Deep Generative Models

Deep Learning Decal

Hosted by Machine Learning at Berkeley

ML@B

Agenda

Background

Directed Generative Nets

- Variational Autoencoders
- Auto-Regressive Models
- Generative Adversarial Networks

Evaluation of Generative Models

Recent Work and Food for Thought

Background

Where are we?

ML@B

- BMs are a powerful class of models, but can be hard to train
- BMs are, according to Yoshua Bengio, fringe research
- Let's take a step back

Big world, big data

ML@B

- We often forget how much we know about the world
- The plethora of information is out there, and usually easy to access
- The hard part is creating models and algorithms which can draw information from the bottomless big data treasure chest

Generative models

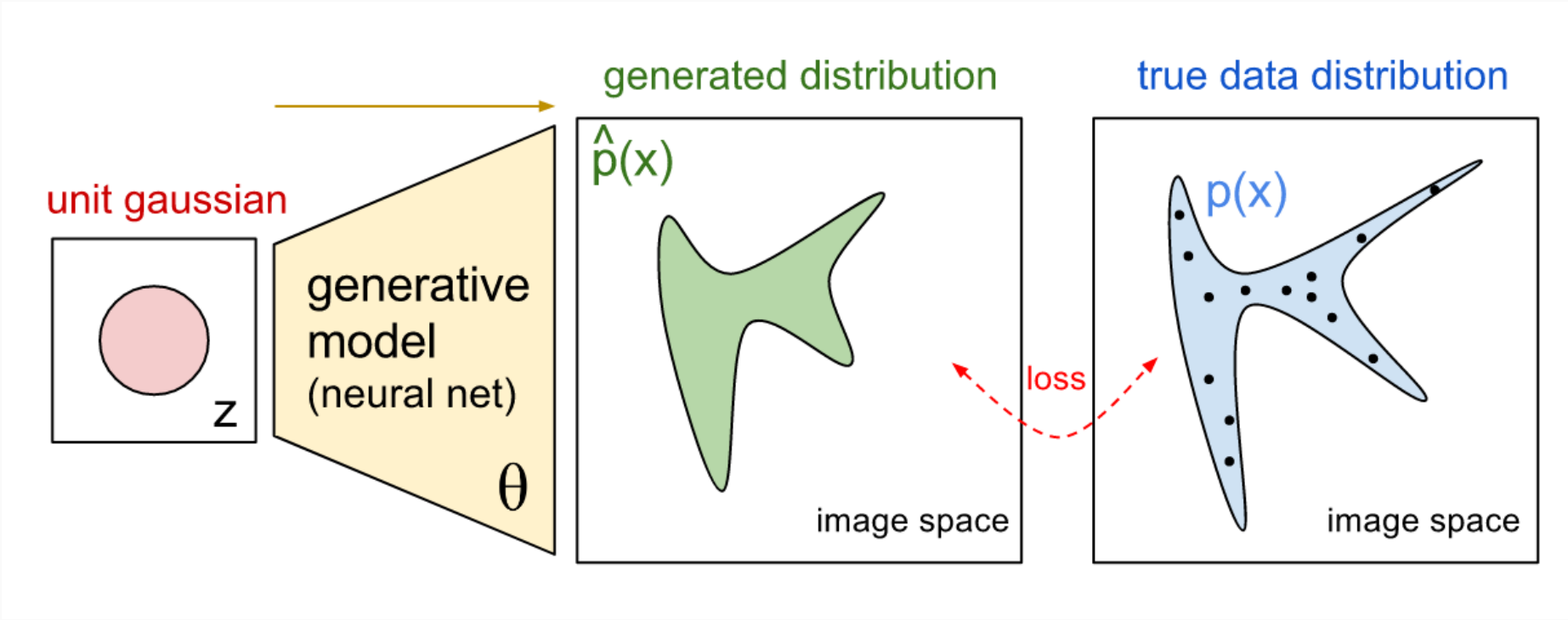
- Generative models are one of the most promising approaches towards the goal of understanding big data

"What I cannot create, I do not understand"

- Richard Feynman

- Neural network-based generative models have a number of parameters much smaller than the amount of data we have
- They are forced to find and efficiently internalize the meaning of the data if they are to generate it

General formulation of generative modeling



Generative models

ML@B

- There are many short-term applications:
  - Image generation, denoising, in-painting, and super-resolution
  - Exploration in RL contexts
  - Pre-training for supervised tasks
  - Many more...
- A **deeper** promise is while training generative models, we grant computers the ability to understand the world and all of its intricacies

Generative models in self-driving cars

ML@B

- Suppose a self-driving car is driving and sees a foot sticking out in front of a bus that is being passed by another car
- The car's neural network might generate a weak but noticeable signal that a human could be in the car's path
- Higher up in the car's system of understanding, the signal may be amplified and cycled back down to the visual generator
- An estimated size, shape, orientation, and speed of the pedestrian can be used to avoid a possible collision

Let’s dig in...

Back-propagation through random operations

- Traditional neural networks implement some deterministic transformation of the input variables **x**
- In generative models, we'd like to extend neural networks to implement stochastic transformations of **x**
- Training them utilizes the method of **stochastic back-propagation**

