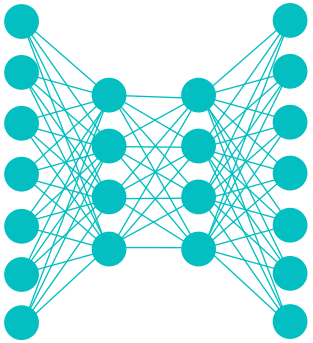
Lecture Notes for

Neural Networks and Machine Learning

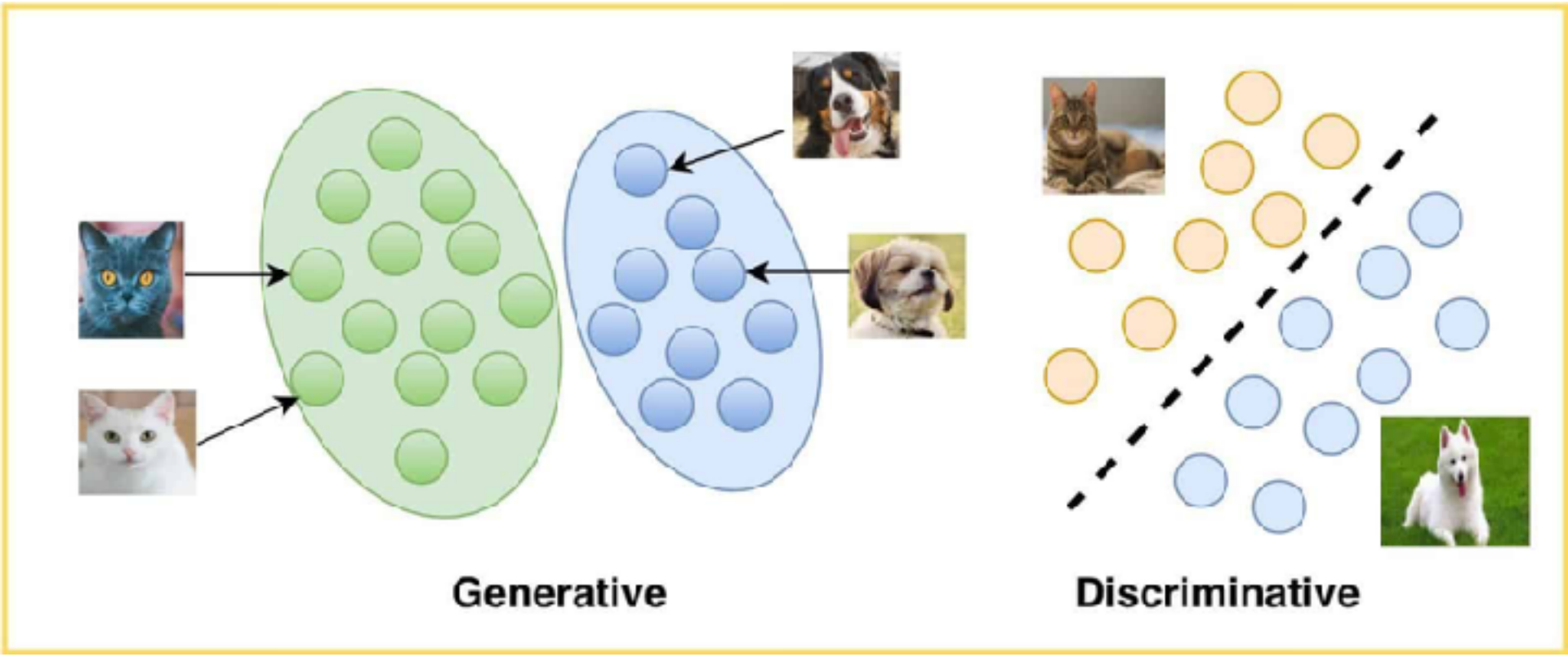
Generative Networks and Auto-Encoding Generators



Logistics and Agenda

- Logistics
  - Office Hours, 12:30-1:30
  - Lab due date
  - Student paper presentation
- Agenda
  - A historical perspective of generative Neural Networks
  - Variational Auto-Encoding
  - VAE in Keras Demo (if time)
  - Adversarial Auto-Encoders (if time)

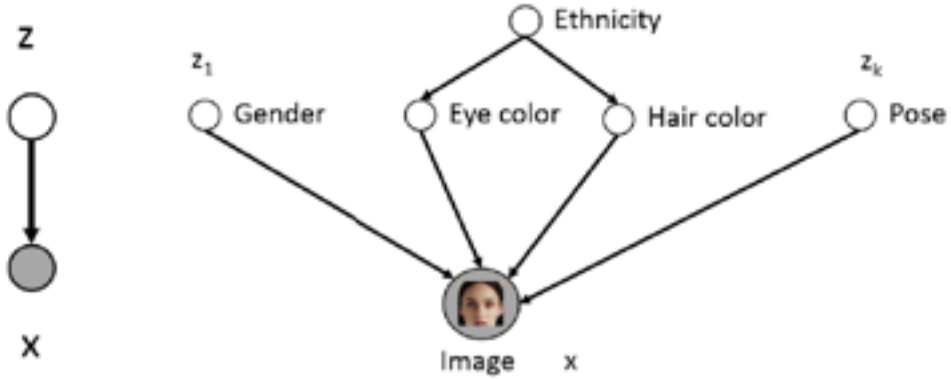
Generative versus Discriminative



https://learnopencv.com/generative-and-discriminative-models/

Motivations: Generative Latent Variables

Latent Space Variables

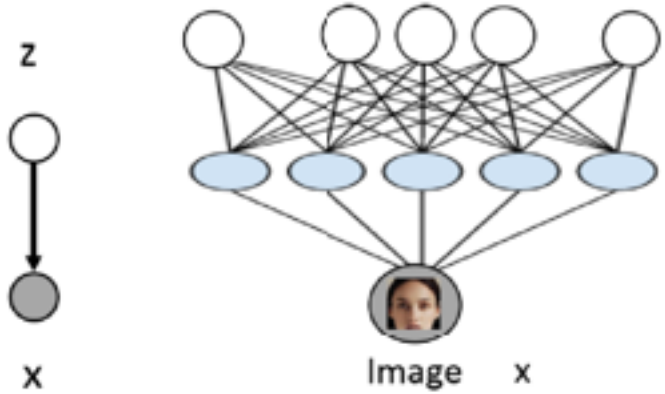


$$p(\mathbf{x} \mid \mathbf{z})$$

Output Observation (e.g., image)

