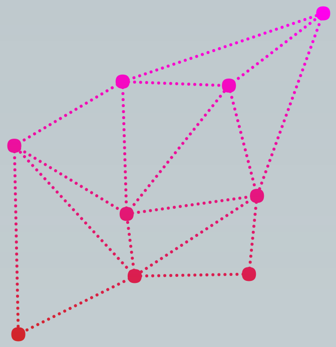
RESEARCH #1

# Generative Models



JUNE 16, 2016
ANDREJ KARPATHY, PIETER ABBEEL, GREG BROCKMAN, PETER CHEN, VICKI CHEUNG, ROCKY DUAN, IAN GOODFELLOW, DURK KINGMA, JONATHAN HO, REIN HOUTHOOFT, TIM SALIMANS, JOHN SCHULMAN, ILYA SUTSKEVER, AND WOJCIECH ZAREMBA.

Our first research results are now live: four projects that share a common theme of enhancing or using generative models, a branch of unsupervised learning techniques in machine learning.

In addition to describing our work, this post will tell you a bit more about generative models: what they are, why they are important, and where they might be going.

One of our core aspirations at OpenAI is to develop algorithms and techniques that endow computers with an understanding of our world.

It's easy to forget just how much you know about the world: you understand that it is made up of 3D environments, objects that move, collide, interact; people who walk, talk, and think; animals who graze, fly, run, or bark; monitors that display information encoded in language about the weather, who won a basketball game, or what happened in 1970.

This tremendous amount of information is out there and to a large extent easily accessible — either in the physical world of atoms or the digital world of bits. The only tricky part is to develop models and algorithms that can analyze and understand this treasure trove of data.

**Generative models are one of the most promising approaches towards this goal.** To train a generative model we first collect a large amount of data in some domain (e.g., think millions of images, sentences, or sounds, etc.) and then train a model to generate data like it. The intuition behind this approach follows a famous quote from Richard Feynman:

> *“What I cannot create, I do not understand.”*

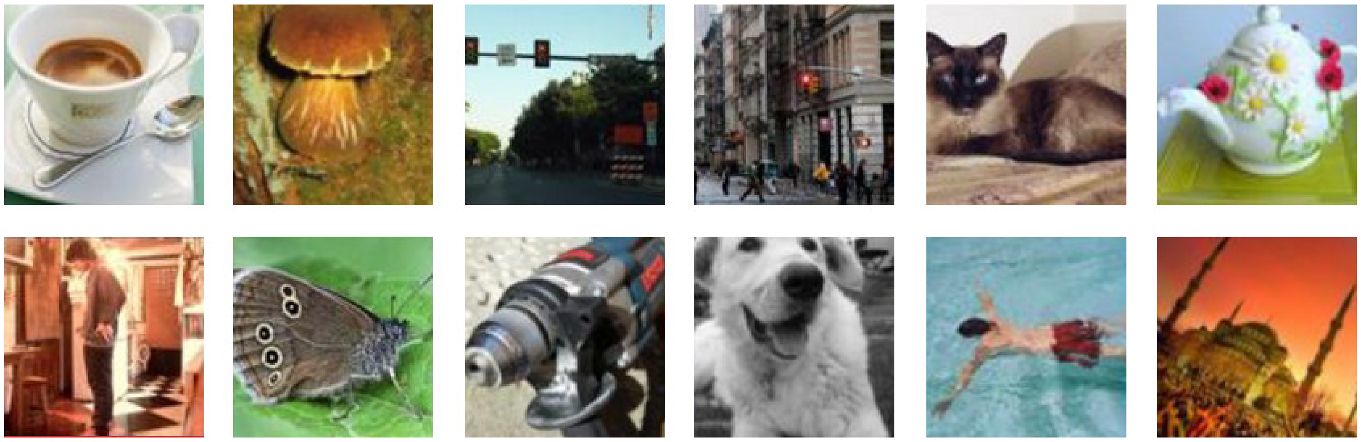
—Richard Feynman

The trick is that the neural networks we use as generative models have a number of parameters significantly smaller than the amount of data we train them on, so the models are forced to discover and efficiently internalize the essence of the data in order to generate it.

Generative models have many short-term applications. But in the long run, they hold the potential to automatically learn the natural features of a dataset, whether categories or dimensions or something else entirely.

## Generating images

Let’s make this more concrete with an example. Suppose we have some large collection of images, such as the 1.2 million images in the ImageNet dataset (but keep in mind that this could eventually be a large collection of images or videos from the internet or robots). If we resize each image to have width and height of 256 (as is commonly done), our dataset is one large `1,200,000x256x256x3` (about 200GB) block of pixels. Here are a few example images from this dataset:

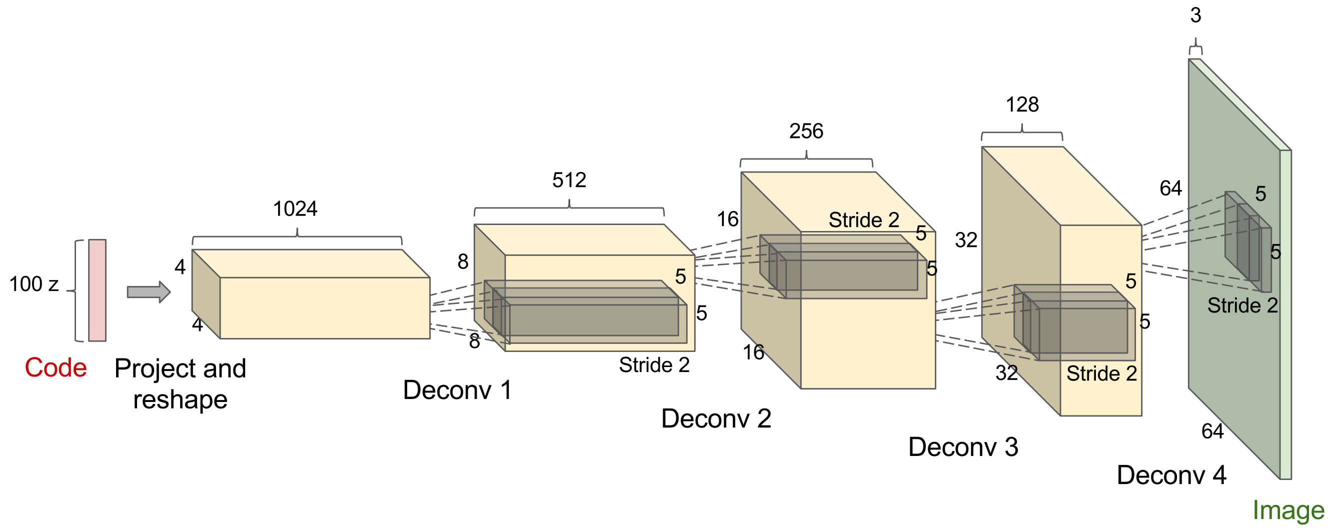


These images are examples of what our visual world looks like and we refer to these as "samples from the true data distribution". We now construct our generative model which we would like to train to generate images like this from scratch. Concretely, a generative model in this case could be one large neural scratch. Concretely, a generative model in this case could be one large neural network that outputs images and we refer to these as "samples from the model".

## DCGAN

One such recent model is the DCGAN network from Radford et al. (shown below). This network takes as input 100 random numbers drawn from a uniform distribution (we refer to these as a *code*, or *latent variables*, in red) and outputs an image (in this case 64x64x3 images on the right, in green). As the code is changed incrementally, the generated images do too — this shows the model has learned features to describe how the world looks, rather than just memorizing some examples.

The network (in yellow) is made up of standard convolutional neural network components, such as deconvolutional layers (reverse of convolutional layers), fully connected layers, etc.:



DCGAN is initialized with random weights, so a random code plugged into the network would generate a completely random image. However, as you might imagine, the network has millions of parameters that we can tweak, and the goal is to find a setting of these parameters that makes samples generated from random codes look like the training data. Or to put it another way, we want the model distribution to match the true data distribution in the space of images.

---

## Training a generative model

Suppose that we used a newly-initialized network to generate 200 images, each time starting with a different random code. The question is: how should we adjust the network's parameters to encourage it to produce slightly more believable samples in the future? Notice that we're not in a simple supervised setting and don't have any explicit *desired targets* for our 200 generated images; we merely want them to look real.

One clever approach around this problem is to follow the Generative Adversarial Network (GAN) approach. Here we introduce a second *discriminator* network (usually a standard convolutional neural network) that tries to classify if an input image is real or generated. For instance, we could feed the 200 generated images and 200 real images into the discriminator and train it as a standard classifier to distinguish between the two sources. But in addition to that — and here's the trick — we can also backpropagate through both the discriminator and the generator to find how we should change the generator's parameters to make its 200 samples slightly more confusing for the discriminator. These two networks are therefore locked in a battle: the discriminator is trying to distinguish real images from fake images and the generator is trying to create images that make the discriminator think they are real. In the end, the generator network is outputting images that are indistinguishable from real images for the discriminator.

There are a few other approaches to matching these distributions which we will discuss briefly below. But before we get there below are two animations that show samples from a generative model to give you a visual sense for the training process. In both cases the samples from the generator start out noisy and chaotic, and over time converge to have more plausible image statistics:

*VAE learning to generate images (log time)*

[illegible]

*GAN learning to generate images (linear time)*

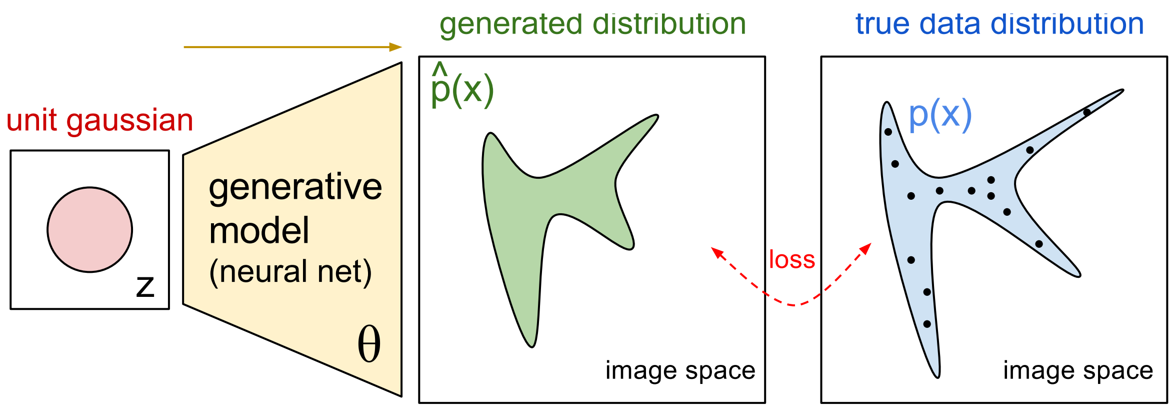
This is exciting — these neural networks are learning what the visual world looks like! These models usually have only about 100 million parameters, so a network trained on ImageNet has to (lossily) compress 200GB of pixel data into 100MB of weights. This incentivizes it to discover the most salient features of the data: for example, it will likely learn that pixels nearby are likely to have the same color, or that the world is made up of horizontal or vertical edges, or blobs of different colors. Eventually, the model may discover many more complex regularities: that there are certain types of backgrounds, objects, textures, that they occur in certain likely arrangements, or that they transform in certain ways over time in videos, etc.