Directed Generative Nets

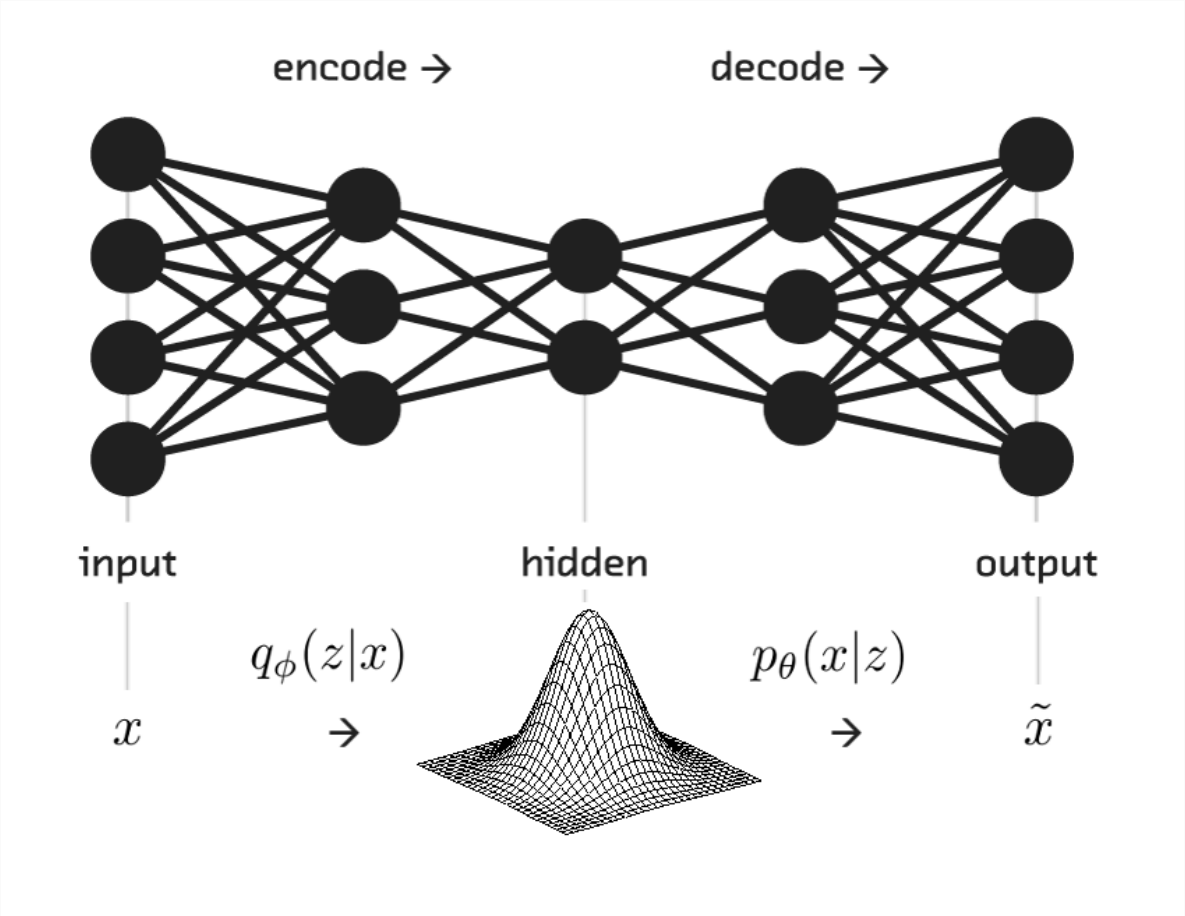
Three approaches to generative modeling

ML@B

- Variational autoencoders
  - Maximizes a lower bound on the log likelihood of the data
- Autoregressive models
  - Model the conditional distribution of every individual pixel given previous pixels
- Generative adversarial networks
  - Pose the training process as a game between two separate networks: a generator network and a second discriminator network

Variational autoencoders (VAE)

- Kingma, 2013; Rezende et al, 2014
- A directed model that uses learned approximate inference and can be trained purely with gradient-based methods



VAEs step by step

ML@B

To generate a sample:

- Step 1: the VAE draws a sample $\mathbf{z}$ from the code distribution $p_{model}(\mathbf{z})$
- Step 2: the sample is run through a differentiable generator network $g(\mathbf{z})$
- Step 3: $\mathbf{x}$ is sampeld from a distribution $p_{model}(\mathbf{x}; g(\mathbf{z})) = p_{model}(\mathbf{x}|\mathbf{z})$
- During training, the approximate inference network (or encoder) $q(\mathbf{z}|\mathbf{x})$ is used to obtain $\mathbf{z}$ and $p_{model}(\mathbf{x}|\mathbf{z})$ is then viewed as a decoder network

VAEs: key insight

- VAEs may be trained by maximizing the variational lower bound $\mathcal{L}(q)$ associated with data point **x**

$$\begin{aligned}\mathcal{L}(q) &= E_{z\sim q(z|x)} log p_{model}(z, x) + \mathcal{H}(q(z|x)) \\ &= E_{z\sim q(z|x)} log p_{model}(x|z) - D_{KL}(q(z|x)||p_{model}(z)) \\ &\leqslant log p_{model}(x)\end{aligned}$$

** All expectations can be approximated by Monte Carlo sampling

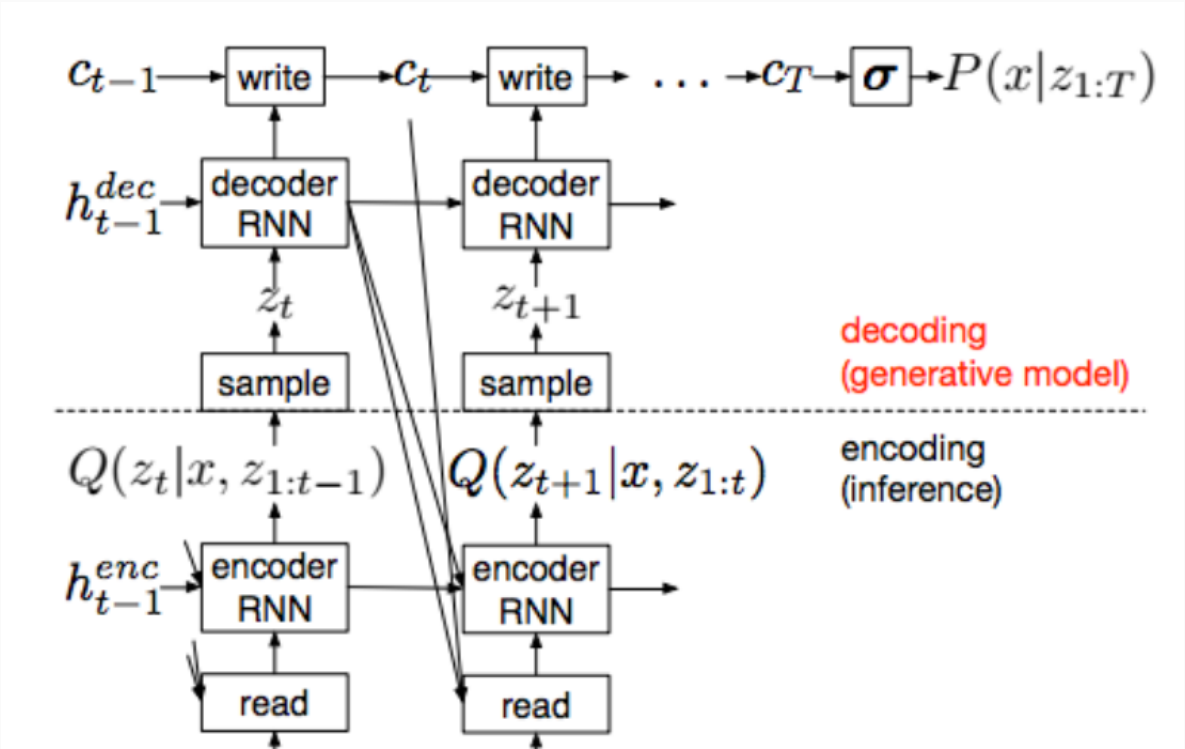
VAEs

ML@B

- Theoretically pleasing, easy to implement, and obtains excellent results on state of the art approaches!
- Drawback: samples from VAEs trained on images tend to be blurry
- Drawback: they tend to use only a small subset of the dimensions of $\mathbf{z}$
- Advantage (over BMs): very easy and straightforward to extend to a wide range of model architectures