Hard: **z** is expertly chosen

Latent Space Variables



$$p(\mathbf{x} \mid \mathbf{z})$$

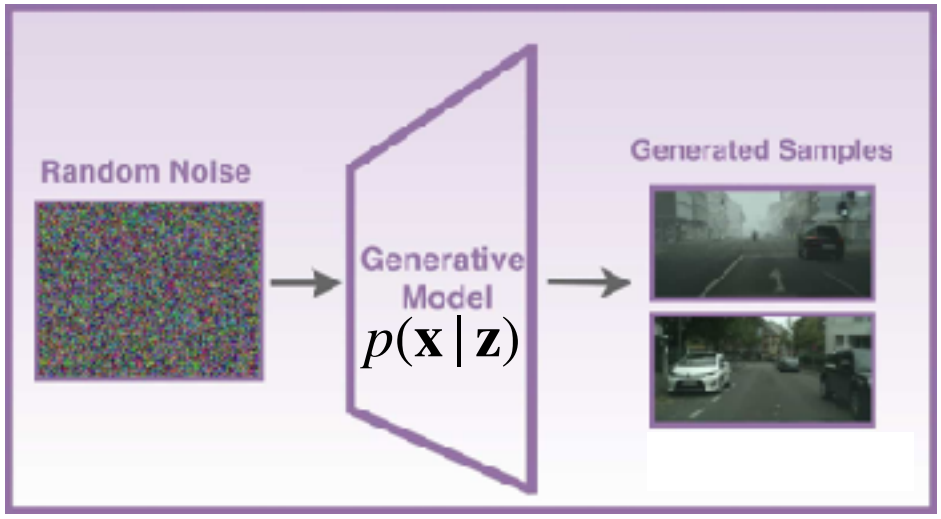
Output Observation (e.g., image)

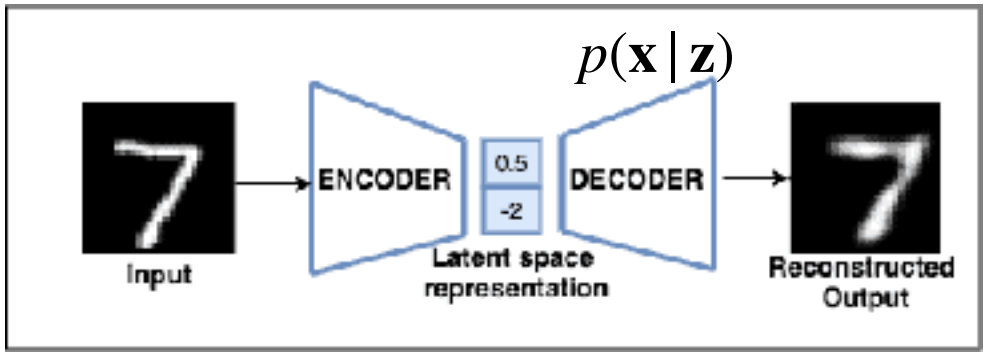
Not as Hard: **z** is trained, latent variables are uncontrolled

Want: $$p(\mathbf{x}) \approx \sum_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z})$$

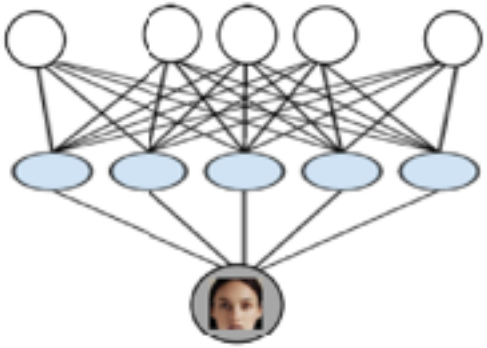
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Motivations: Generative Latent Variables





Latent Space Variables



$$p(\mathbf{x}\,|\,\mathbf{z})$$

Output Observation (e.g., image)

Not as Hard: **z** is trained, latent variables are uncontrolled

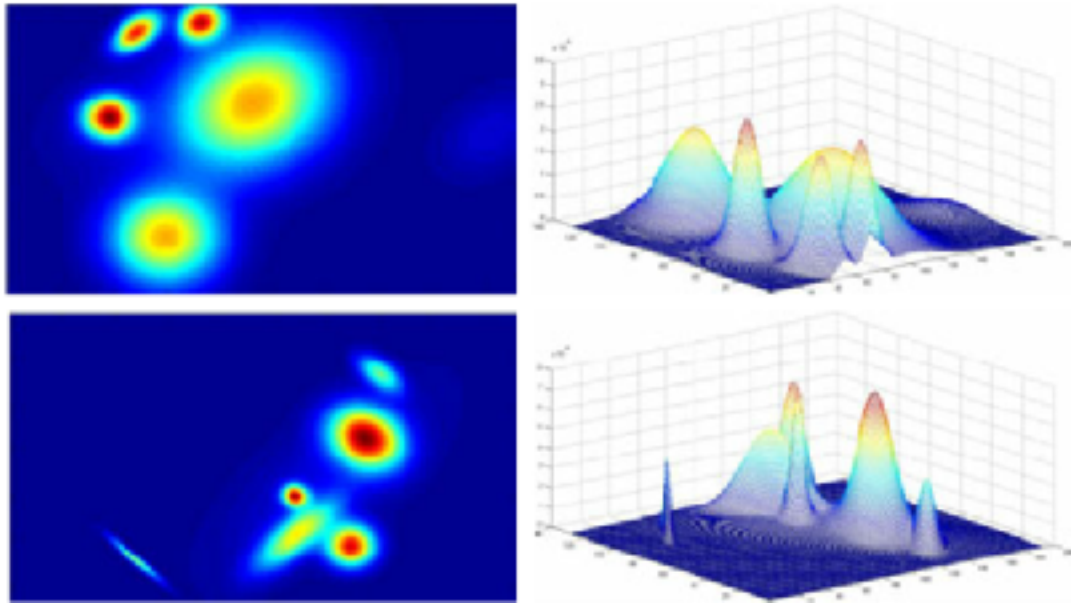
Want: $$p(\mathbf{x}) \approx \sum_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}\,|\,\mathbf{z})$$

https://learnopencv.com/generative-and-discriminative-models/

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Motivation: Mixtures for Simplicity

Want: $$p(\mathbf{x}) \approx \sum_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z})$$



- Each latent variable is mostly independent of other latent variables
- The sum of various mixtures can approximate most any distribution
- Good choice for conditional is Normal Distribution
- Can parameterize $p(x|z)$ to be a Neural Network

$$p_\theta(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mathbf{x}, \mu_k, \Sigma_k)$$

