12. Unsupervised Deep Learning

CS 535 Deep Learning, Winter 2018

Fuxin Li

With materials from Wanli Ouyang, Zsolt Kira, Lawrence Neal, Raymond Yeh, Junting Lou and Teck-Yian Lim

Unsupervised Learning in General

- Unsupervised learning is learning without annotations (labels)
  - No regression targets
  - No class labels
  - No implicit labels (e.g. sequence to sequence)
- The goal is different from supervised learning
  - Supervised learning is usually trying to learn a function $f(\boldsymbol{x}) \approx y$
  - Unsupervised learning is learning a representation to **compactly represent** all the input $\boldsymbol{x}$

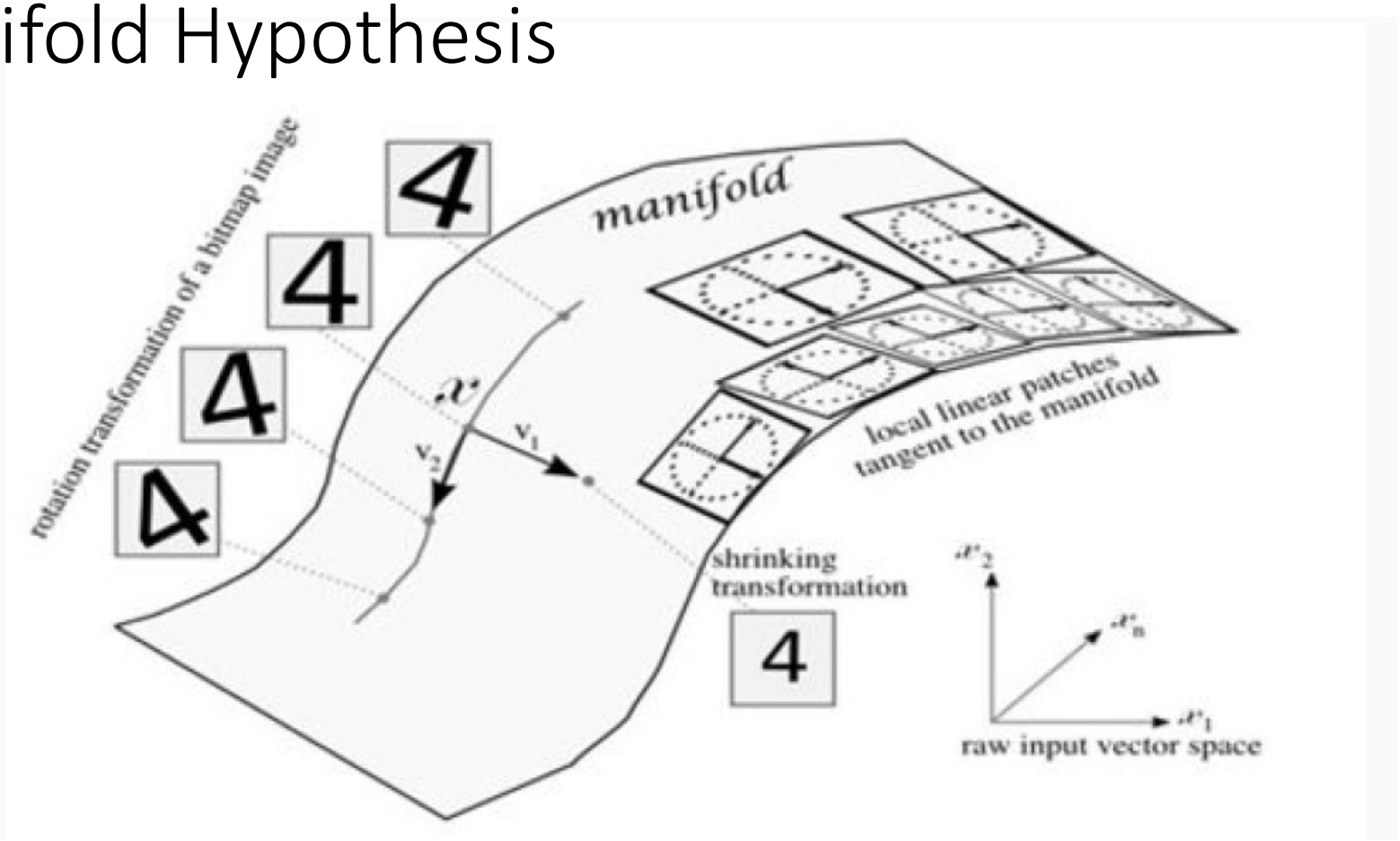
Occam's Razor Again

- In supervised learning, we seek to control overfitting by making the model simple

- In unsupervised learning, this is almost the **only** goal (before GANs)
  - Use a short description to represent the data
  - Minimal Description Length Principle
  - Dimensionality Reduction
  - Clustering

Manifold hypothesis: data comes from a low-dimensional space mapped to high-dim + noise

Manifold Hypothesis

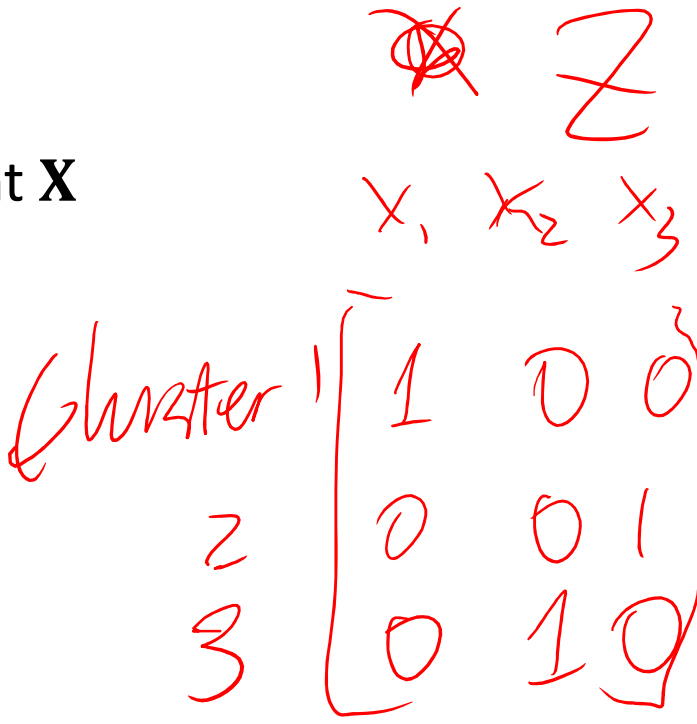


Generic Unsupervised Learning

- The general reconstruction objective:

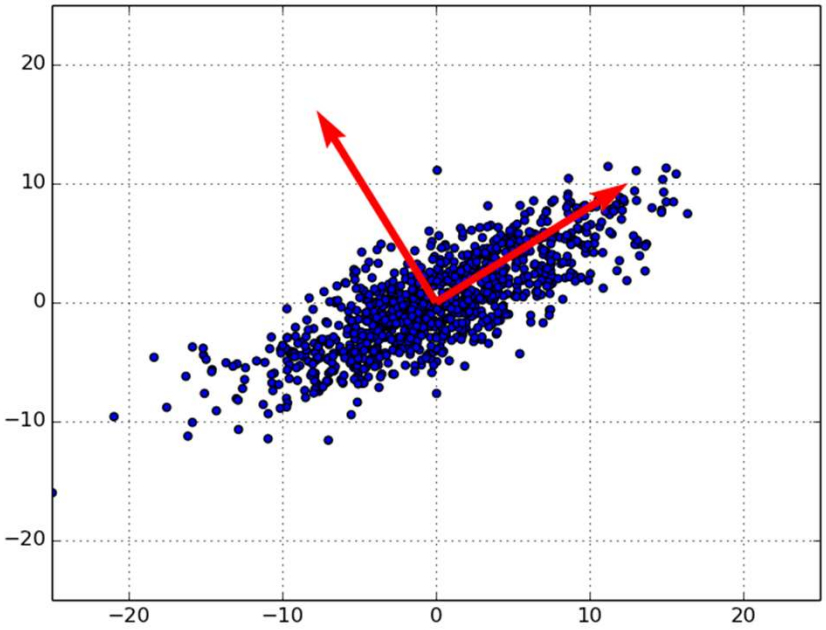
$$\min_{\mathbf{Z},\mathbf{U}} ||\mathbf{X} - \mathbf{ZU}||_F^2$$

- Use a lower-dimensional subspace $\mathbf{U}$ to represent $\mathbf{X}$
  - $\mathbf{Z}$ are the coordinates in the low-dimensional space
  - Reduced curse of dimensionality
  - "Simpler" model
- No constraint: PCA (Xu et al. ICML 2009)
- K-means clustering: $z_{ij} \in \{0,1\}, \sum_j z_{ij} = 1$

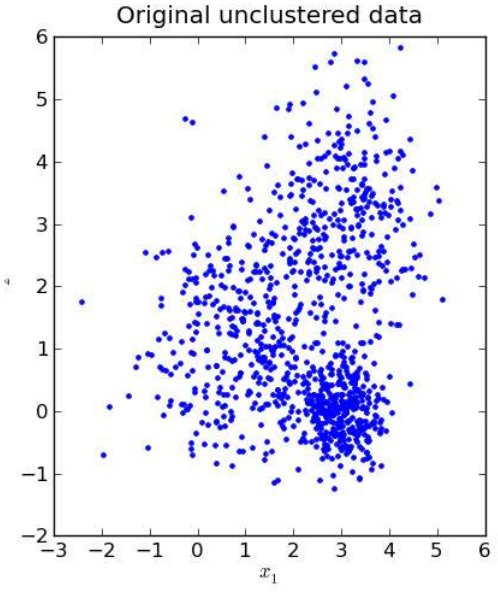


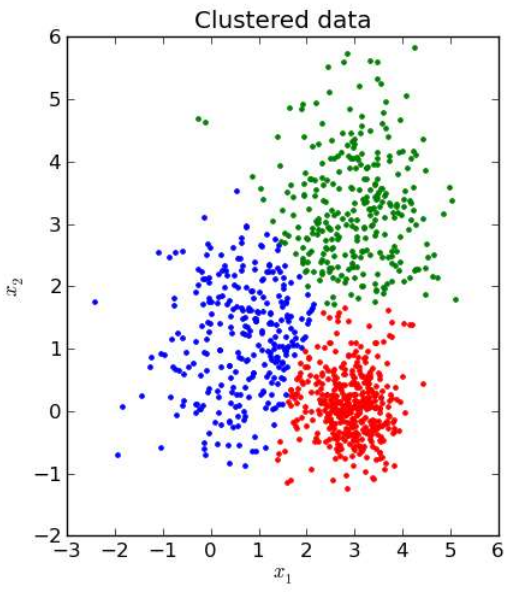
Geometric Representations

PCA



K-Means

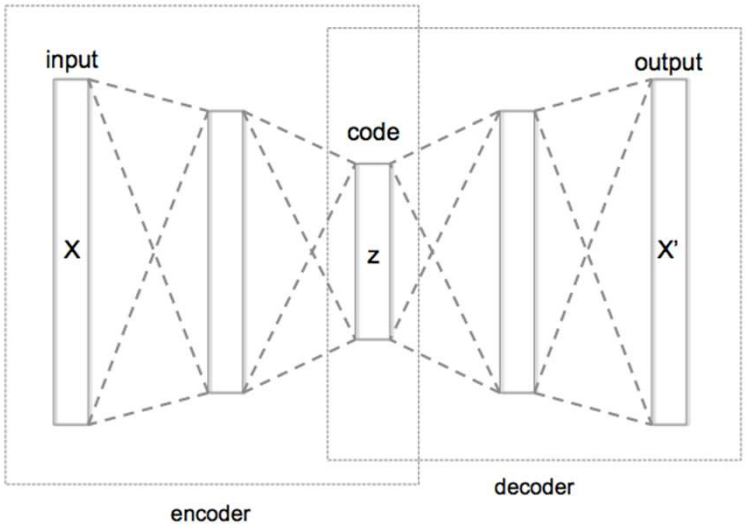




Reconstruction

$$\min_{\mathbf{Z},\mathbf{U}} ||\mathbf{X} - \mathbf{ZU}||_F^2$$

- $\mathbf{ZU}$ is the reconstruction
- PCA
  - Singular value decomposition
- K-means
  - Reconstruct each item with its cluster center
- Can treat as “autoencoder”
  - Encoding and decoding
- Sampling from the “code” space
  - Generative model!

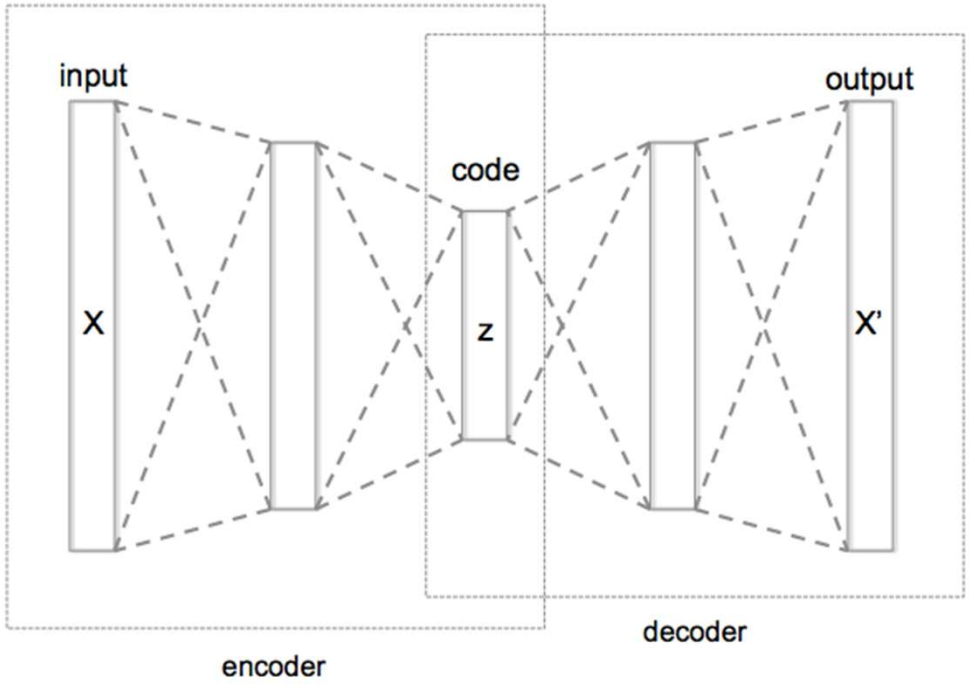


Deep Autoencoders

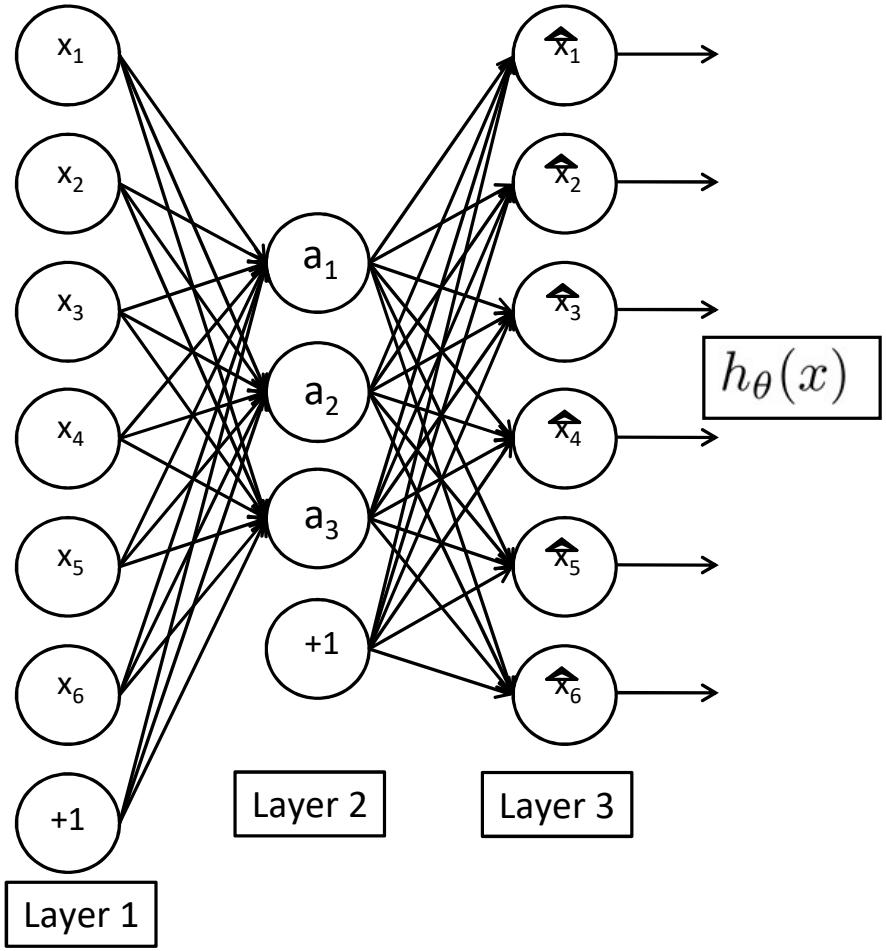
$$z = \sigma(W_1 x + b_1)$$

$$h(x) = \sigma\,(W_2 z + b_2)$$

$$\min_{W_1, W_2, b_1, b_2} \sum_i (h(x_i) - x_i)^2$$



Autoencoders



Autoencoder.

Network is trained to output the input (learn identify function).

$$h_\theta(x) \approx x$$

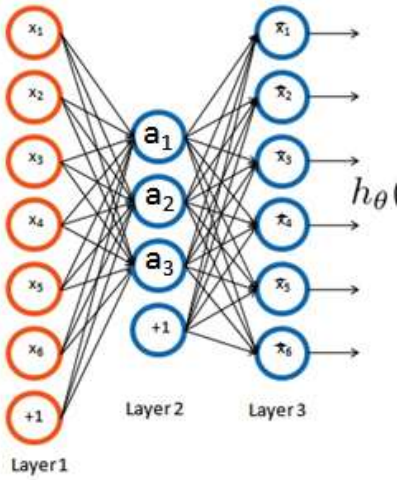
Trivial solution unless:
- Constrain number of units in Layer 2 (learn compressed representation), or
- Constrain Layer 2 to be sparse.

Sparse Autoencoders (SAE)

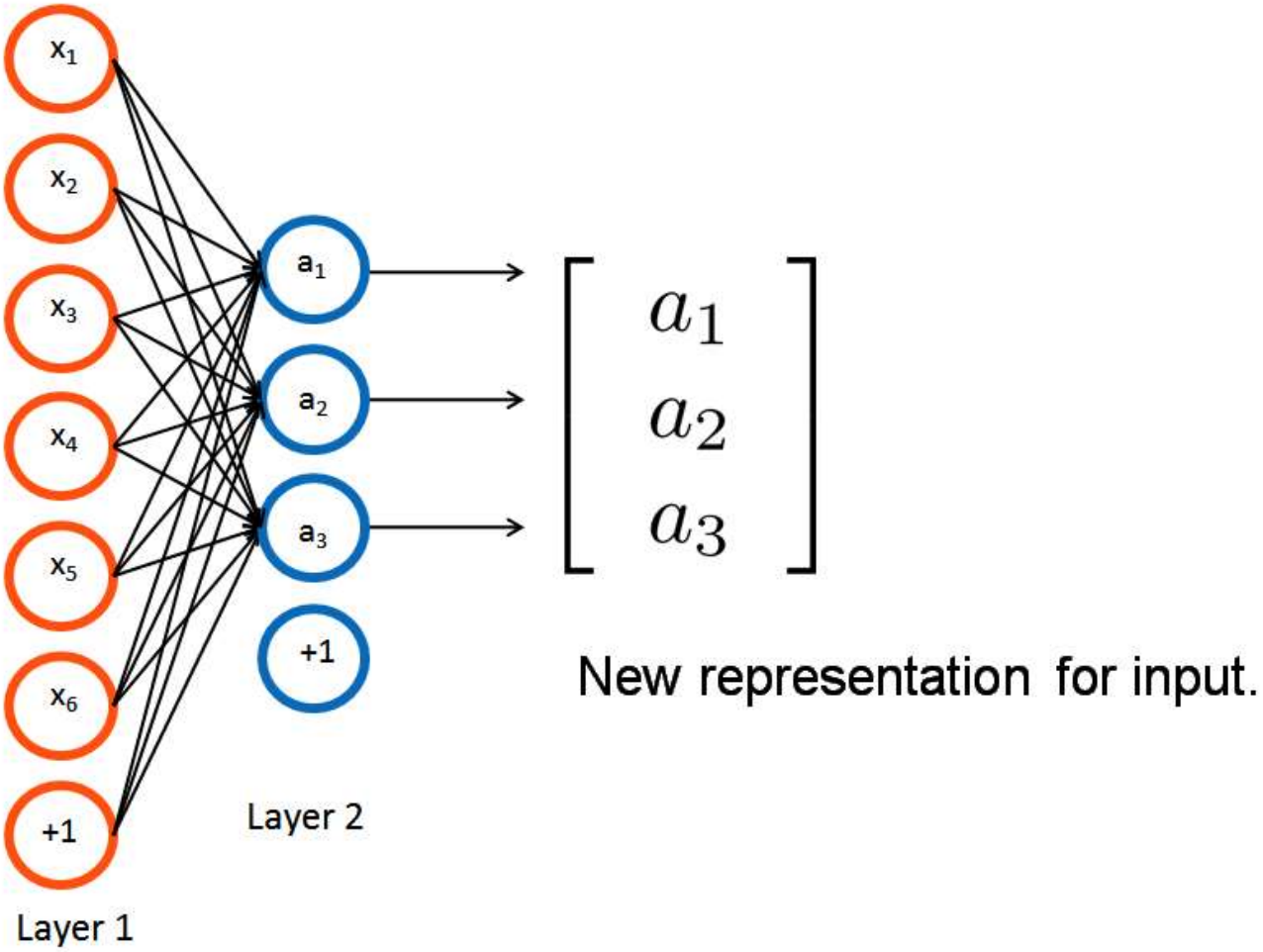
Training a sparse autoencoder.