A sophisticated VAE: DRAW

- Deep recurrent attention writer (Gregor et al, 2015)
- Uses a recurrent encoder and recurrent decoder combined with an attention mechanism
- Generation process consists of sequentially visiting different small image patches and drawing the values of the pixels at those points



Importance weighted autoencoder

- Burda et al extended the VAE framework to maximize the importance weighted autoencoder objective:

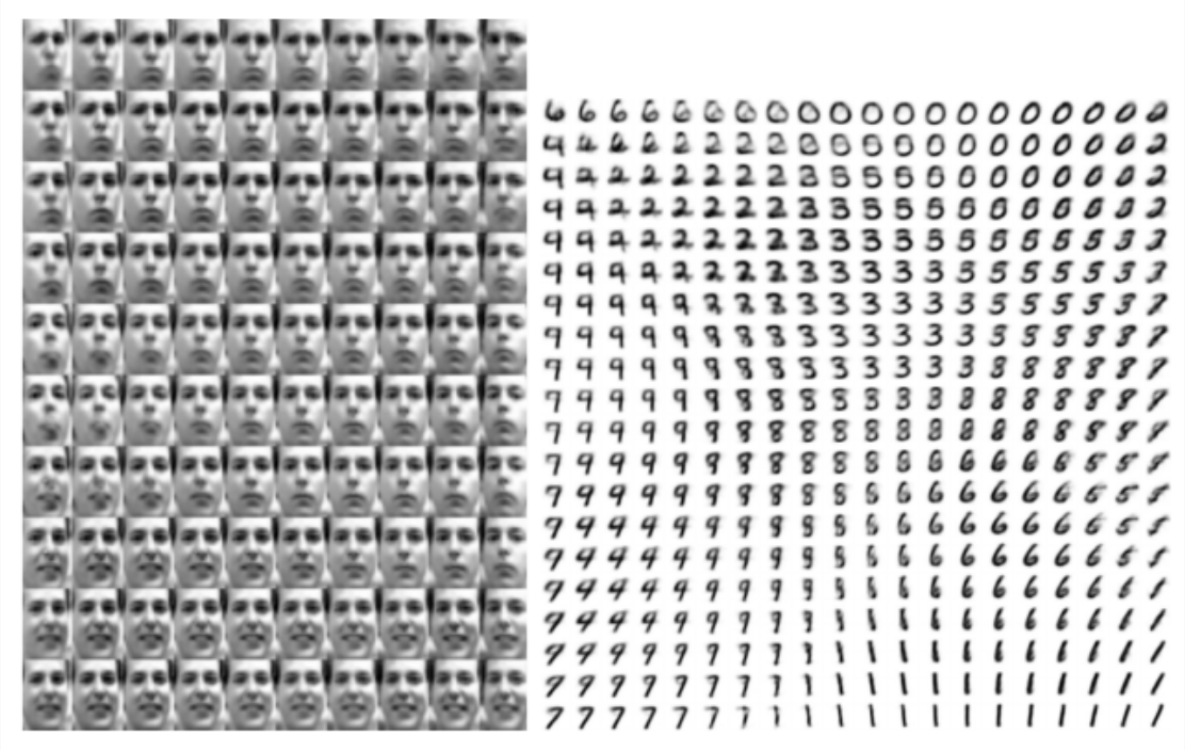
$$\mathcal{L}_k(x, q) = E_{z^{(1)},\ldots,z^{(k)} \sim q(z|x)}[log \frac{1}{k} \sum_{i=1}^{k} \frac{p_{model}(x, z^{(i)})}{q(z^{(i)}|x)}]$$

- Can be interpreted as forming an estimate of the true $logp_{model}(x)$ using importance sampling of z from proposal distribution $q(z|x)$

VAEs as a manifold learning algorithm

ML@B

- Simultaneously training a parametric encoder in combination with the generator network forces the model to learn a predictable coordinate system which the encoder can capture

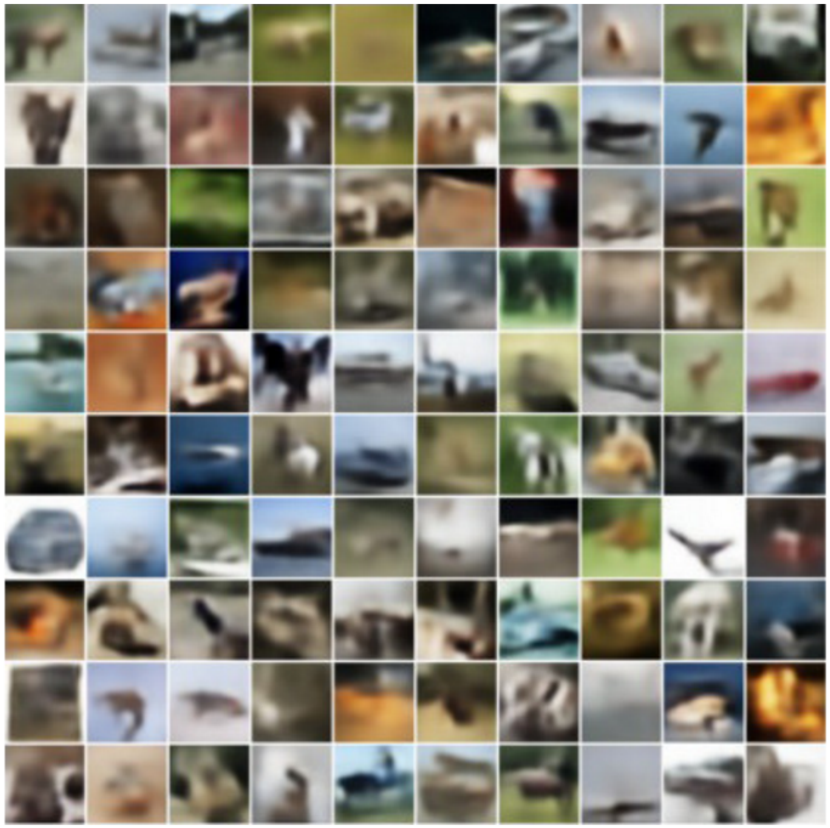


Improving VAEs

ML@B

- Kingma and Salimans introduced a flexible and computationally scalable method for improving the accuracy of variational inference
- Most VAEs have been trained using approximate posteriors, where every latent variable is independent
- Inverse autoregressive flow (IAF) allows us to parallelize the computation of rich approximate posteriors and make them very flexible

DRAW (2015) vs. IAF (2016)