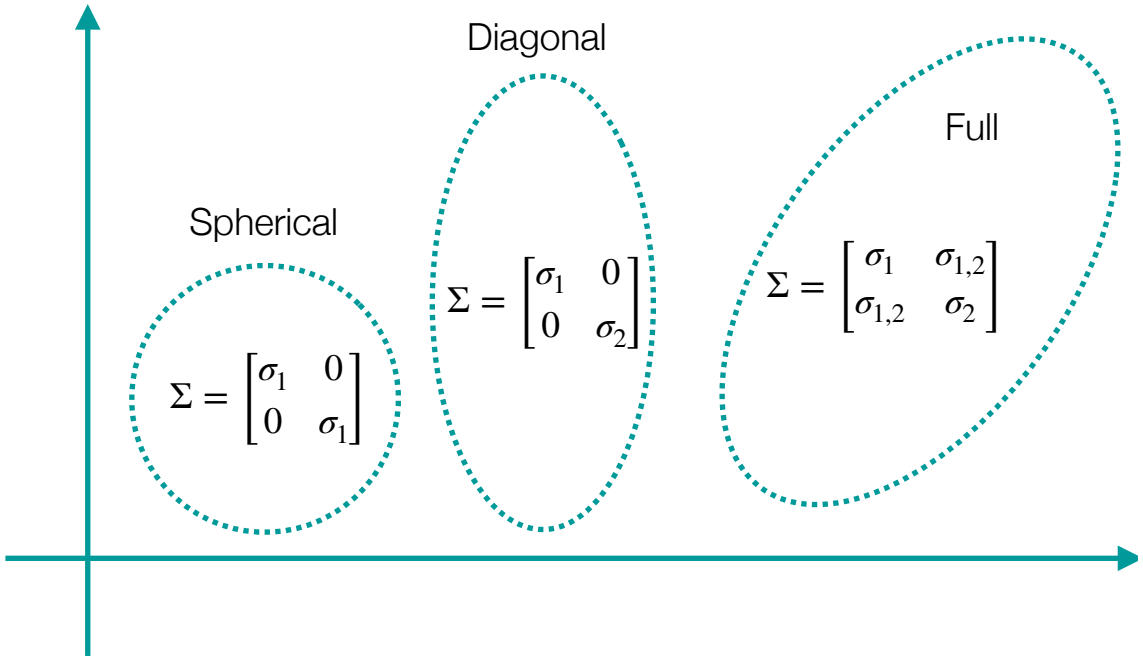
mean and covariance learned

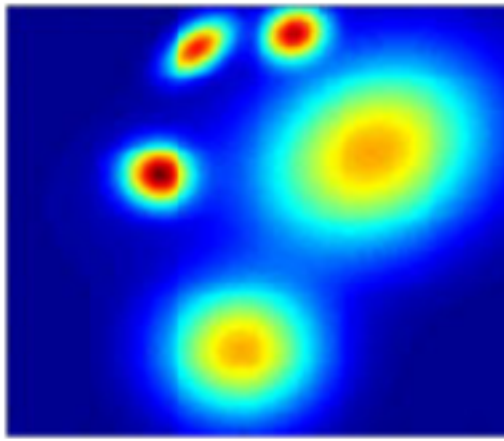
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Motivation: Mixtures for Simplicity

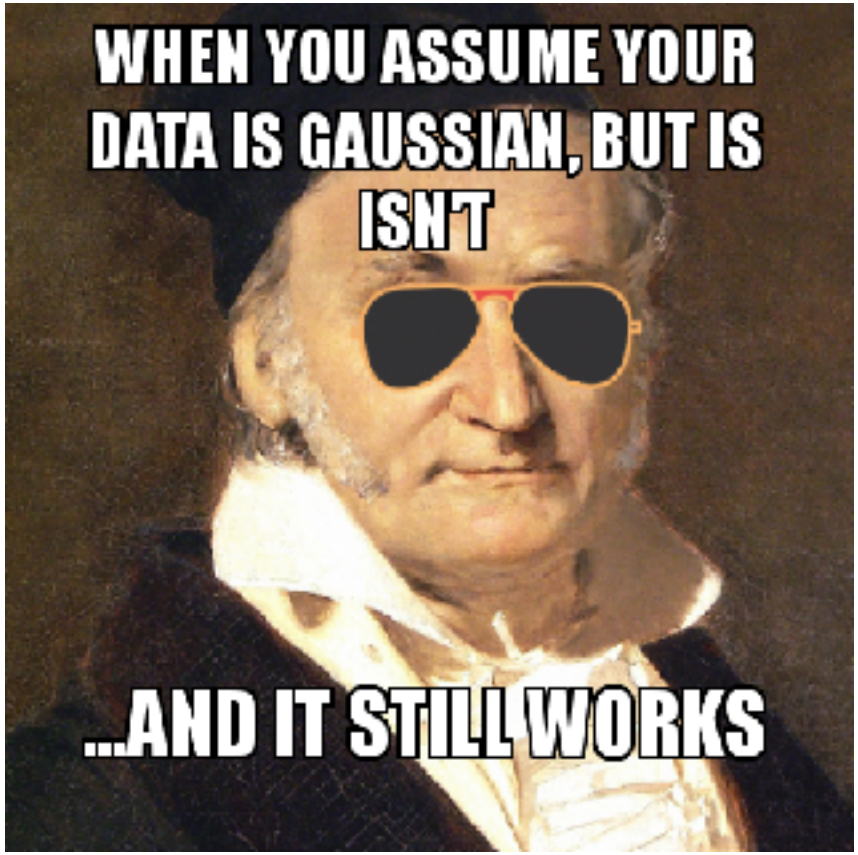
$$= \mathcal{N}(\mathbf{x}, \mu_k, \Sigma_k)$$

