Generative models

Florian Buettner

buettner.florian@siemens.com

Today‘s lecture: NNs as generative models

- Generative models – What and why
- Autoregressive models
- Variational autoencoders
- GANs

Unsupervised learning and generative models

- Just data, no labels
  - PCA, k-means,…
- Learn latent structure of data
- **Density estimation**
  - Training data from distribution p_data
  - Learn a distribution p_model that is similar to p_data

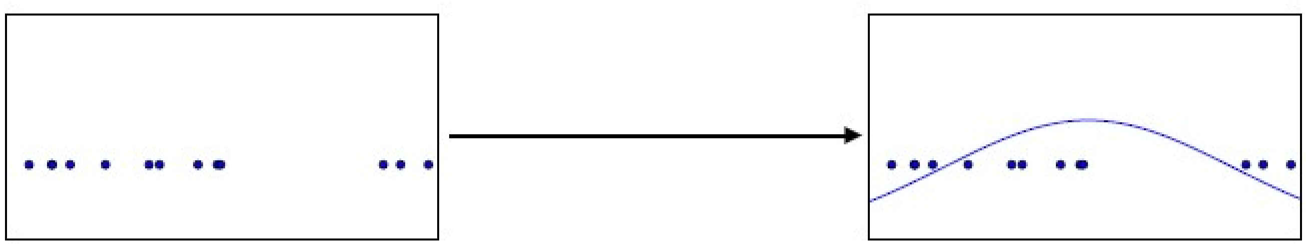


Figure: Goodfellow 2017 *arXiv:1701.00160*

Density modelling

- Simples approach: learn everything about the data
  - Define explicit model and maximise overall likelihood
- Better: focus on what is useful!
  - pixel value Vs image content, n-gram Vs semantics
    - „not all bits are created equal“
  - Curse of dimenstionality
- How can models be used for future taks?
  - Access representations?
  - Get generative model for free

But why?

- Latent variables capturing data manifold as general feature
- Anomaly detection
- Domain transfer: Art, super-resolution, colorisation
- Simulation and planning (RL)
- Creating means understanding
  - What kind of patterns has the model learnt?

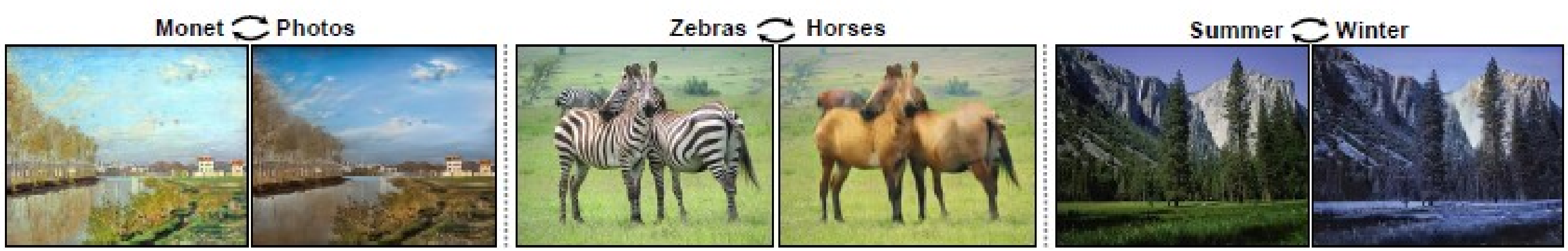


Figure: Zhu et al. *arXiv preprint*, 2017.

Types of generative models

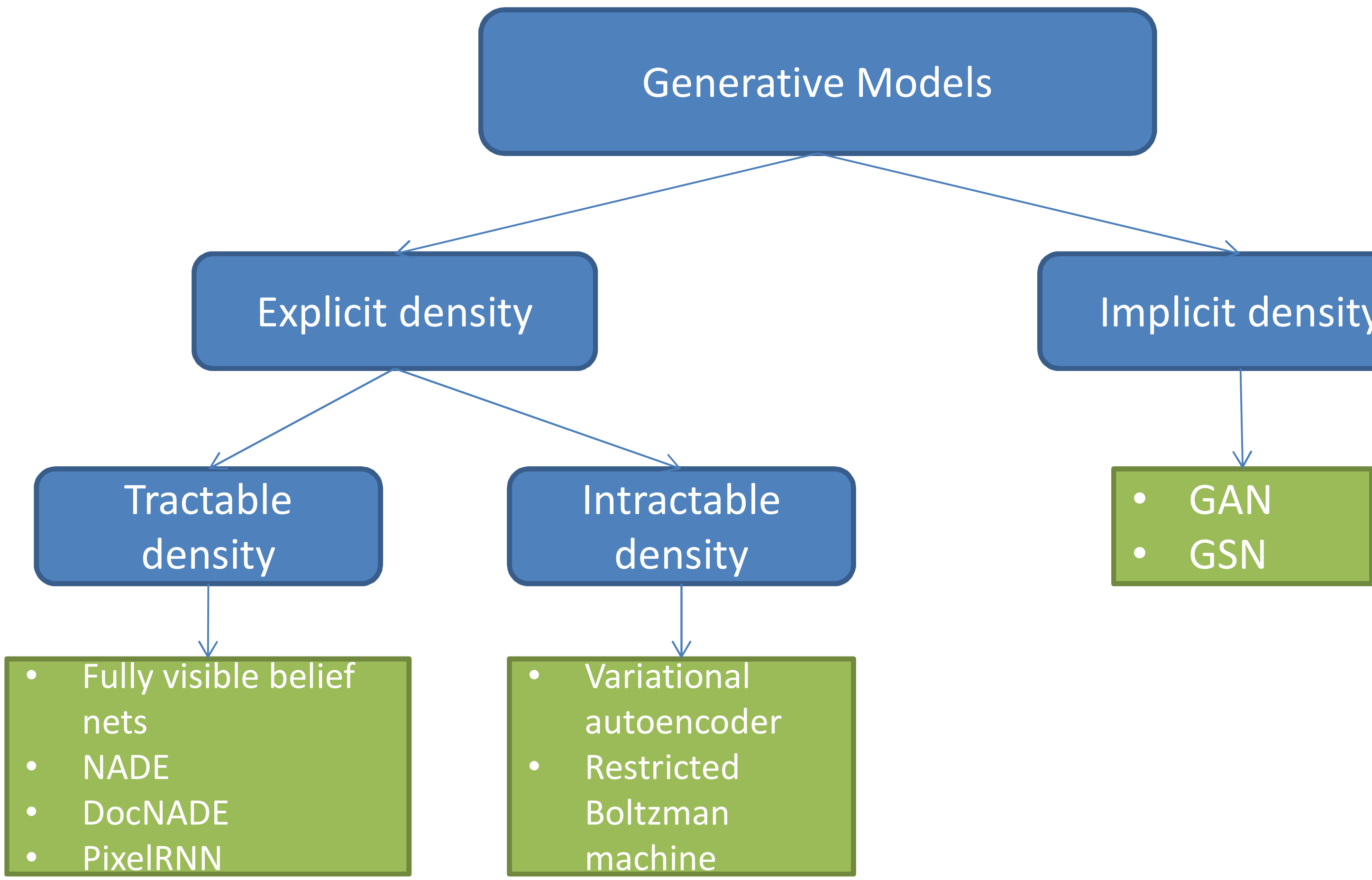


Figure adapted from Goodfellow 2017, arxiv

Types of generative models

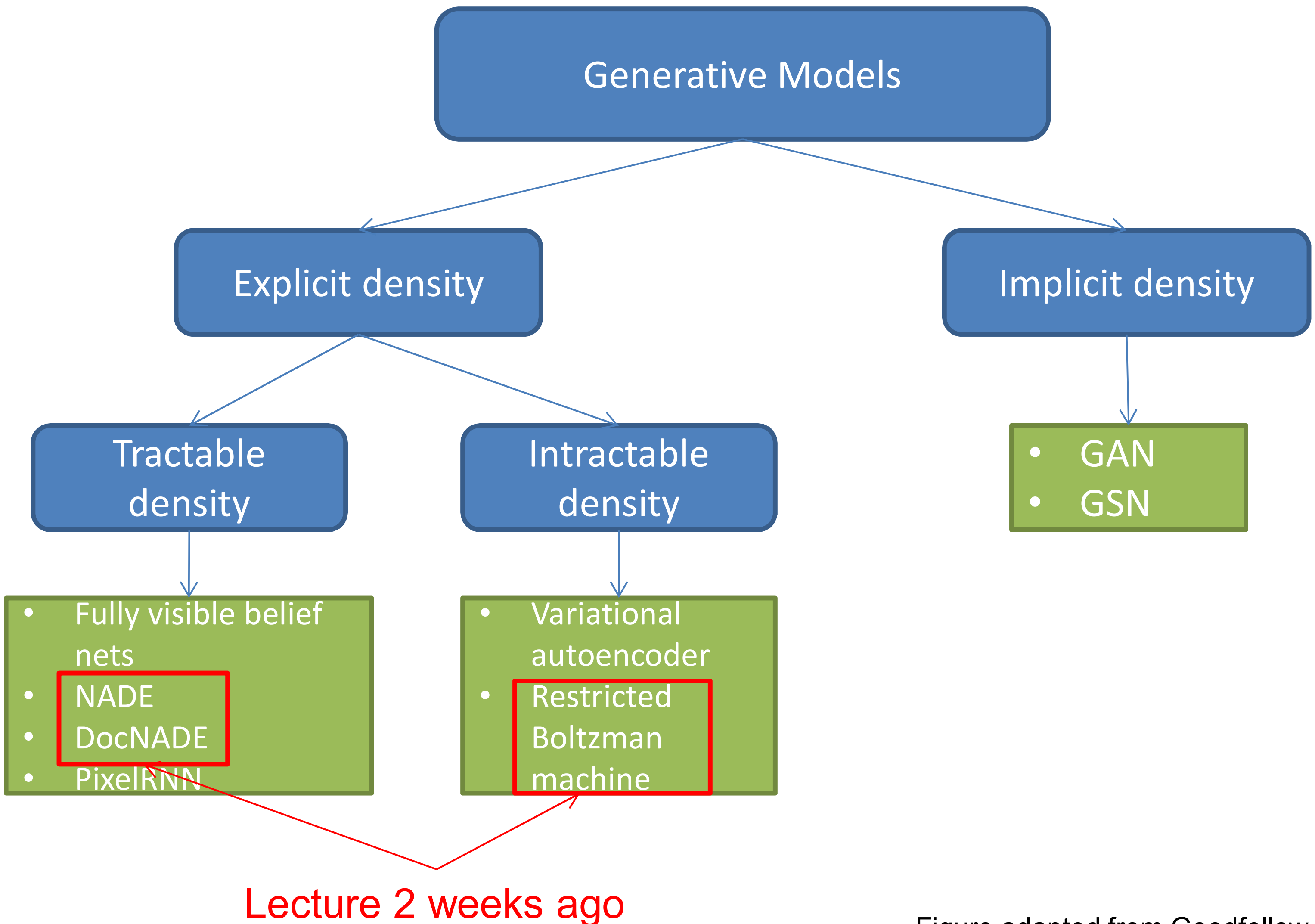
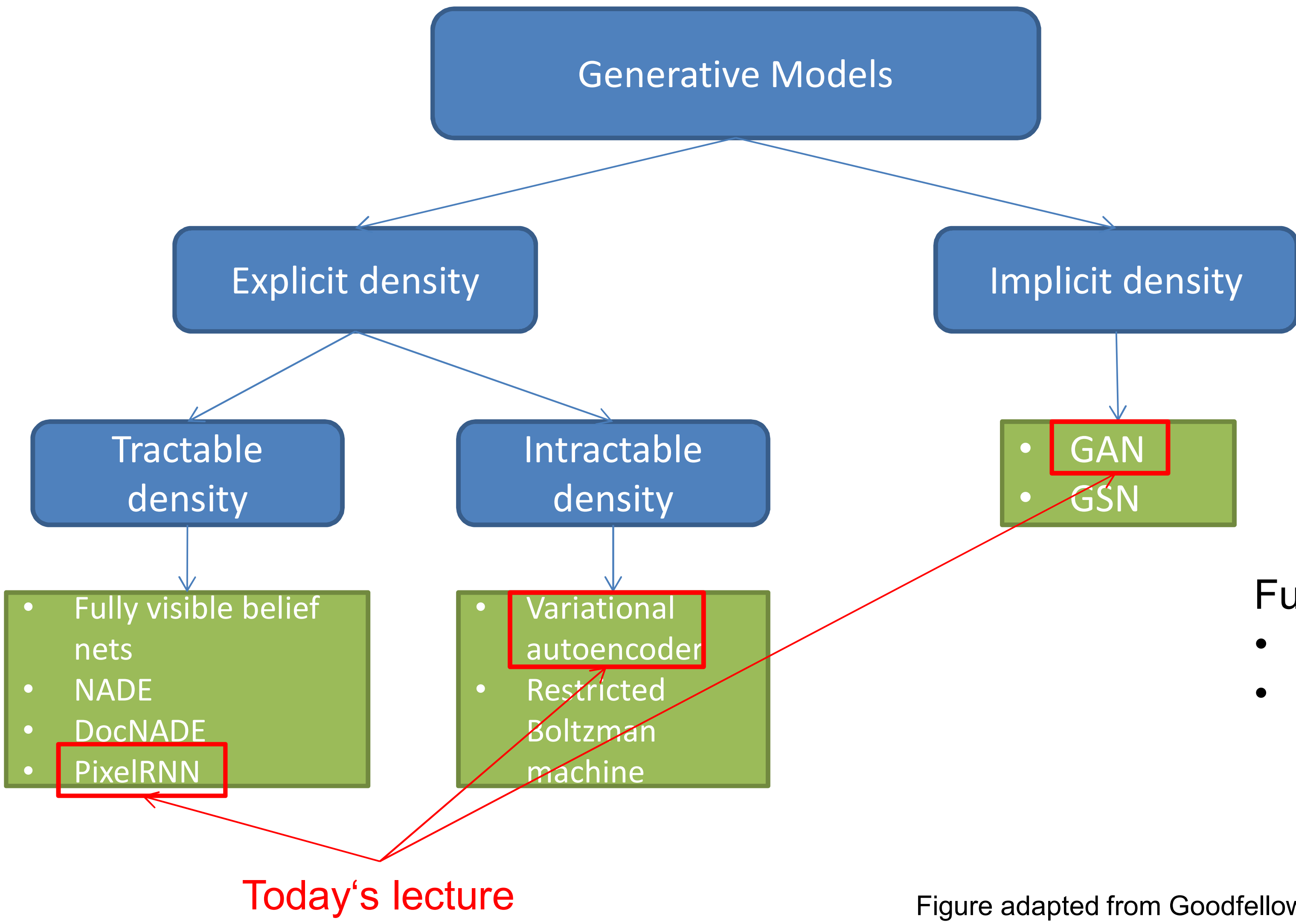