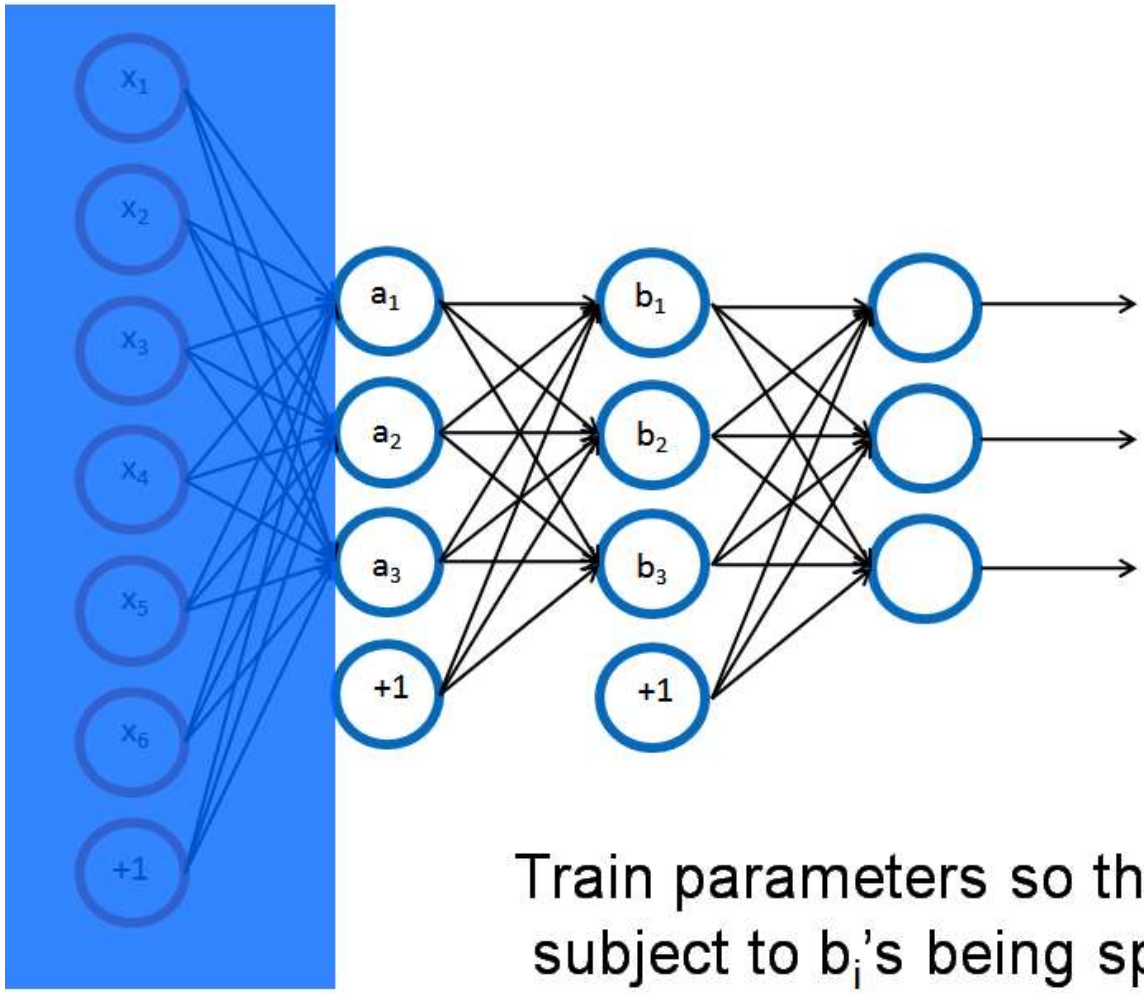
Given unlabeled training set $x_1, x_2, \ldots$

$$\min_{\theta} \underbrace{\|h_\theta(x) - x\|^2}_{\text{Reconstruction error term}} + \lambda \underbrace{\sum_i |a_i|}_{L_1 \text{ sparsity term}}$$



Training deep sparse Autoencoders

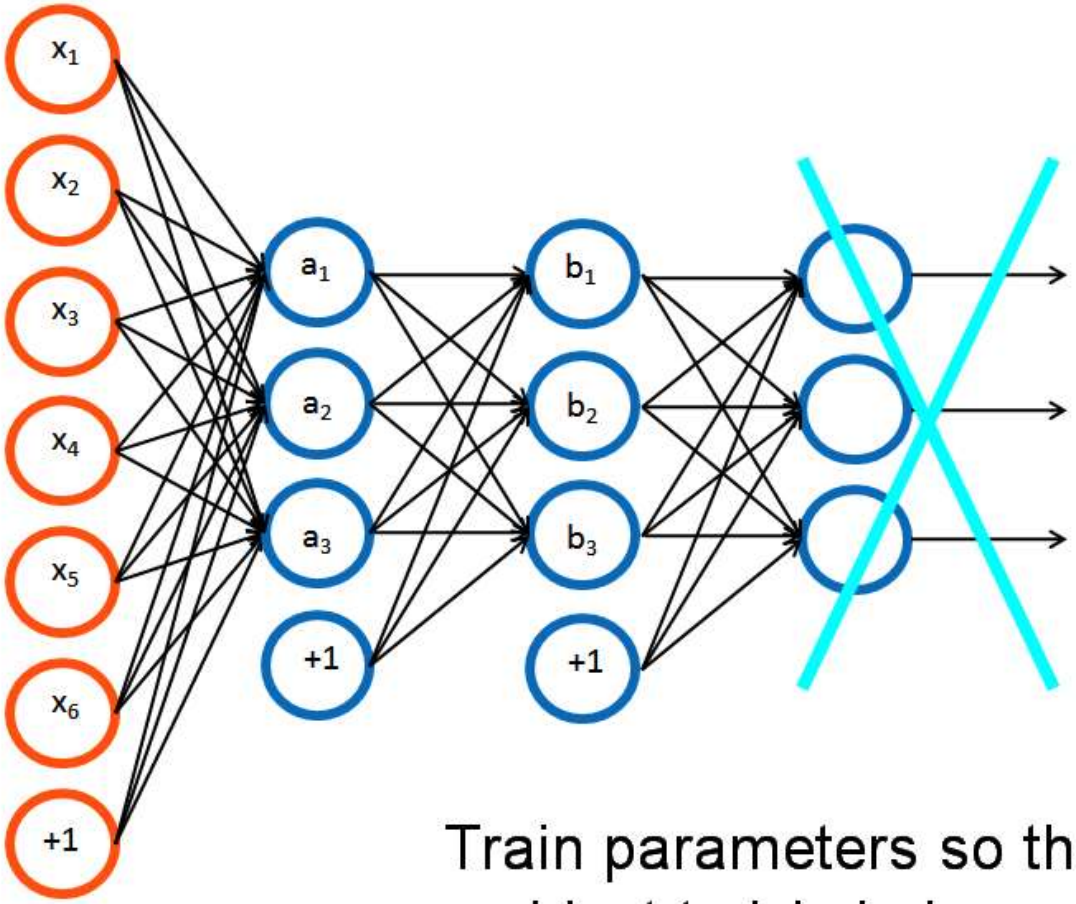




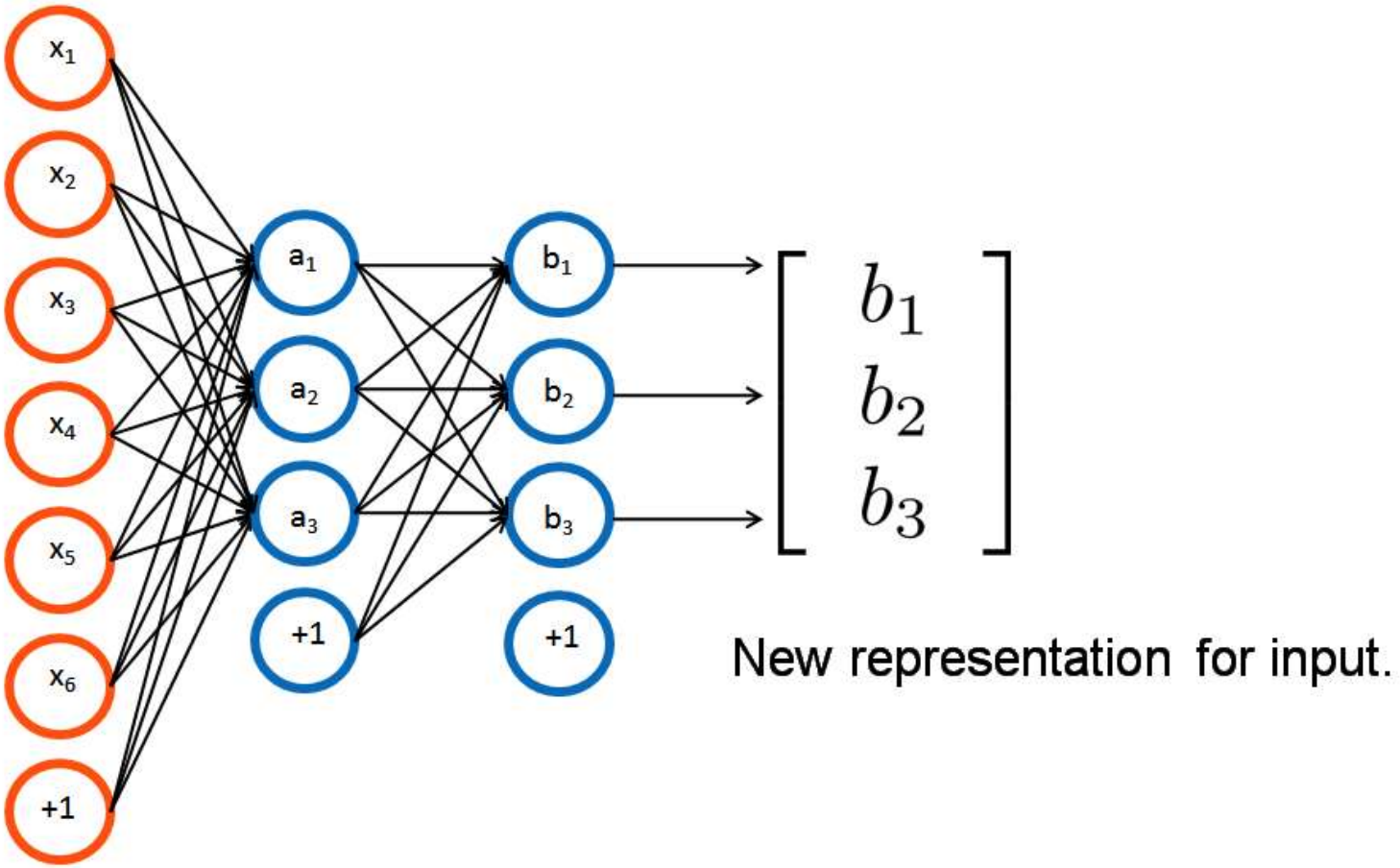
$h_\theta(x)$

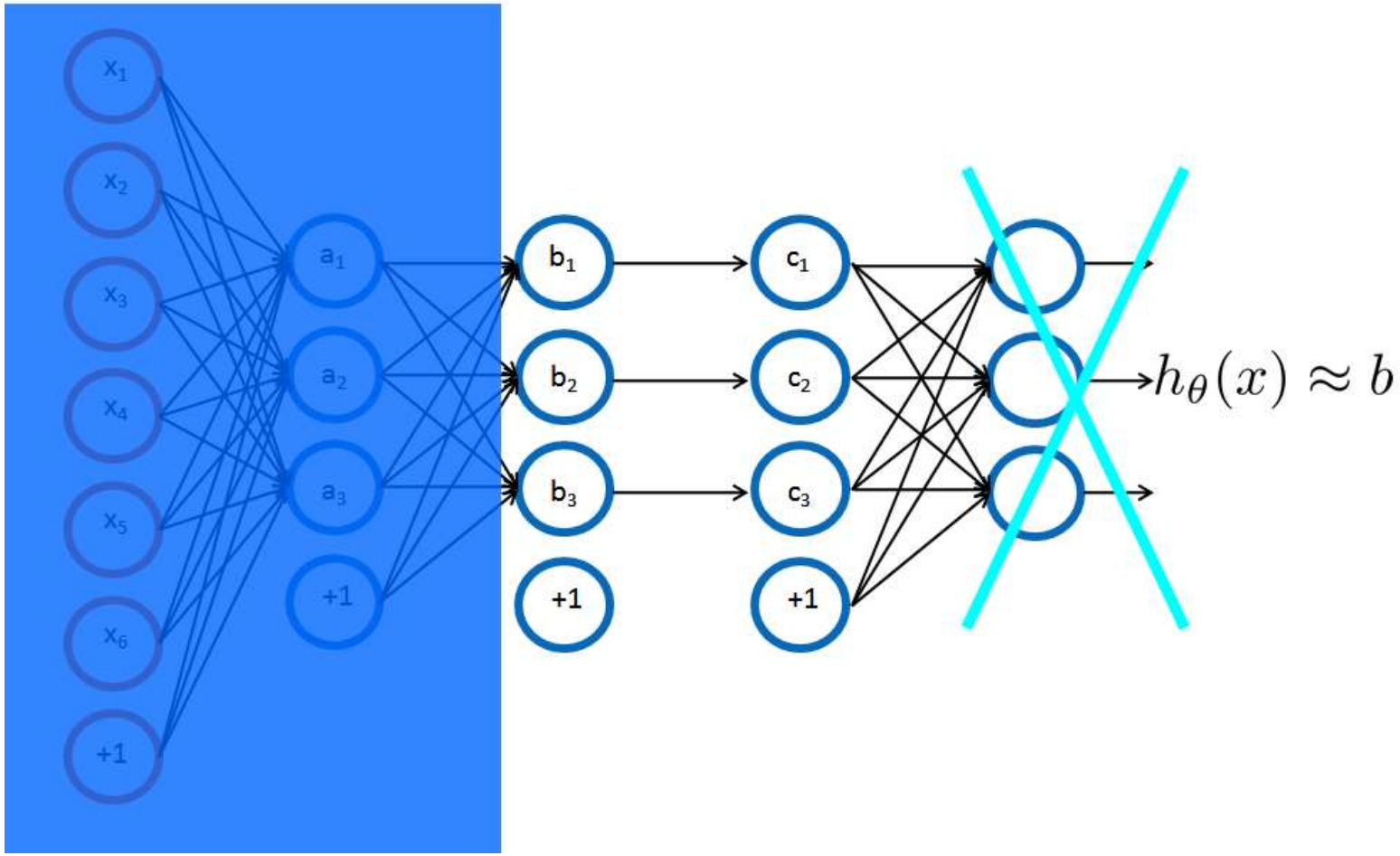
Note we are reconstructing ***a*** at this point, not *x*

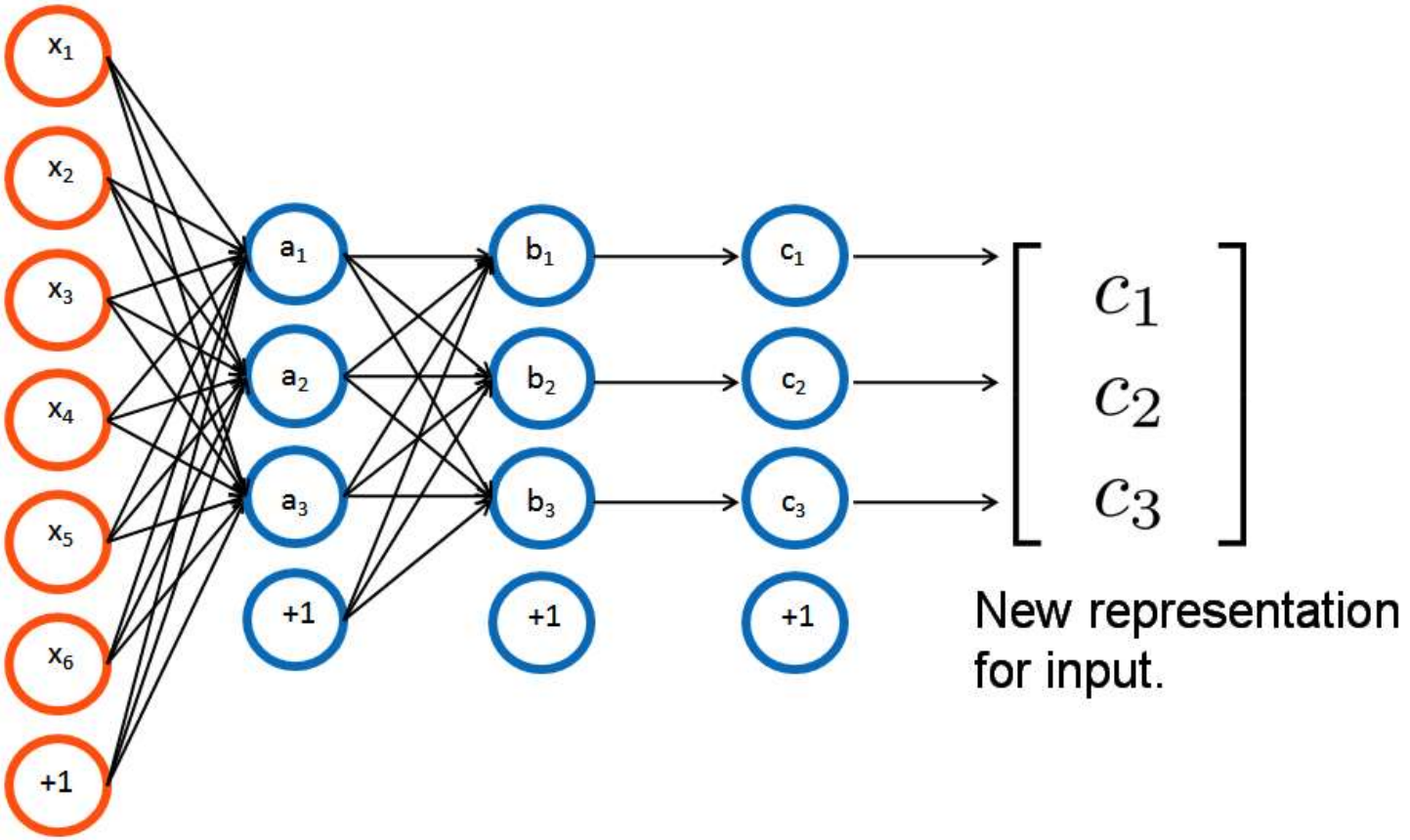
Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse



Train parameters so that $h_\theta(x) \approx a$,
subject to b_i's being sparse.


x1
x2
x3
x4
x5
x6
+1
a1
a2
a3
+1
b1
b2
b3
+1
b_1
b_2
b_3
New representation for input.


x_1
x_2
x_3
x_4
x_5
x_6
+1
a_1
a_2
a_3
+1
b_1
b_2
b_3
+1
c_1
c_2
c_3
+1
$h_\theta(x) \approx b$



Use [c_1, c_3, c_3] as representation to feed to learning algorithm.

A comparison of methods for compressing digit images to 30 real numbers

real data

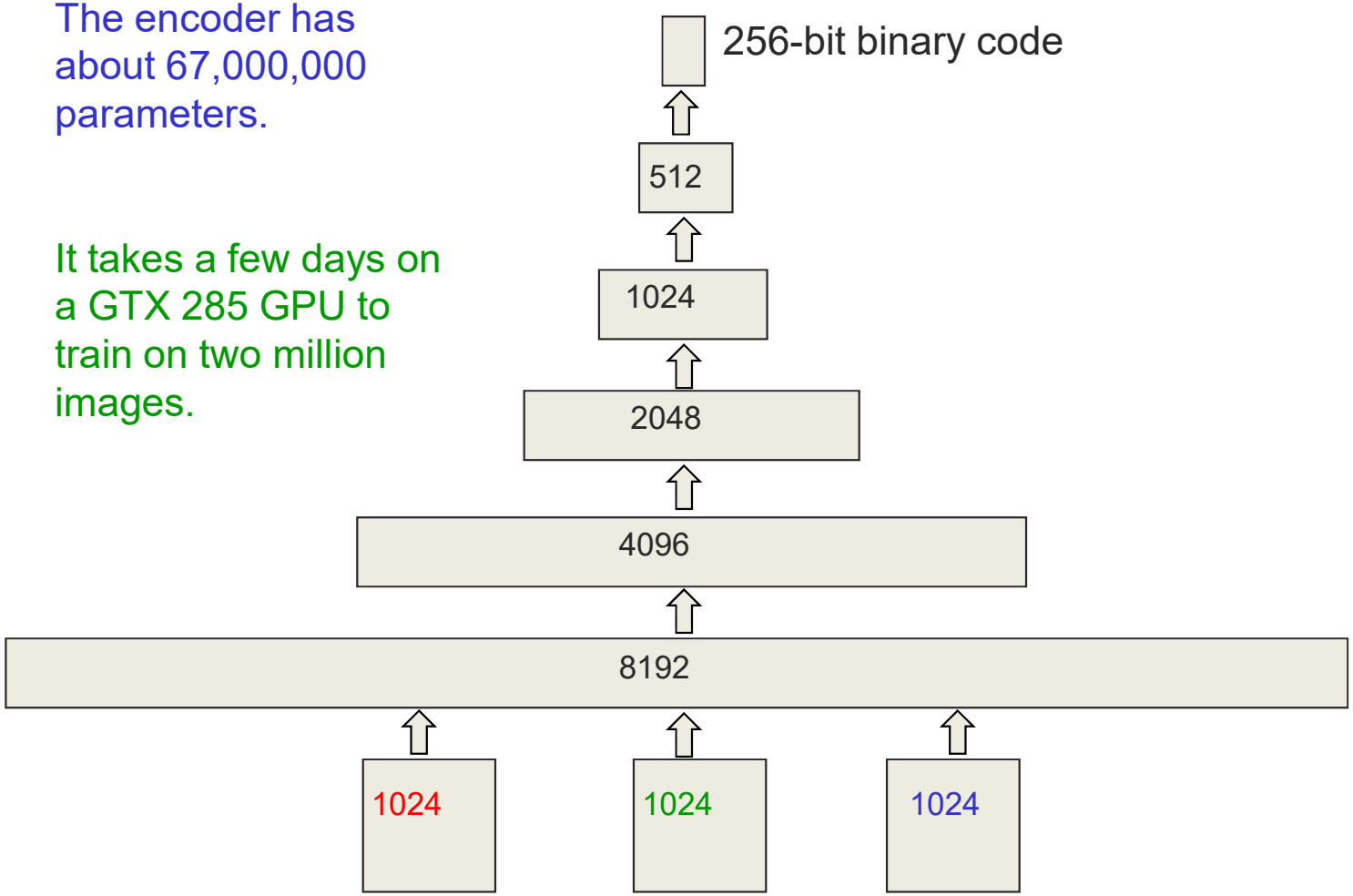
30-D deep auto

30-D PCA

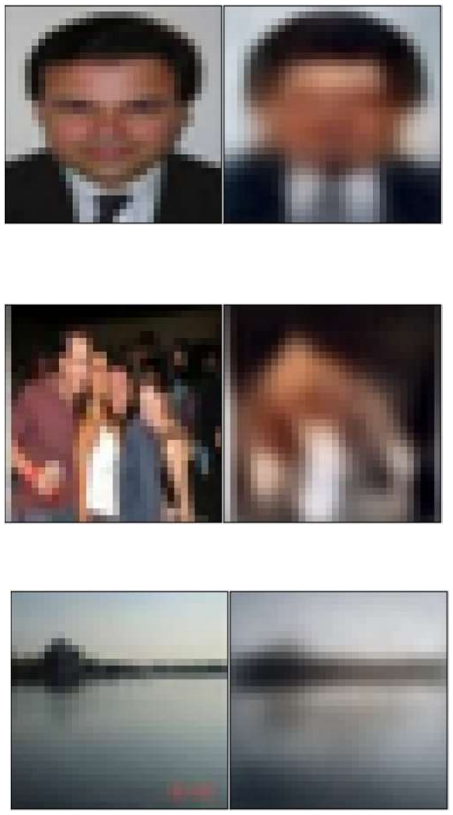
Krizhevsky's deep autoencoder

The encoder has about 67,000,000 parameters.

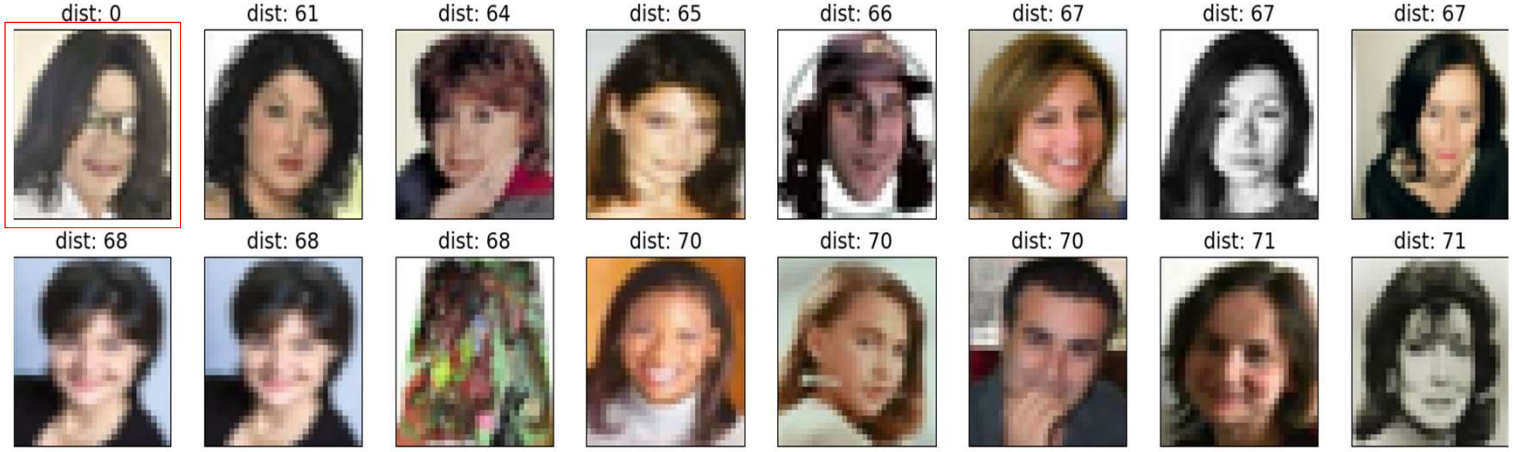
It takes a few days on a GTX 285 GPU to train on two million images.