mean and covariance learned

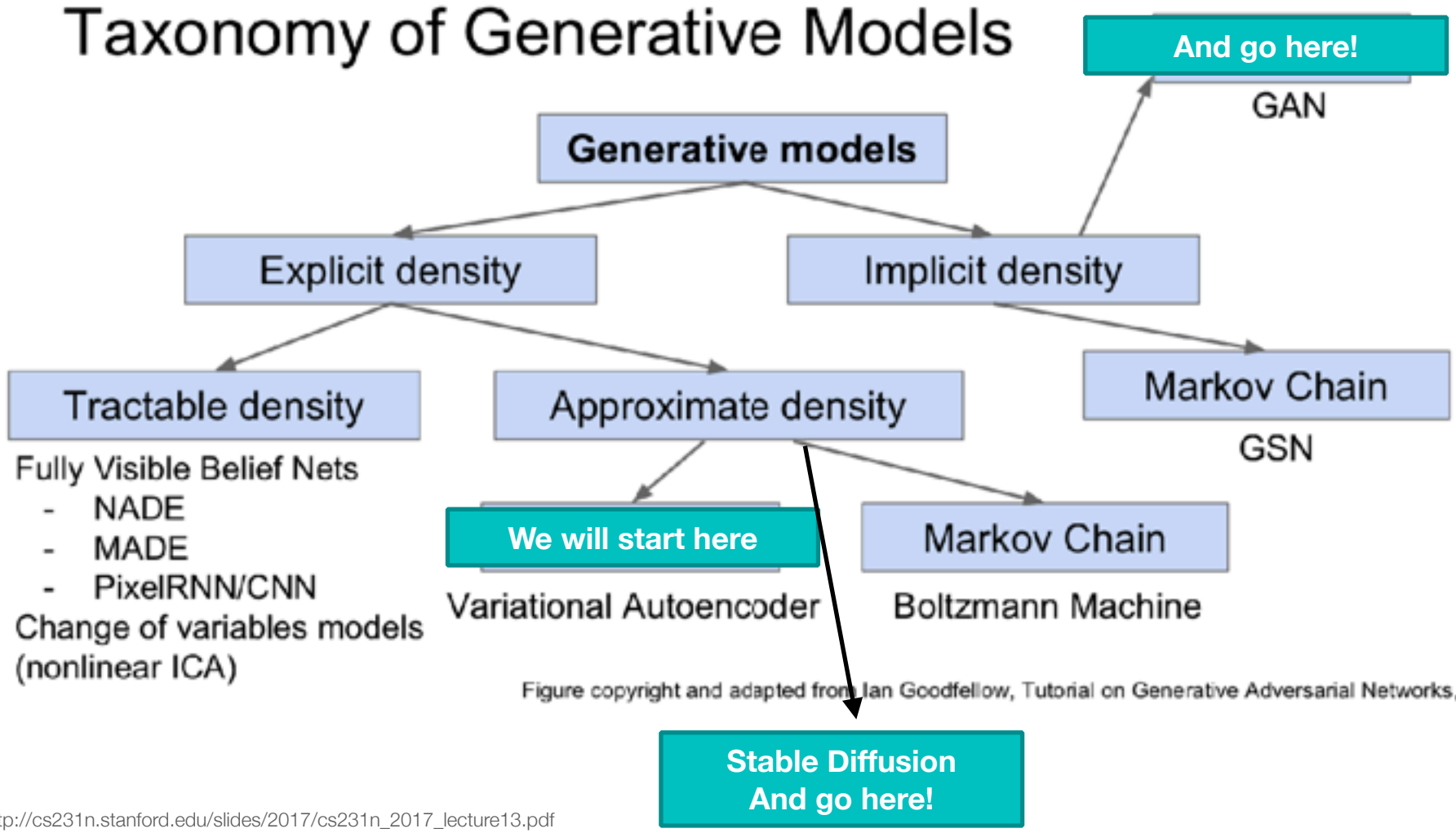




A History of Generative Networks

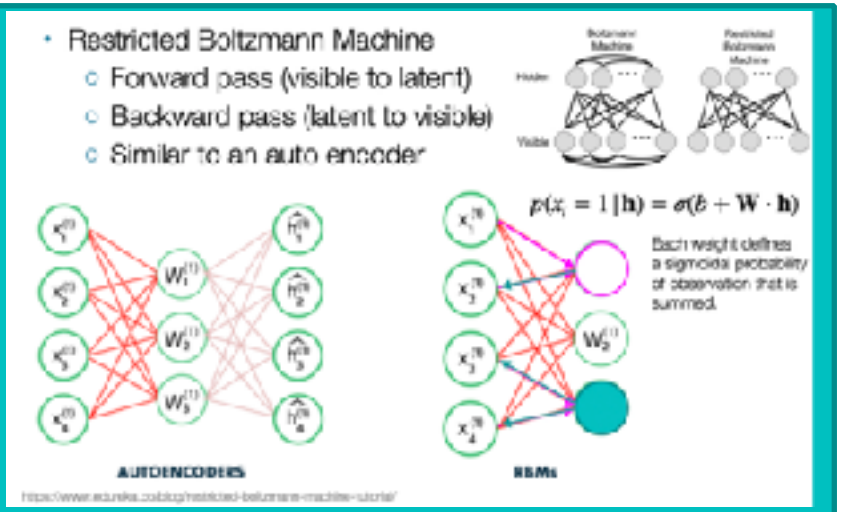


Taxonomy of Generative Models

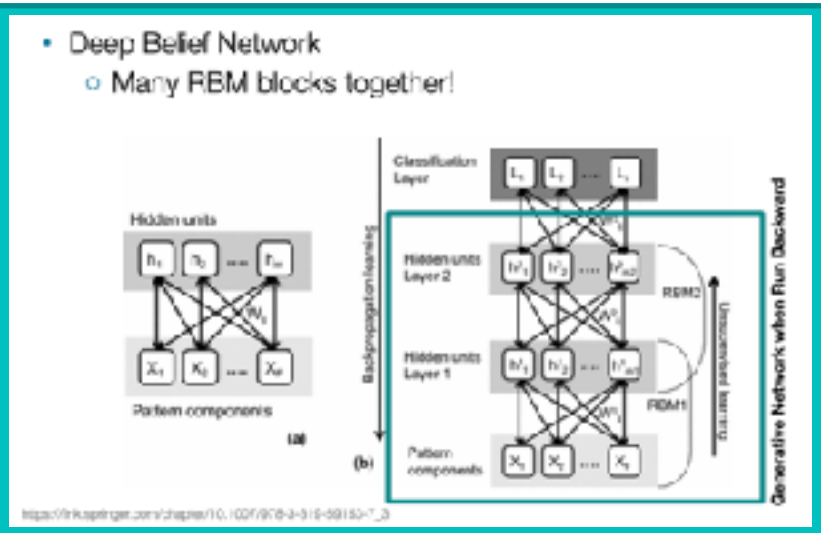


http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

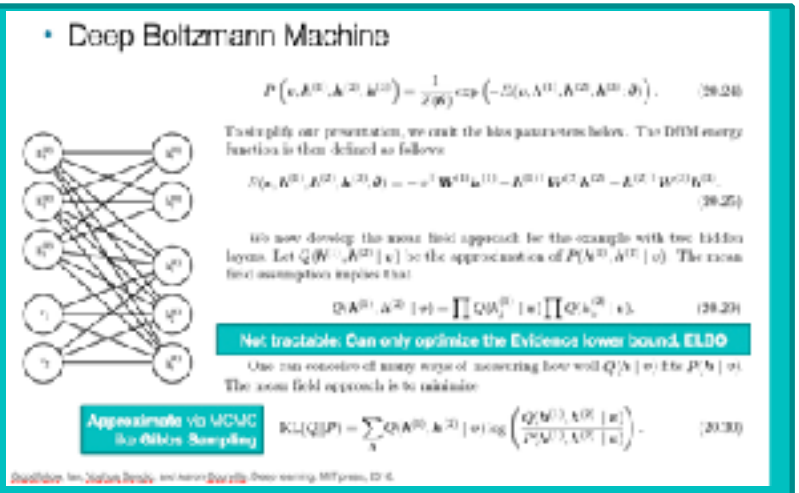
Abridged History of Generative Networks



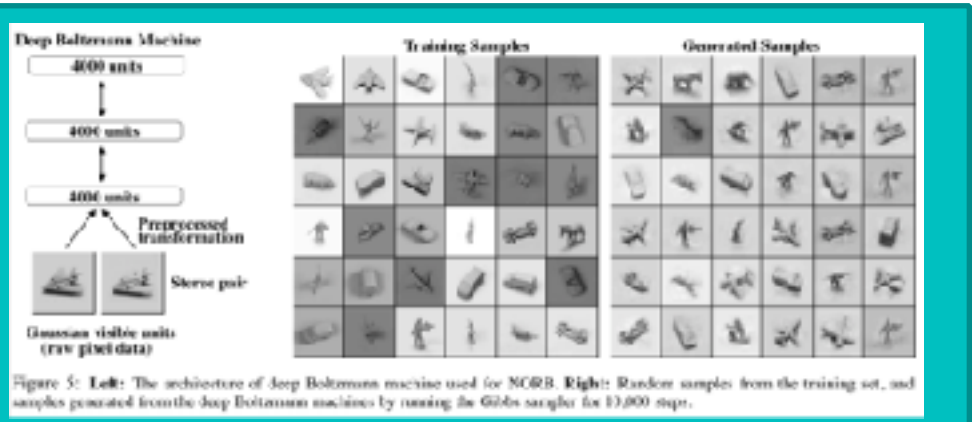
2006 Restricted Boltzmann Machine



2007, Deep Belief Networks
RBMs with many layers

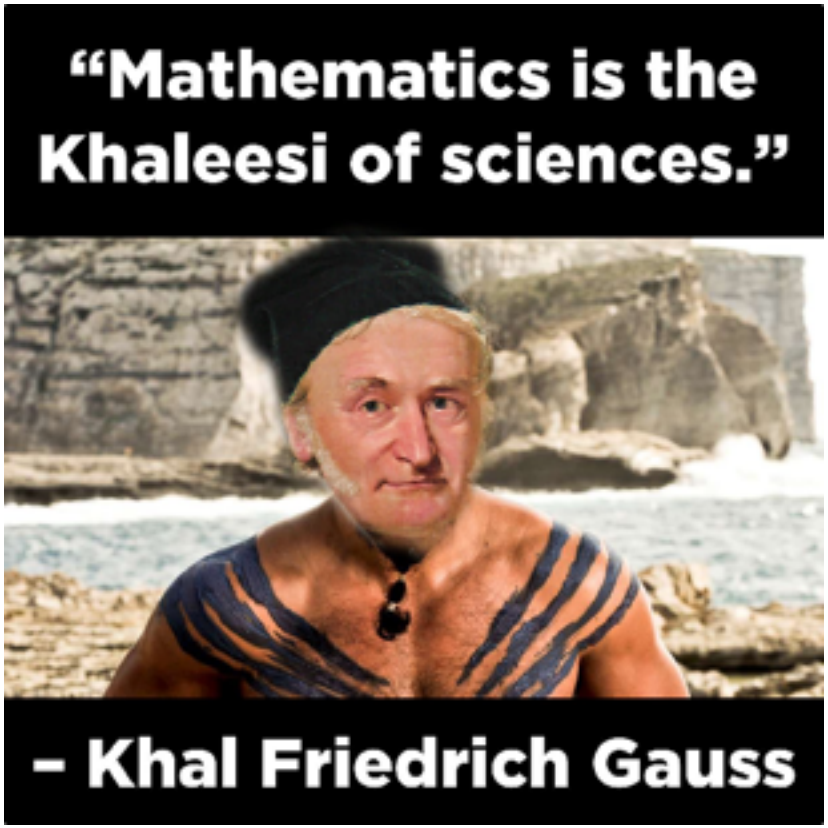


2009 Deep Boltzmann Machine
Goodfellow, Bengio, Courville



2009, Practical Examples
Salakhutdinov and Hinton

Variational Auto Encoding



Aside: Remember These

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

some function

$$\mathbf{E}_{s \leftarrow q(s|x)}[f(\cdot)] = \int q(s \mid x) \cdot f(x)\, dx \approx \sum_{\forall i} q(s \mid x^{(i)}) \cdot f(x^{(i)})$$