Figure adapted from Goodfellow 2017 *arXiv:1701.00160*

Types of generative models



Further reading

- GSN paper
- Belief nets

Figure adapted from Goodfellow 2017, *arXiv:1701.00160*

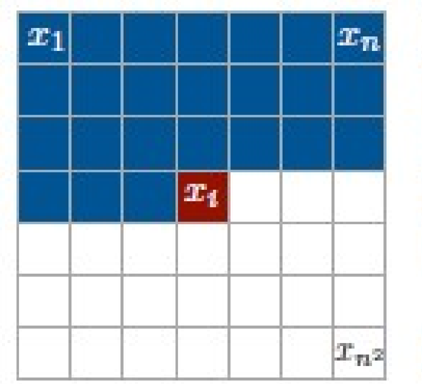
Autoregressive models

- Explicit density model
- Spilt high-dimensional input data into into sequence
  - predict small piece of system (current state previous states)
  - no more curse of dimensionality

$$p(\mathrm{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

PixelRNN algorithm

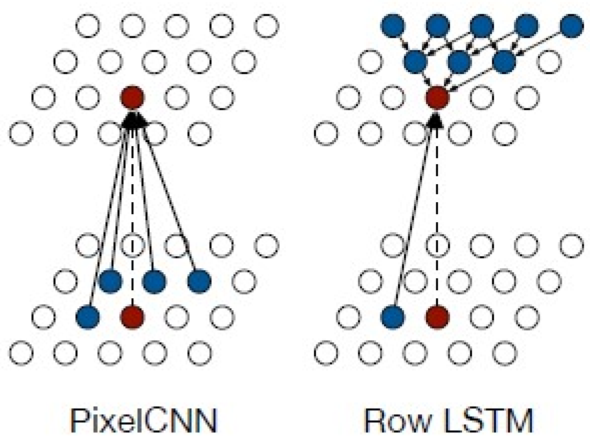
- Use neural net to model distribution over pixel values
- Optimise weights by maximising likelihood of all images
- Need to choose order!
  - Start at a corner
  - Sequentially generate pixel values
- Use LSTM to model dependency on previous pixels



van den Oord et al. *Advances in Neural Information Processing Systems*. 2016.

PixelCNN

- Same as before, but use CNN to model dependency on previous pixels
  - Masked convolutions
- Training more efficient thanks to possible parallelisations
  - Context for convs is known!
- Prediction is still slow

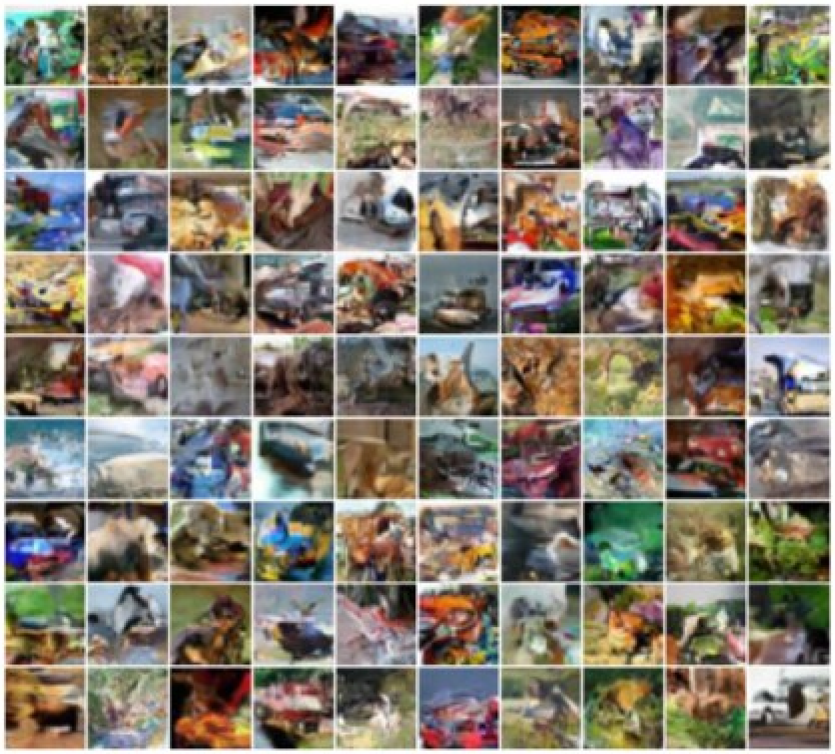


Further Reading

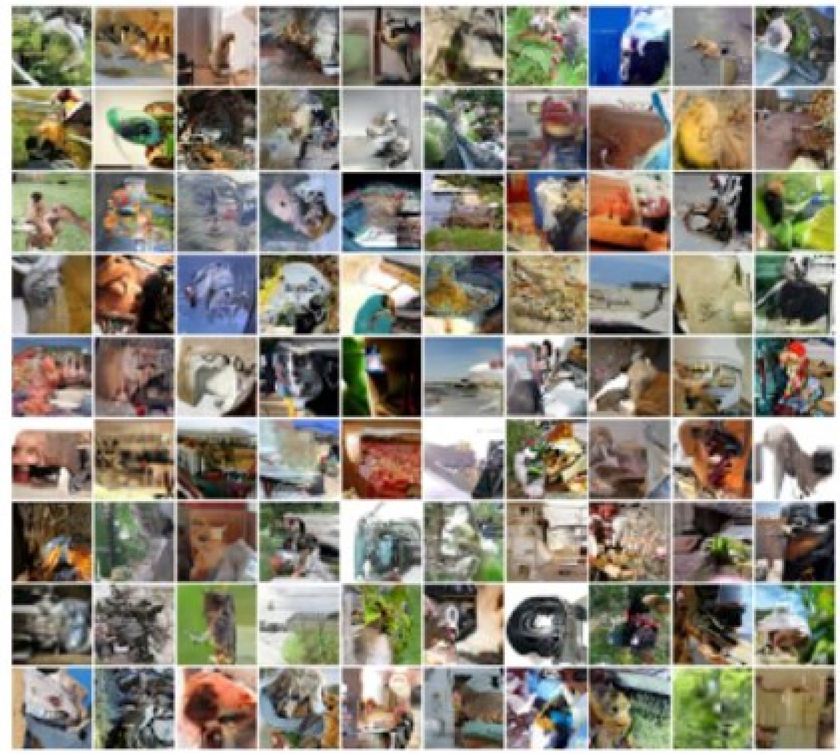
- Mulsti-scale RNN
- Conditonal Image generation
- PixelCNN++
- Gated PixelCNN

van den Oord et al. *Advances in Neural Information Processing Systems*. 2016.

Some results



32x32 CIFAR-10



32x32 ImageNet

Figure from van der Oord et al 2016, arxiv

Pros and Cons

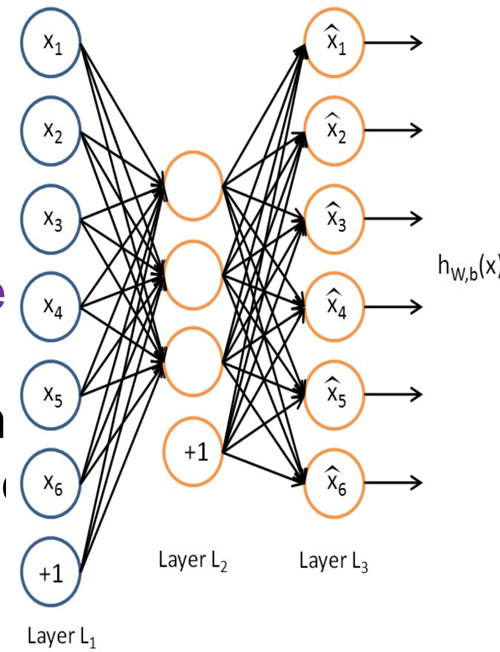
- Pros
  - Simple: just pick an order, no need to define prob distribution
  - easy to generate samples, like dreaming
- Cons
  - very expensive
    - as many predictions as pieces of data
    - parallelise during training but not testing
  - order dependance
    - where to start in an image?
    - how to deal with missing data?
  - Teacher forcing
    - difficult to generate long sequences
    - „blind representation“: not large structure of data that is actually interesting

Recap: Auto-encoders (AEs)

→ a *feed-forward neural network* trained to ***reproduce its input*** at the *output* layer

Key Facts about AEs:

→ unsupervised ML algorithm, similar to PCA
→ neural network's target output is its input
→ learn latent features/encoding of input (no manual fe engineering)
→ represent both **linear** and **non-linear** transformation
→layered to form deep learning network, i.e., distribute representations
→ tractable / easier optimization
→ *applications* in *denoising* and *dimensionality* reduction (*dense representation*)
→ powerful non-linear (i.e., non-linear encoding and decoding) generalization of PCA



http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Slide from last week's lecture

Variational autoencoders

- Probabilistic version of autoencoder
  - Place prior on latent space
  - Sample from prior and use decoder to generate new samples
  - AE as generative model

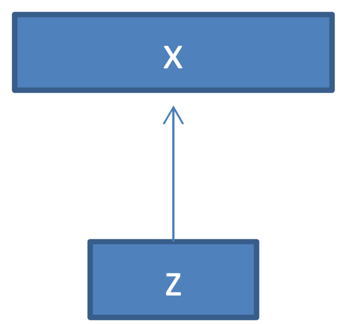
prior

$$p(z) = N(0, I)$$

$$p_\theta(x|z) = N(\mu, \sigma^2)$$

$$\mu = f_\theta(z) = \text{multilayer neural net}$$

likelihood



Inference

- With flexible neural net f_θ(z), the data distribution p_θ(x) can be almost arbitrarily complicated / multi-modal distribution
- But intractable posterior distribution p(z|x)
- Need approximate inference for learning