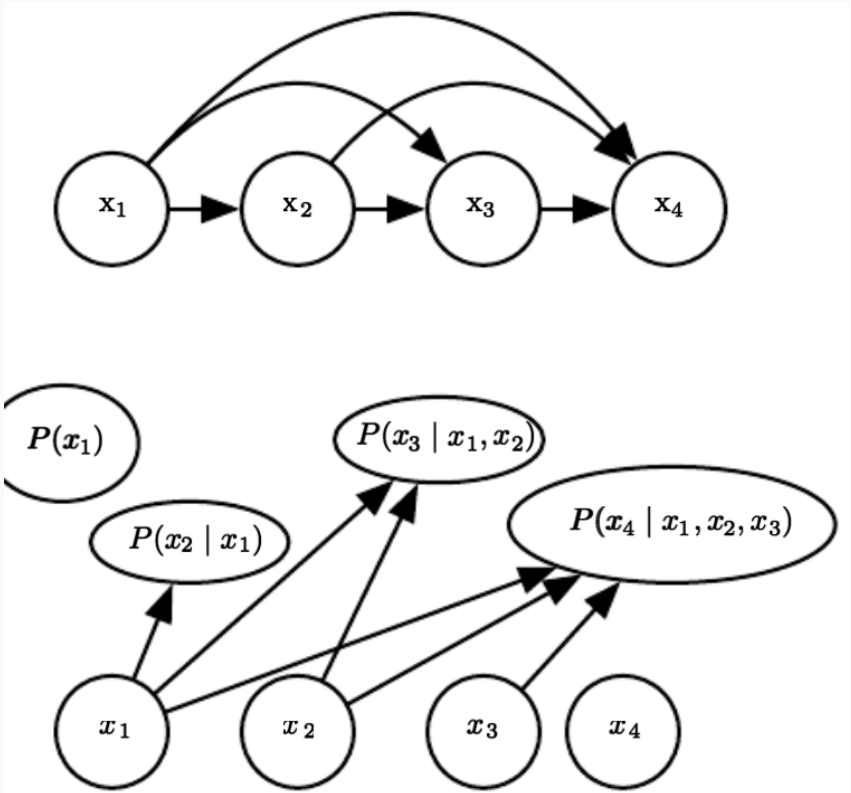


Generated from a DRAW model

Generated from a VAE trained with IAF

Auto-regressive networks

- Directed probabilistic models with no latent random variables
- Conditional distributions are represented by neural networks
- Their graph structure is the complete graph
- They decompose a joint probability over the observed variables using the chain rule of probability



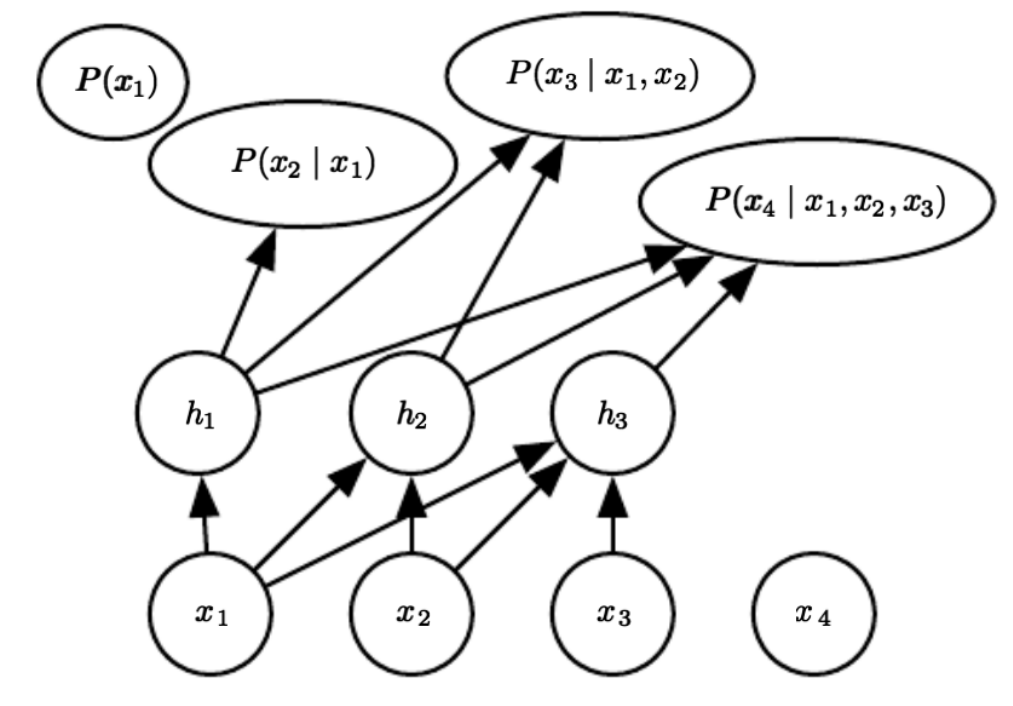
Linear auto-regressive networks

ML@B

- Simplest auto-regressive network (no hidden units, no parameter sharing)
- Each $P(x_i|x_{i-1}, ..., x_1)$ is parameterized as a linear model (linear reg. for continuous data, logistic reg. for binary data, softmax reg. for discrete data)
- Has $O(d^2)$ parameters when there are d variables
- LARNs are the generalization of linear classification methods to generative modeling, bearing the same advantages and disadvantages
- Easy to train, but have low capacity

Neural auto-regressive networks

- Same left-to-right graphical model as LARNs, but use a different parameterization of the conditional distribution
- New parameterization allows for more model capacity, and can improve generalization by introducing a parameter sharing and feature sharing principle