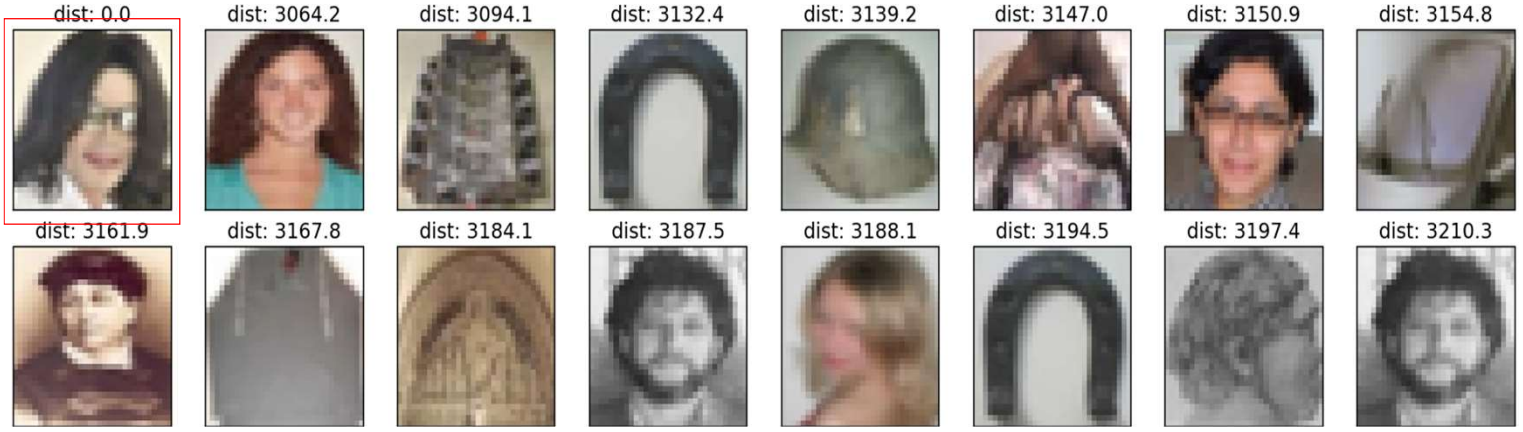
Reconstructions of 32x32 color images from 256-bit codes



retrieved using 256 bit codes



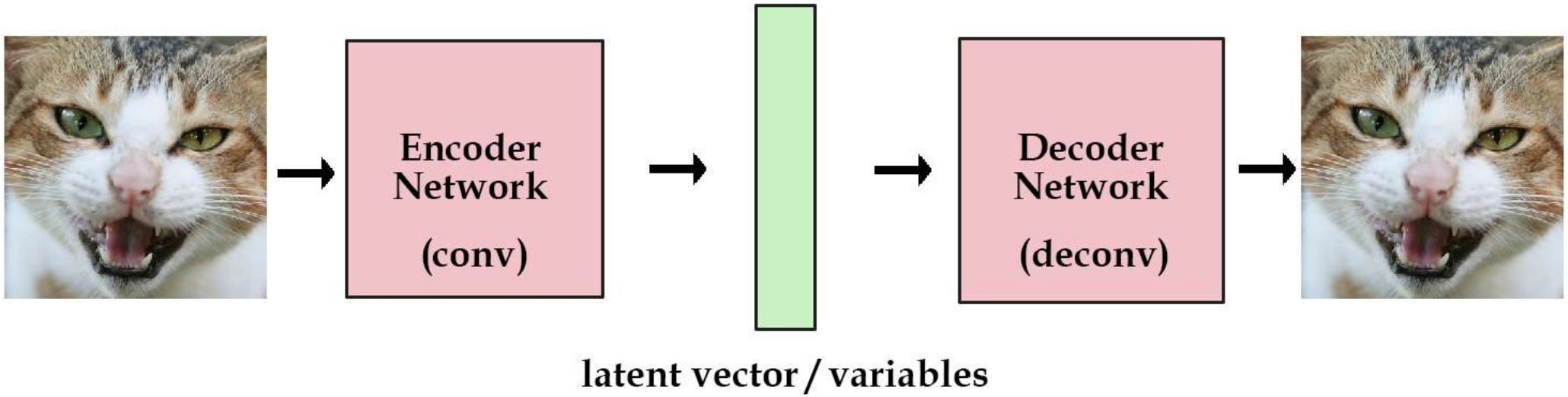
retrieved using Euclidean distance in pixel intensity space



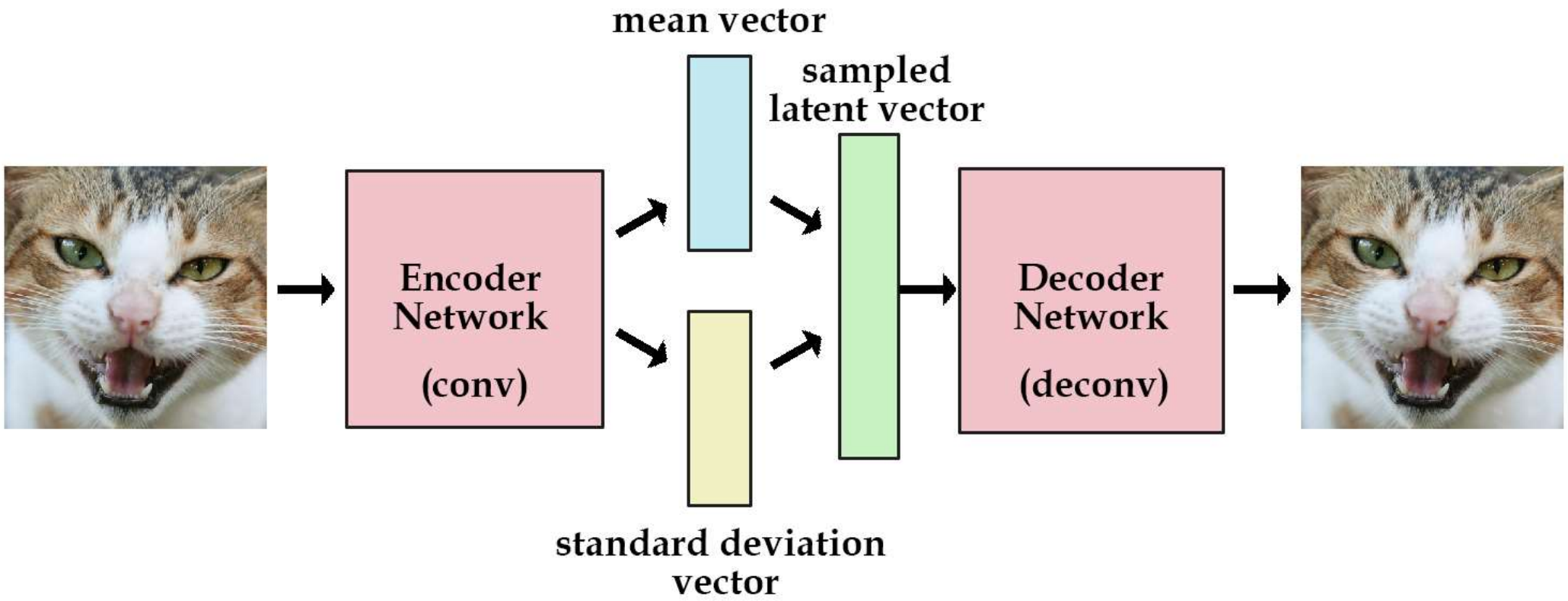
retrieved using 256 bit codes

retrieved using Euclidean distance in pixel intensity space

Convolutional Autoencoders



Variational Autoencoders



The “Variational”

- Suppose there is one probabilistic encoder $Q(z|X)$ and one decoder $P(X|z)$ that generates X from unit Gaussian $N(\mathbf{0}, \mathbf{I})$
- Suppose for P there is an inverse distribution represented by $P(z|X)$
- Goal:

$$\max_X \log P(X) - D_{KL}(Q(z|X)||P(z|X))$$

Generate Training Data Identity between the encoder and decoder

Variational Auto-Encoder

$$\max_X \log P(X) - D_{KL}(Q(z|X)||P(z|X))$$

- Use Bayes Rule, one can convert:

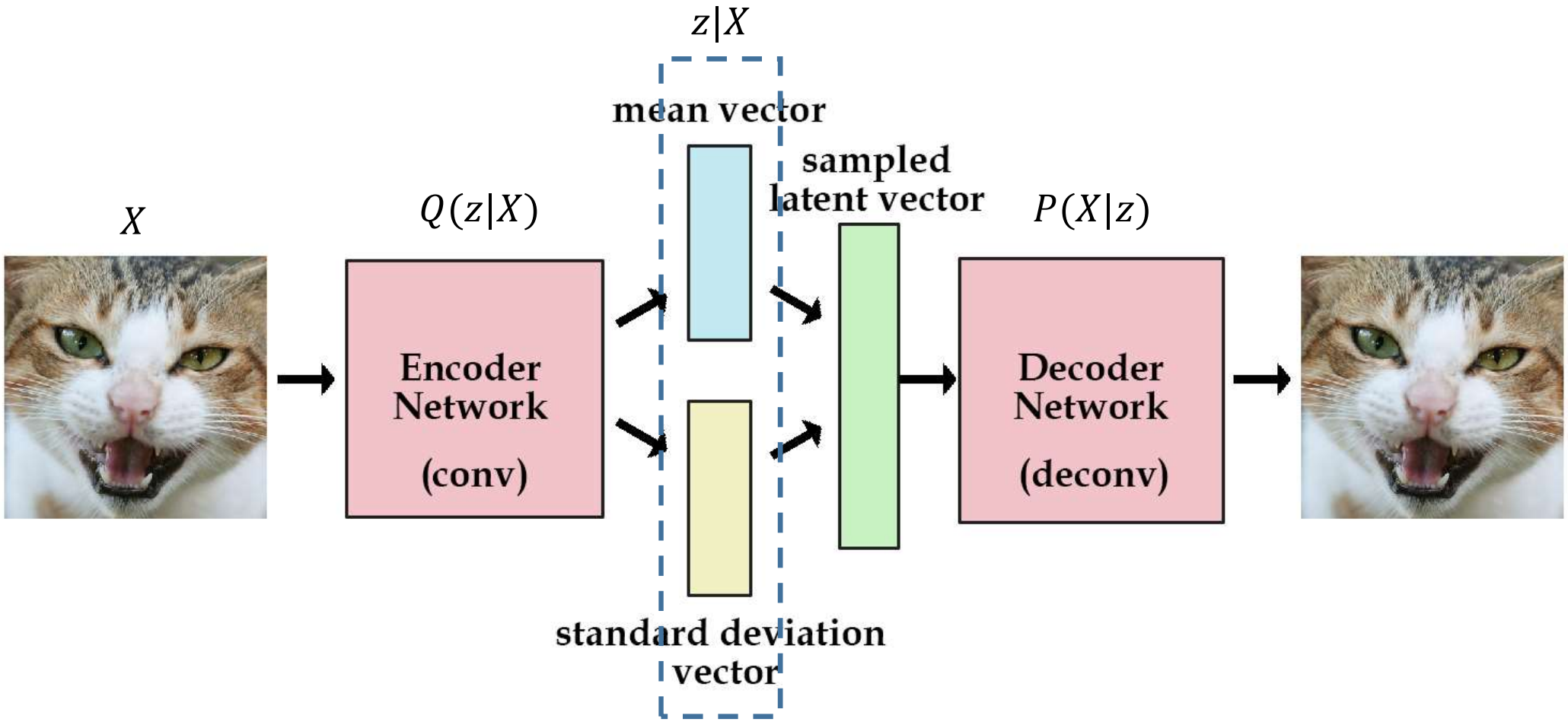
$$\begin{aligned} D_{KL}\big(Q(z|X)||P(z|X)\big) &= E_{z|X\sim Q}\big[\log Q(z|X) - \log\big(P(z|X)\big)\big] \\ &= E_{z|X\sim Q}\left[\log Q(z|X) - \log\frac{P(X|z)P(z)}{P(X)}\right] \\ &= E_{z|X\sim Q}[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X) \end{aligned}$$

- Hence, the optimization goal can be converted to:

$$\max_X E_{z|X\sim Q}\big[L(X,z;P,Q)\big] = E_{z|X\sim Q}[\log P(X|z) - D_{KL}(Q(z|X)||P(z))\big]$$

2 Gaussians! Closed-form solution

Variational Auto-Encoder

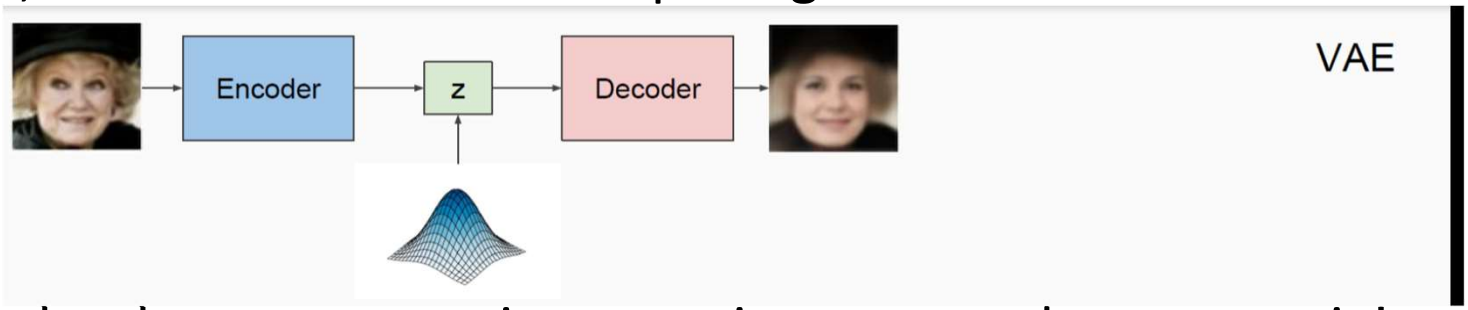


VAE Training

- Given dataset $\boldsymbol{X}$
- Repeat until convergence
  - Sample a mini-batch of M examples from $\boldsymbol{X}$ as $\boldsymbol{X}_M$
  - Sample M noise vetors $\epsilon \sim N(0,1)$
  - Run forward and backward pass on $L(\boldsymbol{X}_M, z + \epsilon, \theta)$ to update θ
- Note:
  - Every iteration we use different ϵ!

GANs

- Still, VAE does not create crisp images



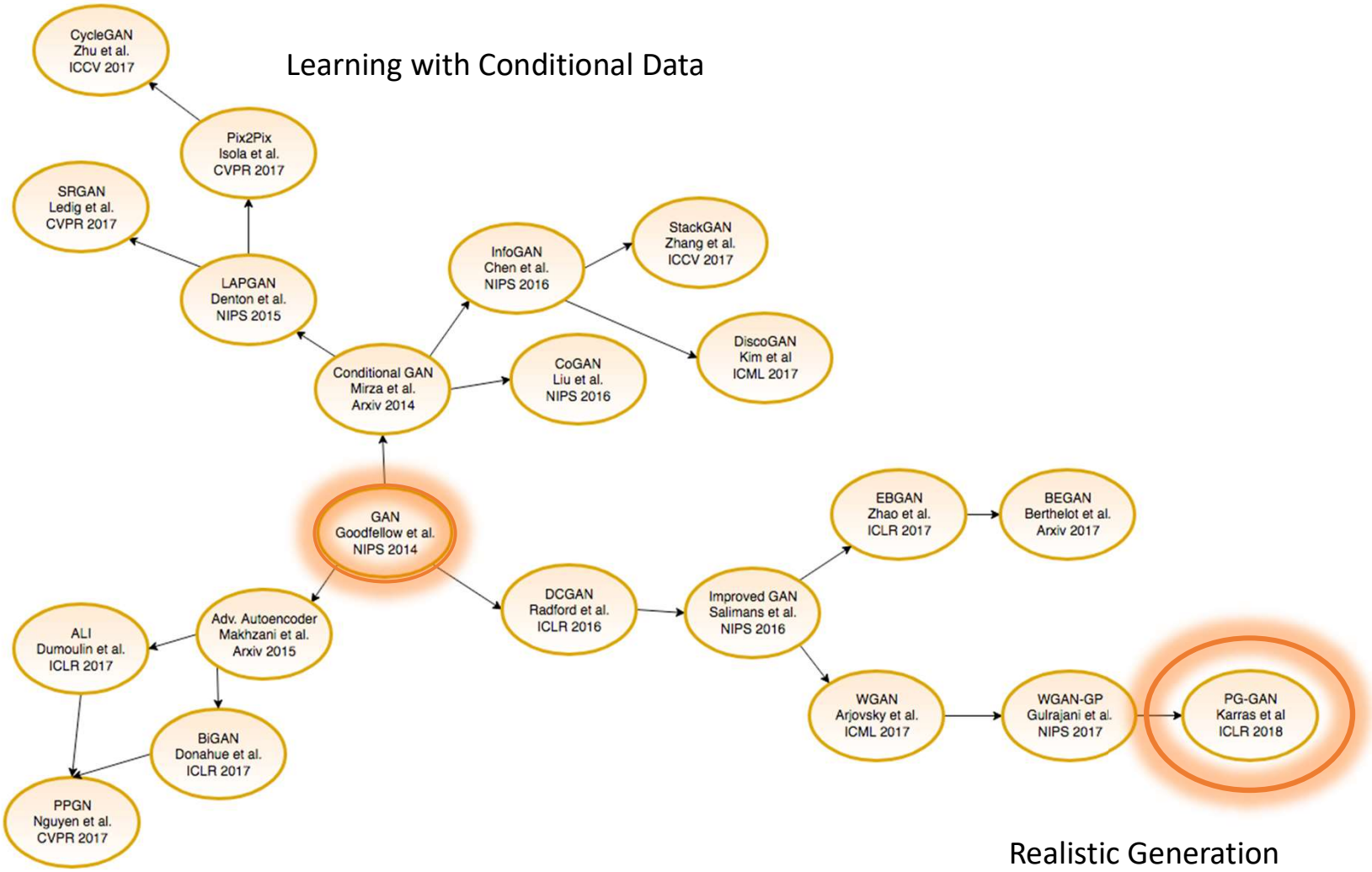
- Maybe the reconstruction error is not a good error metric!

- What's the problem with the reconstruction error?
  - L2 in the image space is not a good distance metric
  - It does not need to generate anything other than the training set

Cf. Larsen et al. arXiv:1512.09300

Generative Adversarial Nets

- Coined by Ian Goodfellow in 2014
- Generative:
  - Models the training distribution
  - Can be sampled to produce “fake” examples
- Adversarial:
  - Training involves “minimax” between two networks
  - Discriminator: Learns to classify “real” vs “fake”
  - Generator: Learns to output “real” images
- Network:
  - Trains by gradient descent with backpropagation


CycleGAN
Zhu et al.
ICCV 2017
Learning with Conditional Data
Pix2Pix
Isola et al.
CVPR 2017
SRGAN
Ledig et al.
CVPR 2017
LAPGAN
Denton et al.
NIPS 2015
InfoGAN
Chen et al.
NIPS 2016
StackGAN
Zhang et al.
ICCV 2017
DiscoGAN
Kim et al
ICML 2017
Conditional GAN
Mirza et al.
Arxiv 2014
CoGAN
Liu et al.
NIPS 2016
GAN
Goodfellow et al.
NIPS 2014
EBGAN
Zhao et al.
ICLR 2017
BEGAN
Berthelot et al.
Arxiv 2017
DCGAN
Radford et al.
ICLR 2016
Improved GAN
Salimans et al.
NIPS 2016
ALI
Dumoulin et al.
ICLR 2017
Adv. Autoencoder
Makhzani et al.
Arxiv 2015
WGAN
Arjovsky et al.
ICML 2017
WGAN-GP
Gulrajani et al.
NIPS 2017
PG-GAN
Karras et al
ICLR 2018
BiGAN
Donahue et al.
ICLR 2017
PPGN
Nguyen et al.
CVPR 2017
Realistic Generation
Learning to Encode

Generative Adversarial Nets

The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. To learn the generator's distribution p_g over data $\boldsymbol{x}$, we define a prior on input noise variables $p_{\boldsymbol{z}}(\boldsymbol{z})$, then represent a mapping to data space as $G(\boldsymbol{z}; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1 - D(G(\boldsymbol{z})))$. In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{1}$$

Generative Adversarial Nets

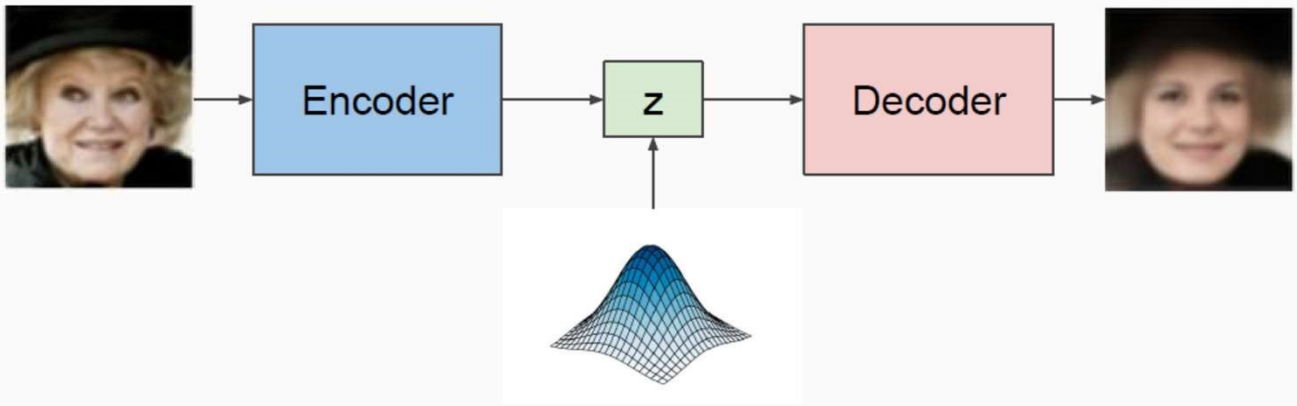
a)

b)

c)

d)