---

## More general formulation

Mathematically, we think about a dataset of examples $x_1, \ldots, x_n$ as samples from a true data distribution $p(x)$. In the example image below, the blue region shows the part of the image space that, with a high probability (over some threshold) contains real images, and black dots indicate our data points (each is one image in our dataset). Now, our model also describes a distribution $\hat{p}_\theta(x)$ (green) that is defined implicitly by taking points from a unit Gaussian distribution (red) and mapping them through a (deterministic) neural network — our generative model (yellow).

Our network is a function with parameters $\theta$, and tweaking these parameters will tweak the generated distribution of images. Our goal then is to find parameters $\theta$ that produce a distribution that closely matches the true data distribution (for example, by having a small KL divergence loss). Therefore, you can imagine the green distribution starting out random and then the training process iteratively changing the parameters $\theta$ to stretch and squeeze it to better match the blue distribution.



## Three approaches to generative models

Most generative models have this basic setup, but differ in the details. Here are three popular examples of generative model approaches to give you a sense of the variation:

- Generative Adversarial Networks (GANs), which we already discussed above, pose the training process as a game between two separate networks: a generator network (as seen above) and a second discriminative network that tries to classify samples as either coming from the true distribution $p(x)$ or the model distribution $\hat{p}(x)$. Every time the discriminator notices a difference between the two distributions the generator adjusts its parameters slightly to make it go away, until at the end (in theory) the generator exactly reproduces the true data distribution and the discriminator is guessing at random, unable to find a difference.
- Variational Autoencoders (VAEs) allow us to formalize this problem in the framework of probabilistic graphical models where we are maximizing a lower bound on the log likelihood of the data.
- Autoregressive models such as PixelRNN instead train a network that models the conditional distribution of every individual pixel given previous pixels (to the left and to the top). This is similar to plugging the pixels of the

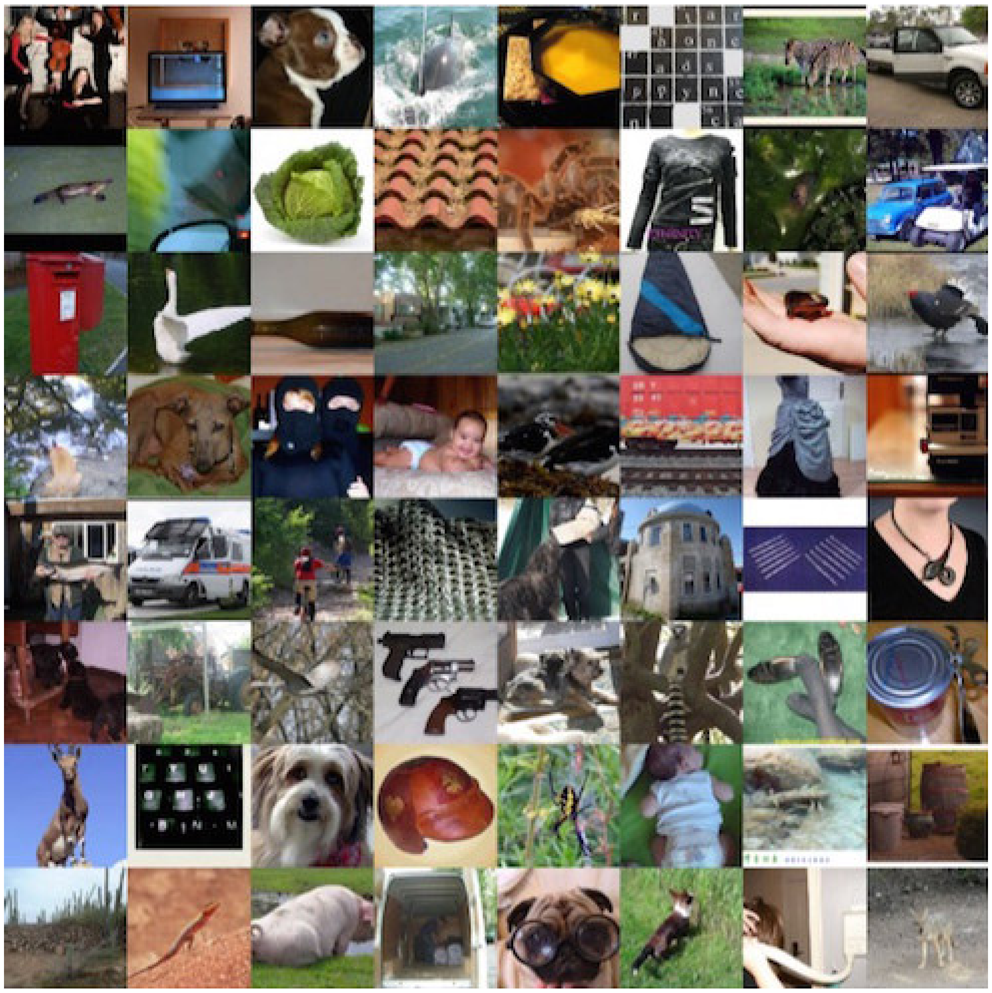
image into a char-rnn, but the RNNs run both horizontally and vertically over the image instead of just a 1D sequence of characters.

