menu, but the amount of oscillation becomes more severe. By checking the interface, he quickly realizes that there are two learning rates in GANs, so he reverts its value and decreases the generator's learning rate. After a few more iterations, the oscillation subsides and the distribution of the fake samples almost matches that for the real samples. This experimentation helps him understand the importance in balancing the power between the discriminator and generator.

**Understanding equilibrium between discriminator and generator.** Bob wonders what would happen if he perturbs the equilibrium between the discriminator and generator. That is, what if either submodel overpowers its complement. Looking into the model overview graph, he finds that some other hyperparameters also come in matched pairs, such as the number of training loops, one for the discriminator and the other for the generator. Originally, both numbers are set to 1 (i.e., the submodels run one training epoch in alternate sequence). Bob decides to increase discriminator's loop count 3 (i.e., 3 discriminator epochs, followed by 1 generator epoch, followed, and repeat). To his

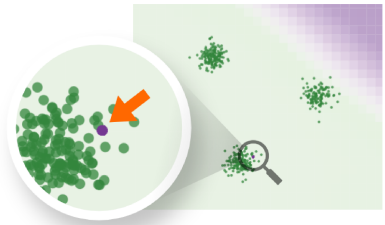



Fig. 13. Mode collapse, a common problem in GANs.

surprise, this "unbalanced" epoch setting (3 vs. 1) causes GAN to converge faster. Comparing this "unbalanced" setting with the original "balanced" (1 vs. 1) setting, Bob starts to understand that a more powerful discriminator can indeed accelerate training, because a stronger discriminator leads to stronger gradients for the generator, which in turns more quickly move the fake samples towards the real distribution, thus faster training convergence.

**Exploring mode collapse.** Bob would like to train a GAN to work with more complex data distributions. He picks one distribution that consists of three disjoint dense regions. He increases the number of layers for both the generator and discriminator, then clicks the play button. After a few seconds, all fake samples seem to have disappeared, as he can only see real samples. He temporarily hides the real samples (by toggling their visibility), thinking that they may be covering the fake samples. Then, he realizes that all fake samples have collapsed into a single point (as shown in Fig. 13). He does not know why this happens, and wonders if it is due to his hyperparameter choices. So he experiments with several other sets of hyperparameters, and observes the pattern that this happens more often when the generators and discriminators are set to use more layers and neurons. He consults the literature for possible causes, and learns that this is in fact a well-known problem in GANs, called *mode collapse*, whose exact cause is still an active research topic [8, 26]. Bob's observation through GAN Lab motivates him to study new variants of GANs, which may overcome this problem [8, 26].

## 9 Informed Design through Iterations

The current design of GAN Lab is the result of 11 months of investigation and development through many iterations. Below we share two key lessons learned from our experience.

The model overview graph is a crucial and effective feature that helps users develop mental models for GANs. Our early design (Fig. 14) did not include the overview graph. Instead, it displayed a long list of hyperparameters. While that design had all the necessary features for training GANs interactively, pilot users, including machine learning experts, commented that the tool was difficult to use and to interpret. The main reason is that, without an overview, users had to develop mental models for GANs (in their heads) to keep track of how the larger number of hyperparameters map to the different model components. This finding prompted us to add the model overview graph, inspired from common architecture diagrams for GANs, which helps users build mental models for the training process of GANs [24].

Animating the generator's transformation (Fig. 5) was helpful in helping users interpret the manifold visualization. Our early version only showed the transformed manifold (e.g., Fig. 5, rightmost). However, many users were puzzled by what they saw because, the manifold could be so severely distorted that they could not tell what its original shape was (a uniform 2D grid), thus they could not make the connection to realize that the manifold visualization was indeed representing the generator's output. We though about adding text to the interface to explain the manifold, but as GAN Lab is intended to be used as a standalone tool, we would like to keep the visual design compact, and we wanted to include textual descriptions only when necessary. Thus, we came up with the idea of visually explaining the transformation as an animated transition, which was immediately clear to all users.