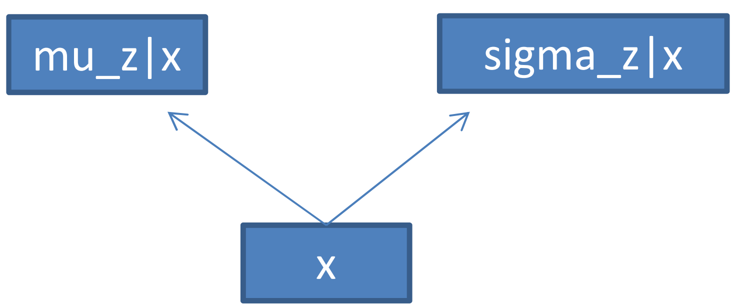
posterior $$p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$$

Data likelihood $$p_\theta(x) = \boxed{\int} p_\theta(z)p_\theta(x|z)dz$$

Variational inference with neural networks

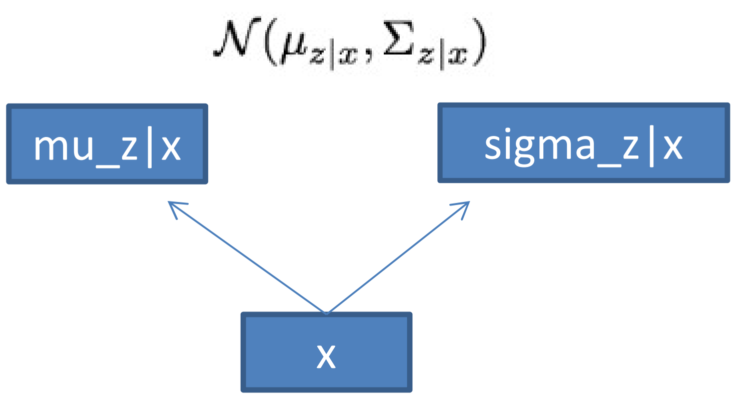
- Posterior p(z|x) is not tractable
- Introduce parametric model q_ϕ(z|x) of true posterior
    - ϕ: variational parameters
    - parameterised by neural networks

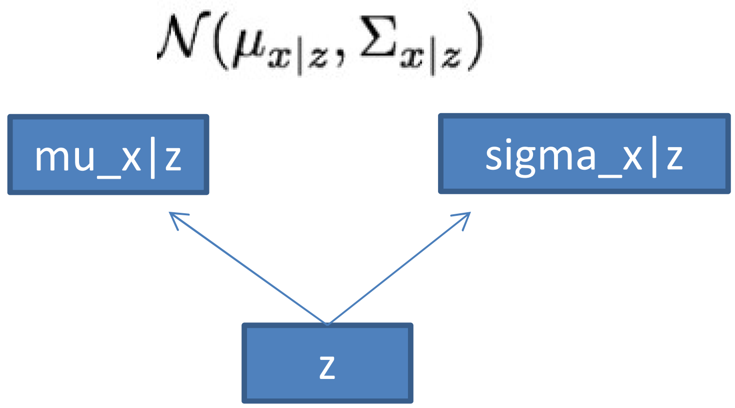
$$q_\varphi(z|x) = N(\mu,\sigma^2)$$
$$[\mu, \sigma^2] = f^{(z|x)}(x,\varphi) = \text{multilayer neural net}$$



Encoder and decoder

- 2 NNs, encoder network q_phi(z|x) and decoder network p_theta(x|z)





Enoder network q_Phi

Decoder network p_theta

Recap: Variational inference

- Approximate posterior p with q-distribution
- Minimize KL divergence between q and p
  - Equivalent: maximise Evidence Lower Bound (ELBO) of data D

$$\mathrm{D}_{KL}(Q_\omega(\boldsymbol{\theta})\,||P(\boldsymbol{\theta}|D)) = \int d\boldsymbol{\theta}.\,Q_\omega(\boldsymbol{\theta})\,\log\frac{Q_\omega(\boldsymbol{\theta})}{P(\boldsymbol{\theta}|D)}$$

$$\nabla_\omega \mathrm{ELBO}(D) = \nabla_\omega \int Q_\omega(\boldsymbol{\theta})\,\log P(D|\boldsymbol{\theta}) - Q_\omega(\boldsymbol{\theta})\log\frac{Q_\omega(\boldsymbol{\theta})}{P(\boldsymbol{\theta})}\,d\boldsymbol{\theta}$$

$$= \nabla_\omega \mathbb{E}_{Q_\omega(\boldsymbol{\theta})}\left[\log P(d|\boldsymbol{\theta}) - \log\frac{Q_\omega(\boldsymbol{\theta})}{P(\boldsymbol{\theta})}\right]$$

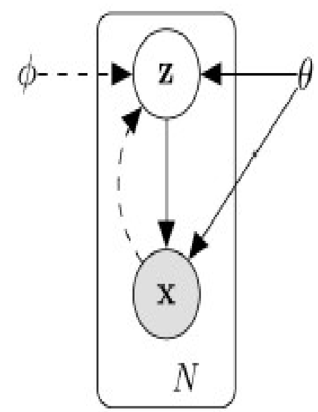
If we could move the derivative into the Expectation, we could approximate it by sampling!

Variational inference with neural networks

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound



- Jointly optimize w.r.t. ϕ and θ
- Simple SGD:
  - Sampling small minibatches of data
  - Sampling from approx. posterior
  - Use reparametrisation trick to approximate gradient of ELBO

Recap: reparameterisation trick

$$\nabla_\omega \mathbb{E}_{p(z|\omega)}[f(z)] = \nabla_\omega \int p(z|\omega) f(z)\, dz$$

Find a way to reparametrize $p(z|\omega)$ such that $z \sim g(\varepsilon, \omega)$ and we just sample ε

$$\nabla_\omega \int p(z|\omega) f(z)\, dz = \nabla_\omega \int p(\varepsilon) f(g(\varepsilon, \omega))\, d\varepsilon$$

We move the derivative inside the integral:

warranted by the dominated convergence theorem

$$= \int p(\varepsilon) \nabla_\omega f(g(\varepsilon, \omega))\, d\varepsilon = \mathbb{E}_{p(\varepsilon)}[\nabla_\omega f(g(\varepsilon, \omega))]$$

Example:

$$z \sim \mathcal{N}(\mu, \sigma) \;\rightarrow\; z \sim \mu + \sigma\varepsilon; \varepsilon \sim \mathcal{N}(0,1)$$

☑ **ε is now independent of μ, σ and backpropagation works!**

Slide from lecture 7

Stochastic Gradient VB

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \int q_{\boldsymbol{\phi}}(\mathbf{z}) \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z})\right] d\mathbf{z}$$

$$\simeq \frac{1}{L} \sum_{l=1}^{L} \left(\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}^{(l)}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}^{(l)})\right)$$

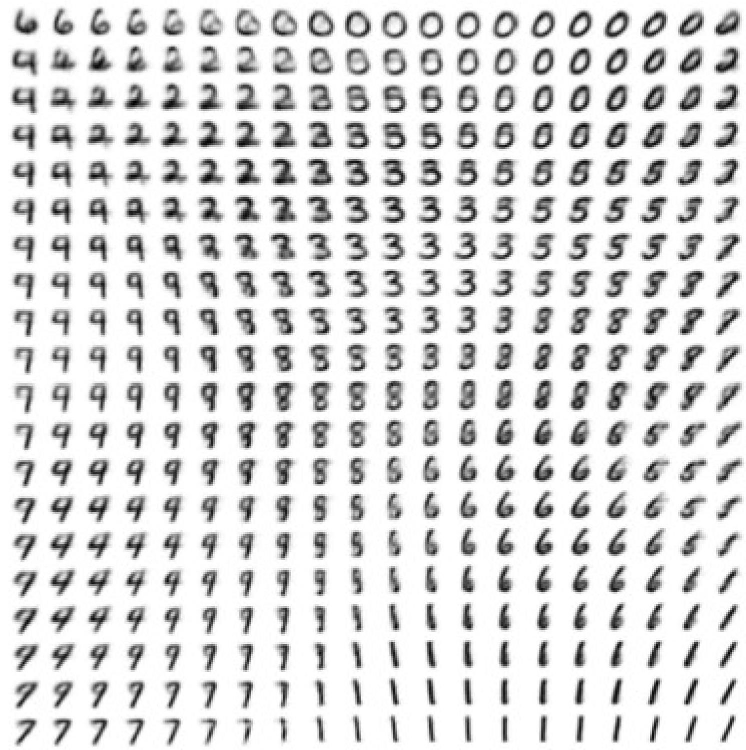
$$\begin{aligned}
\text{where} \quad & \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon}) \quad \text{(samples from noise variable)} \\
& \mathbf{z}^{(l)} = g(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{\phi}) \\
\text{(such that} \quad & \mathbf{z}^{(l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}))
\end{aligned}$$

Stochastic VB in practice

- Draw mini-batch
- Sample from p(eps)
- Compute gradients using backprop
- Update theta and phi

Some results

(a) Learned Frey Face manifold

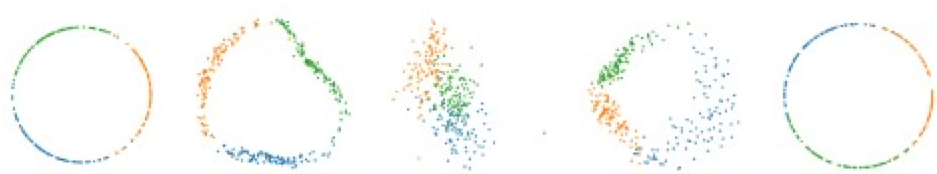


(b) Learned MNIST manifold

- Diagonal prior leads to independant z_i s
- Components are interpretable
- Representation is accesible (via q(z|x))

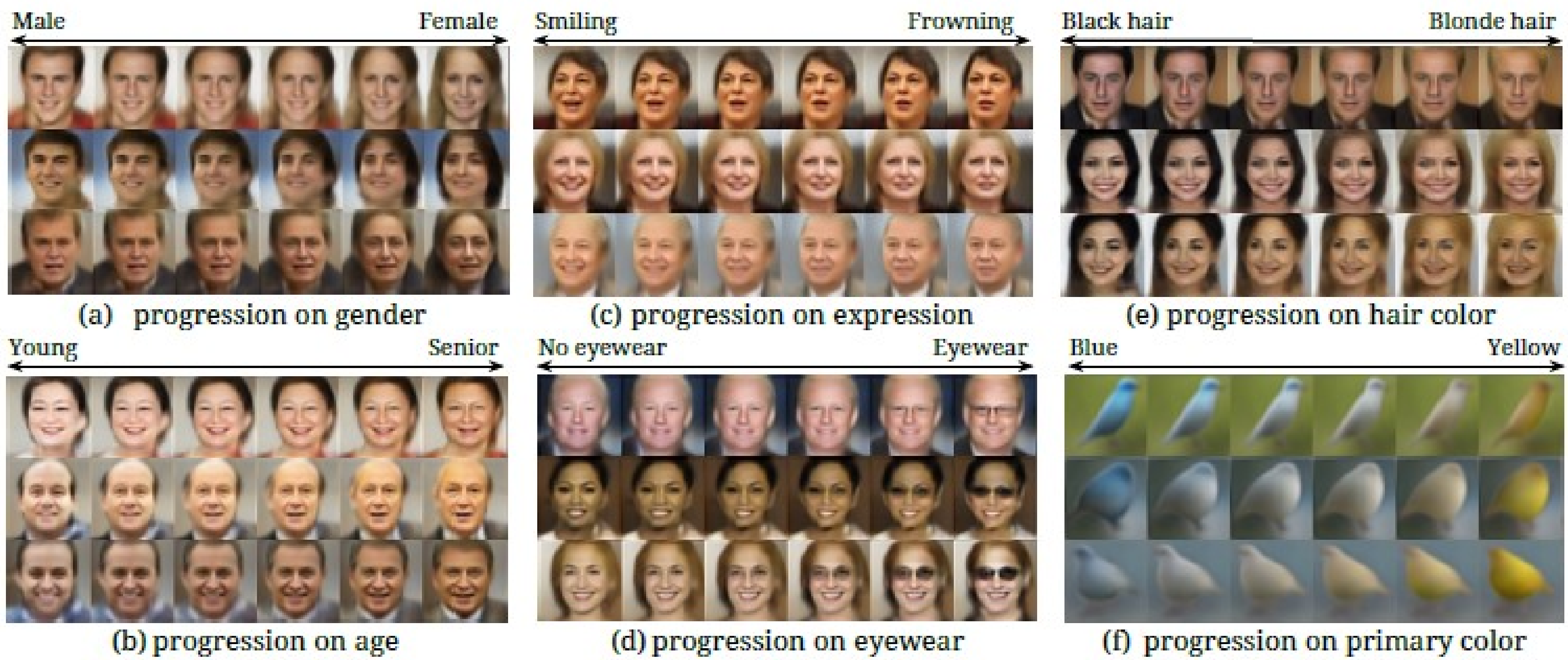
Some improvements

- Improve encoder/decoder
  - Use convolutions
  - Conditional VAE
    - Replace all P(X|z) with P(X|z,Y)
    - Replace all Q(z|X) with Q(z|X,Y)
  - Hierarchical VAE
- Improve prior
  - Problem: mis-match prior and aggregate posterior
    - Bad samples for high density in prior and low density in posterior (hasn't seen samples!)
    - Bad reconstruction error