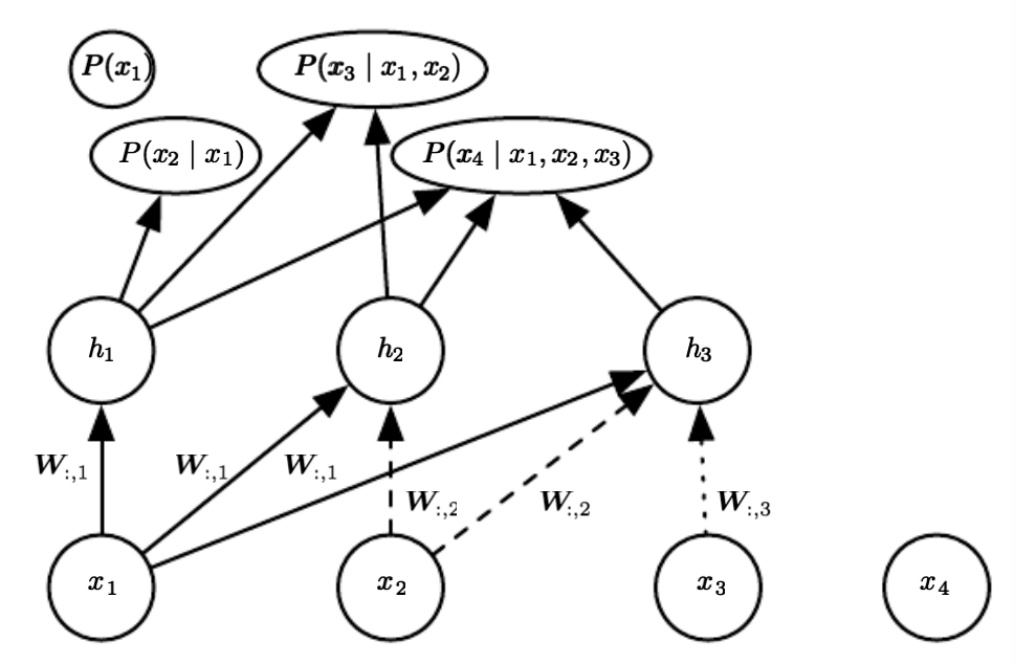


Advantages of using neural networks

- The parameterization of each $P(x_i|x_{i-1}, ..., x_1)$ with a neural network with $(i-1) \times k$ inputs and k outputs (if variables are discrete and are one-hot encoded) allows us to estimate the conditional probability without requiring an exponential number of examples and parameters, but is still able to capture high-order dependencies between random variables
- Instead of having a different network for the prediction of each x_i, the left-to-right connectivity allows the networks to be merged
  - Hidden layer features computed for predicting x_i can be reused for predicting $x_{i+k}(k > 0)$
  - Parameters are jointly optimized

Neural autoregressive density estimator (NADE)

- NADE introduces an additional parameter sharing scheme; the parameters of hidden units of different groups j are shared:



- The weights $W'_{j,k,i}$ from the i-th x_i to the k-th element of the j-th group of hidden unit $h_k^{(j)} (j \geqslant i)$ are shared among the groups:

$$W'_{j,k,i} = W_{k,i}$$

Generative adversarial networks

ML@B

- Goodfellow et al, 2014
- Based on a game theoretic scenario in which a generator network competes against an adversary
- Generator network produces a sample

$$\mathbf{x} = g(\mathbf{z}; \theta^{(g)})$$

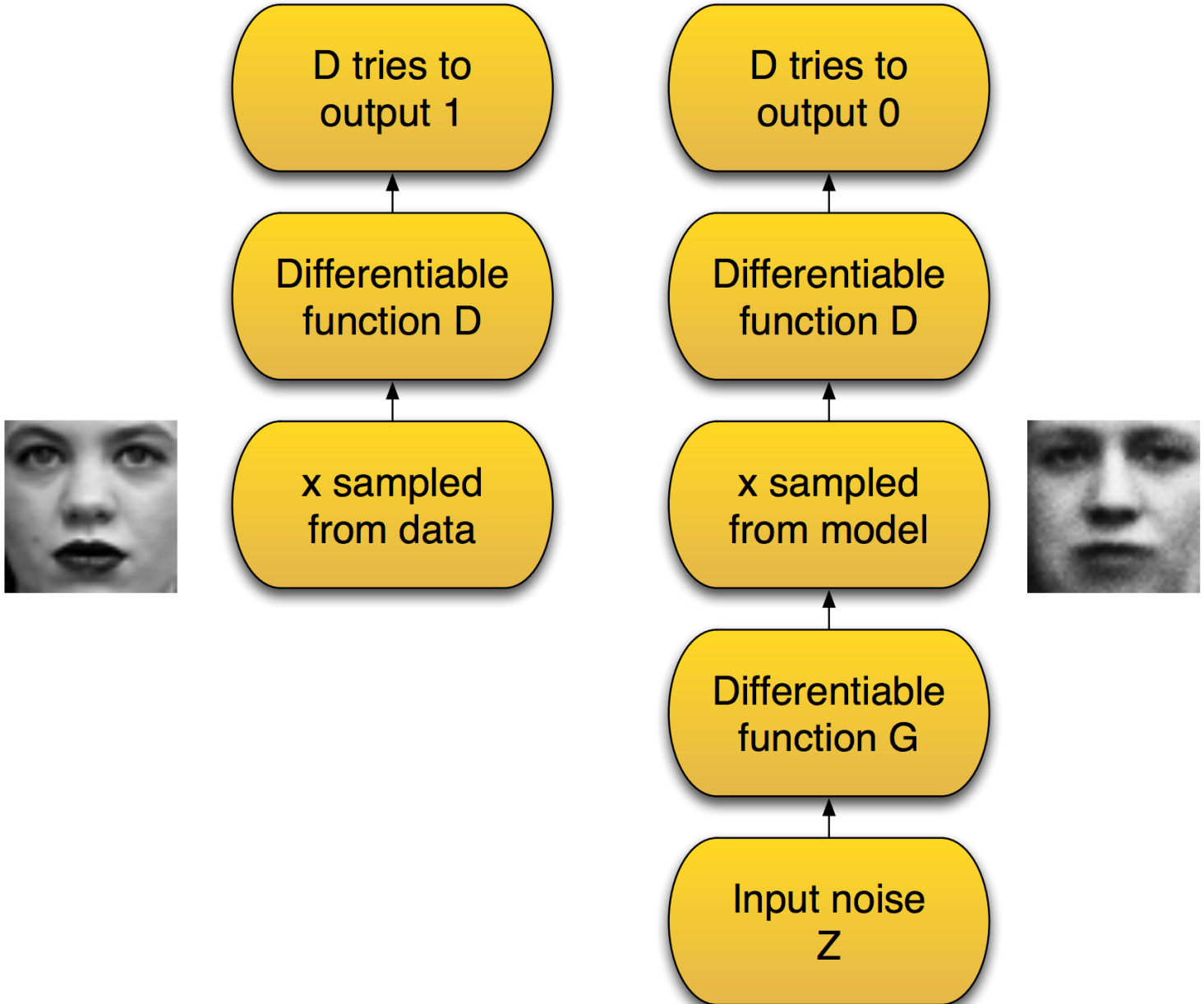
- Its adversary, the discriminator network, emits a probability $d(\mathbf{x}; \theta^{(d)})$, indicating the chance that $\mathbf{x}$ is a real training example or a fake one generated by the generator

Some GAN facts by Ian Goodfellow

ML@B

- GANs use a latent code
- They are asymptotically consistent (unlike variational methods)
- Do not require Markov chains
- Regarded as producing the best samples (hard to quantify this)
- Learning in GANS can be hard in practice when g and d are represented by neural networks and we are faced with non-convex functions to optimize
- Finding Nash equilibria in high-dimensional, continuous, non-convex games is an important open research problem

GAN framework (Goodfellow, 2016)



The generator network (Goodfellow 2016)