VAE vs. GAN

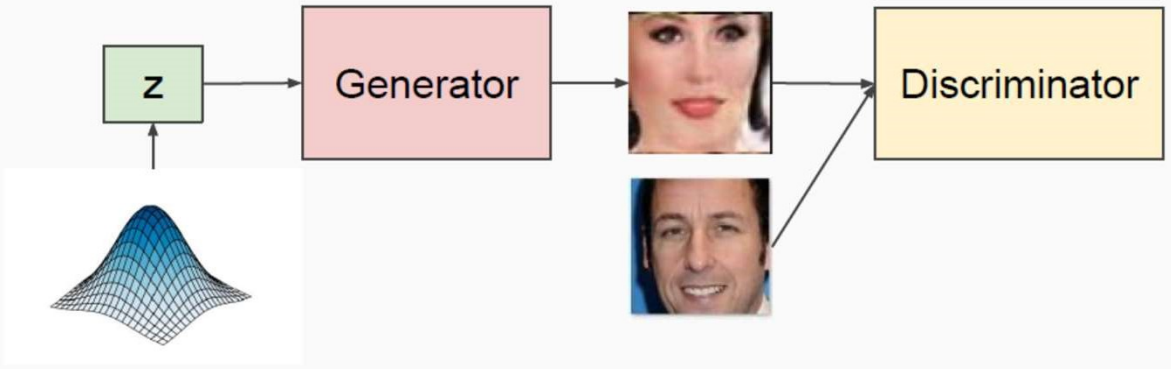


VAE

✓: Given an X **easy** to find z.
✓: Interpretable probability P(X)

X: Usually outputs blurry Images

GAN



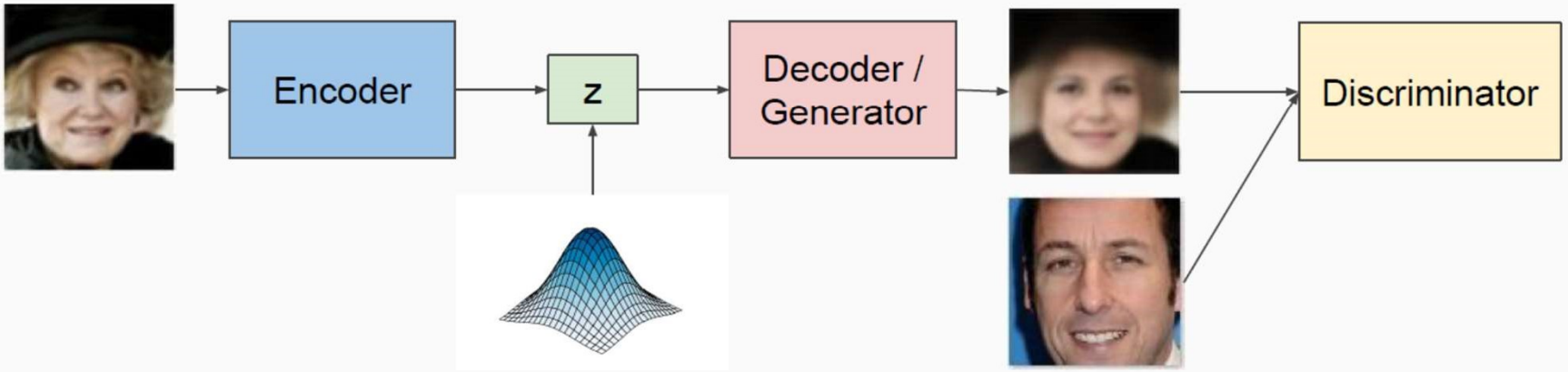
✓: Very sharp images

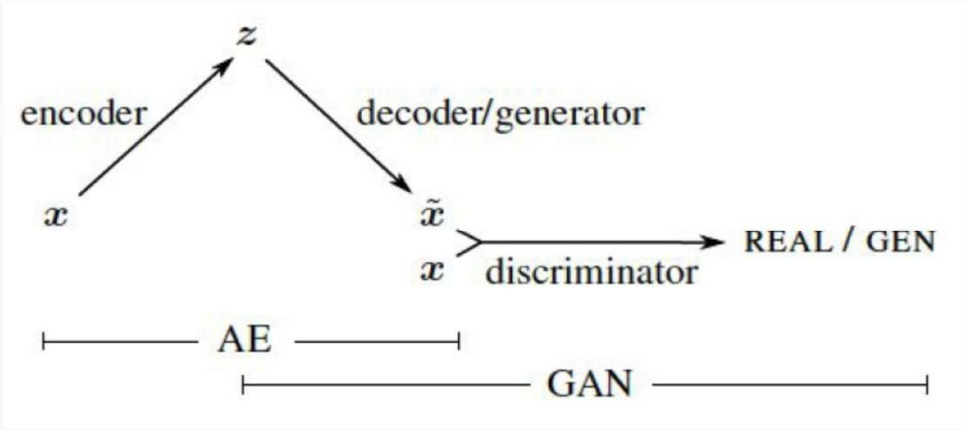
X: Given an X **difficult** to find z. (Need to backprop.)

✓/X: No explicit P(X).

Image Credit: Autoencoding beyond pixels using a learned similarity metric

GAN + VAE (Best of both models)



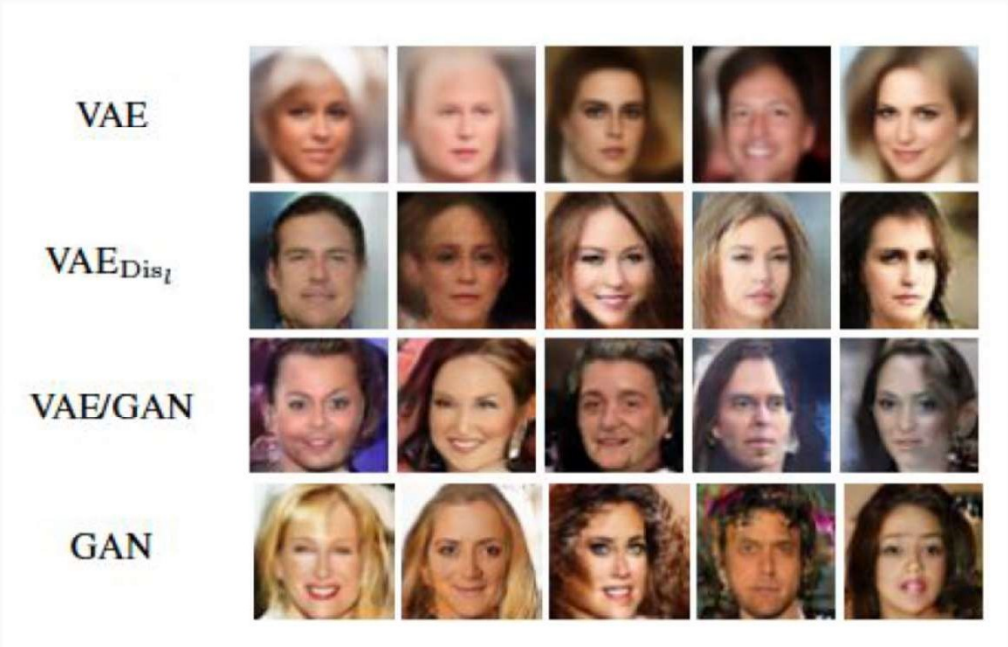


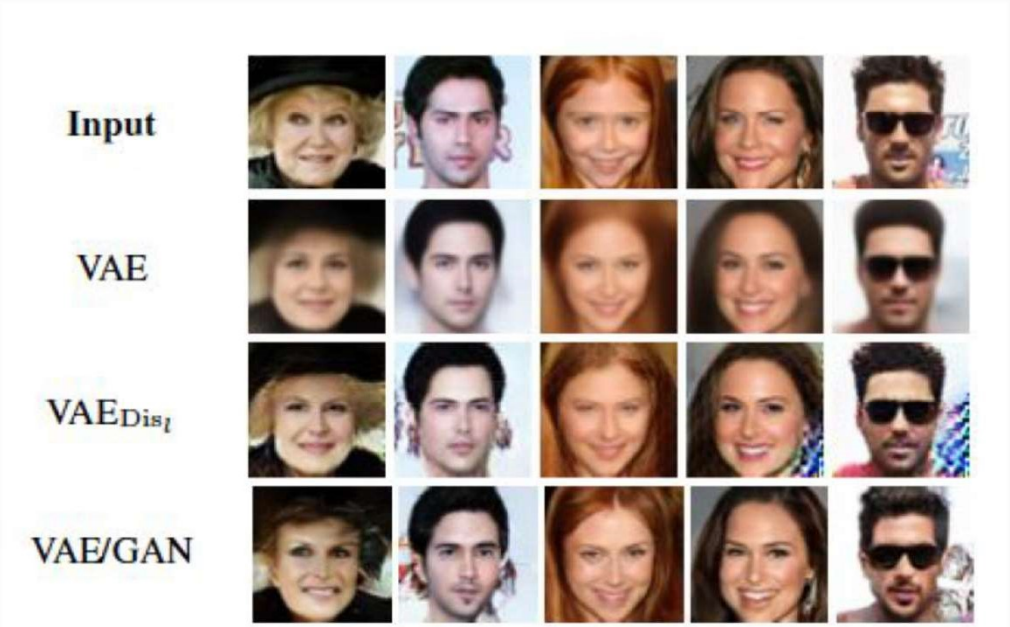
KL Divergence L_2 Difference

$$\mathcal{L} = \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l} + \mathcal{L}_{\text{GAN}}$$

Image Credit: Autoencoding beyond pixels using a learned similarity metric

Results





VAE_{Dis_l} : Train a GAN first, then use the discriminator of GAN to train a VAE.

VAE/GAN: GAN and VAE trained together.

Image Credit: Autoencoding beyond pixels using a learned similarity metric

Conditional VAE (CVAE)

What if we have **labels**? (e.g. digit labels or attributes) Or other inputs we wish to condition on **(Y)**.

- **NONE** of the derivation changes.
- Replace all **P(X|z) with P(X|z,Y)**.
- Replace all **Q(z|X) with Q(z|X,Y).**
- Go through the same KL divergence procedure, to get the same lower bound.

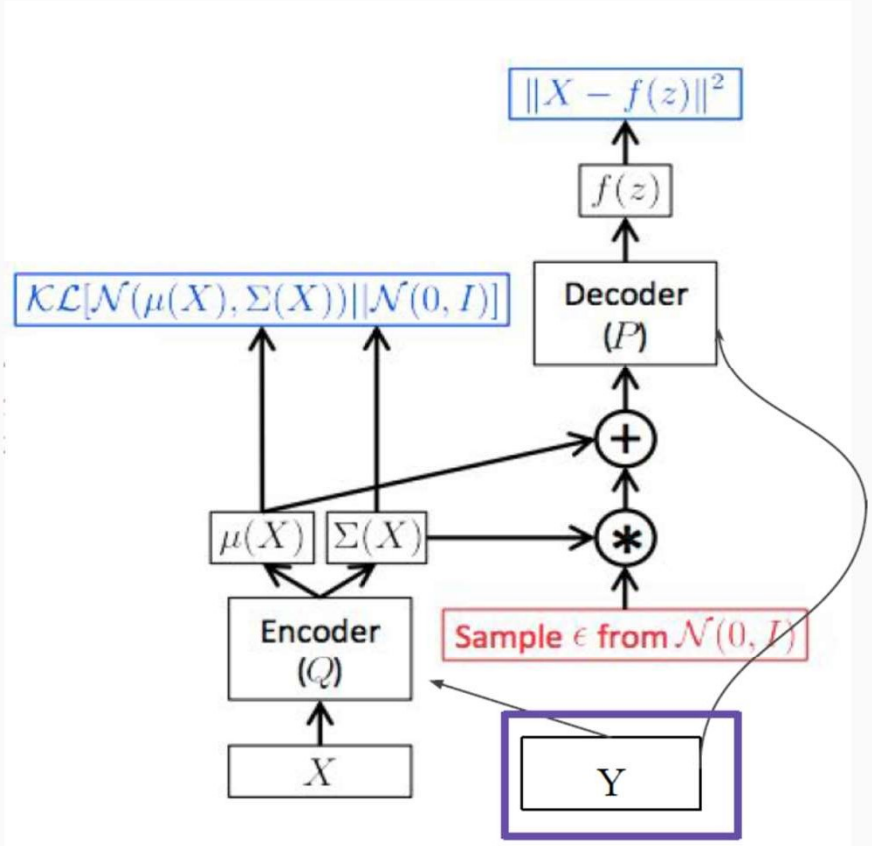


Image Credit: Tutorial on VAEs

Example

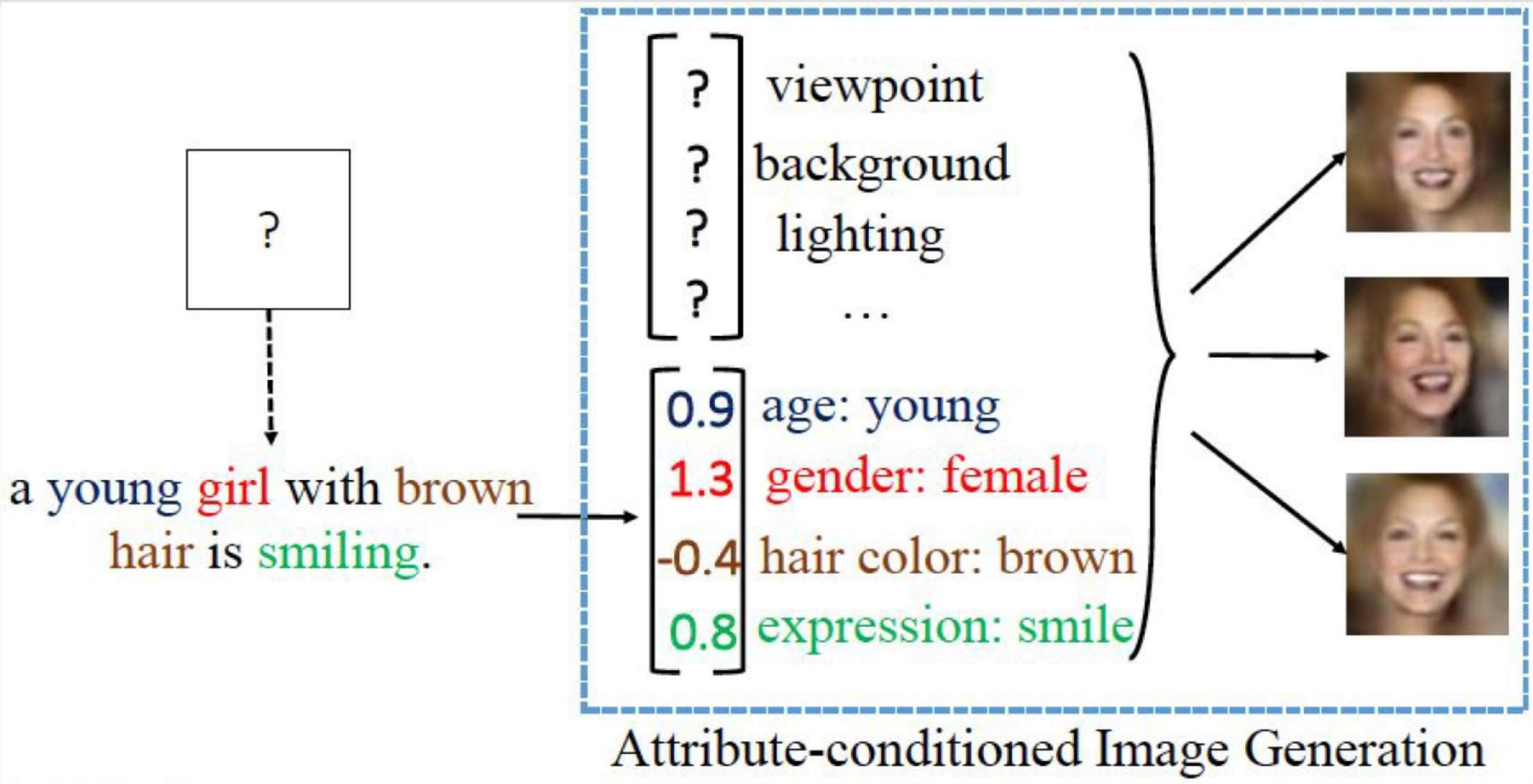
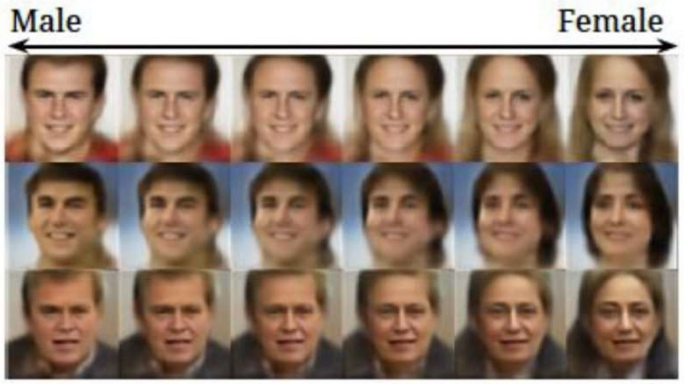
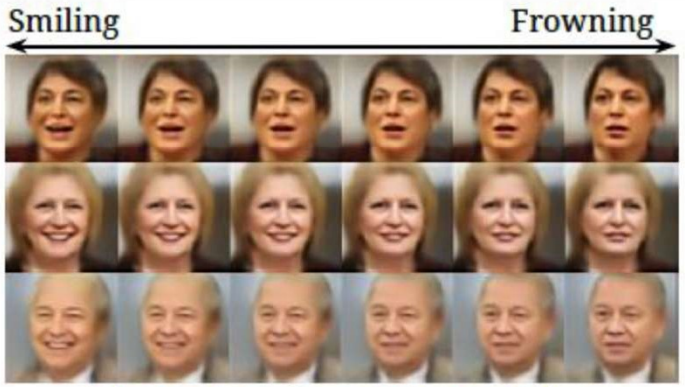


Image Credit: Attribute2Image

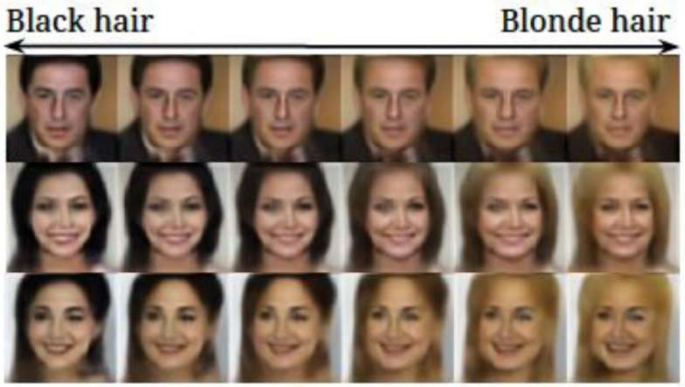
Attribute-conditioned image progression



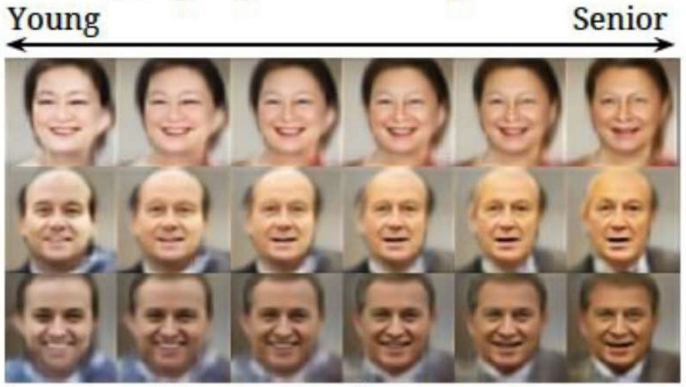
(a) progression on gender



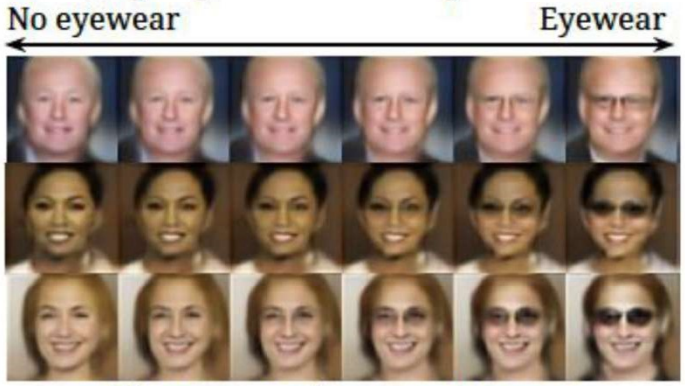
(c) progression on expression



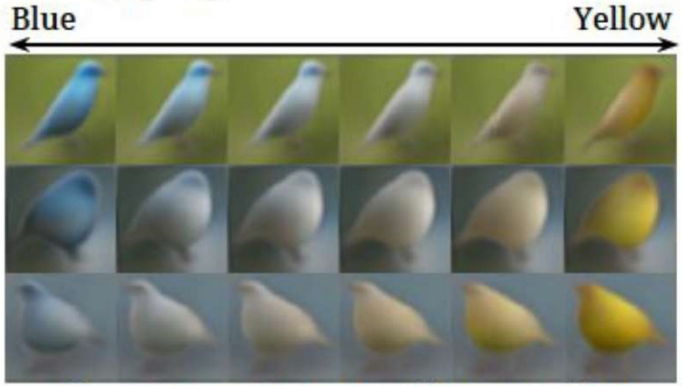
(e) progression on hair color



(b) progression on age



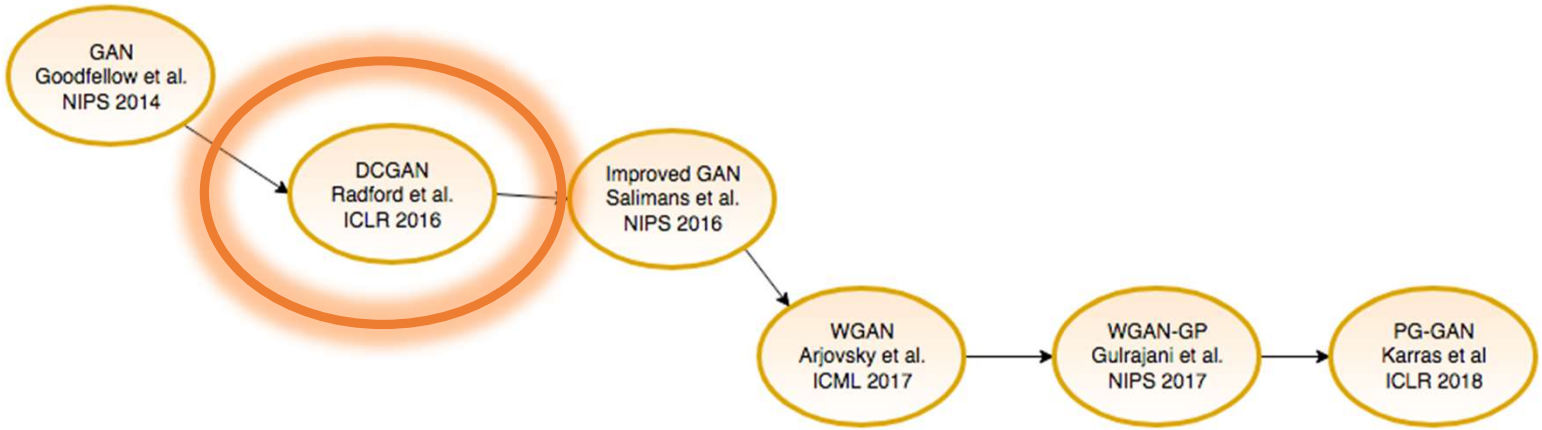
(d) progression on eyewear



(f) progression on primary color

$$p_\theta(x|y,z) \text{ with } z \sim \mathcal{N}(0,I) \text{ and } y = [y_\alpha, y_{rest}], \text{ where } y_\alpha = (1-\alpha)\cdot y_{min} + \alpha \cdot y_{max}$$

Image Credit: Attribute2Image


GAN
Goodfellow et al.
NIPS 2014
DCGAN
Radford et al.
ICLR 2016
Improved GAN
Salimans et al.
NIPS 2016
WGAN
Arjovsky et al.
ICML 2017
WGAN-GP
Gulrajani et al.
NIPS 2017
PG-GAN
Karras et al
ICLR 2018