All of these approaches have their pros and cons. For example, Variational Autoencoders allow us to perform both learning and efficient Bayesian inference in sophisticated probabilistic graphical models with latent variables (e.g. see DRAW, or Attend Infer Repeat for hints of recent relatively complex models). However, their generated samples tend to be slightly blurry. GANs currently generate the sharpest images but they are more difficult to optimize due to unstable training dynamics. PixelRNNs have a very simple and stable training process (softmax loss) and currently give the best log likelihoods (that is, plausibility of the generated data). However, they are relatively inefficient during sampling and don't easily provide simple low-dimensional *codes* for images. All of these models are active areas of research and we are eager to see how they develop in the future!

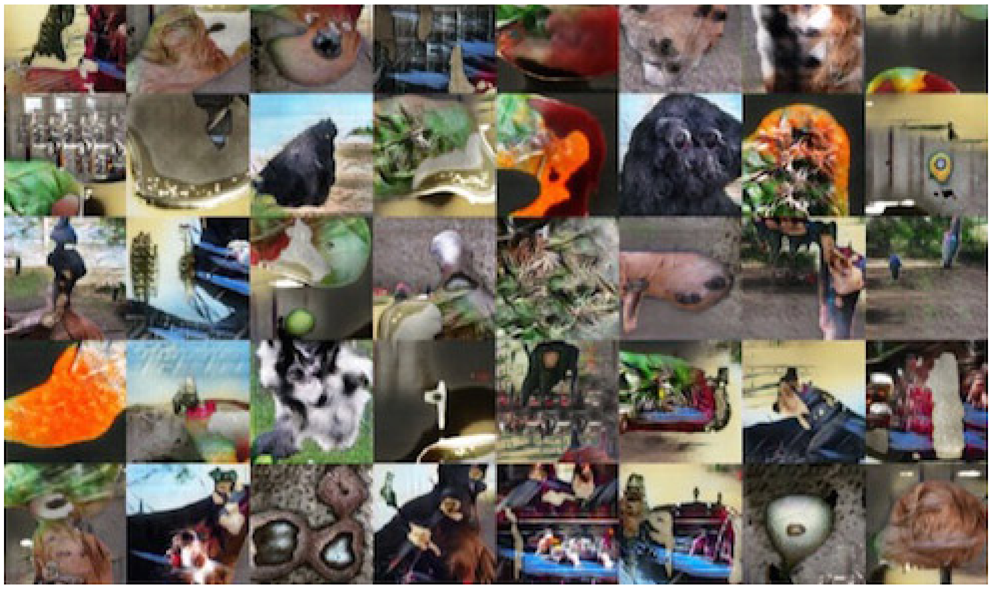
## Our recent contributions

We're quite excited about generative models at OpenAI, and have just released four projects that advance the state of the art. For each of these contributions we are also releasing a technical report and source code.

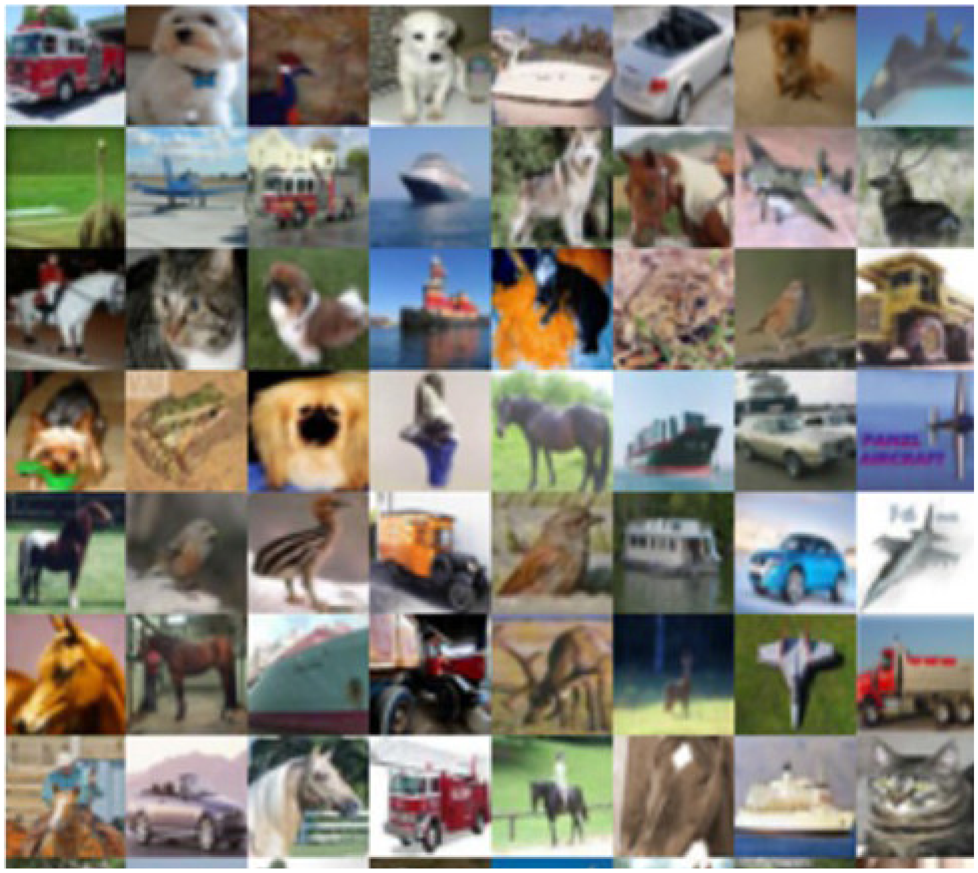
**Improving GANs** (code). First, as mentioned above GANs are a very promising family of generative models because, unlike other methods, they produce very clean and sharp images and learn codes that contain valuable information about these textures. However, GANs are formulated as a game between two networks and it is important (and tricky!) to keep them in balance: for example, they can oscillate between solutions, or the generator has a tendency to collapse. In this work, Tim Salimans, Ian Goodfellow, Wojciech Zaremba and colleagues have introduced a few new techniques for making GAN training more stable. These techniques allow us to scale up GANs and obtain nice 128x128 ImageNet samples:

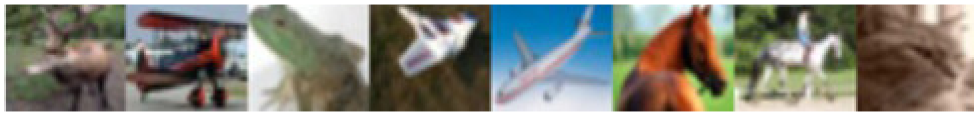


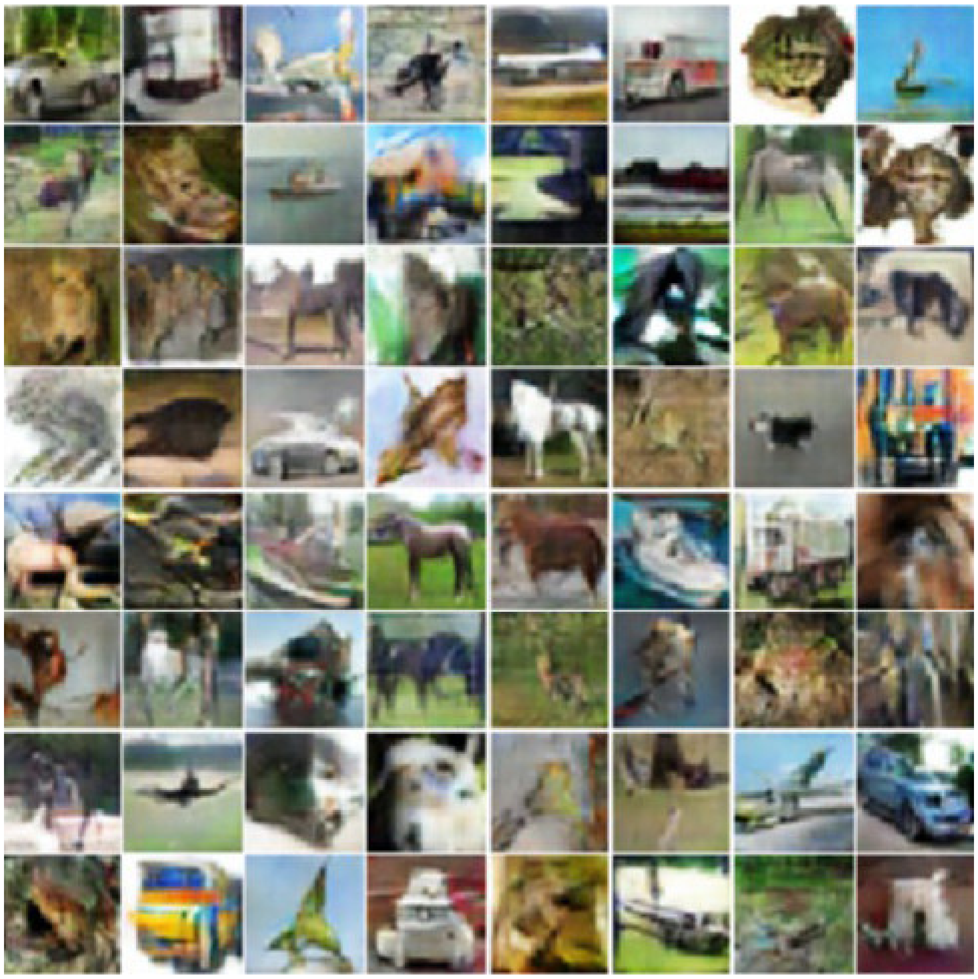
*Real images (ImageNet)*


*Generated images*

Our CIFAR-10 samples also look very sharp - Amazon Mechanical Turk workers can distinguish our samples from real data with an error rate of 21.3% (50% would be random guessing):