ML@B

- $\mathbf{x} = g(\mathbf{z}; \theta^{(g)})$
- g must be differentiable
- No invertibility requirement
- Trainable for any size of z
- Some guarantees require z to have higher dimension than x
- Can make x conditionally Gaussian given z, but not required

GAN learning

ML@B

- Formulate the problem as a zero-sum game
- A function $v(\theta^{(g)}, \theta^{(d)})$ determines the payoff of the discriminator
- The generator receives $-v(\theta^{(g)}, \theta^{(d)})$ as its own payoff
- During learning, each player attempts to maximize its own payoff
- At convergence:

$$g^* = argmin_g max_d v(g, d)$$

- Default choice for v is:

$$v(\theta^{(g)}, \theta^{(d)}) = E_{x \sim p_{data}} logd(\mathbf{x}) + E_{x \sim p_{model}} log(1 - d(\mathbf{x}))$$

The training procedure (Goodfellow 2016)

ML@B

- Use SGD-like algorithm of choice (Adam) on two mini-batches simultaneously
- A mini-batch of training examples
- A mini-batch of generated samples
- Optional: run k steps of one player for every step of the other player
- Dropout very important in the discriminator network

The minimax game (Goodfellow 2016)

- Costs:

$$J^{(d)} = -\frac{1}{2} E_{x \sim p_{data}} log d(\mathbf{x}) - E_{x \sim p_{model}} log(1 - d(\mathbf{x}))$$

$$J^{(g)} = -J^{(d)}$$

- Equilibrium is a saddle point
- Resembles Jensen-Shannon divergence
- Generator minimizes the log-probability of the discriminator being correct

A non-saturating game (Goodfellow 2016)

ML@B

- Costs:

$$J^{(d)} = -\frac{1}{2}E_{x\sim p_{data}} log d(\mathbf{x}) - E_{x\sim p_{model}} log(1 - d(\mathbf{x}))$$

$$J^{(g)} = -\frac{1}{2}E_{z} log(d(g(\mathbf{z}))$$

- Equilibrium not describable with a single loss
- Generator maximizes the log-probability of the discriminator being mistaken
- (Heuristically motivated) generator can still learn even when the discriminator successfully rejects all of the generated samples

A maximum-likelihood game (Goodfellow 2016)

- Costs:

$$J^{(d)} = -\frac{1}{2} E_{x \sim p_{data}} log d(\mathbf{x}) - E_{x \sim p_{model}} log(1 - d(\mathbf{x}))$$

$$J^{(g)} = -\frac{1}{2} E_z exp(\sigma^{-1}(d(g(\mathbf{z}))))$$

- When discriminator is optimal, the generator gradient matches that of maximum likelihood
- Because MLE training converges, this new GAN formulation should converge with enough samples
- In practice, did not seem to improve convergence (high variance around expected gradient)

The non-convergence problem with traditional GANs

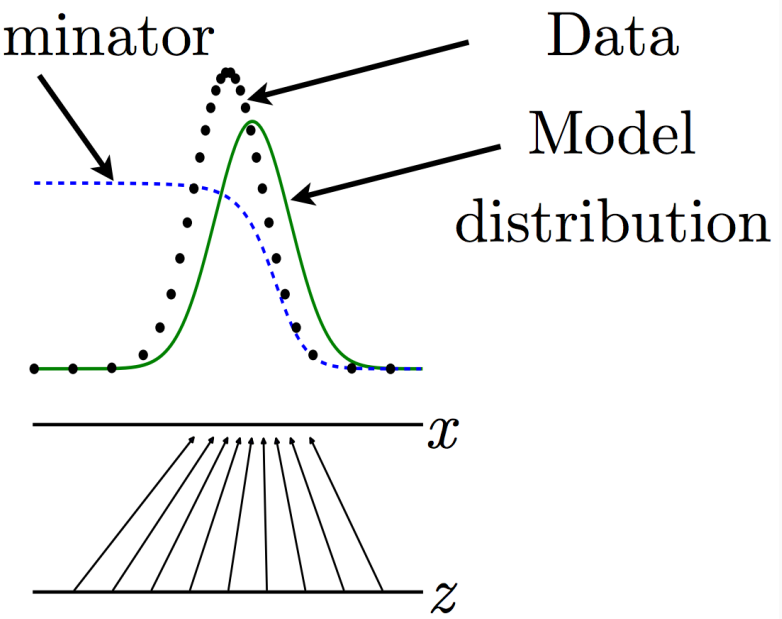
- A byproduct of non-convex functions
- Consider the value function $v(a, b) = ab$ where one player controls a and the incurs cost ab, the other player controls b and incurs cost -ab
- If each player makes infinitesimally small gradient steps, they reduce their own cost at the expense of the other
- Then a and b go in a stable, circular orbit rather than converging to the equilibrium at the origin

The discriminator's strategy

- Optimal $d(\mathbf{x})$ for any $p_{data}(\mathbf{x})$ and $p_{model}(\mathbf{x})$ is:

$$d(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}$$

A *cooperative* rather than adversarial view of GANs: the discriminator tries to estimate the ratio of the data and model distributions, and informs the generator of its estimate in order to guide its improvements.



(Goodfellow 2016)