(a) Original (b) Autoencoder (c) $\mathcal{N}$-VAE (d) $\mathcal{N}$-VAE, $\beta = 0.1$ (e) $\mathcal{S}$-VAE

Results

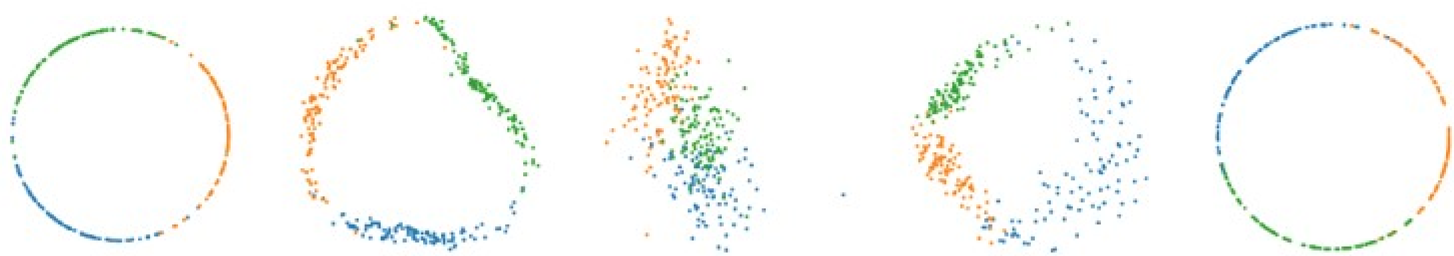


Yan et al., *European Conference on Computer Vision*. Springer, Cham, 2016.

More improvements

- Beta-VAE
  - Encourage disentangled factors
    - introduce an adjustable hyperparameter that balances independence constraints with reconstruction accuracy

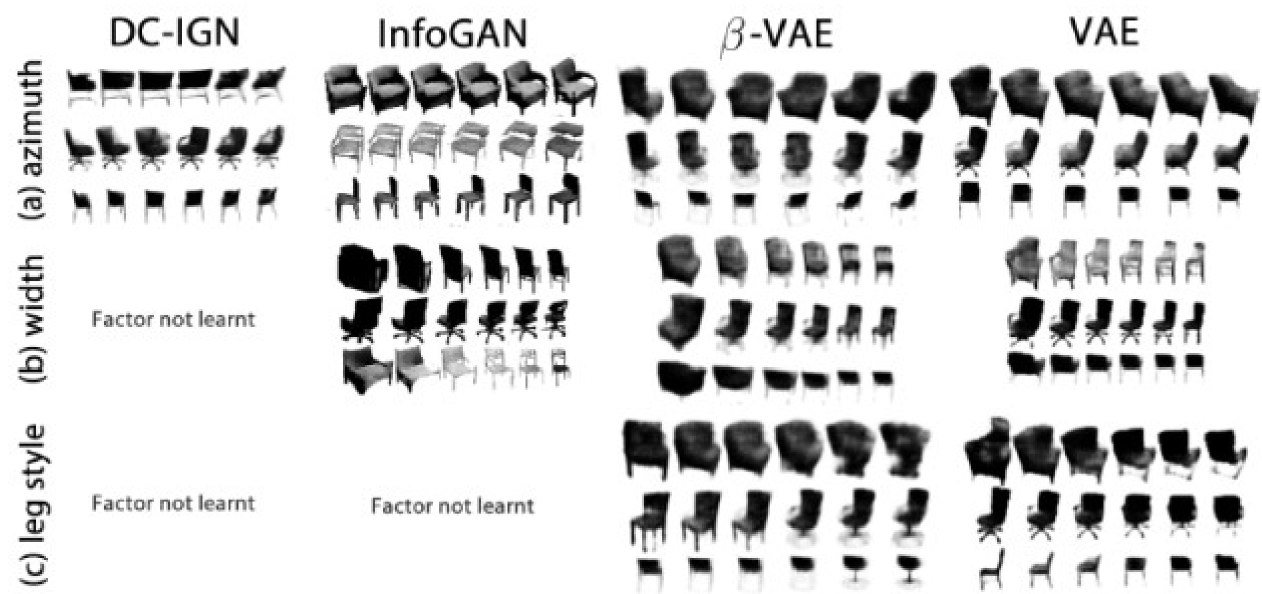
$$\mathcal{F}(\theta, \phi, \beta; \mathbf{x}, \mathbf{z}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \, D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}))$$



(a) Original (b) Autoencoder (c) $\mathcal{N}$-VAE (d) $\mathcal{N}$-VAE, $\beta = 0.1$ (e) $\mathcal{S}$-VAE

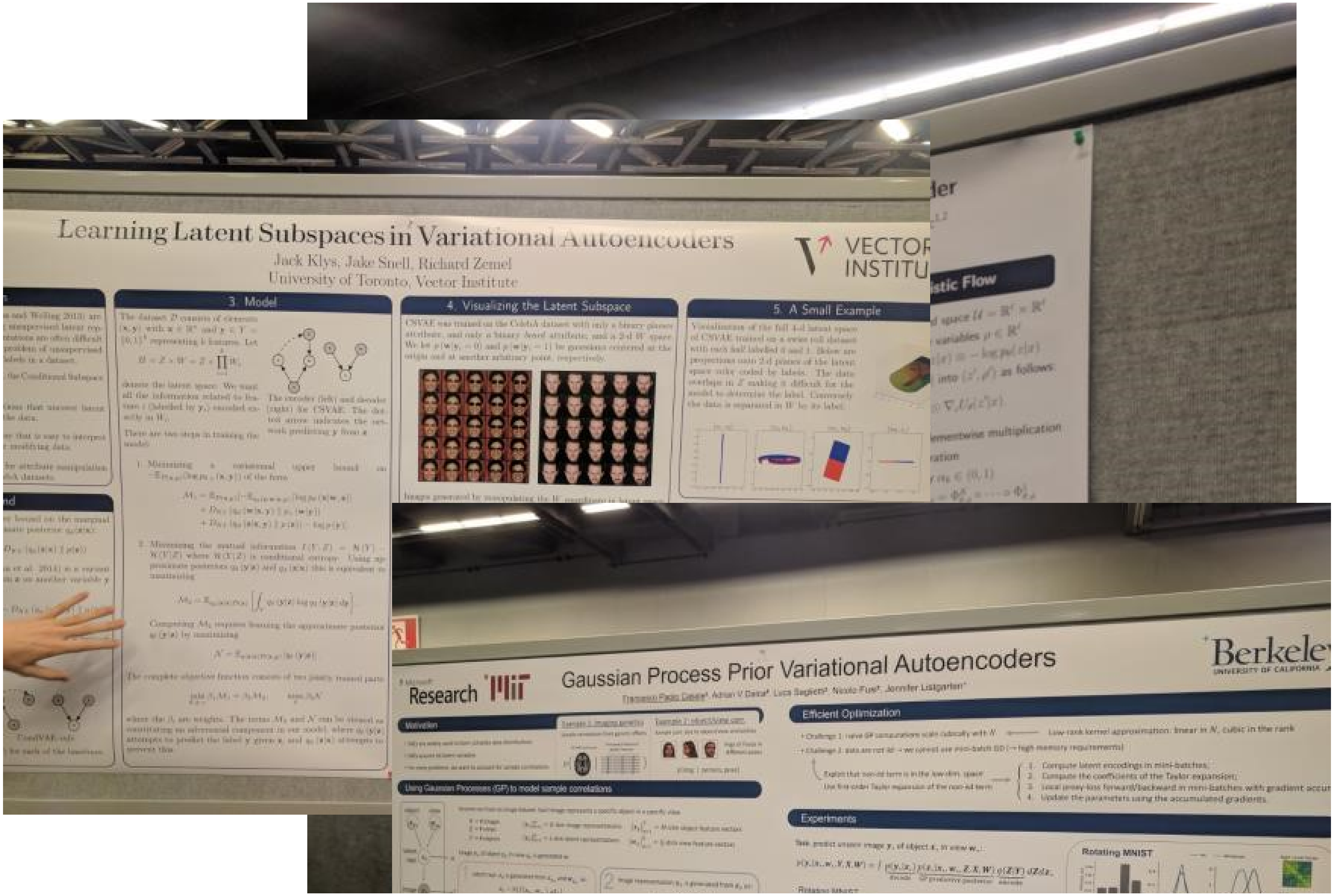
Higgins et al., ICLR 2017; Davidson et al, UAI, 2018

Beta-VAE and S-VAE



Higgins et al., ICLR 2017

Some current research directions



Pros and Cons of VAEs

- Pros
  - Accesible representation
  - Robust and straight-forward to train
- Cons
  - Generated images blurrier than SOTA

GANs

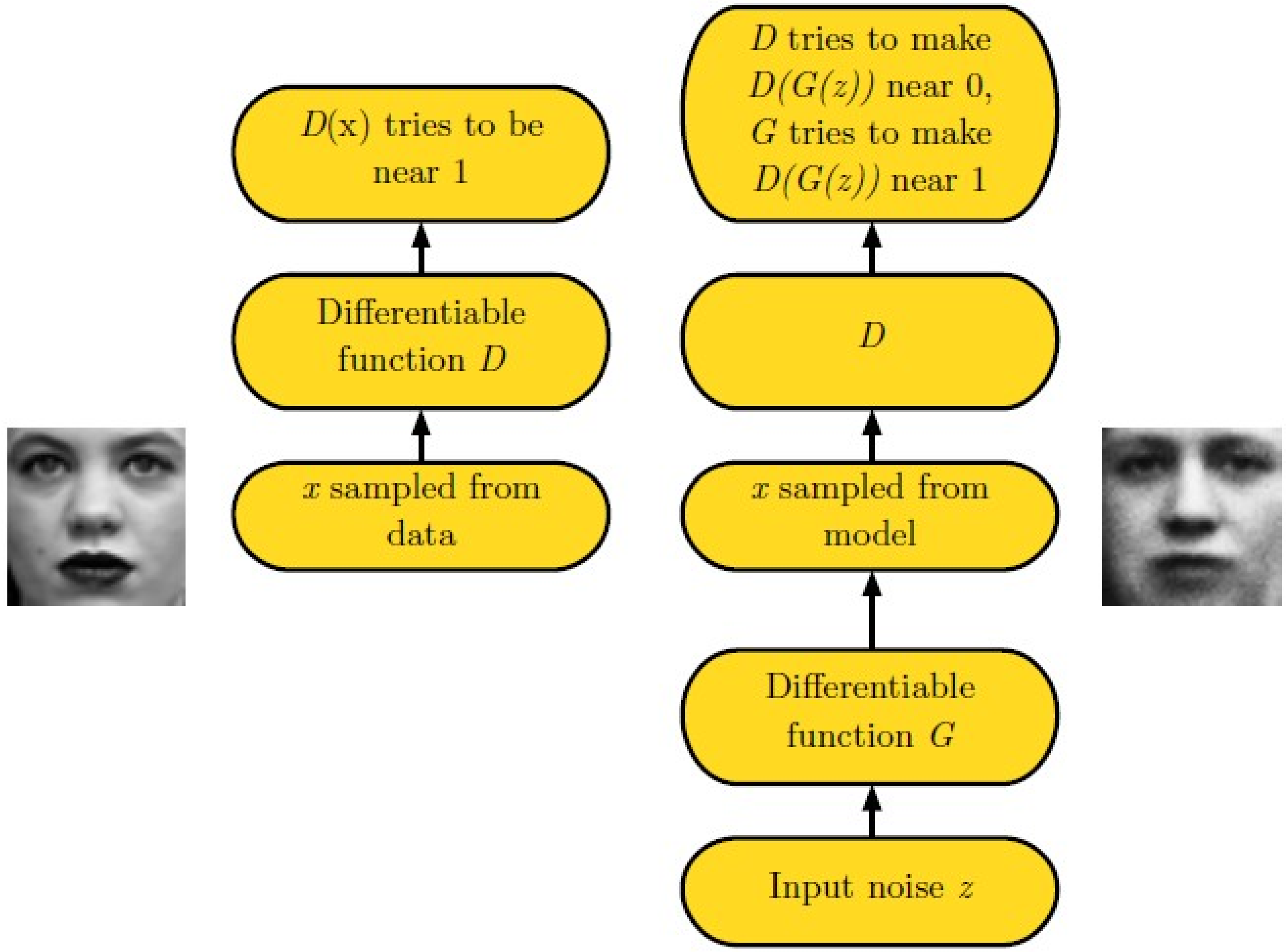
- We would like to sample from a complex, high-dimensional distribution
  - Problem: No direct way for this
  - Solution: Sample random noise and pass it through NN