*Real images (CIFAR-10)*

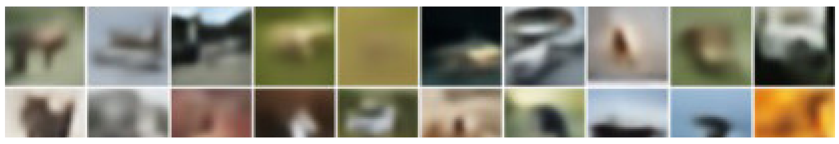

*Generated images*

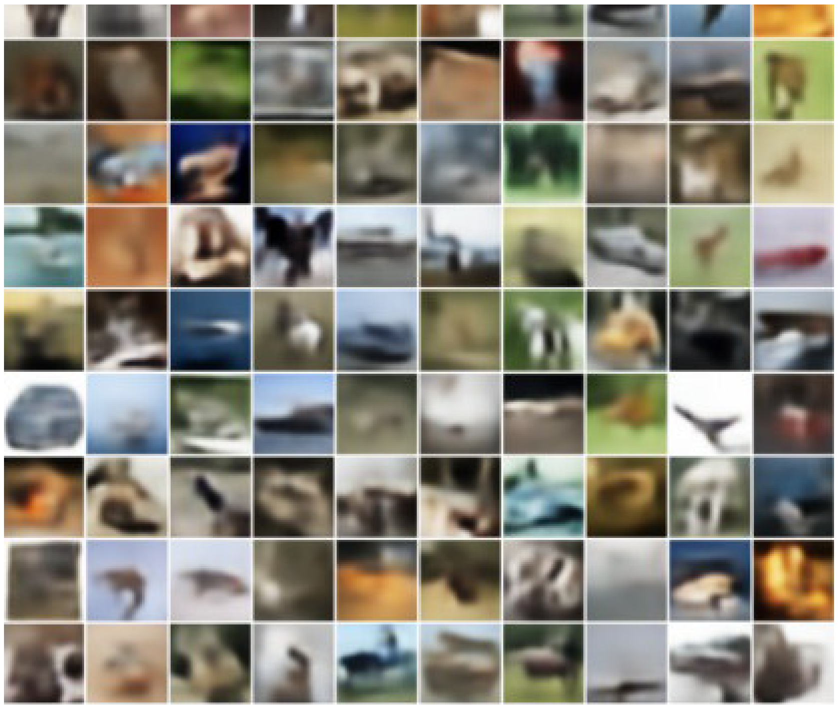
In addition to generating pretty pictures, we introduce an approach for semi-supervised learning with GANs that involves the discriminator producing an additional output indicating the label of the input. This approach allows us to obtain state of the art results on MNIST, SVHN, and CIFAR-10 in settings with very few labeled examples. On MNIST, for example, we achieve 99.14% accuracy with only 10 labeled examples per class with a fully connected neural network — a result that's very close to the best known results with fully supervised approaches using all 60,000 labeled examples. This is very promising because labeled examples can be quite expensive to obtain in practice.

Generative Adversarial Networks are a relatively new model (introduced only two years ago) and we expect to see more rapid progress in further improving the stability of these models during training.

**Improving VAEs** (code). In this work Durk Kingma and Tim Salimans introduce a flexible and computationally scalable method for improving the accuracy of variational inference. In particular, most VAEs have so far been trained using crude approximate posteriors, where every latent variable is independent. Recent extensions have addressed this problem by conditioning each latent variable on the others before it in a chain, but this is computationally inefficient due to the introduced sequential dependencies. The core contribution of this work, termed *inverse autoregressive flow* (IAF), is a new approach that, unlike previous work, allows us to parallelize the computation of rich approximate posteriors, and make them almost arbitrarily flexible.

We show some example 32x32 image samples from the model in the image below, on the right. On the left are earlier samples from the DRAW model for comparison (vanilla VAE samples would look even worse and more blurry). The DRAW model was published only one year ago, highlighting again the rapid progress being made in training generative models.