Fig. 14. Early design of GAN Lab did not include a model overview graph that helps users develop mental models for GANs.

## 10 Limitations and Future Work

**Transferring user knowledge to higher dimensions.** Our main decision to use 2D datasets is to promote comprehension [34]. Through our tool, with 2D datasets, users can gain important knowledge about the overall training process of GANs, and specific details, such as how model components interact over time, how data flow through components, and how losses are recomputed to iteratively update components. These important concepts and knowledge are transferable to practical use cases of GANs where higher dimensional data are used (e.g., images). However, it remains an open research problem whether certain behaviors (e.g., mode collapse) that users may observe when experimenting with 2D datasets would be easily reproducible in higher dimensional datasets, where the larger number of parameters would lead to more-complex interactions and less-predictable results. We plan to conduct studies to develop deeper understanding of how and when such correspondence or mismatch may occur.

**Supporting image data.** To extend GAN Lab to support image data, some modifications and optimizations will be needed. Training on image data is often time consuming. To speed this up, pre-trained models may be provided to users so they can skip the earlier training steps. As for visual design, projection methods (e.g., t-SNE) may be used to replace some views in GAN Lab to visualize the distribution of generated image samples [41].

**Speed and scalability.** GAN Lab leverages TensorFlow.js to accelerate GAN training for browser-based deployment. For models with many parameters, this can be time consuming. In the short term, we believe rapid advances in JavaScript and hardware will shorten this by a good amount. A longer-term challenge to overcome is browsers' inability to render visualization and perform computation at the same time (i.e., single-threaded). Developers need to strike a good balance in planning and interleaving these actions, to maximize model computation speed and visual responsiveness.

**Supporting more GAN variants.** While GAN Lab currently implements a few different loss functions, other GAN variants exist [12]. Through open-sourcing GAN Lab, we look forward to seeing the community to build on GAN Lab to implement more variants, enabling users to interactively and visually compare them, easing the challenges in evaluating GANs [8]. Some variants may require minor design changes of the interface (e.g., adding new nodes to overview graph).

**In-depth evaluation of educational benefits.** Longitudinal studies of GAN Lab will help us better understand how it helps with learning of GANs. It would be particularly valuable to investigate how different types of users (e.g., students, practitioners, and researchers) would benefit from the tool.

## Acknowledgments

We thank Shan Carter, Daniel Smilkov, Google Big Picture Group and People + AI Research (PAIR), Georgia Tech Visualization Lab, and the anonymous reviewers for their feedback. This work was supported in part by NSF grants IIS-1563816, CNS-1704701, and TWC-1526254.

## REFERENCES


[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] S. Amershi, N. Arksey, G. Carenini, C. Conati, A. Mackworth, H. Maclaren, and D. Poole. Designing CIspace: Pedagogy and usability in a learning environment for AI. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 178–182. ACM, 2005. doi: 10.1145/1067445.1067495

[3] S. Bachthaler and D. Weiskopf. Continuous scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1428–1435, 2008. doi: 10.1109/TVCG.2008.119

[4] A. Bilal, A. Jourabloo, M. Ye, X. Liu, and L. Ren. Do convolutional neural networks learn class hierarchy? *IEEE Transactions on Visualization and Computer Graphics*, 24(1):152–162, 2018. doi: 10.1109/TVCG.2017.2744683

[5] S. Carter and M. Nielsen. Using artificial intelligence to augment human intelligence. *Distill*, 2017. doi: 10.23915/distill.00009

[6] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018. doi: 10.1109/MSP.2017.2765202

[7] G. Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006

[8] I. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[9] 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 (NIPS)*, pp. 2672–2680, 2014.

[10] P. J. Guo. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pp. 579–584. ACM, 2013. doi: 10.1145/2445196.2445368

[11] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *Proceedings of the 11th International Symposium on Visual Computing*, pp. 867–877, 2015. doi: 10.1007/978-3-319-27857-5_77

[12] A. Hindupur. The GAN Zoo: A list of all named GANs! `https://deephunt.in/the-gan-zoo-79597dc8c347`, 2017. Accessed: 2018-03-31.