A two-player game

- Set up a game between two players
- **Generator G**: generate samples that are intended to come from the same distribution as the training data
- **Discriminator D**: determines whether a determine whether a sampel is real or fake
  - use traditional supervised learning techniques
- Generator is trained to fool discriminator

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Illustration of the game



Ian Goodfellow arxiv 2016

Learning the parameters of G and D

- Cost functions of generator and discriminator (J_D and J_G) depend on both sets of parameters (theta_D and theta_G)
  - But: Each network has only access to it‘s own parameters
  - Not optimisation, but game!
- Solution to game
  - Nash equilibrium: Tuple (theta_G, theta_D) that is a local optimum of J_D wrt theta_D and a local optimum of J_G wrt theta_G

Cost function of D

- Minimize cross-entropy
  - Train on 2 mini-batches
    - one coming from the dataset, where the label is 1 for all examples
    - one coming from the generator, where the label is 0 for all examples
- Co-operative view
  - Discriminator more like a teacher instructing the generator in how to improve

Zero-sum game

- Sum of all players' costs is always zero
  - J_G = - J_D
  - Also referred to as minmax
  - minimization in outer loop and maximization in inner loop

$$\boldsymbol{\theta}^{(G)*} = \underset{\boldsymbol{\theta}^{(G)}}{\arg\min} \max_{\boldsymbol{\theta}^{(D)}} V\left(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}\right)$$

Value function

$$V\left(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}\right) = -J^{(D)}\left(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}\right)$$

Heuristic non-saturating game

- Cost from zero-sum game does not perform well in practice
  - D minimizes a cross-entropy but G maximises same cross-entropy
  - When D rejects sampels with high confidence, gradient of G vanishes
- Solution: flip target of cross-entropy for cost for G
  - Maximise log-prob of D being mistaken

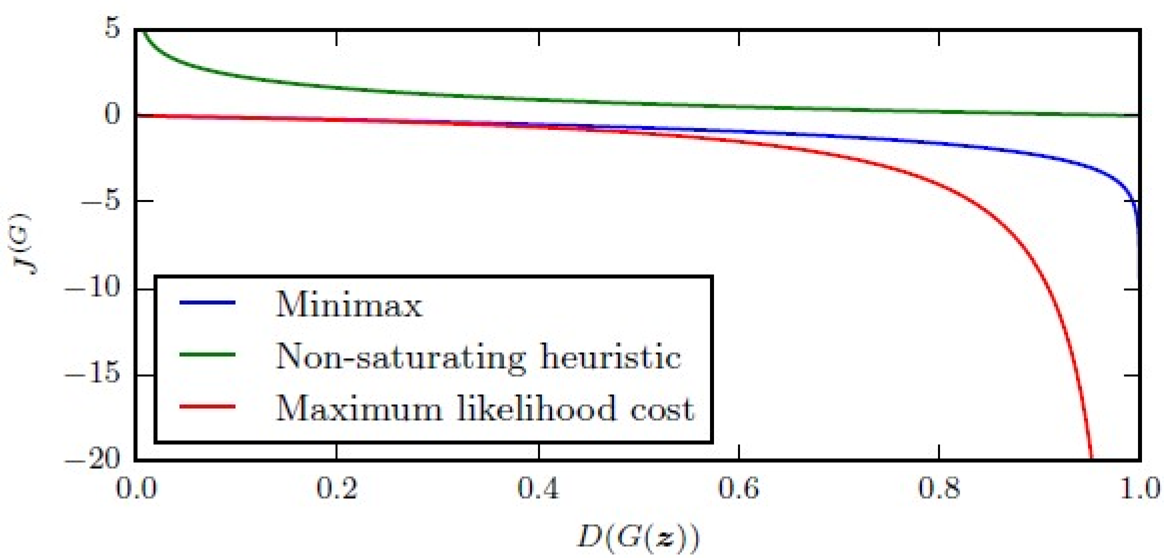
$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \log D(G(z))$$

  - Each player has strong gradient when he loses the game

Maximum likelihood game

- Minimizing the KL divergence between the data and the model
  - Equivalent (if D is optimal)

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \exp\left(\sigma^{-1}\left(D(G(z))\right)\right)$$



Putting it together

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

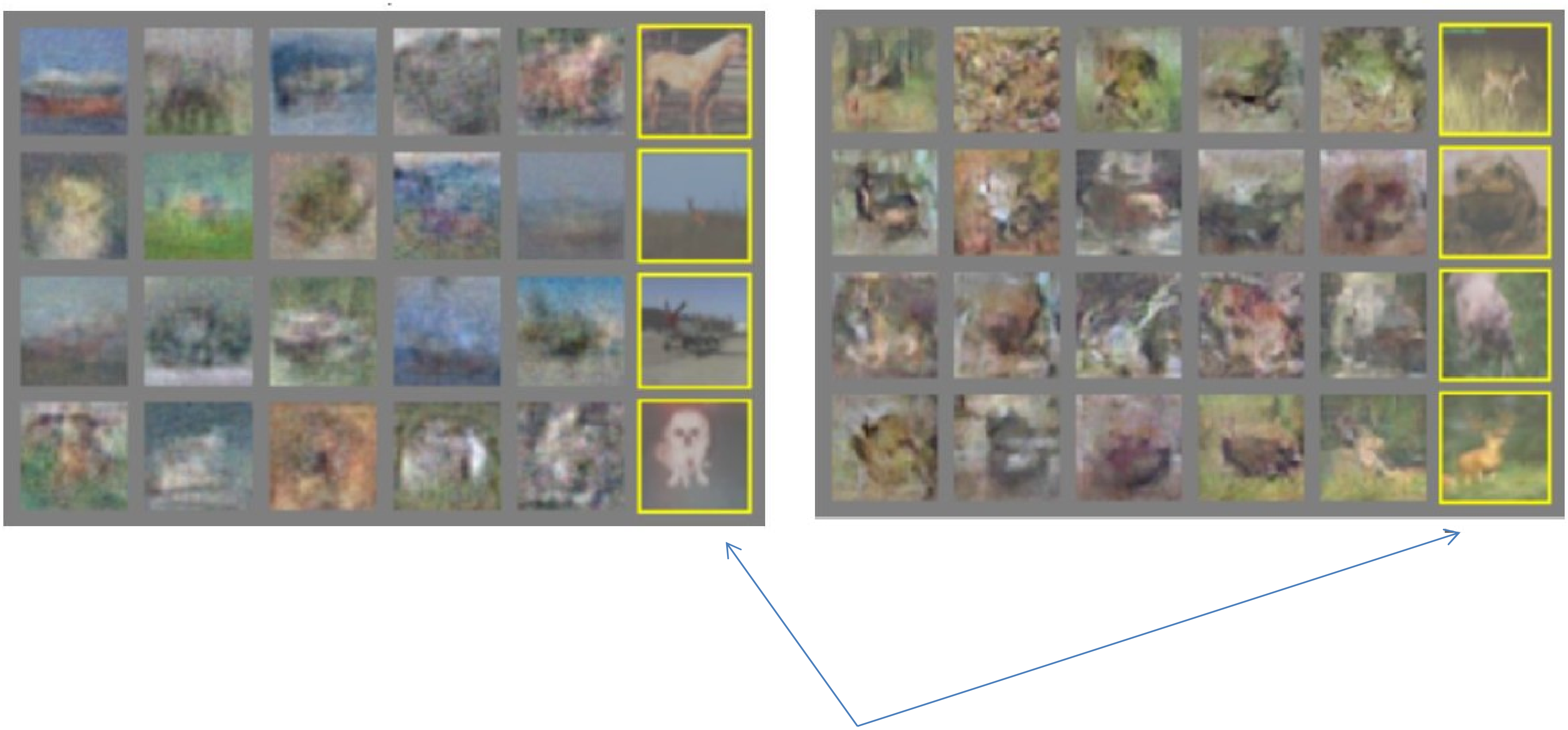
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Goodfellow 2014

Samples from generator

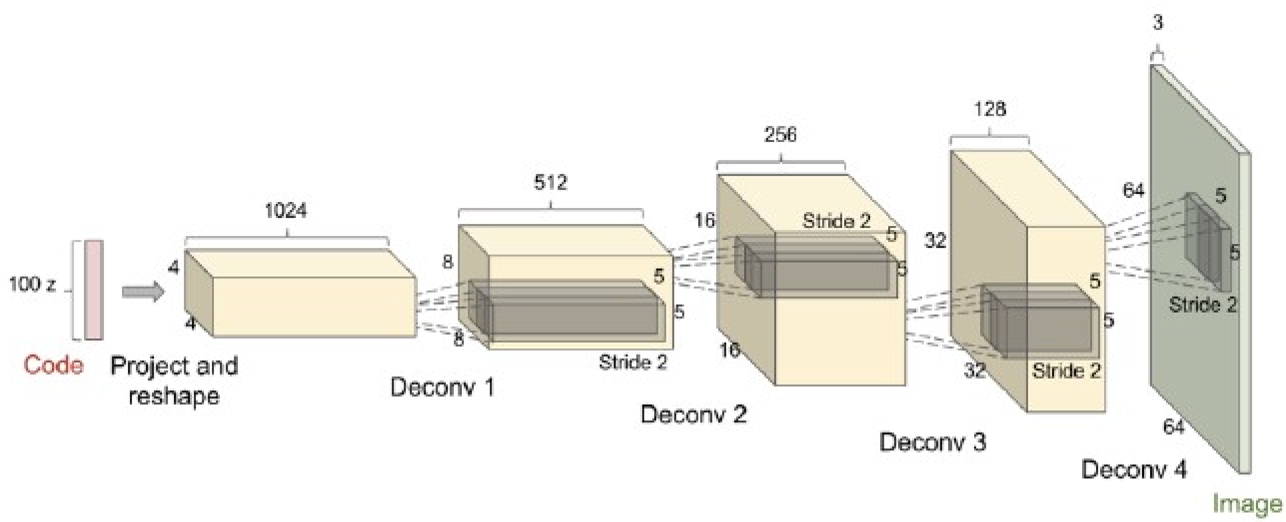


Nearest neighbour from training set

DGAN

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.