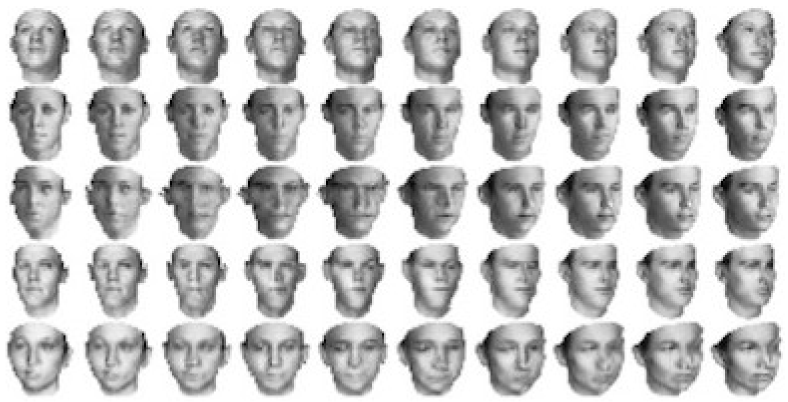


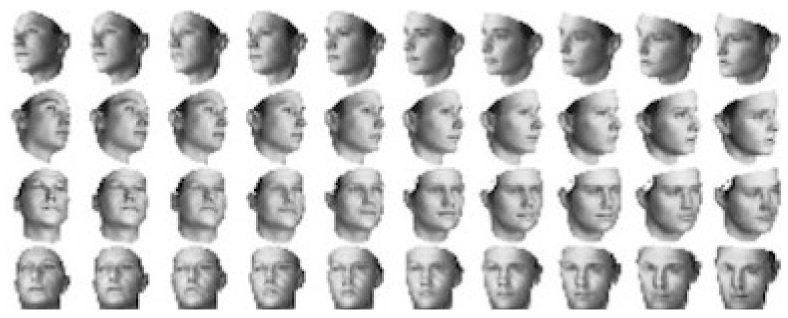


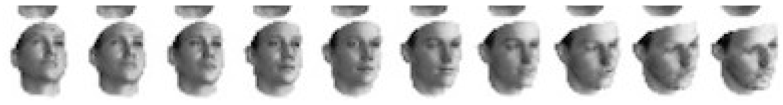
*Generated from DRAW model*

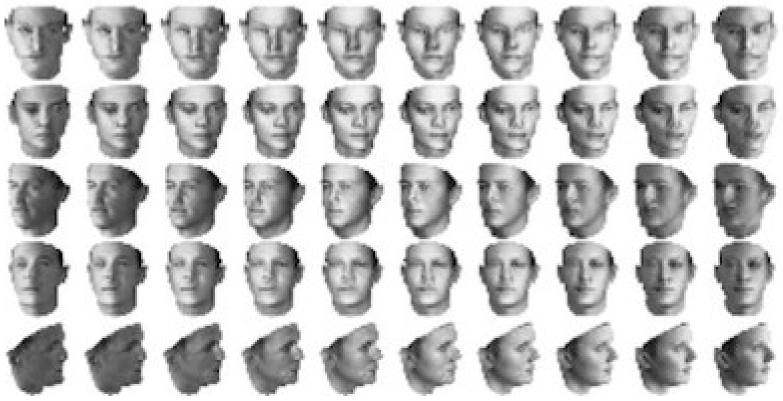
*Generated from a VAE trained with IAF*

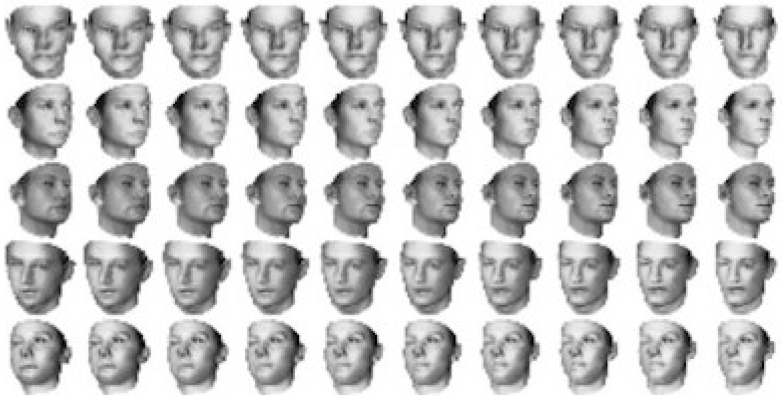
**InfoGAN** (code). Peter Chen and colleagues introduce InfoGAN — an extension of GAN that learns disentangled and interpretable representations for images. A regular GAN achieves the objective of reproducing the data distribution in the model, but the layout and organization of the code space is *underspecified* — there are many possible solutions to mapping the unit Gaussian to images and the one we end up with might be intricate and highly entangled. The InfoGAN imposes additional structure on this space by adding new objectives that involve maximizing the mutual information between small subsets of the representation variables and the observation. This approach provides quite remarkable results. For example, in the images of 3D faces below we vary one continuous dimension of the code, keeping all others fixed. It's clear from the five provided examples (along each row) that the resulting dimensions in the code capture interpretable dimensions, and that the model has perhaps *understood* that there are camera angles, facial variations, etc., without having been told that these features exist and are important:


*(a) Azimuth (pose)*




*(b) Elevation*

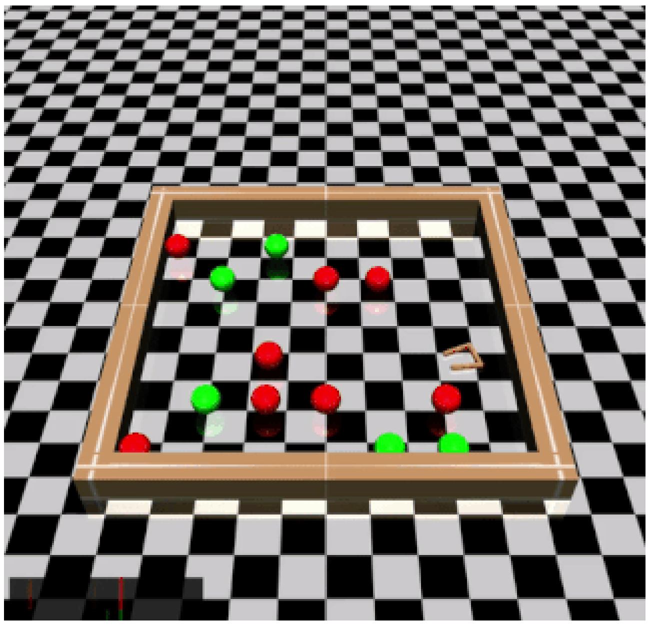

*(c) Lighting*


*(d) Wide or Narrow*

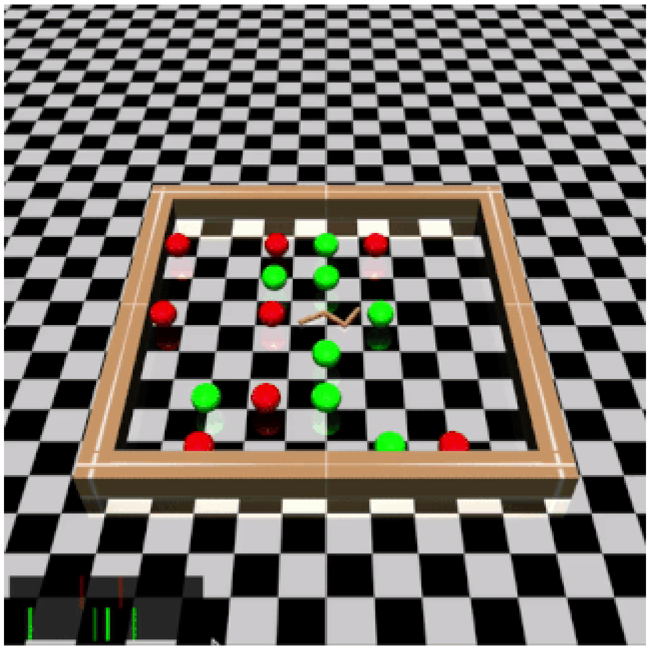
We also note that nice, disentangled representations have been achieved before (such as with DC-IGN by Kulkarni et al.), but these approaches rely on additional supervision, while our approach is entirely unsupervised.