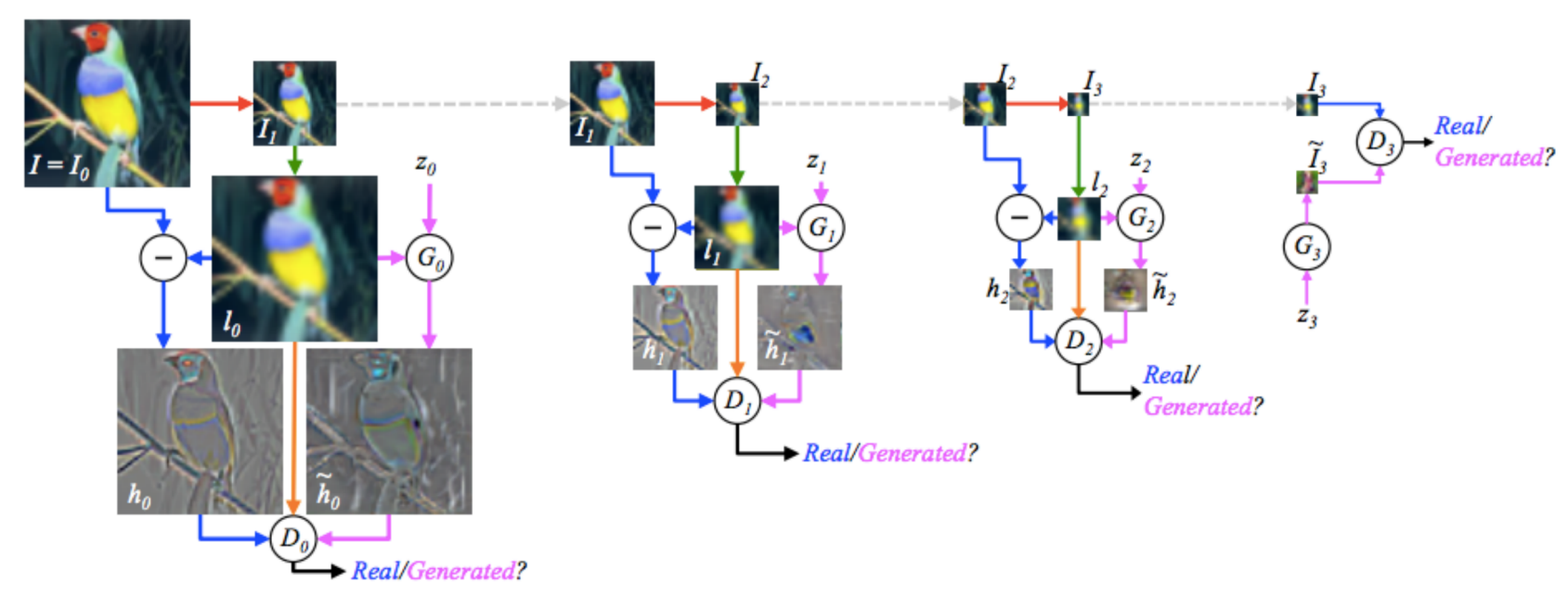
Simplifying GAN learning

ML@B

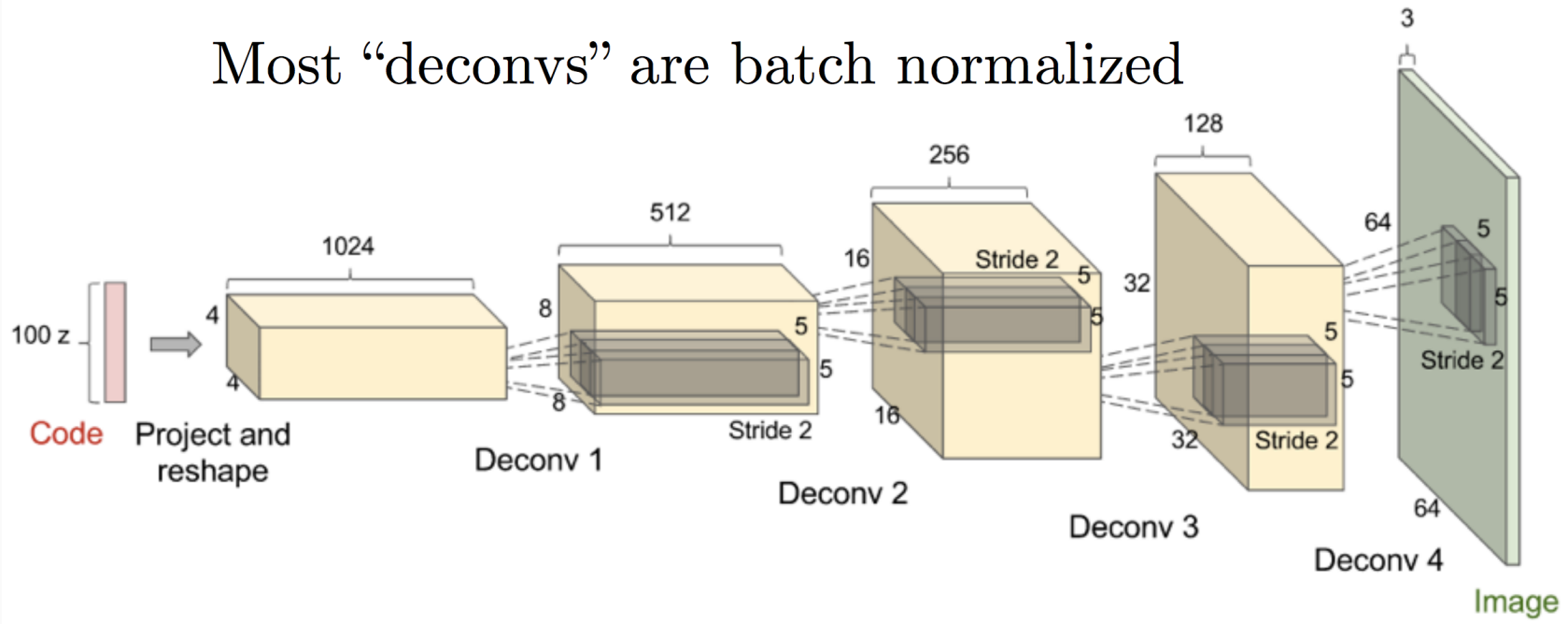
- Can break down the generation process into many levels of detail
- Possible to train conditional GANs (Mirza and Osindero, 2014) that sample from $p(\mathbf{x}|\mathbf{y})$ rather than from $p(\mathbf{x})$
- Denton et al (2015) showed that a series of conditional GANs can be trained to first generate a very low-resolution of an image, and then incrementally add details to it
- The above is called LAPGAN, as it uses a Laplacian pyramid to generate images contained varying levels of detail

The Laplacian pyramid

- Denton + Chintala et al, 2015
- Results: 40% of samples mistaken by humans for real photos



DCGAN



(Radford et al 2015)

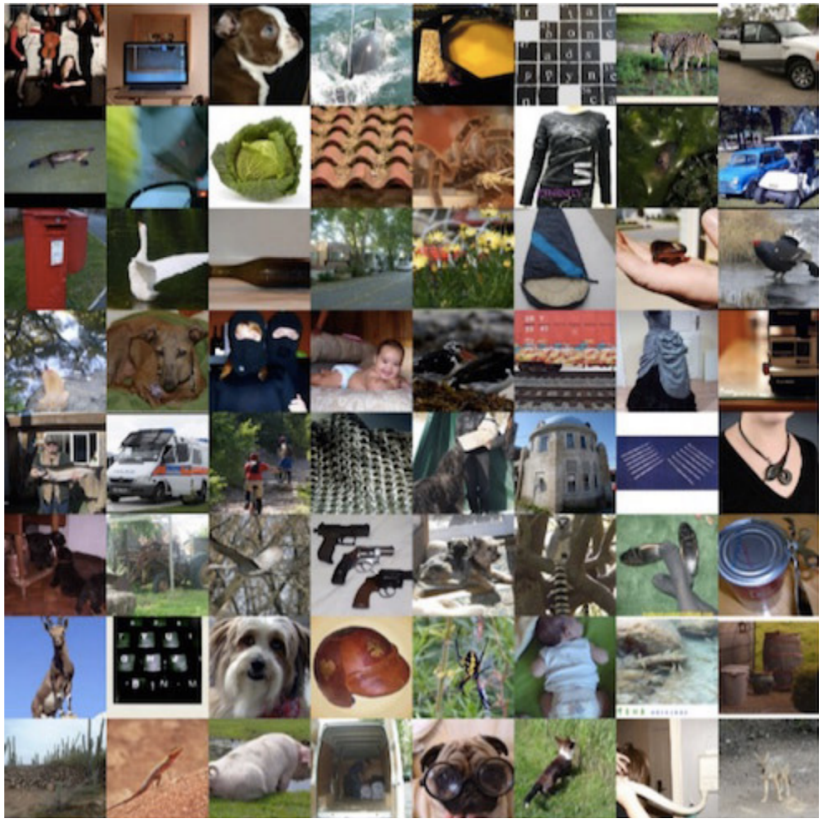
DCGANs can generate bedrooms (LSUN dataset)