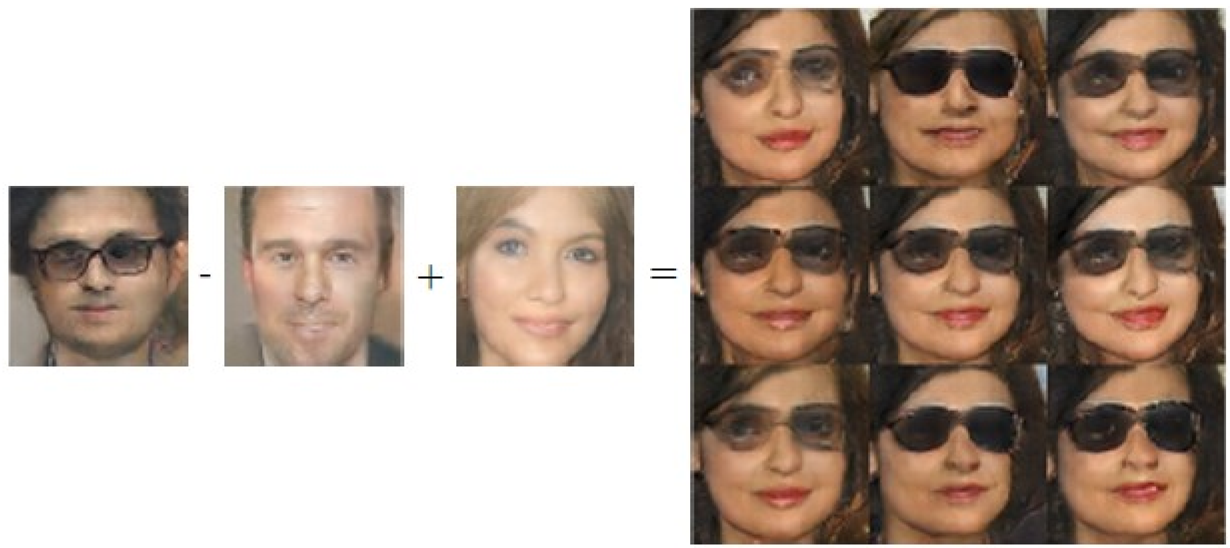


Goodfellow 2016, Radford 2015

Sample again

Goodfellow 2016, Radford 2015

GAN maths

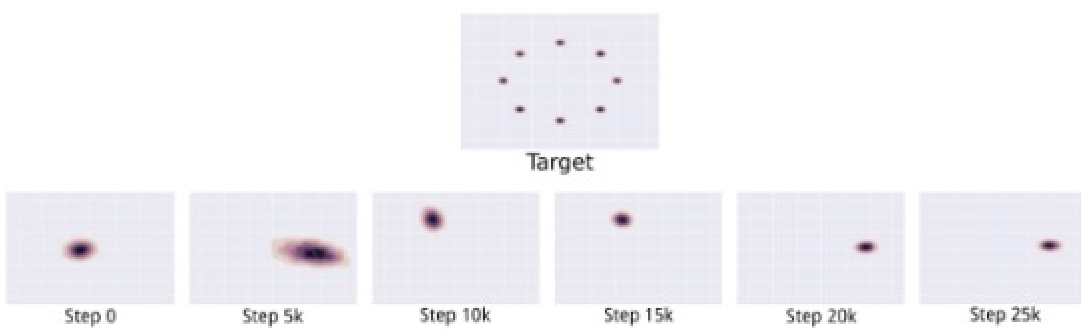


Goodfellow 2016, Radford 2015

Mode collapse

- Most severe problem in terms of non-convergence
- Issue: maximin solution to the GAN game is different from the minimax solution
  - Generator asked to map every z value to the single x coordinate that discriminator believes is most likely to be real
  - Simultaneous gradient descent doesn‘t favour one over the other

$$\boldsymbol{\theta}^{(G)*} = \arg\min_{\boldsymbol{\theta}^{(G)}} \max_{\boldsymbol{\theta}^{(D)}} V\left(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}\right)$$



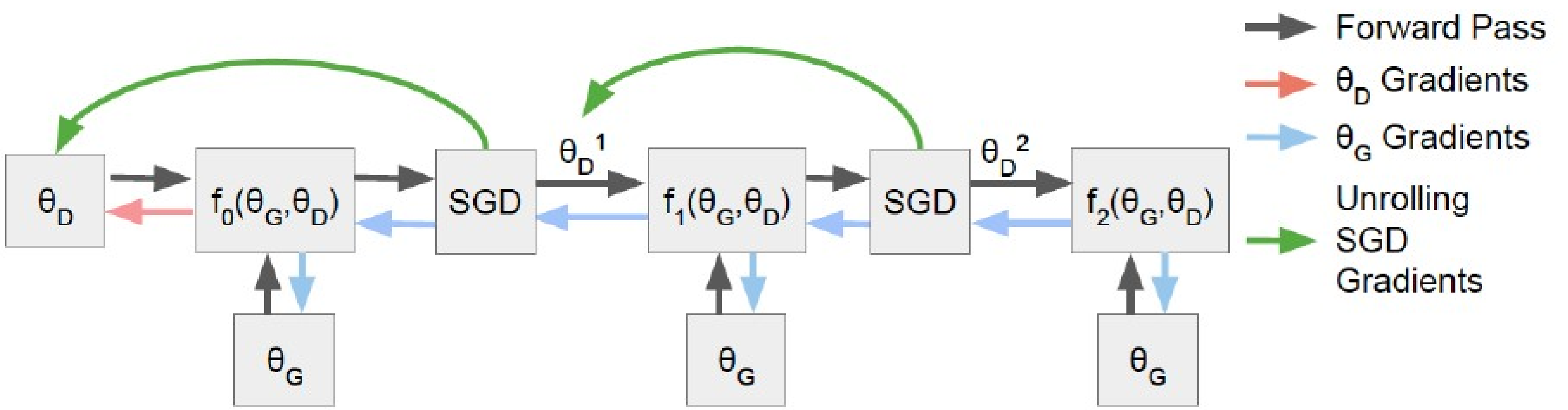
Metz et al., 2016

Solutions

- Minibatch features
  - Compare one example to batches of real/fake examples
  - D can detect if sample is unusually similar to other samples
- Unrolled GAN
  - back-propagate through the maximization operation

Unrolled GAN

- Consider several updates of the generator when updating the discriminator and vice versa
  - k steps in the discriminator
  - backpropagate all steps when computing the gradient on the generator



Metz et al., 2016

Better generators: LSGAN,

- Least squares GAN
  - X. Mao, Q. Li, H. Xie, R. Lau, Z. Wang, “Least squares generative adversarial networks” 2016
  - Still use a classifier but replace cross-entropy loss with Euclidean loss

Discriminator

GAN $$\min_D E_{x\sim p_X}[-\log D(x)] + E_{z\sim p_Z}[-\log(1-D(G(z)))]$$

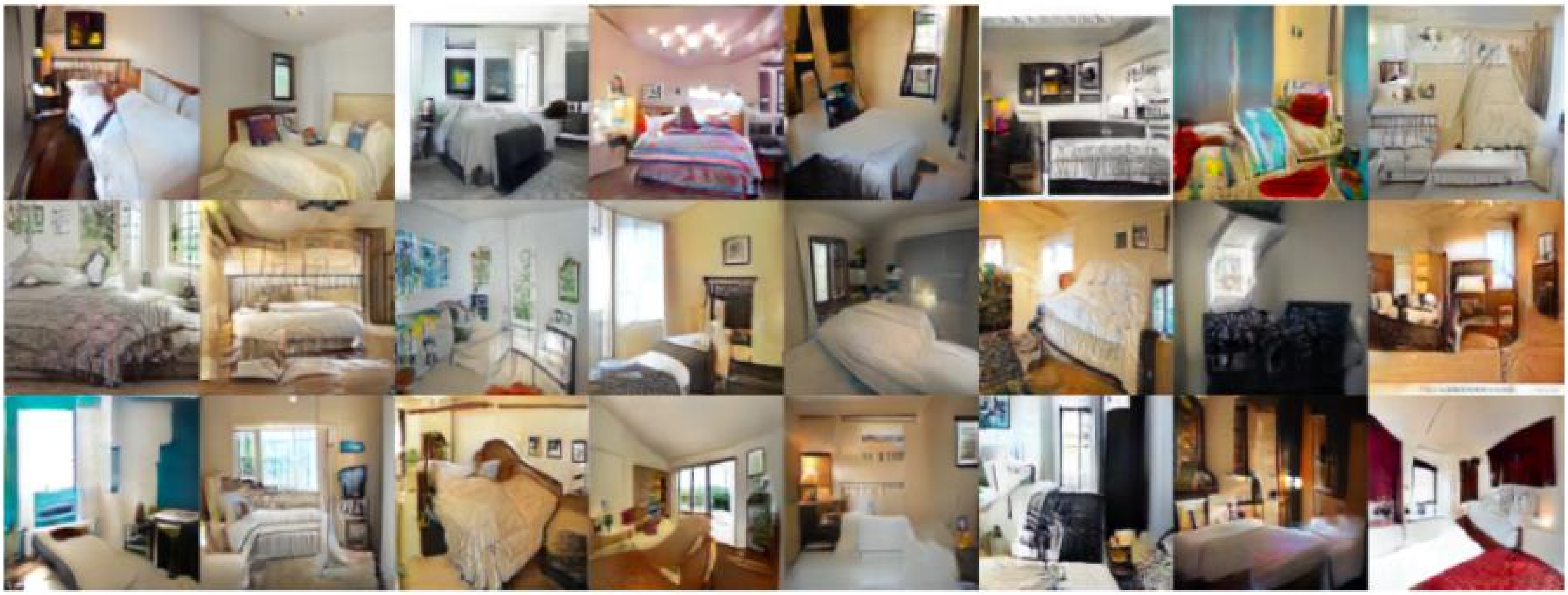
LSGAN $$\min_D E_{x\sim p_X}[(D(x)-1)^2] + E_{z\sim p_Z}[D(G(z))^2]$$

Generator

GAN $$\min_G E_{z\sim p_Z}[-\log D(G(z))]$$

LSGAN $$\min_G E_{z\sim p_Z}[(D(G(z))-1)^2]$$

LSGAN



Wasserstein GAN

- M. Arjovsky, S. Chintala, L. Bottou "Wasserstein GAN" 2016
- Use critic instead of discriminator
  - Discriminator can output real number (same as before w/o sigmoid)

Discriminator

GAN $\max_D E_{x \sim p_X}[\log D(x)] + E_{z \sim p_Z}[\log(1 - D(G(z))]$

WGAN $\max_D E_{x \sim p_X}[D(x)] - E_{z \sim p_Z}[D(G(z))]$

Generator

GAN $\max_G E_{z \sim p_Z}[\log D(G(z))]$

WGAN $\max_G E_{z \sim p_Z}[D(G(z))]$

64

WGAN

WGAN-GP

- I. Gulrajani, F. Ahmed, M. Arjovsky, V. Domoulin, A. Courville "Improved Training of Wasserstein GANs" 2017

$$\min_G \max_D E_{x\sim p_X}[D(x)] - E_{z\sim p_Z}[D(G(Z))] + \lambda E_{y\sim p_Y}[(||\nabla_y D(y)||_2 - 1)^2]$$

$$y = ux + (1-u)G(z)$$

- *y:* imaginary samples

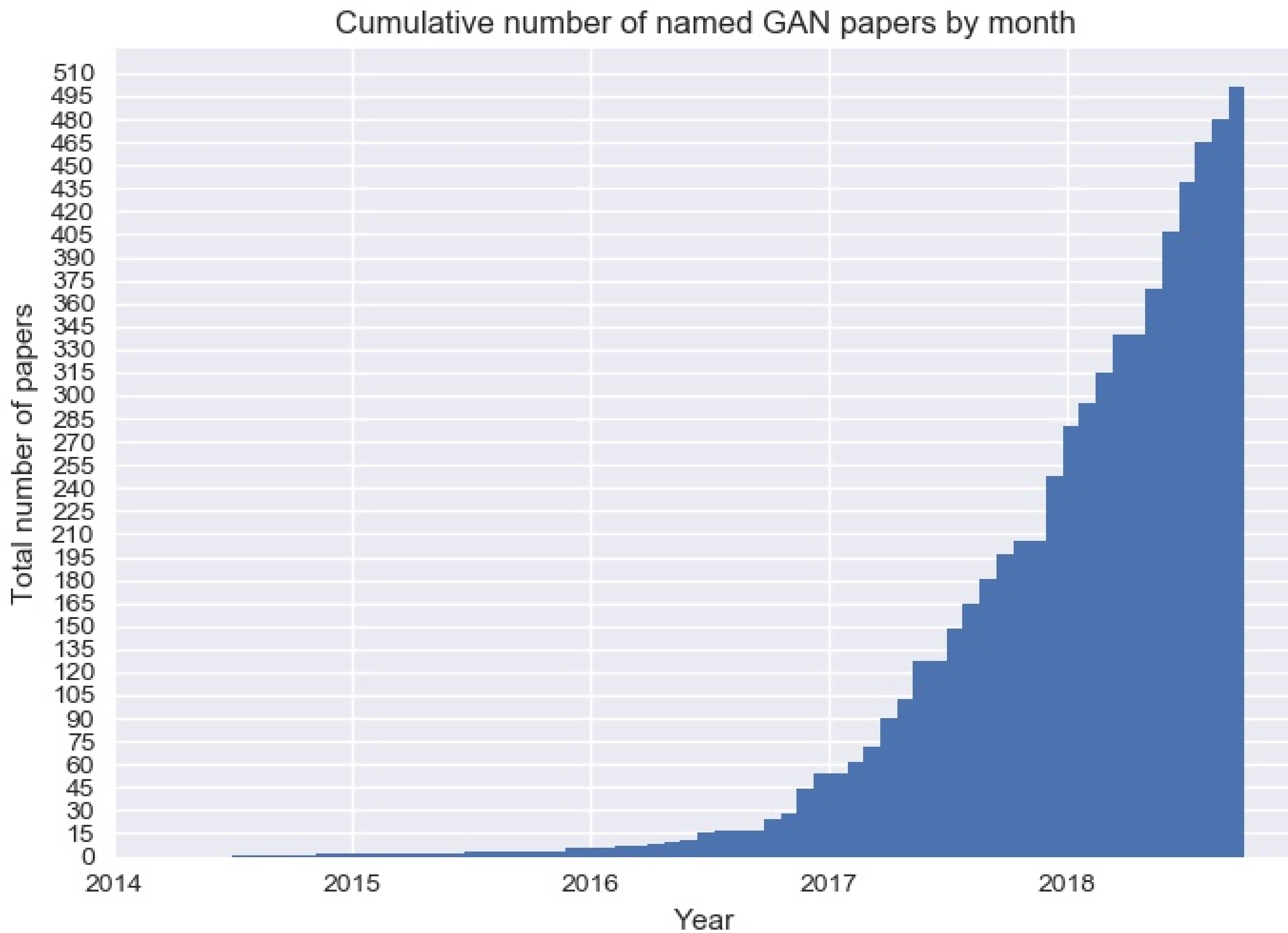
Optimal critic has unit gradient norm almost everywhere

BEGAN and DRAGAN

- DRAGAN: Add gradient norm to standard GAN and evaluate around the data manifold
- BEGAN: use autoencoder as discriminator and optimize lower bound of the Wasserstein distance between auto-encoder loss distributions on real and fake data.