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

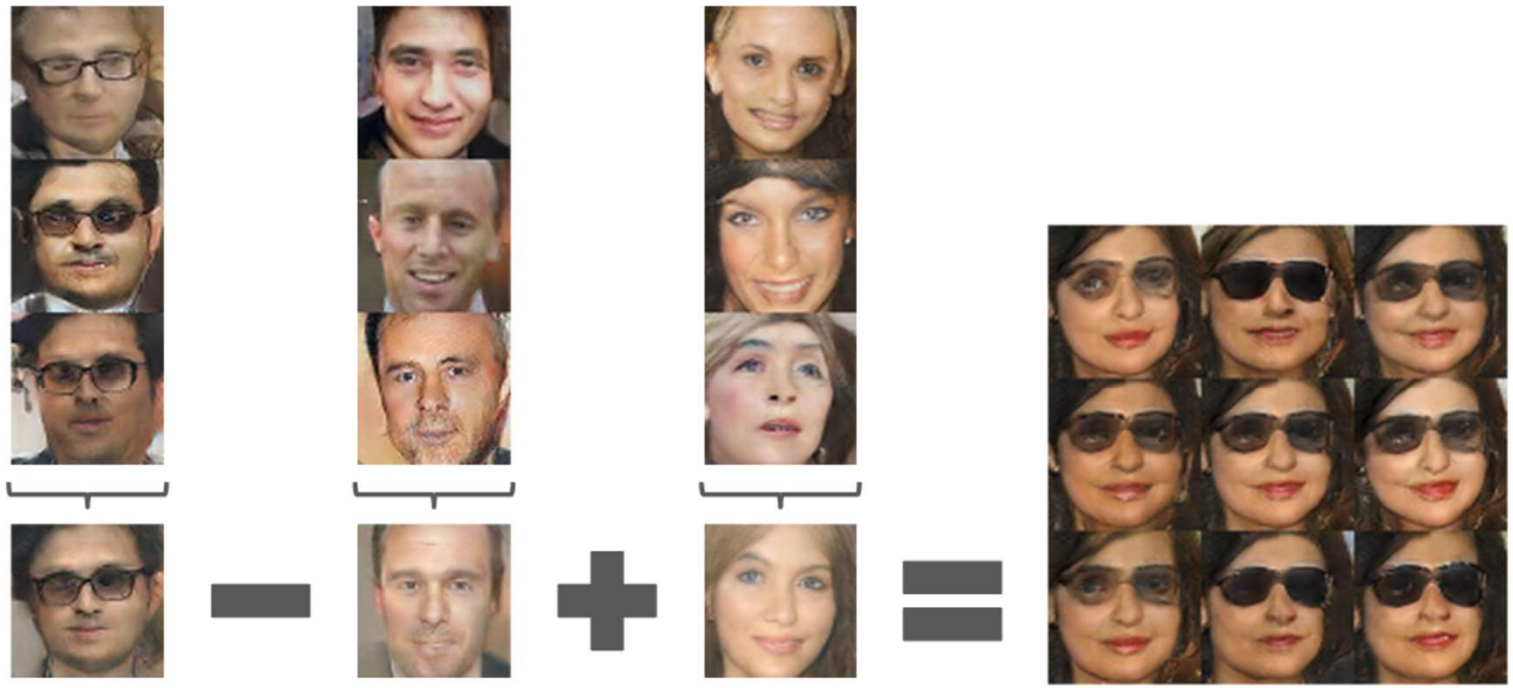
Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks



Improved Techniques for Training GANs

Tim Salimans
tim@openai.com

Ian Goodfellow
ian@openai.com

Wojciech Zaremba
woj@openai.com

Vicki Cheung
vicki@openai.com

Alec Radford
alec.radford@gmail.com

Xi Chen
peter@openai.com

Improved Techniques for Training GANs

- Unimportant Hacks
  - Feature Matching
  - Historical Averaging
  - Label Smoothing
  - Virtual Batch Normalization
- Very Important Hack
  - **Minibatch Discrimination**
- Very Important Metric
  - Inception Score

Improved Techniques for Training GANs

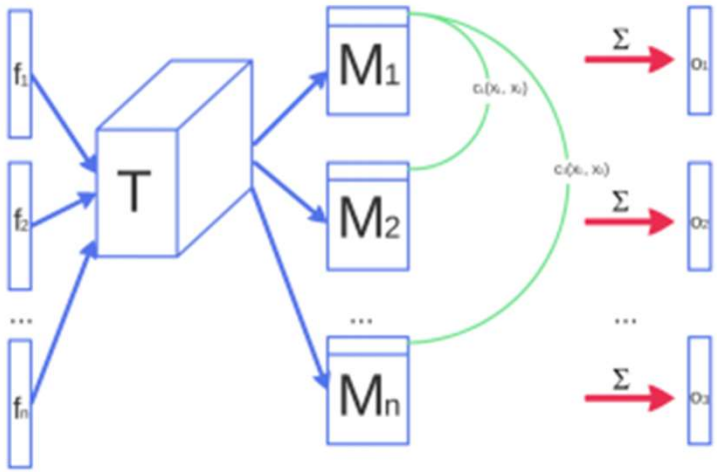


Figure 1: Figure sketches how mini-batch discrimination works. Features $\mathbf{f}(\boldsymbol{x}_i)$ from sample $\boldsymbol{x}_i$ are multiplied through a tensor T, and cross-sample distance is computed.

Improved Techniques for Training GANs

$$\exp(\mathbb{E}_{\boldsymbol{x}}\mathrm{KL}(p(y|\boldsymbol{x})||p(y)))$$

As an alternative to human annotators, we propose an automatic method to evaluate samples, which we find to correlate well with human evaluation: We apply the Inception model[1] [19] to every generated image to get the conditional label distribution $p(y|\boldsymbol{x})$. Images that contain meaningful objects should have a conditional label distribution $p(y|\boldsymbol{x})$ with low entropy. Moreover, we expect the model to generate varied images, so the marginal $\int p(y|\boldsymbol{x} = G(z))dz$ should have high entropy. Combining these two requirements, the metric that we propose is: $\exp(\mathbb{E}_{\boldsymbol{x}}\mathrm{KL}(p(y|\boldsymbol{x})||p(y)))$, where we exponentiate results so the values are easier to compare. Our *Inception score* is closely related to the objective used for training generative models in CatGAN [14]: Although we had less success using such an objective for training, we find it is a good metric for evaluation that correlates very well with human judgment. We find that it's important to evaluate the metric on a large enough number of samples (i.e. $50k$) as part of this metric measures diversity.

Improved Techniques for Training GANs

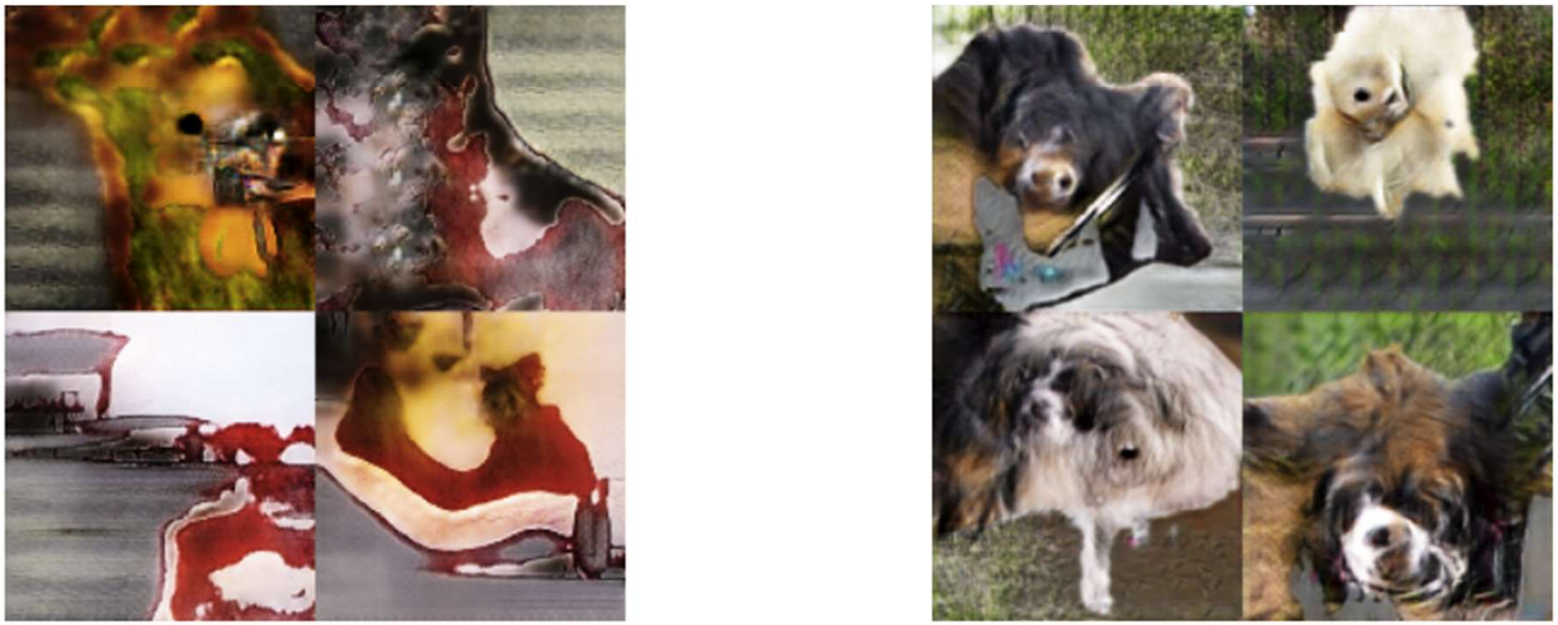
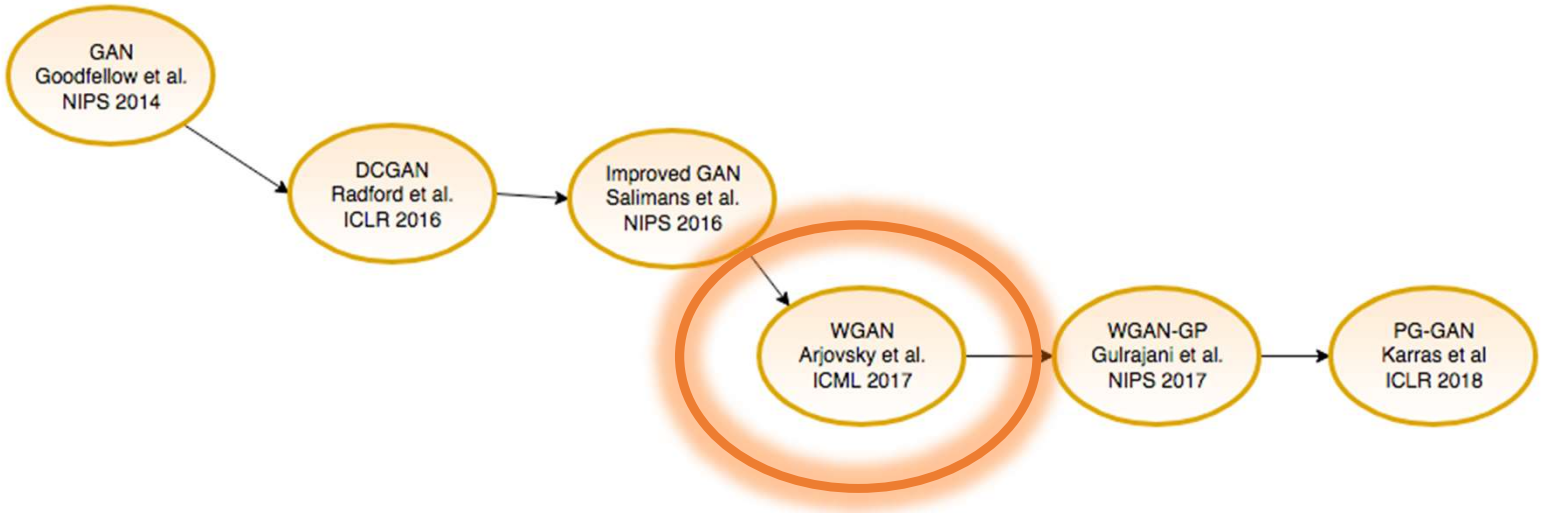
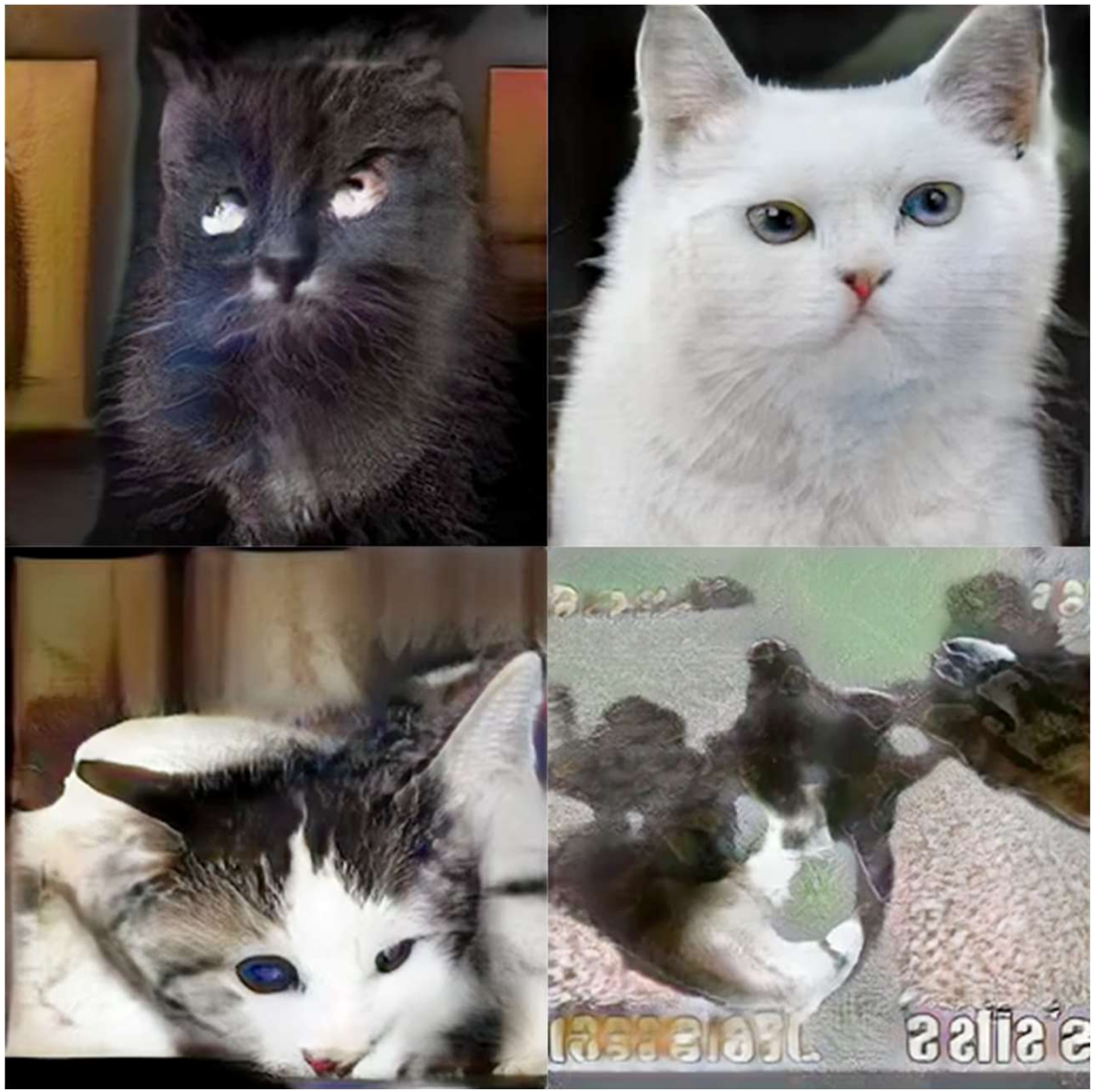
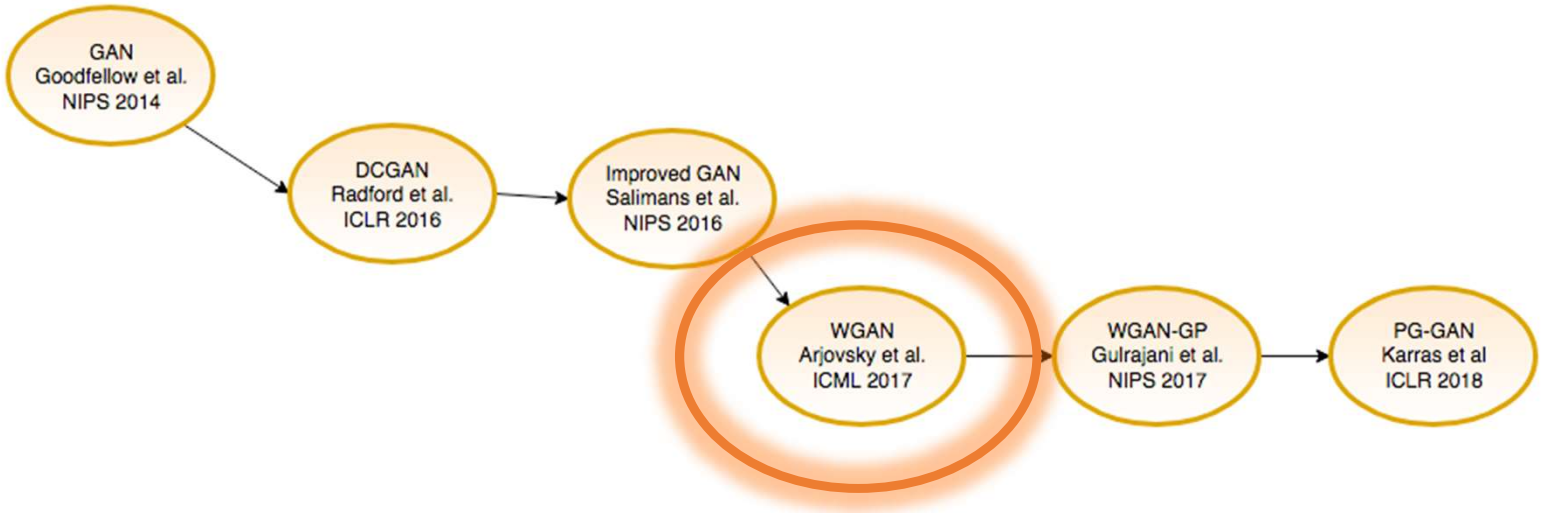


Figure 6: Samples generated from the ImageNet dataset. *(Left)* Samples generated by a DCGAN. *(Right)* Samples generated using the techniques proposed in this work. The new techniques enable GANs to learn recognizable features of animals, such as fur, eyes, and noses, but these features are not correctly combined to form an animal with realistic anatomical structure.


GAN
Goodfellow et al.
NIPS 2014
DCGAN
Radford et al.
ICLR 2016
Improved GAN
Salimans et al.
NIPS 2016
WGAN
Arjovsky et al.
ICML 2017
WGAN-GP
Gulrajani et al.
NIPS 2017
PG-GAN
Karras et al
ICLR 2018

Intermission: Cat Videos




GAN
Goodfellow et al.
NIPS 2014
DCGAN
Radford et al.
ICLR 2016
Improved GAN
Salimans et al.
NIPS 2016
WGAN
Arjovsky et al.
ICML 2017
WGAN-GP
Gulrajani et al.
NIPS 2017
PG-GAN
Karras et al
ICLR 2018

Wasserstein GAN

Martin Arjovsky[1], Soumith Chintala[2], and Léon Bottou[1,2]

[1]Courant Institute of Mathematical Sciences
[2]Facebook AI Research

Wasserstein GAN

- Training DCGAN is unstable!
- WGAN: Replace classification with regression
- Estimate Earth-Mover's Distance

The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right] , \tag{1}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. The EM distance then is the "cost" of the optimal transport plan.

Wasserstein GAN

- Linear, not sigmoid output for discriminator
- Conceptually “Discriminator” is now a “Critic”
  - D(x) is now regression, not classification
- Discriminator must be Lipschitz-continuous
  - WGAN: Limit all weights to [-.01, .01]

W-GAN Theory

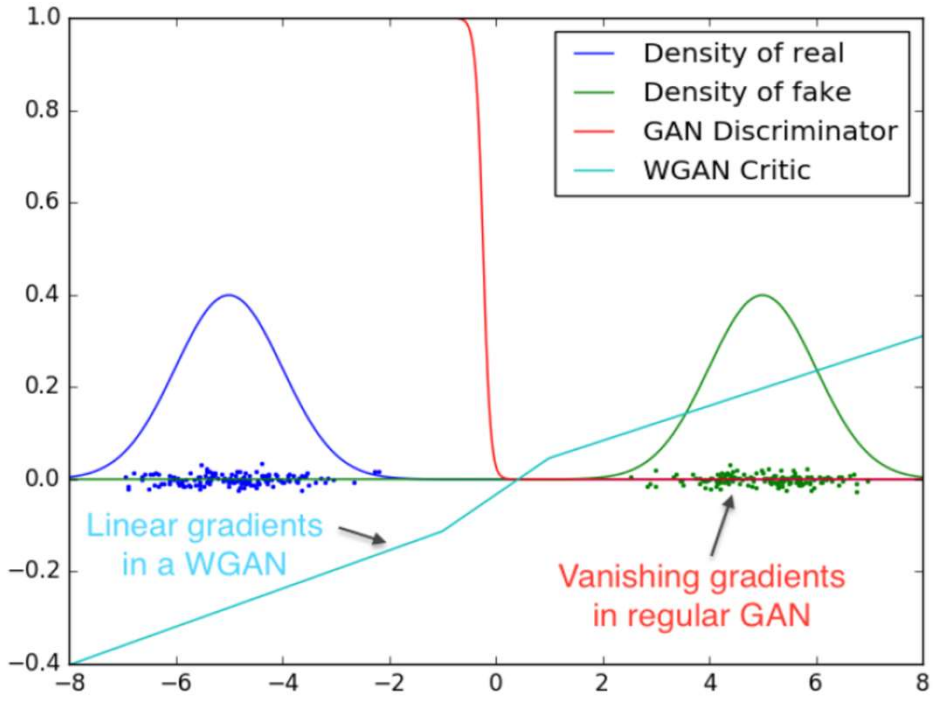
- Kontorovich-Rubinstein duality:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

- Hence we want to maximize the difference of P_r and P_θ we solve
  - f_w a deep network (approximates any function)

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z)]$$

W-GAN critic vs. GAN discriminator



Wasserstein GAN

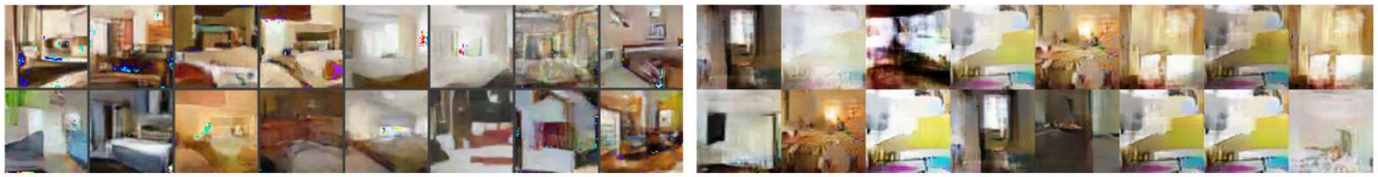
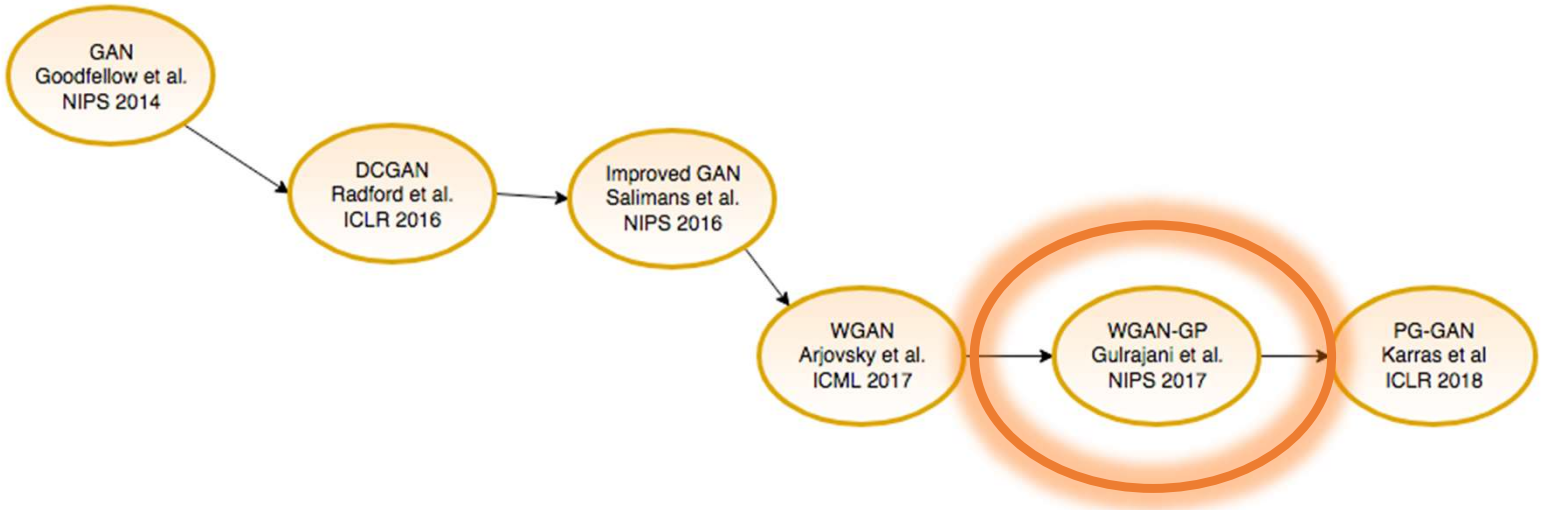


Figure 7: Algorithms trained with an MLP generator with 4 layers and 512 units with ReLU nonlinearities. The number of parameters is similar to that of a DCGAN, but it lacks a strong inductive bias for image generation. Left: WGAN algorithm. Right: standard GAN formulation. The WGAN method still was able to produce samples, lower quality than the DCGAN, and of higher quality than the MLP of the standard GAN. Note the significant degree of mode collapse in the GAN MLP.