could be neural networks

Expected value of f under conditional distribution, q

s is latent variable, $x^{(i)}$ is an observation

$$\therefore \mathbf{E}_{s \leftarrow q(s|x)}[\log f(\cdot)] = \sum_{\forall i} q(s \mid x^{(i)}) \cdot \log\left(f(x^{(i)})\right)$$

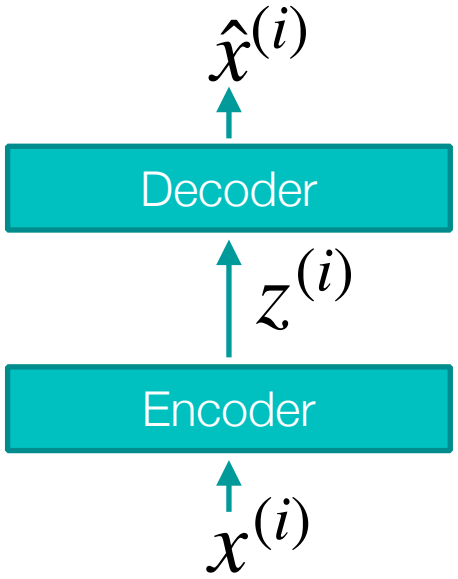
If function is a probability, this is just the negative of cross entropy of distributions:

$$H(q, p) = -\sum_{x} q(x) \cdot \log(p(x))$$

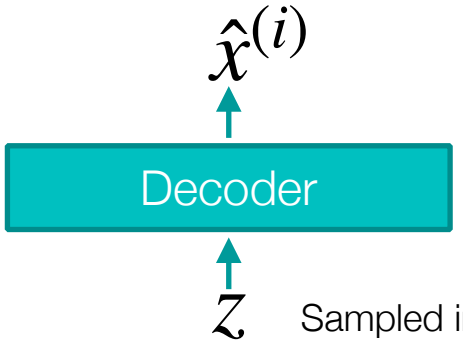
Recall that KL divergence is a measure of difference in two distribution, and is just:

$$D(p \| q) = \sum_{x} p(x) \cdot \log\left(\frac{p(x)}{q(x)}\right) = \mathbf{E}_p\left[\log\left(\frac{p(x)}{q(x)}\right)\right]$$

Can Auto Encoding Generate Samples?



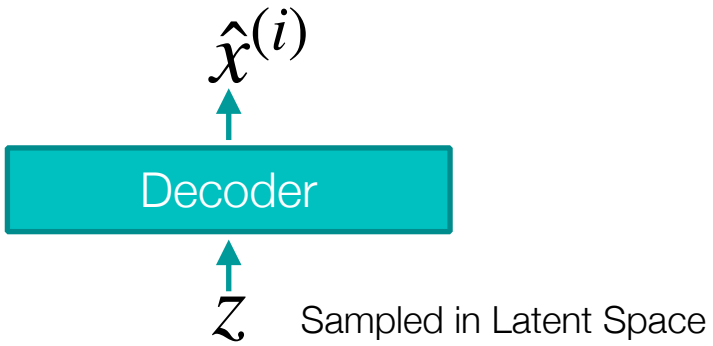
Once trained, is it possible to generate data?