DRAGAN	$\mathcal{L}_{\text{D}}^{\text{DRAGAN}} = \mathcal{L}_{\text{D}}^{\text{GAN}} + \lambda \mathbb{E}_{\hat{x} \sim p_d + \mathcal{N}(0,c)}[(\|\|\nabla D(\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{DRAGAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\text{D}}^{\text{BEGAN}} = \mathbb{E}_{x \sim p_d}[\|\|x - \text{AE}(x)\|\|_1] - k_t \mathbb{E}_{\hat{x} \sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\text{G}}^{\text{BEGAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$

The GAN zoo



https://github.com/hindupuravinash/the-gan-zoo

So....which one is best?

Are GANs Created Equal? A Large-Scale Study

Mario Lucic*   Karol Kurach*   Marcin Michalski   Olivier Bousquet   Sylvain Gelly
Google Brain

than others. We conduct a neutral, multi-faceted large-scale empirical study on state-of-the art models and evaluation measures. We find that most models can reach similar scores with enough hyperparameter optimization and random restarts. This suggests that improvements can arise from a higher computational budget and tuning more than fundamental algorithmic changes. To overcome some limitations

NeurIPS 2018

Evaluating GANs

- Challenging to define appropriate metric
  - Maximum likelihood and other classical metrics not applicable
  - Subjective comparisons (visual quality) may be misleading
    - Inception score: discriminator has low entropy, while producing samples from all classes when passed through a classifier

$$\exp(\mathbb{E}_{x\sim G}[d_{KL}(p(y \mid x), p(y)])$$

    - Fréchet Inception Distance: difference in embedding of true and fake data (assuming MVN in embedded space)
      - Strong negative correlation between visual quality and FID

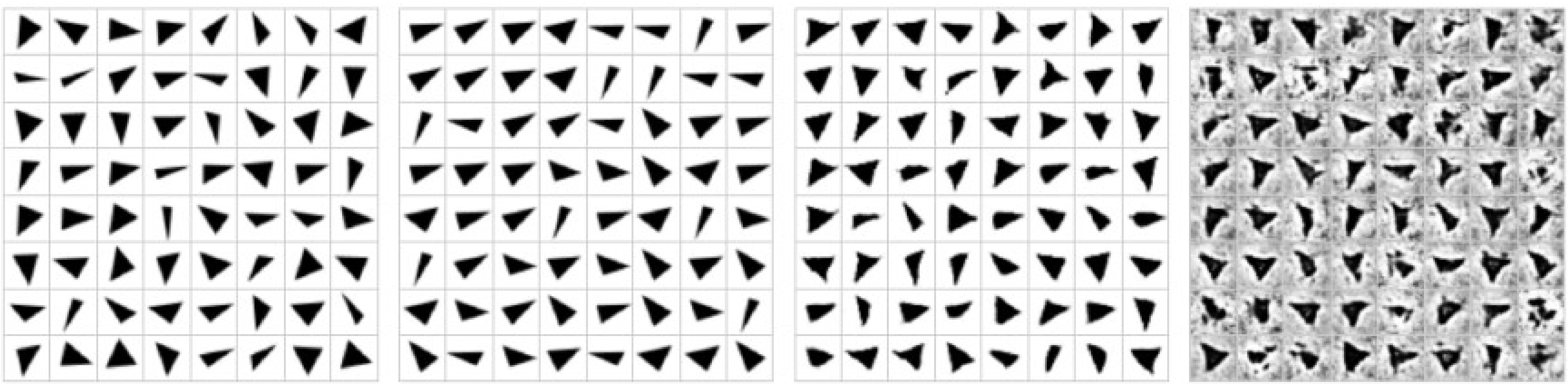
$$\mathtt{FID}(x,g) = ||\mu_x - \mu_g||_2^2 + \mathrm{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}}),$$

      - Precision/recall

Metrics

- Precision measures fraction of relevant retrieved instances among the retrieved instances
- Recall measures fraction of retrieved instances among relevant instances
- F1 score is harmonic average of precision and recall.
- IS captures precision: no penalization for not producing all modes of the data distribution
  - Only for not producing all classes
- FID captures both precision and recall

Precision-Recall for GANs



(a) High precision, high recall (b) High precision, low recall (c) Low precision, high recall (d) Low precision, low recall

Fair assessment

- Compare state-of-the-art approaches

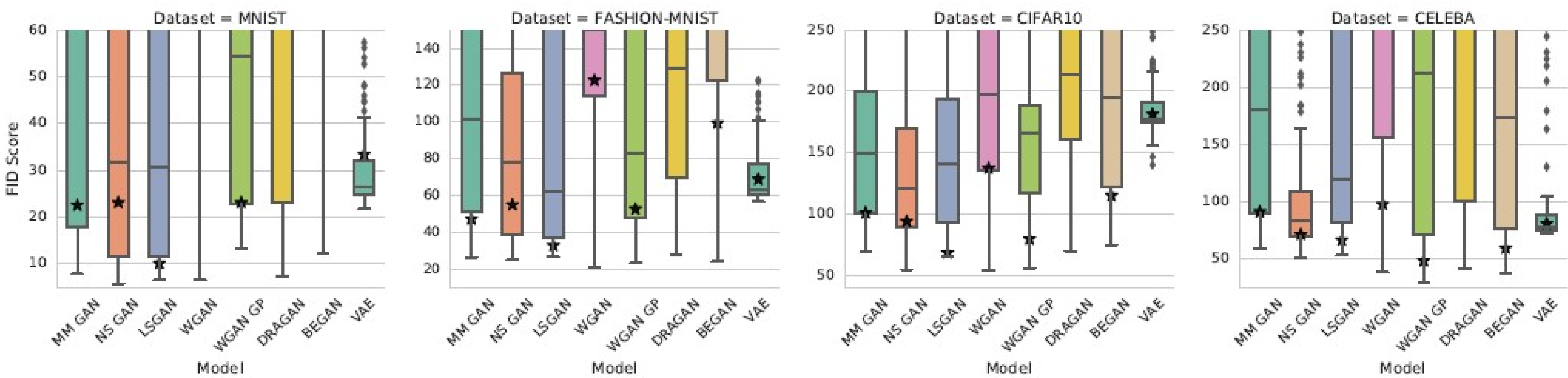
GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x \sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x \sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x} \sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x \sim p_d}[D(x)] + \mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda \mathbb{E}_{\hat{x} \sim p_g}[(\|\|\nabla D(\alpha x + (1 - \alpha \hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x \sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x} \sim p_g}[(D(\hat{x} - 1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda \mathbb{E}_{\hat{x} \sim p_d + \mathcal{N}(0,c)}[(\|\|\nabla D(\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x \sim p_d}[\|\|x - \mathrm{AE}(x)\|\|_1] - k_t \mathbb{E}_{\hat{x} \sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$

Fair comparisons

- Use same architecture
- Optimize hyperparameters on each dataset OR on one dataset only (infer for new datasets)
- Computational budget
  - Dependence on number of optimised hyperparameters

Are GANs created equal?

- Asterisk
  - Default hyperparameters

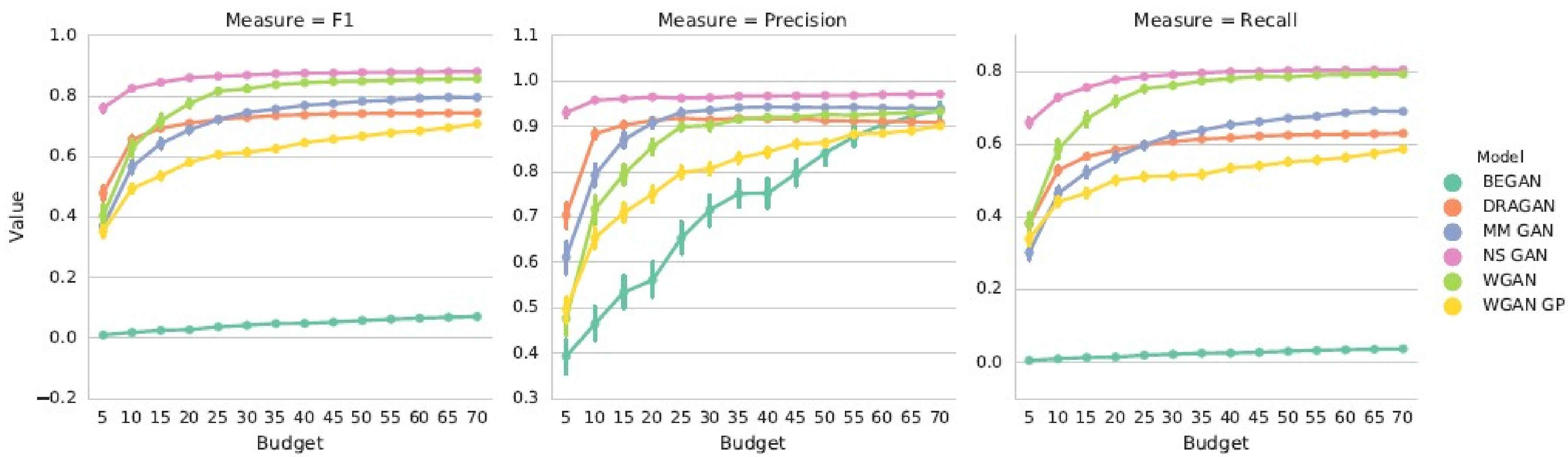


Large-scale hyperparmeter optimisation

- No model strictly dominates the others
  - Strong dependence on dataset
- But: performance not SOTA
  - Larger networks would perform better
  - Authors report best FID (random seed optimisation!)