(Radford et al 2015)

The "mode collapse" problem (Goodfellow 2016)

- Fully optimizing the discriminator with the generator held constant is safe
- Fully optimizing the generator with the discriminator held constant results in mapping points to the argmax of the discriminator
- Can partially fix this by adding nearest-neighbor features constructed from the current mini-batch to the discriminator ("mini-batch GAN"; Salimans et al, 2016)

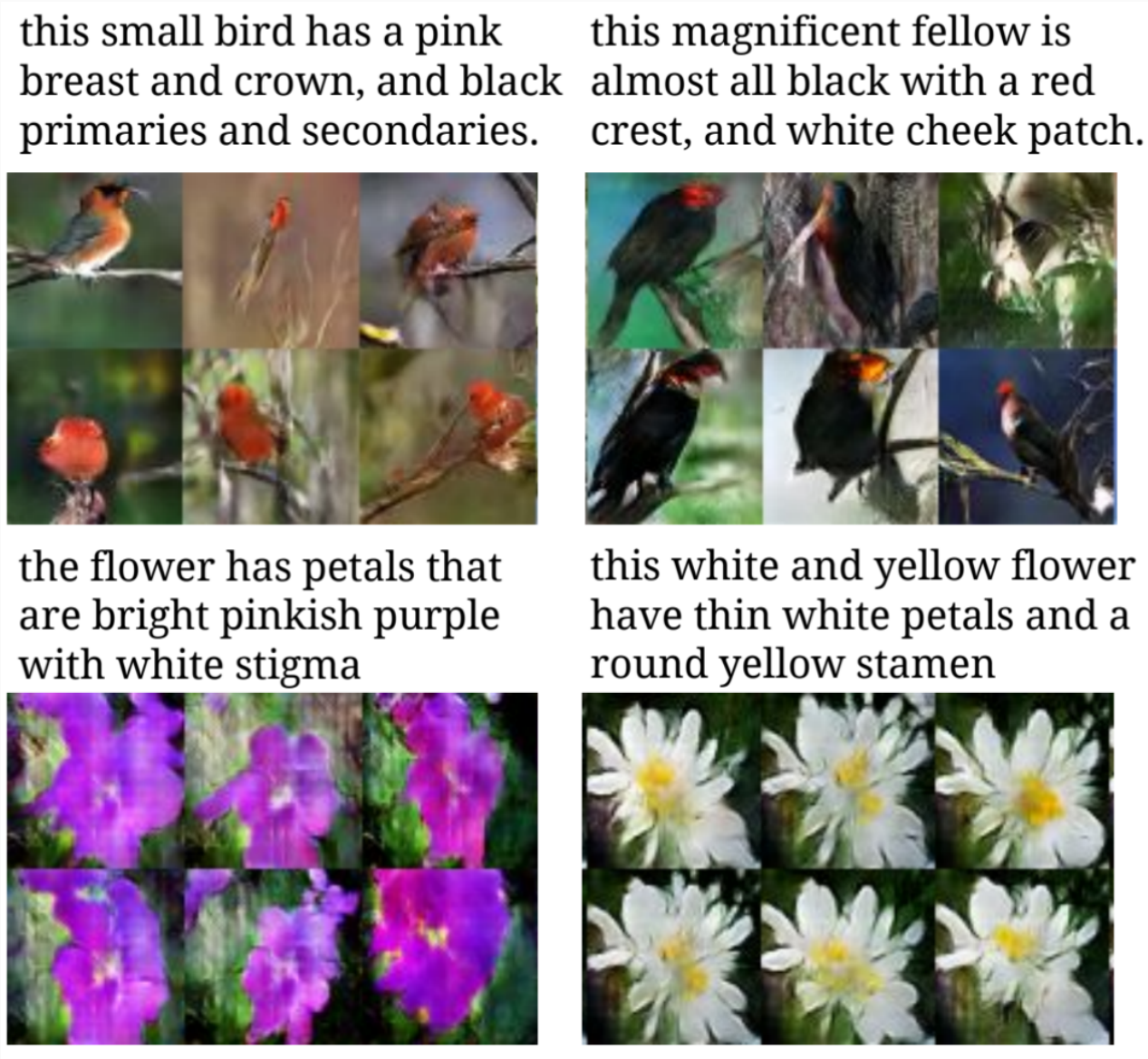
Mini-batch GAN on ImageNet



(Salimans et al 2016)

(Goodfellow 2016)

GANs work best when output entropy is low (Goodfellow 2016)



Other types of GANs

ML@B

- Wasserstein GAN (WGAN)
- Boundary equilibrium GAN (BEGAN)
- Bidirectional GAN (BiGAN)
- InfoGAN
- StackGAN
- CycleGAN
- Super-resolution GAN (SRGAN)
- SeqGAN
- ...DLD-GAN?

Evaluation of Generative Models

Evaluating generative models

ML@B

- How do we compare generative models to one another?
- We need to do so in order to show that our new generative model is better than pre-existing ones...
- This is a difficult and subtle task...
- We usually cannot actually evaluate the log-probability of the data under the model, but only an approximation
- Evaluation metrics are often hard research projects themselves
- Changes in preprocessing fundamentally alter the task

Evaluating generative models

ML@B

- Practitioners often evaluate by visual inspection
- Possible for a poor probabilistic model to produce very good samples (copies training examples)
- To test this, can check nearest neighbor of generated sample in the training set
- A generative model trained on data with many modes may ignore a small number and a human observer would not notice
- Theis et al. (2015) note that choice of evaluation metric must match the intended use of the model

Recent Work and Food for Thought

CycleGAN and Pix2pix

CycleGAN:
https://www.youtube.com/watch?v=9reHvktowLY

Pix2pix:
https://affinelayer.com/pixsrv/

Github repo:
https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

Closing thoughts

ML@B

- Finding Nash equilibrium in high-dimensional, continuous, non-convex games is an important research problem
- Generative models may hold the future to more general AI

Questions?