[13] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics*, 2018. doi: 10.1109/TVCG.2018.2843369

[14] C. D. Hundhausen and J. L. Brown. What you see is what you code: A "live" algorithm development and visualization environment for novice learners. *Journal of Visual Languages & Computing*, 18(1):22–47, 2007. doi: 10.1016/j.jvlc.2006.03.002

[15] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002. doi: 10.1006/jvlc.2002.0237

[16] M. Kahng, P. Andrews, A. Kalro, and D. H. Chau. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018. doi: 10.1109/TVCG.2017.2744718

[17] A. Karpathy. ConvNetJS MNIST demo. `https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html`. Accessed: 2018-03-31.

[18] Y. LeCun. Answer to "what are some recent and potentially upcoming breakthroughs in deep learning?". `http://qr.ae/TU1FeA`, 2016. Accessed: 2018-03-31.

[19] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991. doi: 10.1109/18.61115

[20] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, 2018. doi: 10.1109/TVCG.2017.2744938

[21] 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. doi: 10.1109/TVCG.2016.2598831

[22] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1249–1268, 2017. doi: 10.1109/TVCG.2016.2640960

[23] 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. doi: 10.1016/j.visinf.2017.01.006

[24] Z. Liu and J. Stasko. Mental models, visual reasoning and interaction in information visualization: A top-down perspective. *IEEE Transactions on Visualization and Computer Thank Graphics*, 16(6):999–1008, 2010. doi: 10.1109/TVCG.2010.177

[25] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2813–2821. IEEE, 2017. doi: 10.1109/ICCV.2017.304

[26] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[27] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu. Understanding hidden memories of recurrent neural networks. In *IEEE Conference on Visual Analytics Science and Technology*, 2017.

[28] C. Olah. Neural networks, manifolds, and topology. `http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`, 2014. Accessed: 2018-03-31.

[29] C. Olah and S. Carter. Research debt. *Distill*, 2017. doi: 10.23915/distill.00005

[30] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. DeepEyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):98–108, 2018. doi: 10.1109/TVCG.2017.2744358

[31] B. Rosenberg. Machine learning crash course. `https://developers.googleblog.com/2018/03/machine-learning-crash-course.html`, 2018. Accessed: 2018-03-31.

[32] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2234–2242, 2016.

[33] P. Saraiya, C. A. Shaffer, D. S. McCrickard, and C. North. Effective features of algorithm visualizations. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, pp. 382–386. ACM, 2004. doi: 10.1145/971300.971432

[34] D. Schweitzer and W. Brown. Interactive visualization for the active learning classroom. *ACM SIGCSE Bulletin*, 39(1):208–212, 2007. doi: 10.1145/1227504.1227384

[35] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE)*, 10(3):9, 2010. doi: 10.1145/1821996.1821997

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

[37] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, 2018. doi: 10.1109/TVCG.2017.2744158

[38] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[39] B. Victor. Explorable explanations. `http://worrydream.com/ExplorableExplanations/`, 2011. Accessed: 2018-03-31.

[40] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3156–3164. IEEE, 2015. doi: 10.1109/CVPR.2015.7298935

[41] J. Wang, L. Gou, H. Yang, and H.-W. Shen. GANViz: A visual analytics approach to understand the adversarial game. *IEEE Transactions on Visualization and Computer Graphics*, 24(6):1905–1917, 2018. doi: 10.1109/TVCG.2018.2816223

[42] M. Wattenberg, F. Vigas, and I. Johnson. How to use t-SNE effectively. *Distill*, 2016. doi: 10.23915/distill.00002

[43] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in TensorFlow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1–12, 2018. doi: 10.1109/TVCG.2017.2744878

[44] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. In *Deep Learning Workshop at the 31st International Conference on Machine Learning (ICML)*, 2015.