GAN
Goodfellow et al.
NIPS 2014
DCGAN
Radford et al.
ICLR 2016
Improved GAN
Salimans et al.
NIPS 2016
WGAN
Arjovsky et al.
ICML 2017
WGAN-GP
Gulrajani et al.
NIPS 2017
PG-GAN
Karras et al
ICLR 2018

Improved Training of Wasserstein GANs

Ishaan Gulrajani[1]*, **Faruk Ahmed**[1], **Martin Arjovsky**[2], **Vincent Dumoulin**[1], **Aaron Courville**[1,3]
[1] Montreal Institute for Learning Algorithms
[2] Courant Institute of Mathematical Sciences
[3] CIFAR Fellow
igul222@gmail.com
{faruk.ahmed,vincent.dumoulin,aaron.courville}@umontreal.ca
ma4371@nyu.edu

Improved Wasserstein GAN

- WGAN: Limit all weights to [-.01, .01]
- WGAN-GP: Apply Gradient Penalty instead

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}}. \tag{3}$$

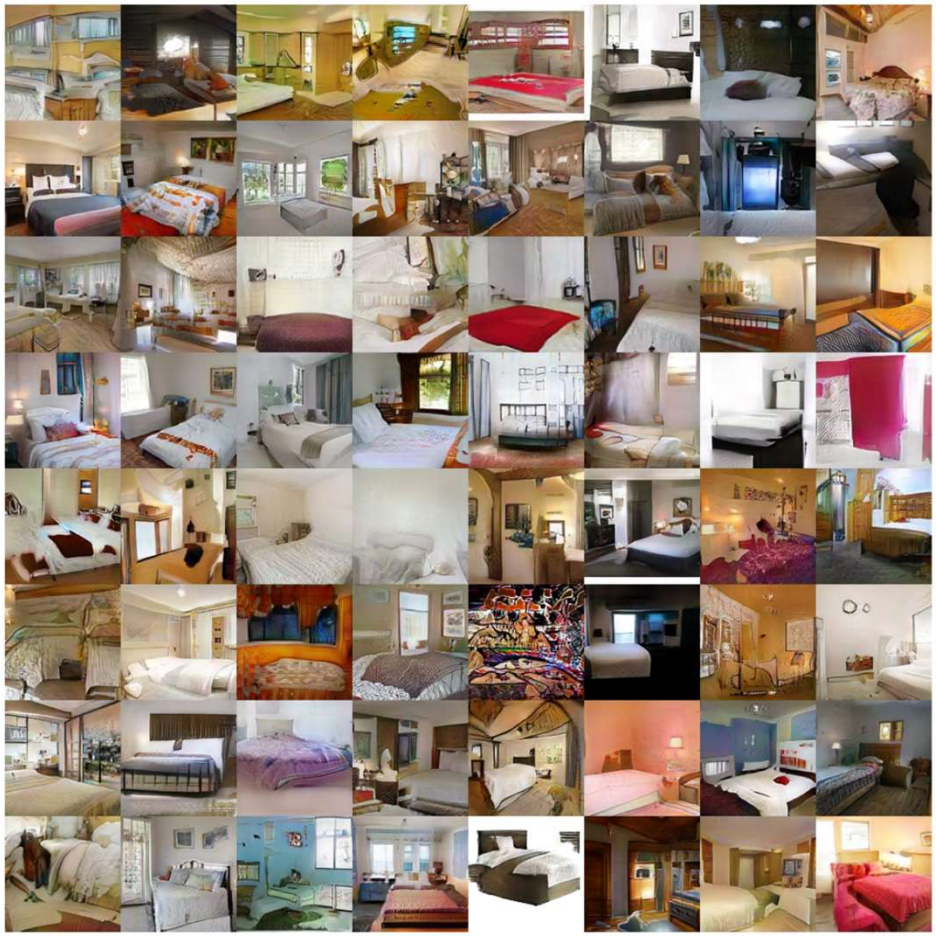
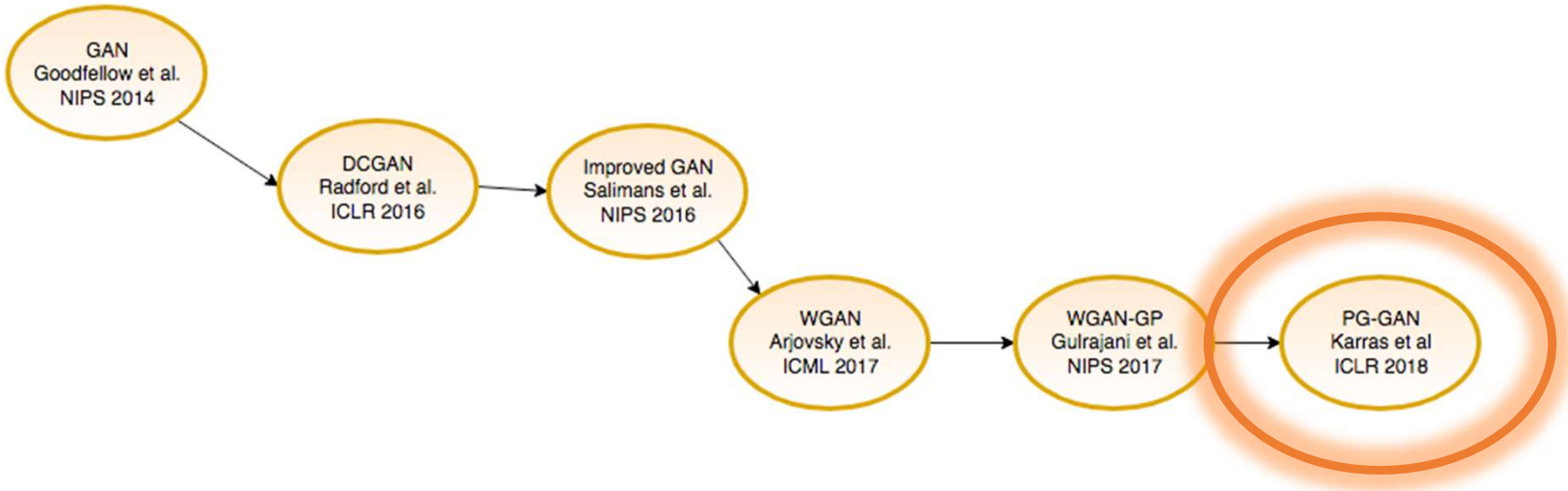


Figure 4: Samples of 128×128 LSUN bedrooms. We believe these samples are at least comparable to the best published results so far.


GAN
Goodfellow et al.
NIPS 2014
DCGAN
Radford et al.
ICLR 2016
Improved GAN
Salimans et al.
NIPS 2016
WGAN
Arjovsky et al.
ICML 2017
WGAN-GP
Gulrajani et al.
NIPS 2017
PG-GAN
Karras et al
ICLR 2018

PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION

Tero Karras NVIDIA
Timo Aila NVIDIA
Samuli Laine NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
{tkarras,taila,slaine,jlehtinen}@nvidia.com

Progressive Growing of GANs

- Contributions
  - Progressive Per-Layer Training
  - Minibatch Standard Deviation
  - Weight and Feature Normalization
- Metrics
  - Multiscale Structural Similarity
  - Sliced Wasserstein Distance
- Experiments
  - Ablation Studies
  - CIFAR-10 Inception Score
  - Nearest-Neighbor Comparisons

Contributions

Progressive Training

- Standard GAN: Generator and Discriminator
- Uses the WGAN-GP loss function
- Learns one layer at a time
  - Trains until convergence on tiny 8x8 images
  - Then appends a layer to G, D
  - Trains until convergence on 16x16 images…
- Learn global structure first, then details
- Related to curriculum learning
- Like Deep Belief Nets from ancient history (2008)

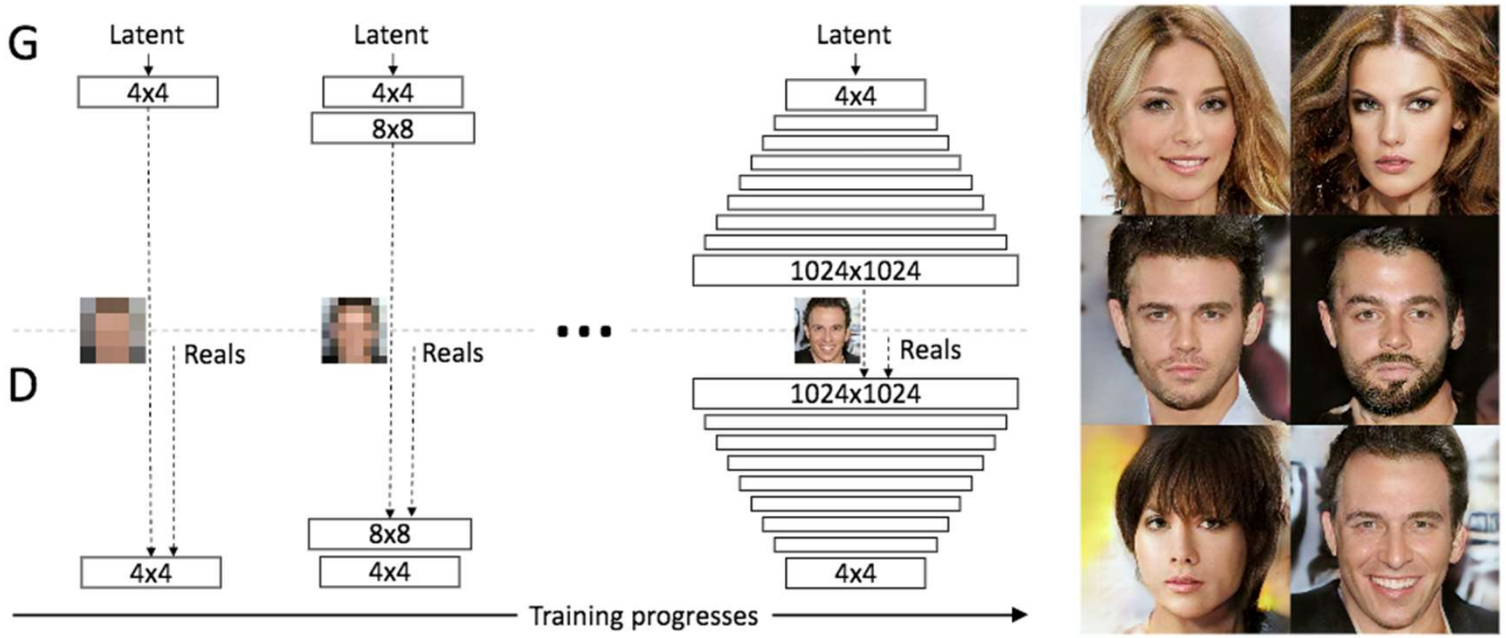


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

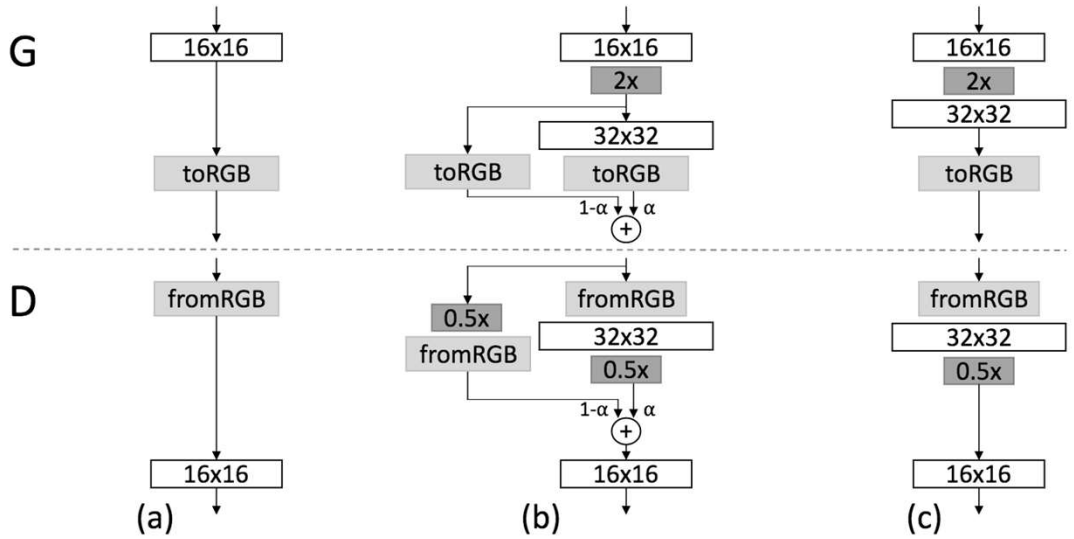


Figure 2: When doubling the resolution of the generator (G) and discriminator (D) we fade in the new layers smoothly. This example illustrates the transition from 16×16 images (a) to 32×32 images (c). During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight α increases linearly from 0 to 1. Here $\boxed{2\times}$ and $\boxed{0.5\times}$ refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The $\boxed{\text{toRGB}}$ represents a layer that projects feature vectors to RGB colors and $\boxed{\text{fromRGB}}$ does the reverse; both use 1×1 convolutions. When training the discriminator, we feed in real images that are downscaled to match the current resolution of the network. During a resolution transition, we interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.

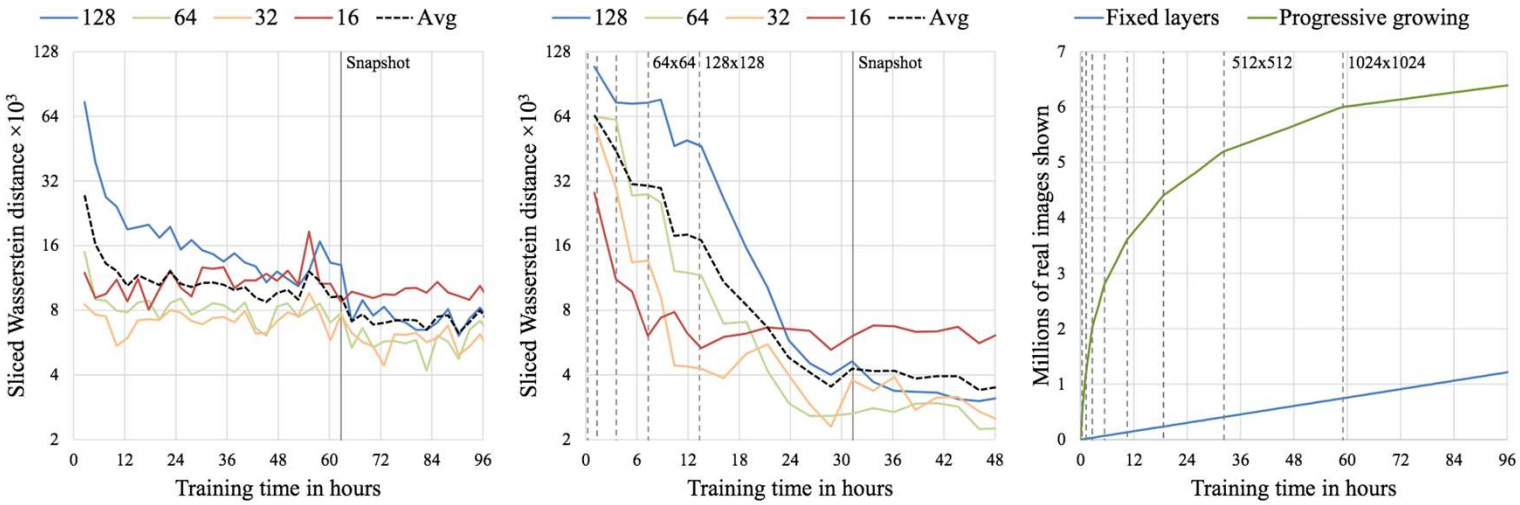


Figure 4: Effect of progressive growing on training speed and convergence. The timings were measured on a single-GPU setup using NVIDIA Tesla P100. (a) Statistical similarity with respect to wall clock time for Gulrajani et al. (2017) using CELEBA at 128×128 resolution. Each graph represents sliced Wasserstein distance on one level of the Laplacian pyramid, and the vertical line indicates the point where we stop the training in Table 1. (b) Same graph with progressive growing enabled. The dashed vertical lines indicate points where we double the resolution of G and D. (c) Effect of progressive growing on the raw training speed in 1024×1024 resolution.

Minibatch Standard Deviation

- Recall Minibatch Discrimination (Improved GAN)
  - Attempts to limit mode collapse by showing the discriminator entire batches, not individual images
  - If all images in a batch are identical, they are all fake
- Works as an extra layer in the discriminator
- Inserted near the end, before a FC layer

Minibatch Standard Deviation

- Recall Minibatch Discrimination (Improved GAN)

The output $o(x_i)$ for this *minibatch layer* for a sample $\boldsymbol{x}_i$ is then defined as the sum of the $c_b(\boldsymbol{x}_i, \boldsymbol{x}_j)$'s to all other samples:

$$o(\boldsymbol{x}_i)_b = \sum_{j=1}^{n} c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}$$

$$o(\boldsymbol{x}_i) = \left[o(\boldsymbol{x}_i)_1, o(\boldsymbol{x}_i)_2, \ldots, o(\boldsymbol{x}_i)_B\right] \in \mathbb{R}^B$$

$$o(\mathbf{X}) \in \mathbb{R}^{n \times B}$$

$$c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) \;=\; \exp(-||M_{i,b} - M_{j,b}||_{L_1}) \;\in\; \mathbb{R}.$$

Minibatch Standard Deviation

- Minibatch Standard Deviation: Simpler approach
  - Compute variance of each feature at spatial location
  - Average the std. dev. among all features, locations
  - Arrive at a single scalar value (mean of std. dev)
- Broadcast that scalar value to all images
  - Take the mean of all those values
- Single scalar value represents “diversity”
- Discriminator quickly learns that diversity is good

Weight Normalization

- Less important but worth mentioning
- Replacement for batch norm, pixel norm, etc
- During training, weights are explicitly scaled

$$\hat{w}_i = w_i / c,$$

- Interacts with Adam/RMSProp momentum
- Ensures equal dynamic range for all layers

Feature Normalization

- Clamp feature vectors to the unit sphere, ie.
  - Divide each vector by its Euclidean norm
  - Normalize features by their magnitude
- Other papers do this to the latent noise vector
  - Here we do it **everywhere,** works surprisingly well

$$b_{x,y} = a_{x,y} / \sqrt{\frac{1}{N} \sum_{j=0}^{N-1} (a_{x,y}^{j})^2 + \epsilon}$$

Metrics

Evaluating Generative Models

- Recall Inception Score (Improved GAN)
  - Single scalar value, larger is better
  - Increases with increasing “objectness”
  - Increases with diversity (of classifications)
- Problem: Entangles realism and diversity
  - Is Inception 5.0 more realistic than 4.9?
- Problem: Only measures inter-class diversity
  - Score is unaffected by variation within class
  - A generator could output one realistic image per Imagenet class, and get a perfect Inception score

Evaluating Generative Models

- Better method: Use two separate metrics

- Multiscale Structural Similarity (MS-SSIM)
  - Measures diversity within a set of images

- Sliced Wasserstein Distance (SWD)
  - Measures statistical similarity between **two** sets

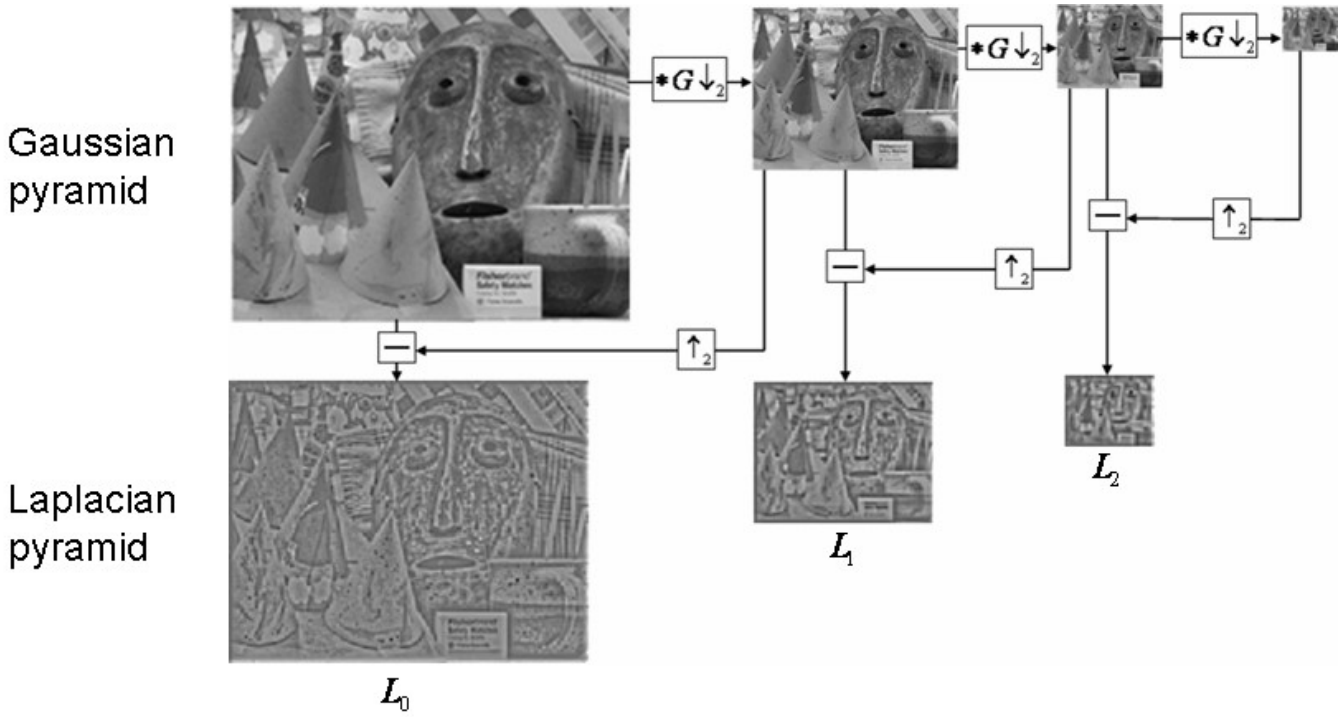
Multiscale Structural Similarity

- Similarity metric used in image processing
  - Ranges from 0 (no similarity) to 1 (identical)
  - Works at multiple downsampled scales

- Here, MS-SSIM applies to generator output
  - Average of many sampled MS-SSIM(x, y) values
  - **Lower** scores (more variety) are **good**
  - Measures **diversity**

Sliced Wasserstein Distance

- For each image, build a Laplacian pyramid



Sliced Wasserstein Distance

- For each image, build a Laplacian pyramid
  - Sample many patches from these pyramids
  - Normalize them by their mean/variance
  - Yields R/G/B histograms at each scale
- Measures difference in distributions
  - Used as SWD(real_images, generated_images)
  - **Lower** score (more similarity) is **better**
  - Measures **realism**

Experiments

Training configuration	CELEBA						LSUN BEDROOM					
	Sliced Wasserstein distance $\times 10^3$					MS-SSIM	Sliced Wasserstein distance $\times 10^3$					MS-SSIM
	128	64	32	16	Avg		128	64	32	16	Avg	
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	**0.0587**
(b) + Progressive growing	4.62	**2.64**	3.78	6.06	4.28	**0.2838**	7.09	6.27	7.40	9.64	7.60	0.0615
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	**3.61**	3.32	**2.71**	6.44	4.02	0.0668
(g) + Pixelwise normalization	**4.06**	3.04	**2.02**	**5.13**	**3.56**	0.2845	3.89	**3.05**	3.24	**5.87**	**4.01**	0.0640
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636

Table 1: Sliced Wasserstein distance (SWD) between the generated and training images (Section 5) and multi-scale structural similarity (MS-SSIM) among the generated images for several training setups at 128×128. For SWD, each column represents one level of the Laplacian pyramid, and the last one gives an average of the four distances.