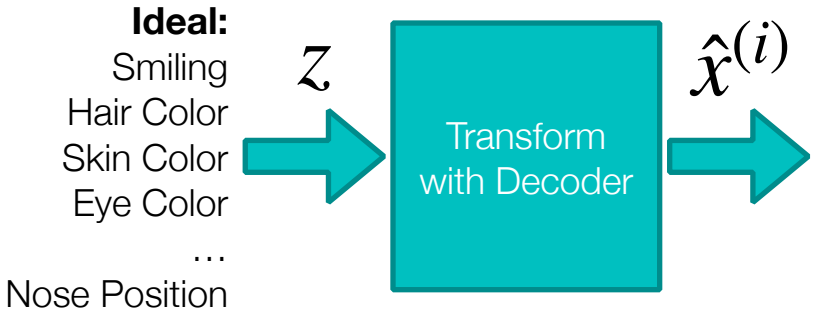
- Does this work for simple auto encoding?
  - Yes, but not satisfactory results
- Learned space is not continuous
- Features could be highly correlated, related in complex ways
  - So, how to sample from the latent space?
- Need to define constraints on latent space…

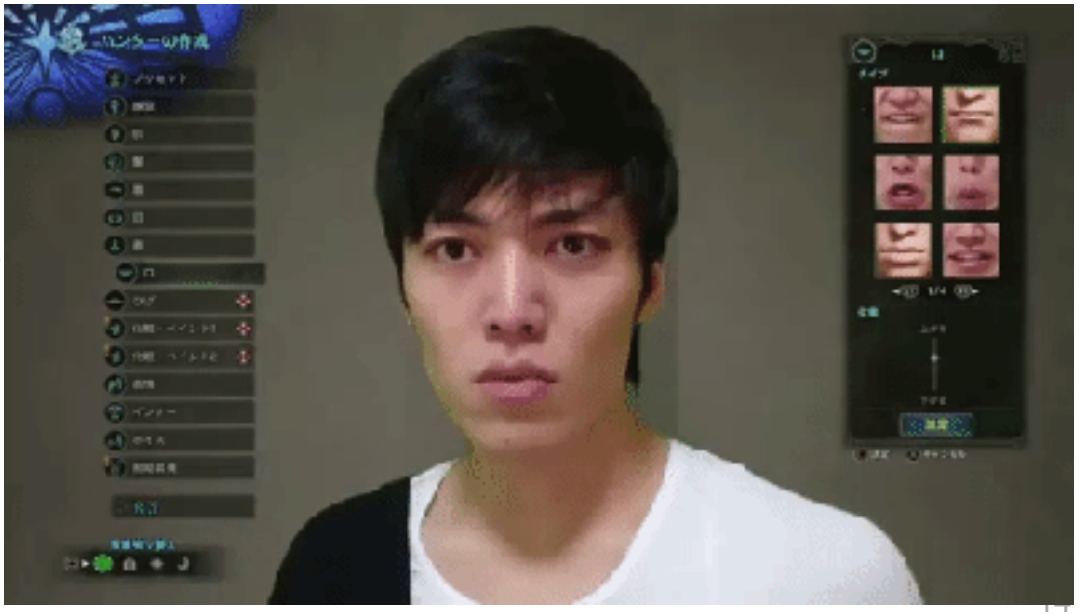
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Reasonable constraints for p(z)?



- Should be simple, easy to sample from: **Normal**
- Each component should be independent and identically distributed (i.i.d).: **Diag. Covariance**
  - Encourages features that may be semantic, like expert might select





Mathematical Motivation

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

We need this inference in order to compute latent variable

$$p(x) = \int p(x \mid z)p(z)dz$$

Denominator is of this form

- We can't compute! **Intractable computation** for all "z"
- So let's define this with **variational inference**:
  - AKA: Find the best approximation of desired distribution using a parametrized set of distributions (usually normal distributions)
  - Only needs to work for z **with observed** $x^{(i)}$
  - 1. **Encode** observed $x^{(i)}$ via network $q(z \mid x^{(i)})$ (with some constraints)
  - 2. Use $q(z \mid x^{(i)})$ to sample z appropriately, then **decode** with another neural network, $p(x^{(i)} \mid z^{(i)})$
  - 3. Make $q(z \mid x^{(i)})$ largest probability possible via Gaussian Distributions

KL Divergence

$$p(x) = \frac{p(z,x)}{p(z|x)} = \frac{p(x|z)p(z)}{p(z|x)}$$

$$\log p(x) = \log p(x) \cdot \int q(z)dz$$

$$\log p(x) = \int \log p(x) \cdot q(z)dz$$

$$\log p(x) = \int q(z) \log \left[\frac{p(x,z)}{p(z|x)} \right] dz$$

$$\log p(x) = \int q(z) \log \left[\frac{p(x,z) \cdot q(z)}{p(z|x) \cdot q(z)} \right] dz$$

$$\log p(x) = \int q(z) \cdot \left(\log \left[\frac{p(x,z)}{q(z)} \right] + \log \left[\cdot \frac{q(z)}{p(z|x)} \right] \right) dz$$

$$\log p(x) = \int q(z) \cdot \log \left[\frac{p(x,z)}{q(z)} \right] dz + \int q(z) \log \left[\frac{q(z)}{p(z|x)} \right] dz$$