	MNIST	FASHION	CIFAR	CELEBA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6*	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8*
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.0 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

Budget matters



Combinging VAEs and GANs

- VEEGAN
  - Combine likelihood-based and likelihood-free models
  - variational inference with synthetic likelihoods
- IntroVAE
  - minimize the divergence of the approximate posterior with the prior for real data while maximizing it for the generated samples
  - generator model attempts to mislead the inference model by minimizing the divergence of the generated samples
- Adversarial Autoencoder
- Adversarial Variational Bayes
- ALI/BiGAN
- AlphaGAN

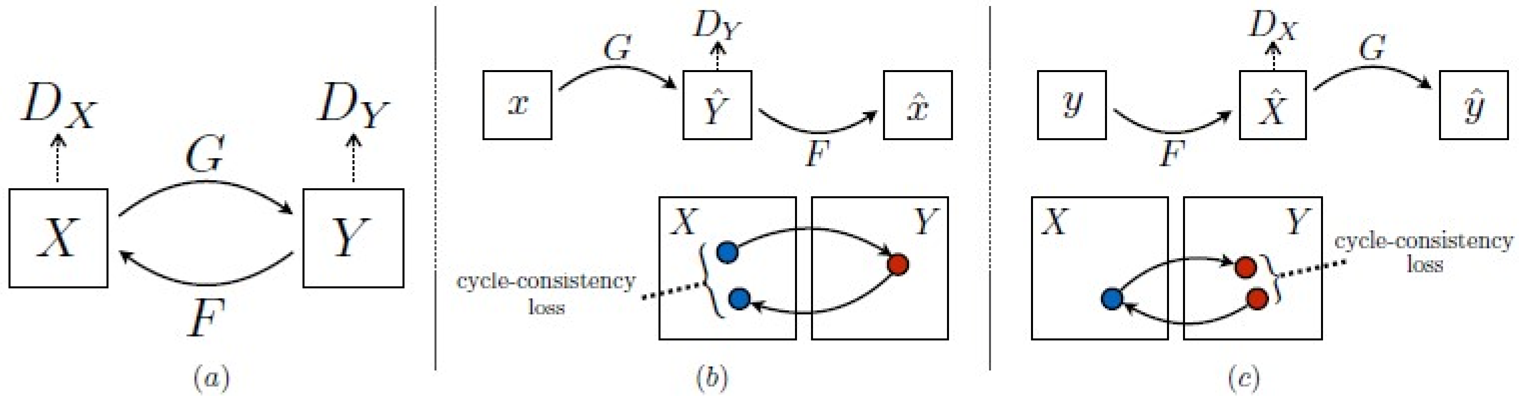
Rosca et al 2017; Huang et al 2018

Unpaired Image-to-Image Translation with CycleGAN

- Unpaired data is cheap
- How to use unpaired data for paired image-to-image translation?
- Idea:
  - Capture special characteristics of one image collection and translate into another image collection
  - Cycle consistency
    - Define additional mapping from generated space to data space
      - Translator $G : X \rightarrow Y$ and translator $F : Y \rightarrow X$
      - F and G inveseres of each other

Zhu et al. *arXiv preprint*, 2017.

CycleGAN



$\|F(G(x)) - x\|_1$ $\|G(F(y)) - y\|_1$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y\sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \quad (1)$$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda\mathcal{L}_{\text{cyc}}(G, F),$$

Style transfer

Figure: Zhu et al. *arXiv preprint*, 2017.

More aplications

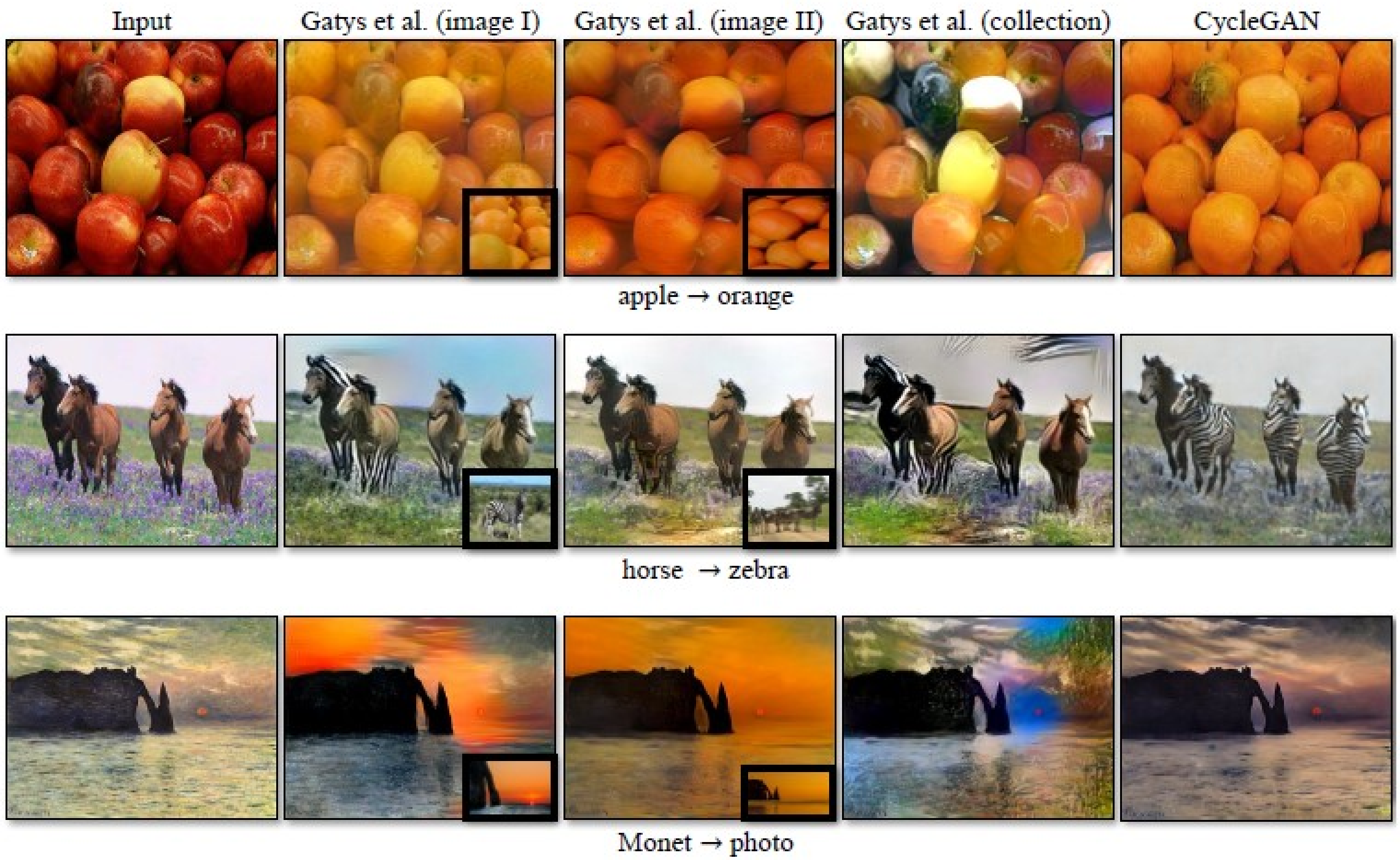


Figure: Zhu et al. *arXiv preprint*, 2017.

Failures

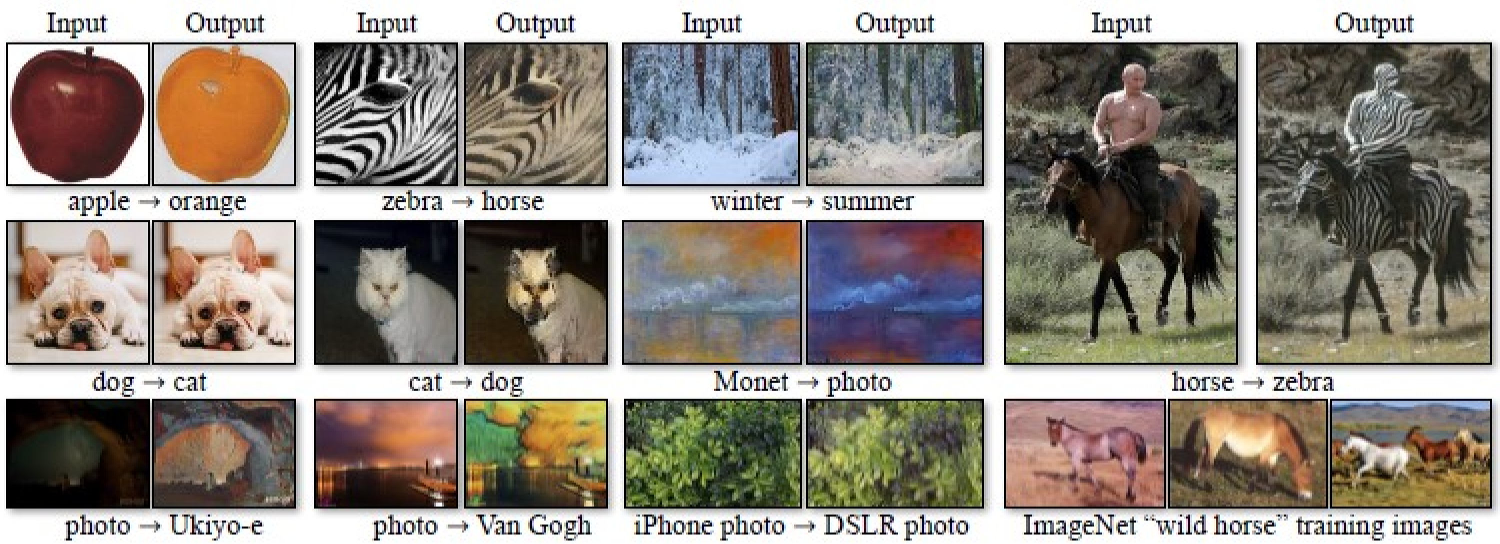


Figure: Zhu et al. *arXiv preprint*, 2017.

CycleGAN

- Excellent qualitative results on several tasks where paired training data does not exist, including collection style transfer, object transfiguration, season transfer, photo enhancement, etc.

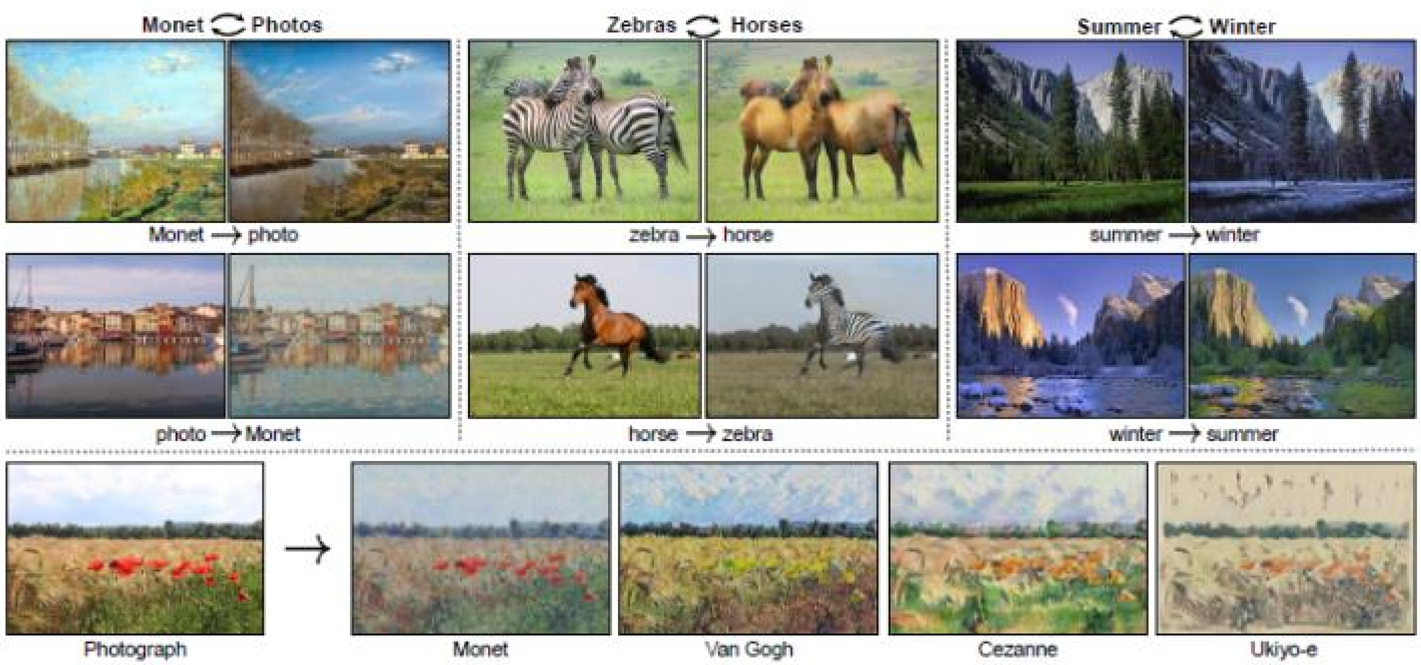


Figure: Zhu et al. *arXiv preprint*, 2017.

Conclusion

- Various types of models
  - Autoregressive models,
    - Explicit density model, opimizes exact likelihood, good samples. Slow
  - VAEs
    - Optimises lower bound on likelihood. Useful representation and inference queries. Blurry samples.
  - GANs
    - Game-theoretic approach, best samples. Tricky and unstable to train
- Large variation between datasets, no one-size-fits-all model
- Lots of open research question and ongoing research