(a) (b) (c) (d) (e∗) (e) (f) (g) (h) Converged

Figure 3: (a) – (g) CELEBA examples corresponding to rows in Table 1. These are intentionally non-converged. (h) Our converged result. Notice that some images show aliasing and some are not sharp – this is a flaw of the dataset, which the model learns to replicate faithfully.

Figure 9: CIFAR10 images generated using a network that was trained unsupervised (no label conditioning), and achieves a record 8.80 inception score.

UNSUPERVISED		
Method		Inception score
ALI	(Dumoulin et al., 2016)	5.34 ± 0.05
GMAN	(Durugkar et al., 2016)	6.00 ± 0.19
Improved GAN	(Salimans et al., 2016)	6.86 ± 0.06
CEGAN-Ent-VI	(Dai et al., 2017)	7.07 ± 0.07
LR-AGN	(Yang et al., 2017)	7.17 ± 0.17
DFM	(Warde-Farley & Bengio, 2017)	7.72 ± 0.13
WGAN-GP	(Gulrajani et al., 2017)	7.86 ± 0.07
Splitting GAN	(Grinblat et al., 2017)	7.90 ± 0.09
Our (best run)		$\mathbf{8.80 \pm 0.05}$
Our (computed from 10 runs)		$\mathbf{8.56 \pm 0.06}$

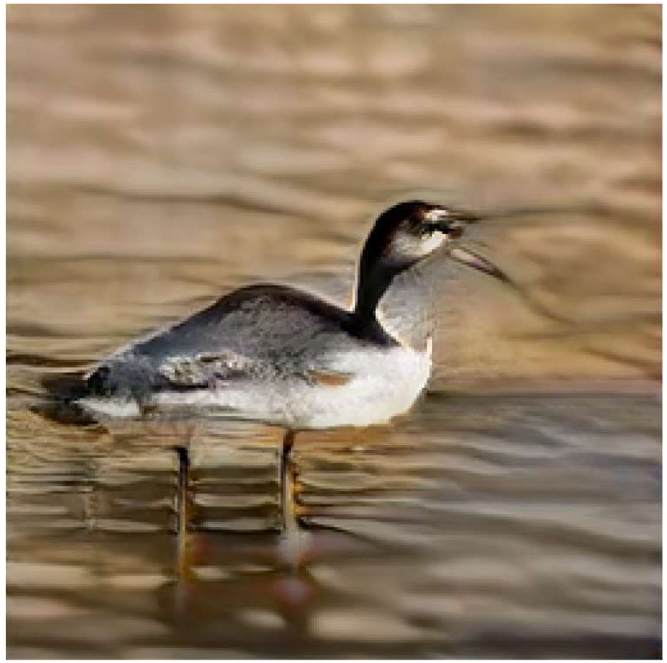
LABEL CONDITIONED		
Method		Inception score
DCGAN	(Radford et al., 2015)	6.58
Improved GAN	(Salimans et al., 2016)	8.09 ± 0.07
AC-GAN	(Odena et al., 2017)	8.25 ± 0.07
SGAN	(Huang et al., 2016)	8.59 ± 0.12
WGAN-GP	(Gulrajani et al., 2017)	8.67 ± 0.14
Splitting GAN	(Grinblat et al., 2017)	8.87 ± 0.09

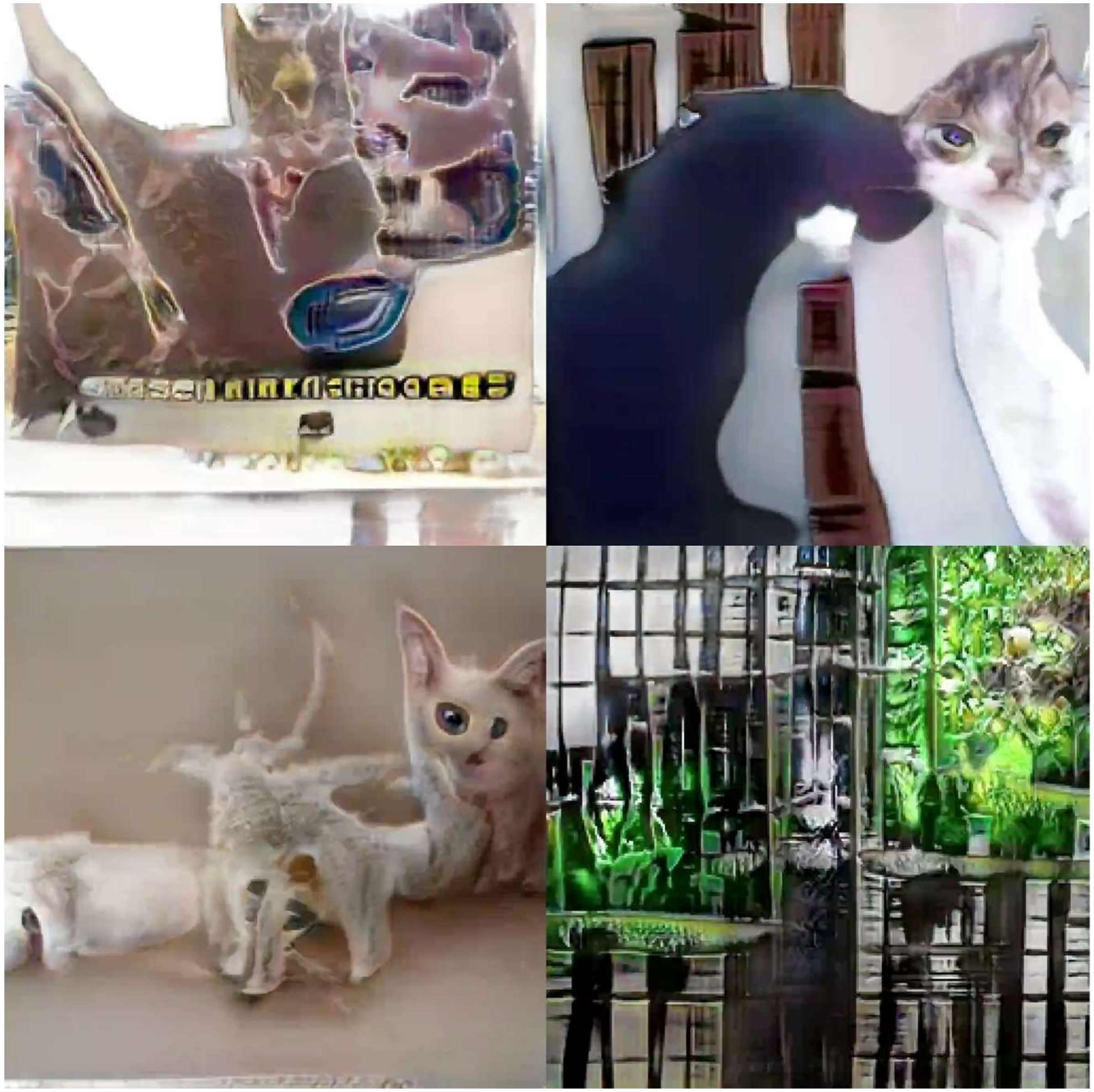
Table 3: CIFAR10 inception scores, higher is better.

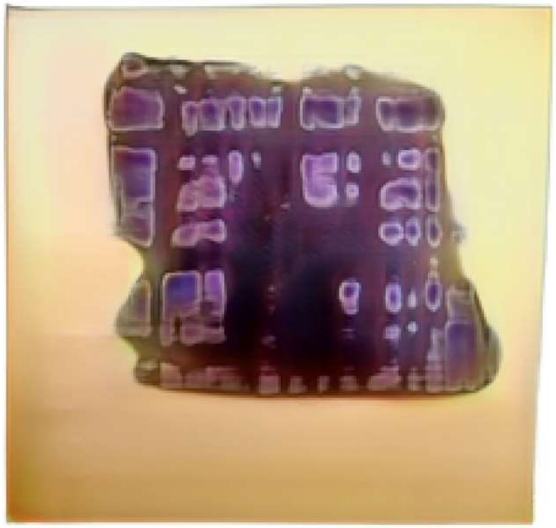
Nearest Neighbors

Comparison with training set images

Results

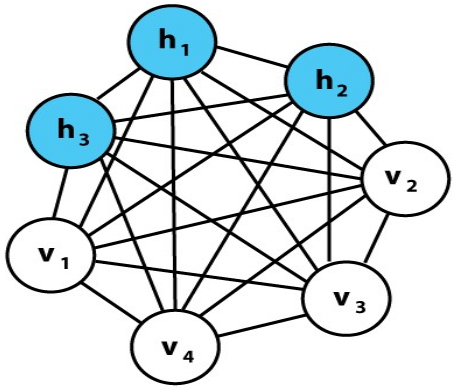






Boltzmann Machine (Fully-connected MRF/CRF)

- Undirected graphical model
- Binary values on each variable
  - $x \in \{0,1\}$
  - $P(x=1) = x$
- Consider only binary interactions



$$E(\mathbf{x};\theta) = -\sum_{i<j} w_{ij} x_i x_j - \sum_i \lambda_i x_i$$

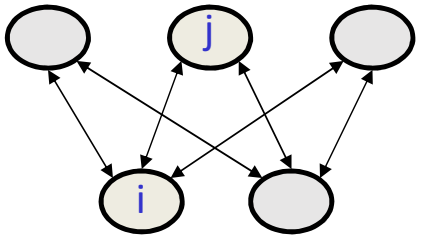
$$P(\mathbf{x};\theta) = \frac{\prod_m f_m(\mathbf{x}_m;\theta_m)}{\sum_{\mathbf{x}} \prod_m f_m(\mathbf{x}_m;\theta_m)} = \frac{e^{-E(\mathbf{x};\theta)}}{\sum_{\mathbf{x}} e^{-E(\mathbf{x};\theta)}} = \frac{f(\mathbf{x};\theta)}{Z(\theta)},$$

$$\theta : \{w_{ij}, \lambda_i\}$$

Boltzmann machine: $E(\boldsymbol{v},\boldsymbol{h}) = \boldsymbol{b}^\top \boldsymbol{v} + \boldsymbol{c}^\top \boldsymbol{h} + \boldsymbol{h}^\top \mathbf{W} \boldsymbol{v} + \boldsymbol{v}^\top \mathbf{U} \boldsymbol{v} + \boldsymbol{h}^\top \mathbf{V} \boldsymbol{h}$

Restricted Boltzmann Machines

- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped.
  - So we can quickly get the exact value of : $< v_i h_j >_{\mathbf{v}}$



$$p(h_j = 1) = \frac{1}{1 + e^{-\left(b_j + \sum_{i \in vis} v_i w_{ij}\right)}}$$

What you gain

1) Conditional probabilities factor nicely

$$P(h|v) = \Pi_i P(h_i|v) \qquad \text{and} \qquad P(v|h) = \Pi_i P(v_i|h)$$

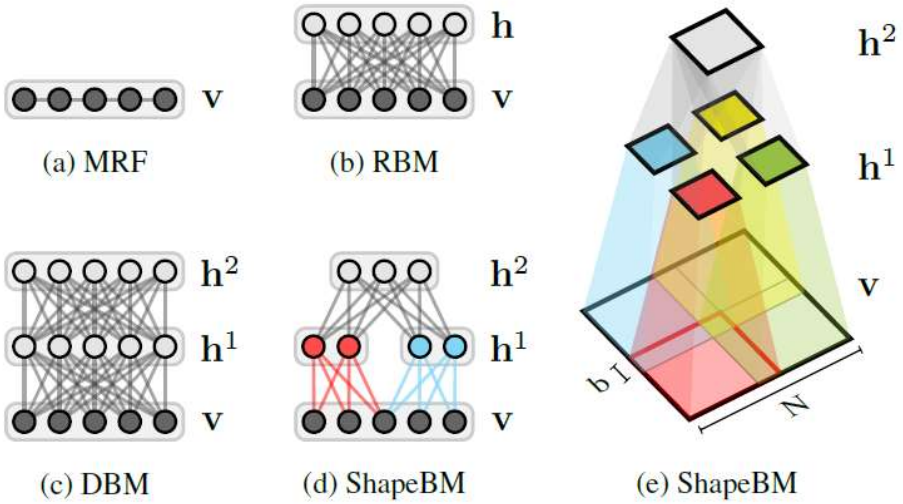
2) Using binary units, we also can get

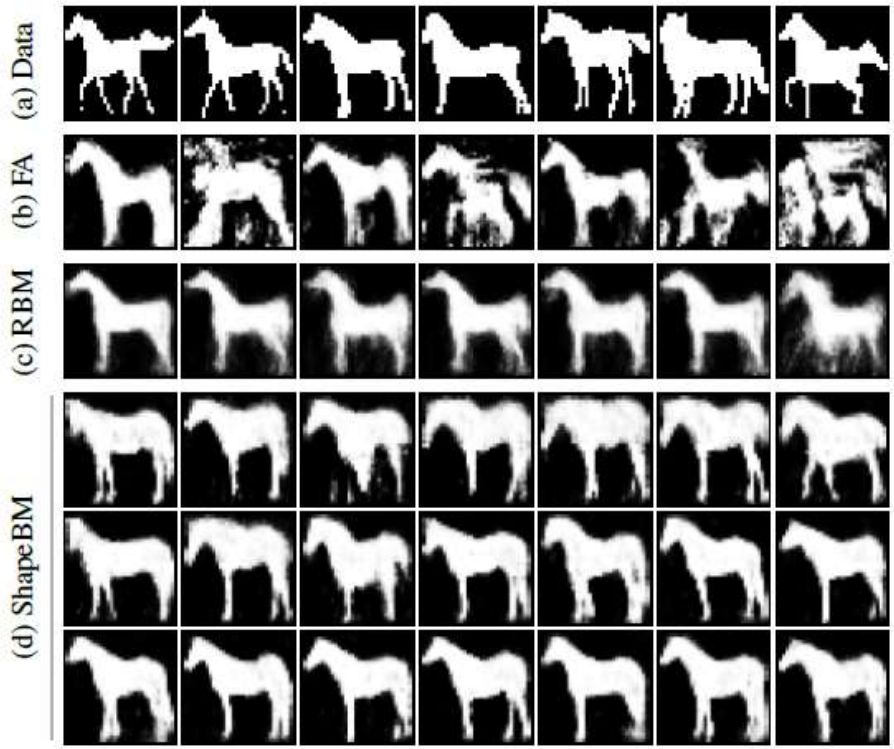
$$P(v_j = 1|h) = \sigma(b_j + W_j^{'} h)$$
$$P(h_i = 1|v) = \sigma(c_i + W_j v)$$

So we can get a sample of the visible or hidden nodes easily...

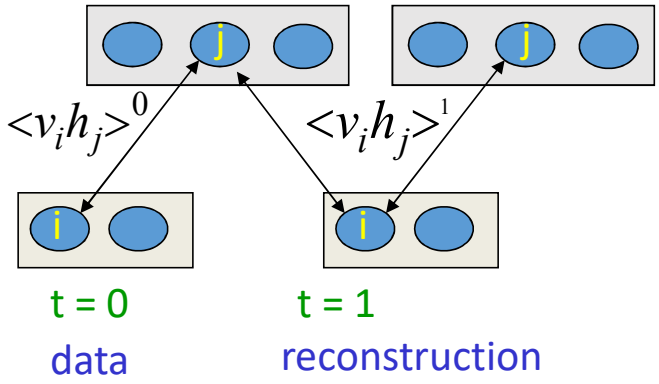
Example: ShapeBM (Eslami et al. 2012)

- Generating shapes
  - 2-layer RBM with local connections
- Learning from many horses





Training: Contrastive divergence



Start with a training vector on the visible units.

Update all the hidden units in parallel.

Update the all the visible units in parallel to get a "reconstruction".

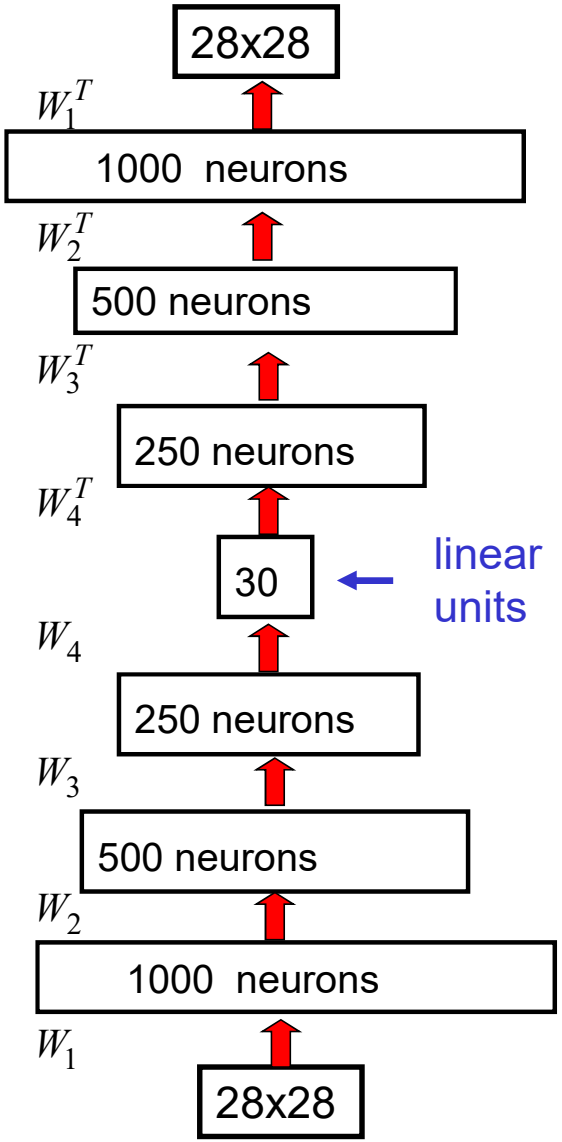
Update the hidden units again.

$$\Delta w_{ij} = \varepsilon(<v_i h_j>^0 - <v_i h_j>^1)$$

This is not following the gradient of the log likelihood. But it works well.

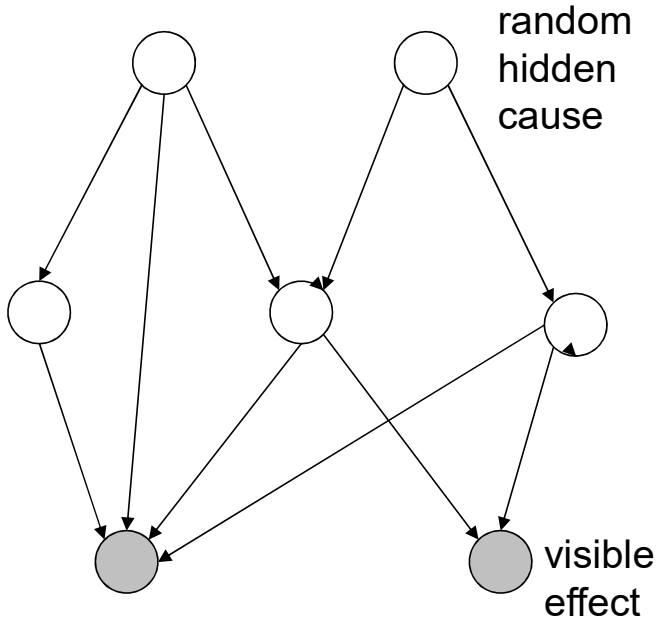
Layerwise Pretraining
(Hinton & Salakhutdinov, 2006)

- They always looked like a really nice way to do non-linear dimensionality reduction:
  - But it is very difficult to optimize deep autoencoders using backpropagation.
- We now have a much better way to optimize them:
  - First train a stack of 4 RBM's
  - Then "unroll" them.
  - Then fine-tune with backprop.



Belief Nets

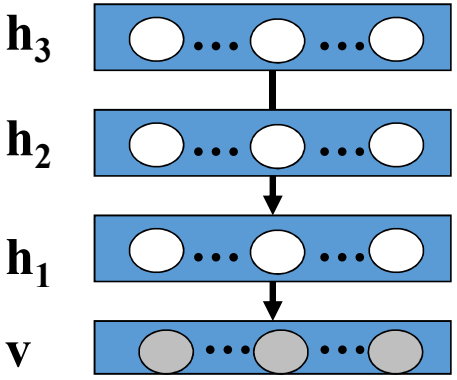
- A belief net is a directed acyclic graph composed of random variables.



Deep Belief Net

- Belief net that is deep
- A generative model
  - $P(v,h_1,\ldots,h_l) = p(v|h_1)\, p(h_1|h_2) \ldots p(h_{l-2}|h_{l-1})\, p(h_{l-1},h_l)$
- Used for unsupervised training of multi-layer deep model.