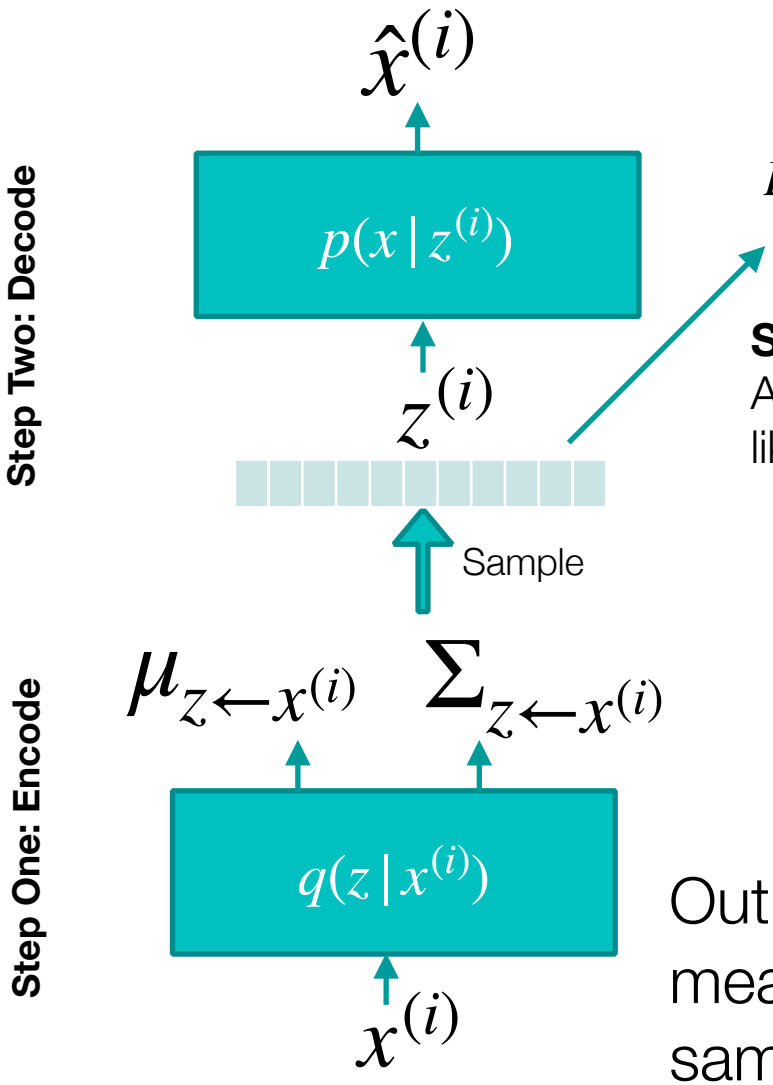
> 0

$$\log p(x) \geq \int q(z) \cdot \log \left[\frac{p(x,z)}{q(z)} \right] dz$$

equal only if
p(z|x) and q(z) are essentially
the same distribution

$$\therefore \min D_{KL} \left[q(z) \| p(z|x) \right]$$

Need a new formulation



Step Three: Make conditional p and q Similar

$$D_{KL}\left[q(z \mid x^{(i)}) \| p(z \mid x^{(i)})\right] = \mathbf{E}_{q(z|x)}\left[\log\left(\frac{q(z \mid x^{(i)})}{p(z \mid x^{(i)})}\right)\right]$$

Step Four: Use Variational Inference

Assume that a family of distributions can maximize likelihood of observing $x^{(i)}$:

$$\log p(x)_{\forall i} \approx \mathbf{E}_{\mathbf{z} \leftarrow q(z|x^{(i)})}\left[\log p(x^{(i)})\right]$$

Max Log Lik:: maximize probability of observed $x^{(i)}$ given family of distributions q hope this is a good approximation

Output of network, q, are the mean and covariance for sampling a variable z

Need a new formulation

$$\log p(x)_{\forall i} \approx \mathbf{E}_{\mathbf{z} \leftarrow q(z|x)} \left[\log p(x^{(i)}) \right]$$ Maximize!

$$= \mathbf{E}_q \left[\log \frac{p(x^{(i)} \mid z) p(z)}{p(z \mid x^{(i)})} \frac{q(z \mid x^{(i)})}{q(z \mid x^{(i)})} \right]$$ Variational + multiply by one

$p(z \mid x^{(i)})$ this is still a problem

$$= \mathbf{E}_q \left[\log p(x^{(i)} \mid z) \right] + \mathbf{E}_q \left[\log \frac{p(z)}{q(z \mid x^{(i)})} \right] + \mathbf{E}_q \left[\log \frac{q(z \mid x^{(i)})}{p(z \mid x^{(i)})} \right]$$

$$= \mathbf{E}_q \left[\log p(x^{(i)} \mid z) \right] - \mathbf{E}_q \left[\log \frac{q(z \mid x^{(i)})}{p(z)} \right] + \mathbf{E}_q \left[\log \frac{q(z \mid x^{(i)})}{p(z \mid x^{(i)})} \right]$$

$$= \mathbf{E}_q \left[\log p(x^{(i)} \mid z) \right] - D_{KL} \left[q(z \mid x^{(i)}) \| p(z) \right] + D_{KL} \left[q(z \mid x^{(i)}) \| p(z \mid x^{(i)}) \right]$$

always non-negative

$$\log p(x)_{\forall i} \geq \mathbf{E}_q \left[\log p(x^{(i)} \mid z) \right] - D_{KL} \left[q(z \mid x^{(i)}) \| p(z) \right]$$ Will Maximize Lower Bound

Can we motivate this in a different way?

The Loss Function

Maximize through
Error of Reconstruction
Same as minimizing cross entropy

want $p(z)$ to be $\mathcal{N}(\mu = 0, \Sigma = I)$
because it makes nice latent space

$$q(z \mid x^{(i)}) \rightarrow (\mu_{z|x}, \Sigma_{z|x}) \quad p(z) \rightarrow \mathcal{N}(0,1)$$

$$D_{KL}\left((\mu, \Sigma) \| \mathcal{N}(0,1)\right) = \frac{1}{2}\left(\text{tr}(\Sigma) + \mu \cdot \mu^T - \underbrace{k}_{|z|} - \log\left(\det(\Sigma)\right)\right)$$

Can get this by manipulating the KL for normal distribution

Determinant of diagonal matrix is simple. Motivates diagonal covariance…

$$= \frac{1}{2}\left(\sum_k \Sigma_{k,k} + \sum_k \mu_k^2 - \sum_k 1 - \log\left(\prod_k \Sigma_{k,k}\right)\right)$$

$$= \frac{1}{2}\left(\sum_k \Sigma_{k,k} + \sum_k \mu_k^2 - \sum_k 1 - \sum_k \log \Sigma_{k,k}\right)$$

$$\geq \mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)} \mid z_k)\right] - D_{KL}\left[q(z \mid x_k^{(i)}) \| p(z_k)\right]$$

$$= \frac{1}{2}\sum_k \left(\Sigma_{k,k} + \mu_k^2 - 1 - \log \Sigma_{k,k}\right)$$

The Covariance Output

$$\geq \mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right] - D_{KL}\left[q(z|x^{(i)})\|p(z)\right]$$

Maximize through
Error of Reconstruction
Same as minimizing cross entropy

want $p(z)$ to be $\mathcal{N}(\mu = 0, \Sigma = I)$
because it makes nice latent space

$q(z|x^{(i)}) \rightarrow (\mu_{z|x}, \Sigma_{z|x}) \quad p(z) \rightarrow \mathcal{N}(0,1)$

$$= \frac{1}{2}\sum_{k}\left(\Sigma_{k,k} + \mu_k^2 - 1 - \log \Sigma_{k,k}\right)$$