The next two recent projects are in a reinforcement learning (RL) setting (another area of focus at OpenAI), but they both involve a generative model component.

**Curiosity-driven Exploration in Deep Reinforcement Learning via Bayesian Neural Networks** (code). Efficient exploration in high-dimensional and continuous spaces is presently an unsolved challenge in reinforcement learning. Without effective exploration methods our agents thrash around until they randomly stumble into rewarding situations. This is sufficient in many simple toy tasks but inadequate if we wish to apply these algorithms to complex settings with high-dimensional action spaces, as is common in robotics. In this paper, Rein Houthooft and colleagues propose VIME, a practical approach to exploration using uncertainty on generative models. VIME makes the agent self-motivated; it actively seeks out surprising state-actions. We show that VIME can improve a range of policy search methods and makes significant progress on more realistic tasks with sparse rewards (e.g. scenarios in which the agent has to learn locomotion primitives without any guidance).

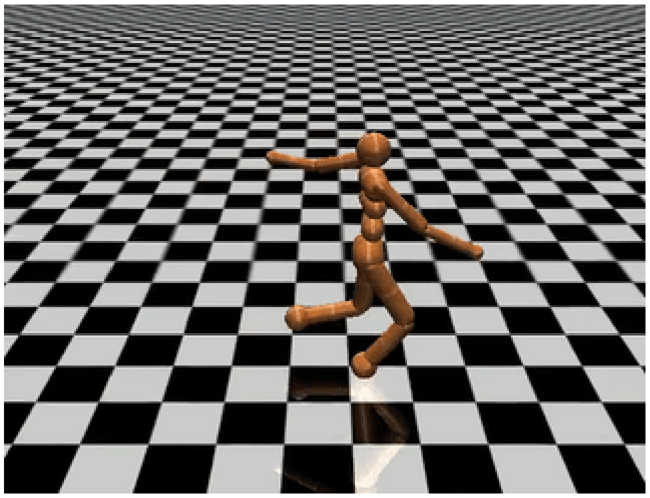


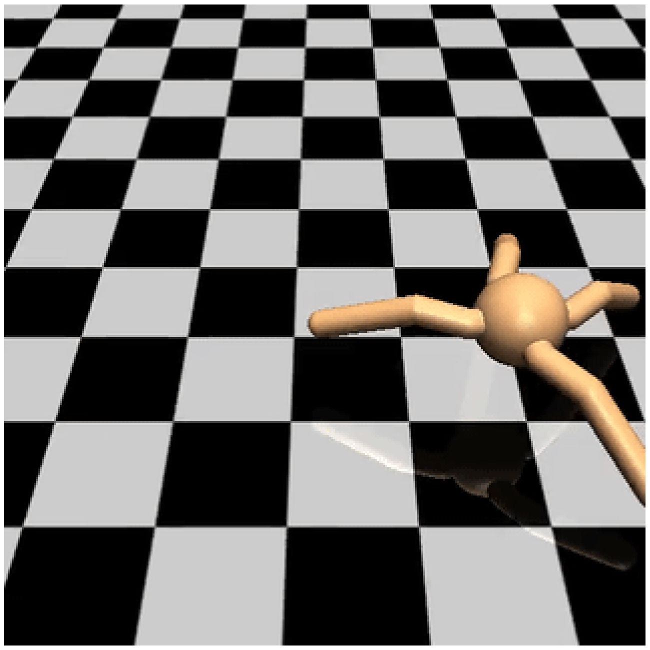
Policy trained with VIME


Policy trained with naive exploration

Finally, we would like to include a bonus fifth project: Generative Adversarial Imitation Learning (code), in which Jonathan Ho and colleagues present a new approach for *imitation learning*. Jonathan Ho is joining us at OpenAI as a summer intern. He did most of this work at Stanford but we include it here as a related and highly creative application of GANs to RL. The standard reinforcement learning setting usually requires one to design a reward function that describes the desired behavior of the agent. However, in practice this can sometimes involve expensive trial-and-error process to get the details right. In contrast, in imitation learning the agent learns from example demonstrations

(for example provided by teleoperation in robotics), eliminating the need to design a reward function.





Popular imitation approaches involve a two-stage pipeline: first learning a reward function, then running RL on that reward. Such a pipeline can be slow, and because it's indirect, it is hard to guarantee that the resulting policy works well. This work shows how one can directly extract policies from data via a connection to GANs. As a result, this approach can be used to learn policies from expert demonstrations (without rewards) on hard OpenAI Gym environments, such as Ant and Humanoid.

## Going forward

Generative models are a rapidly advancing area of research. As we continue to advance these models and scale up the training and the datasets, we can expect to eventually generate samples that depict entirely plausible images or videos. This may by itself find use in multiple applications, such as on-demand generated art, or Photoshop++ commands such as "make my smile wider". Additional presently known applications include image denoising, inpainting, super-resolution, structured prediction, exploration in reinforcement learning, and neural network pretraining in cases where labeled data is expensive.

However, the deeper promise of this work is that, in the process of training generative models, we will endow the computer with an understanding of the world and what it is made up of.