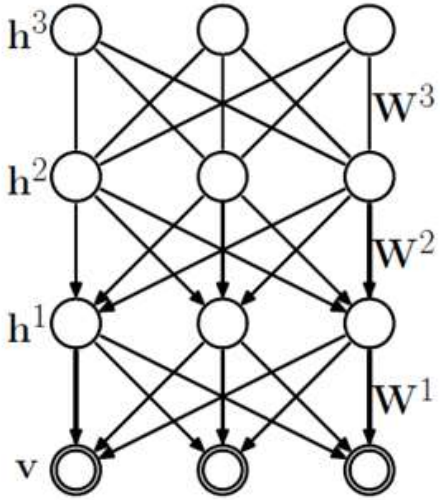
Pixels=>edges=> local shapes=> object parts



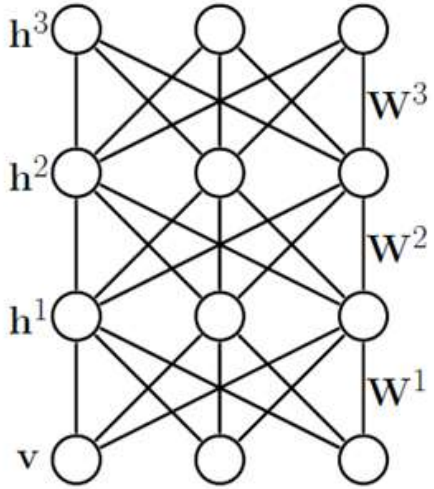
$$P(v,h_1,h_2,h_3) = p(x|h_1)\, p(h_1|h_2)\, p(h_2,h_3)$$

DBNs vs. DBMs

Deep Belief Network



Deep Boltzmann Machine

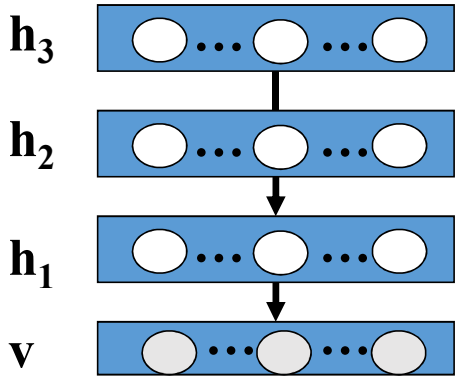


DBNs are hybrid models:

- Inference in DBNs is problematic due to **explaining away**.
- Only greedy pretrainig, **no joint optimization over all layers**.
- Approximate inference is feed-forward: **no bottom-up and top-down**.

Deep Belief Net

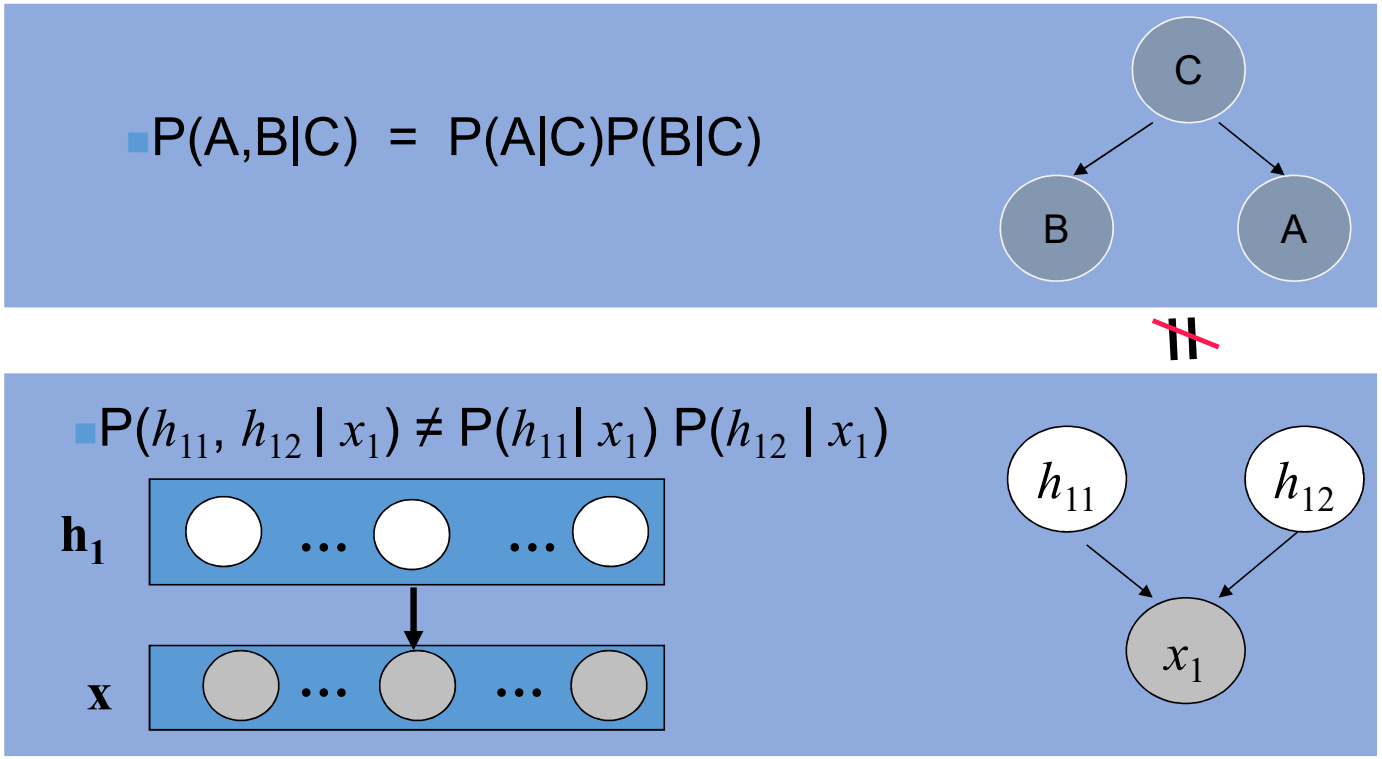
- Learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data
- Inference problem: Infer the states of the unobserved variables.



$$P(v,h_1,h_2,h_3) = p(v|h_1)\, p(h_1|h_2)\, p(h_2,h_3)$$

Deep Belief Net

- Inference problem (the problem of explaining away):

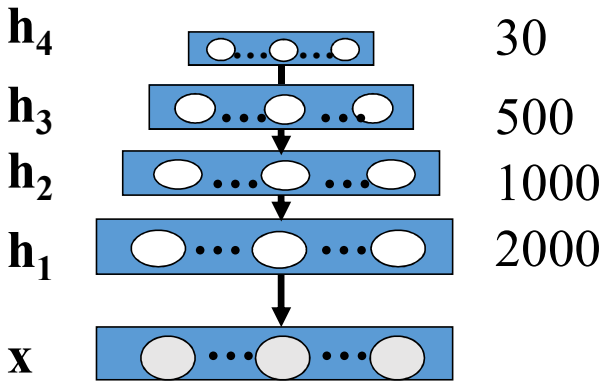


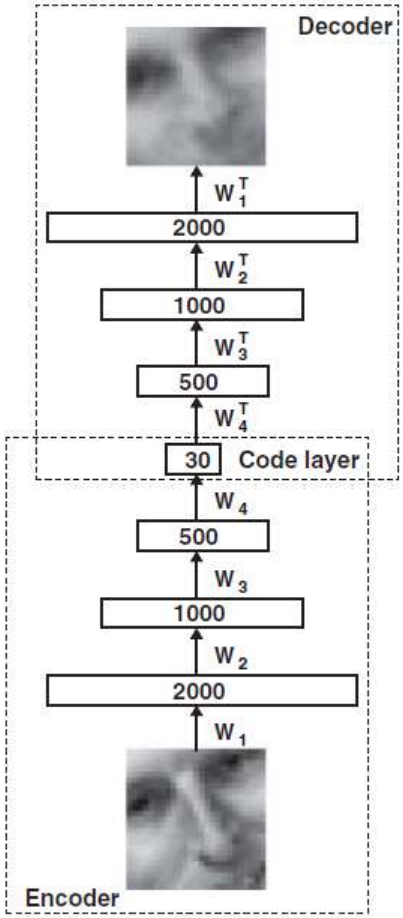
An example from manuscript

Sol: Complementary prior

Deep Belief Net

- Inference problem (the problem of explaining away)
  - Sol: Complementary prior



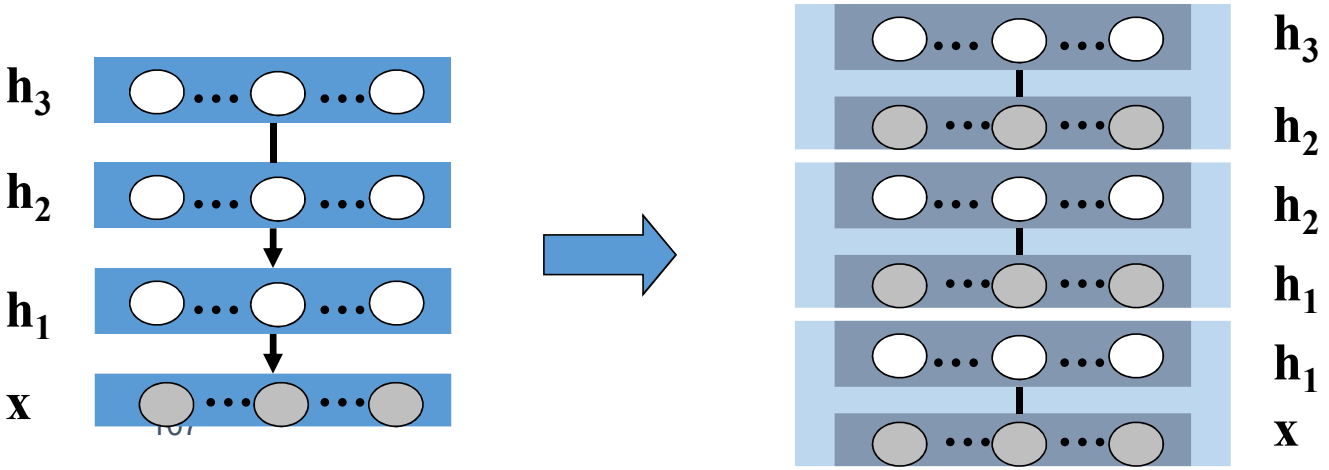


Sol: Complementary prior

Deep Belief Net

$$P(h_i = 1|\mathrm{x}) = \sigma(c_i + W_i \cdot \mathrm{x})$$

- Explaining away problem of Inference (see the manuscript)
  - Sol: Complementary prior, see the manuscript
- Learning problem
  - Greedy layer by layer RBM training (optimize lower bound) and fine tuning
  - Contrastive divergence for RBM training



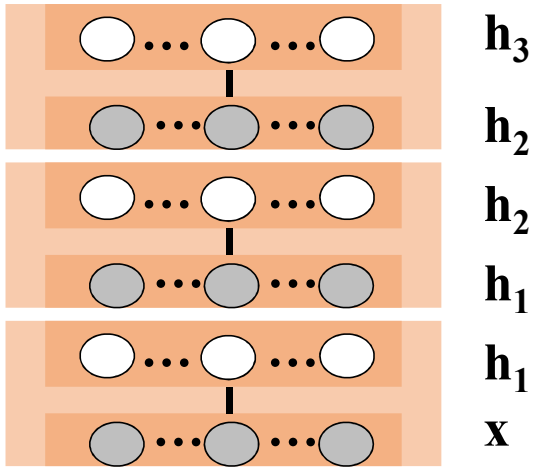
Deep Belief Net

- Why greedy layerwise learning work?
- Optimizing a lower bound:

$$\log P(\mathbf{x}) = \log \sum_{h} P(\mathbf{x}, \mathbf{h}_1)$$

$$\geq \sum_{\mathbf{h}_1} \{Q(\mathbf{h}_1 \mid \mathbf{x})[\log P(\mathbf{h}_1) + \log P(\mathbf{h}_1 \mid \mathbf{x})] - Q(\mathbf{h}_1 \mid \mathbf{x}) \log Q(\mathbf{h}_1 \mid \mathbf{x})]\} \quad (1)$$

- When we fix parameters for layer 1 and optimize the parameters for layer 2, we are optimizing the $P(\mathbf{h}_1)$ in (1)



How many layers should we use?

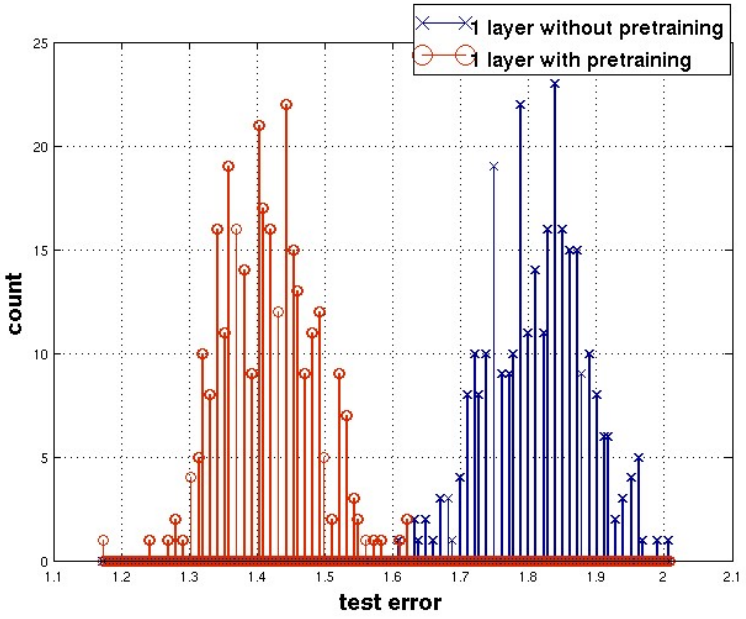
- There might be no universally right depth
  - Bengio suggests that several layers is better than one
  - Results are robust against changes in the size of a layer, but top layer should be big
  - A parameter. Depends on your task.
  - With enough narrow layers, we can model any distribution over binary vectors [1]

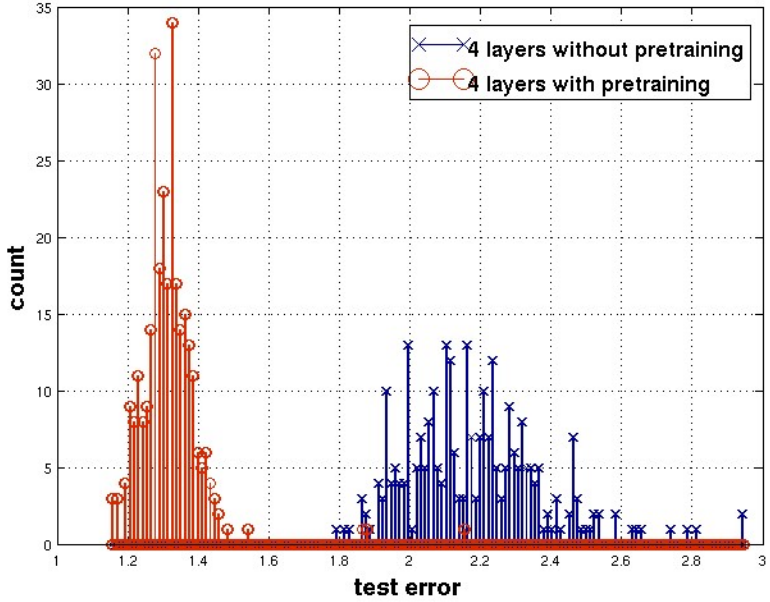
[1] Sutskever, I. and Hinton, G. E., Deep Narrow Sigmoid Belief Networks are Universal Approximators. Neural Computation, 2007

Copied from http://videolectures.net/mlss09uk_hinton_dbn/

Effect of Unsupervised Pre-training (take with a grain of salt)

Erhan et. al. AISTATS'2009

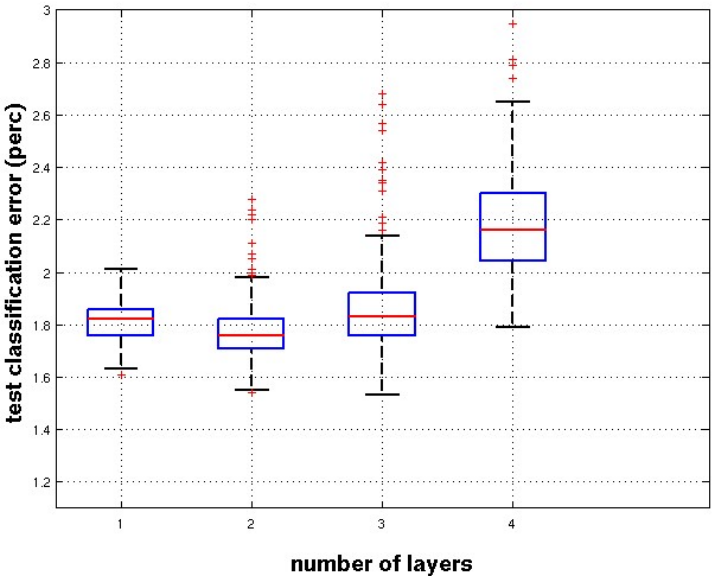




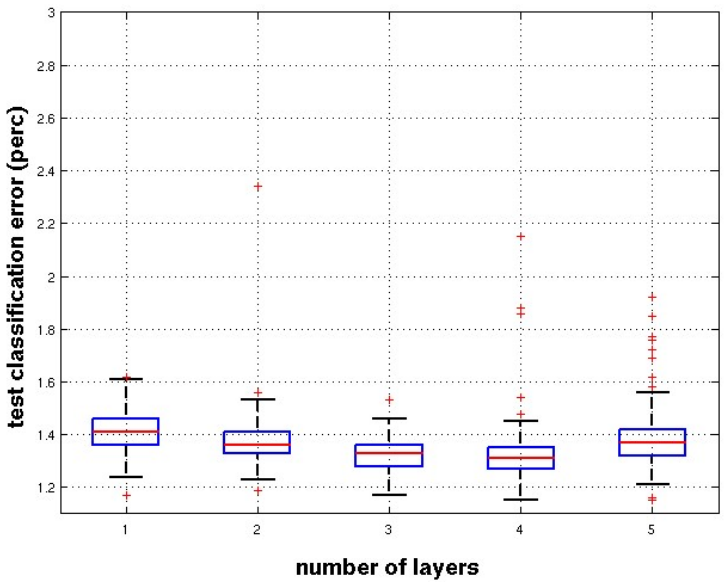
Effect of Depth

without pre-training

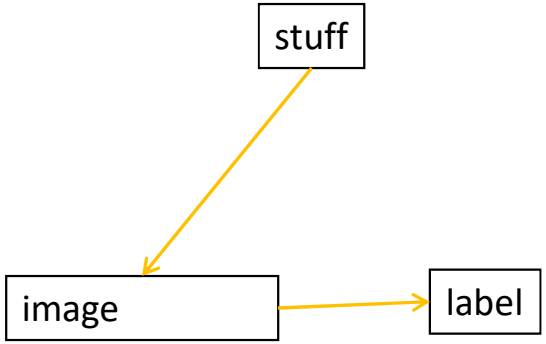
w/o pre-training

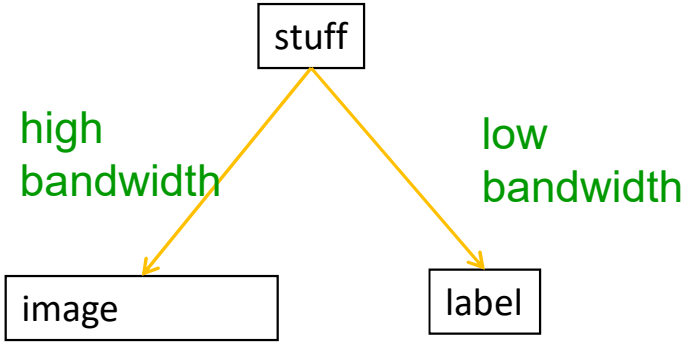


with pre-training



Why unsupervised pre-training makes sense





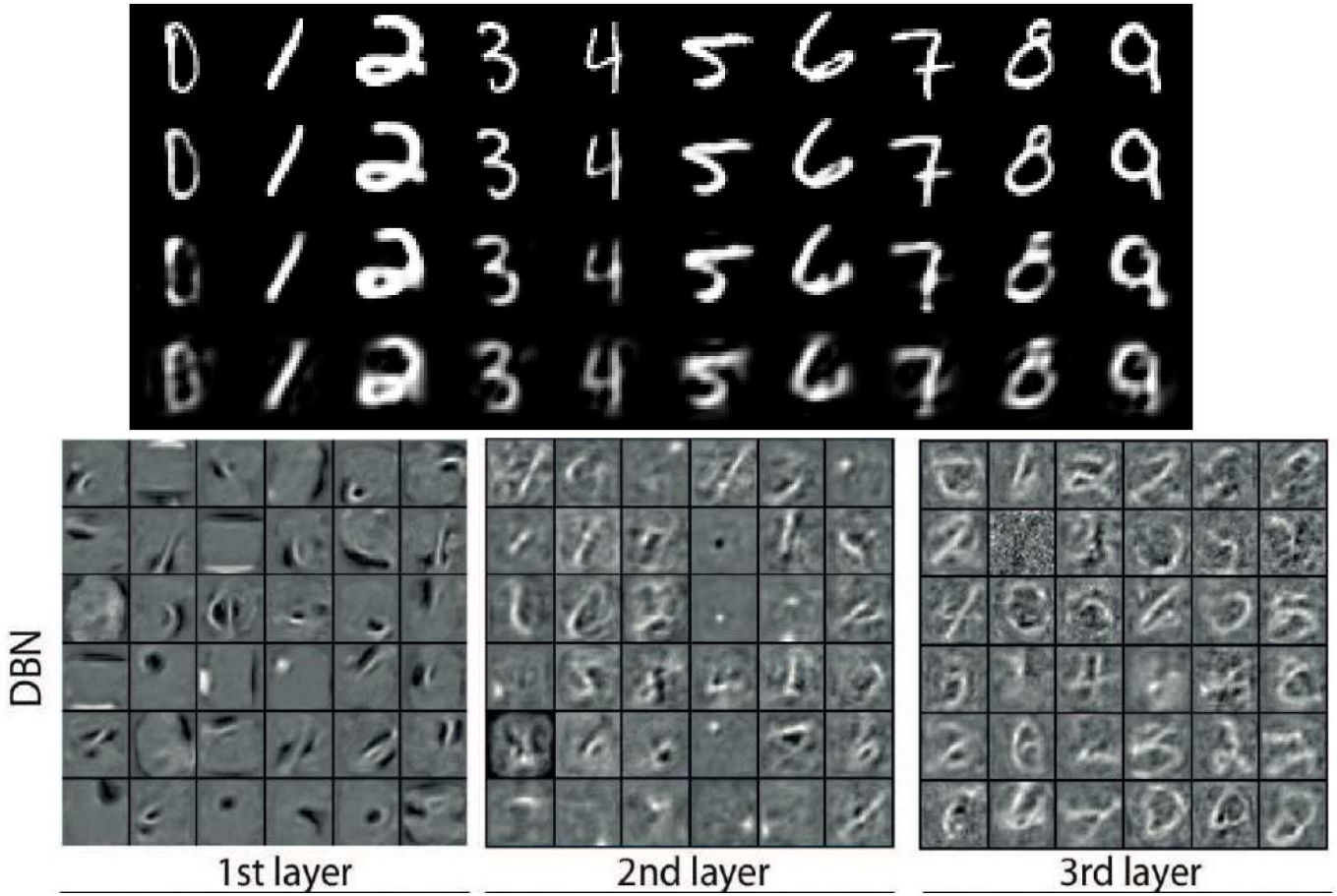
If image-label pairs were generated this way, it would make sense to try to go straight from images to labels.
For example, do the pixels have even parity?

If image-label pairs are generated this way, it makes sense to first learn to recover the stuff that caused the image by inverting the high bandwidth pathway.

Beyond layer-wise pretraining

- Layer-wise pretraining is efficient but not optimal.
- It is possible to train parameters for all layers using a wake-sleep algorithm.
  - Bottom-up in a layer-wise manner
  - Top-down and reffiting the earlier models

Representation of DBN



Finally: Topics covered in this course:

- Basic Neural Networks
- Convolutional Networks
- Recurrent Neural Networks and Long Short-Term Memory
- Optimization and Regularization tricks
- Unsupervised Deep Models (fairly sparsely)