raw covariance is not numerically stable because of underflow

$$\log \Sigma_{k,k} = \widehat{\Sigma_{k,k}}$$

predicted by $q(z|x^{(i)})$

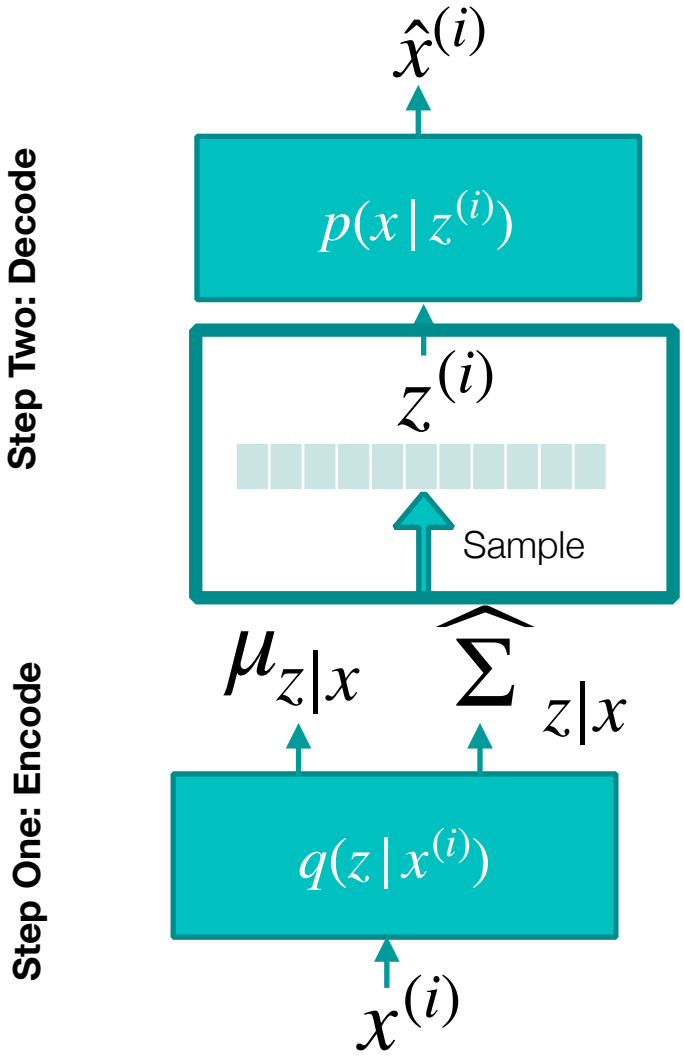
$$= \frac{1}{2}\sum_{k}\left(\exp\left(\widehat{\Sigma_{k,k}}\right) + \mu_k^2 - 1 - \widehat{\Sigma_{k,k}}\right)$$

so we will have the neural network output log variance

Also, remember we assume **diagonal covariance**, so z's are not correlated
This means covariance is only a vector of variances (the diagonal of Σ)

Back Propagating

$$\geq \mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}\,|\,z)\right] - D_{KL}\left[q(z\,|\,x^{(i)})\|p(z)\right]$$



This is partially differentiable by chain rule…

$$\mathcal{N}(\mu_{z|x}, \exp(\widehat{\Sigma_{z|x}})) = z$$

$$= \mu(x^{(i)}) + \exp(\widehat{\Sigma(x^{(i)})}) \cdot \mathcal{N}(0,1)$$

To update q,
we need to back propagate
through sampling layer. How?

The Loss Function Implementation

```
# Encode the input into a mean and variance parameter
z_mean, z_log_variance = encoder(input_img)
# Draw a latent point using a small random epsilon
z = z_mean + exp(z_log_variance) * epsilon

# Then decode z back to an image
reconstructed_img = decoder(z)
# Instantiate a model
model = Model(input_img, reconstructed_img)
```

$\mu(x^{(i)})$ $\widehat{\Sigma(x^{(i)})}$ $q(z|x^{(i)})$

$$z = \mu(x^{(i)}) + \exp(\widehat{\Sigma(x^{(i)})}) \cdot \mathcal{N}(0,1)$$

$\hat{x}^{(i)} = p(x^{(i)}|z)$

```
def vae_loss(self, x, z_decoded):
    x = K.flatten(x)
    z_decoded = K.flatten(z_decoded)
    xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
    kl_loss = -5e-4 * K.mean(
        1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
    return K.mean(xent_loss + kl_loss)
```

$$-\mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right]$$

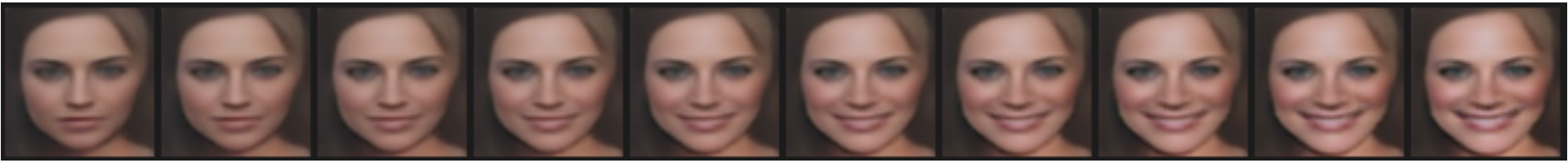
$$-\lambda \sum_k 1 + \widehat{\Sigma(x^{(i)})} - \mu(x^{(i)})^2 - \exp(\widehat{\Sigma(x^{(i)})})$$

Note:
Flipped from maximization to minimization
and added lambda for tradeoff in reconstruction, normal latent space

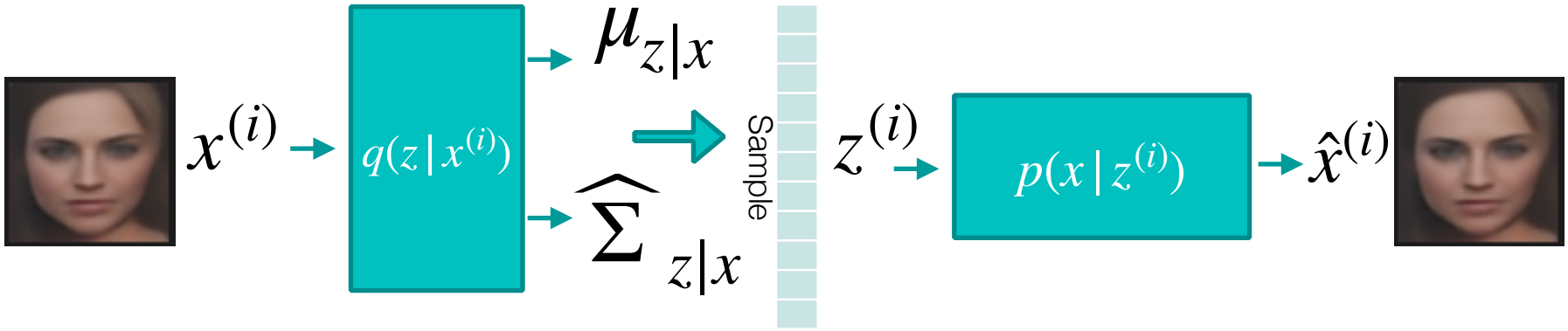
$$= -\mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right] - \lambda \sum_k 1 + \widehat{\Sigma(x^{(i)})} - \mu(x^{(i)})^2 - \exp(\widehat{\Sigma(x^{(i)})})$$

Now that its trained, so what?

Encoding faces, then adjust the “z” that relates to smiling.



Investigate what happens by moving around each z_i

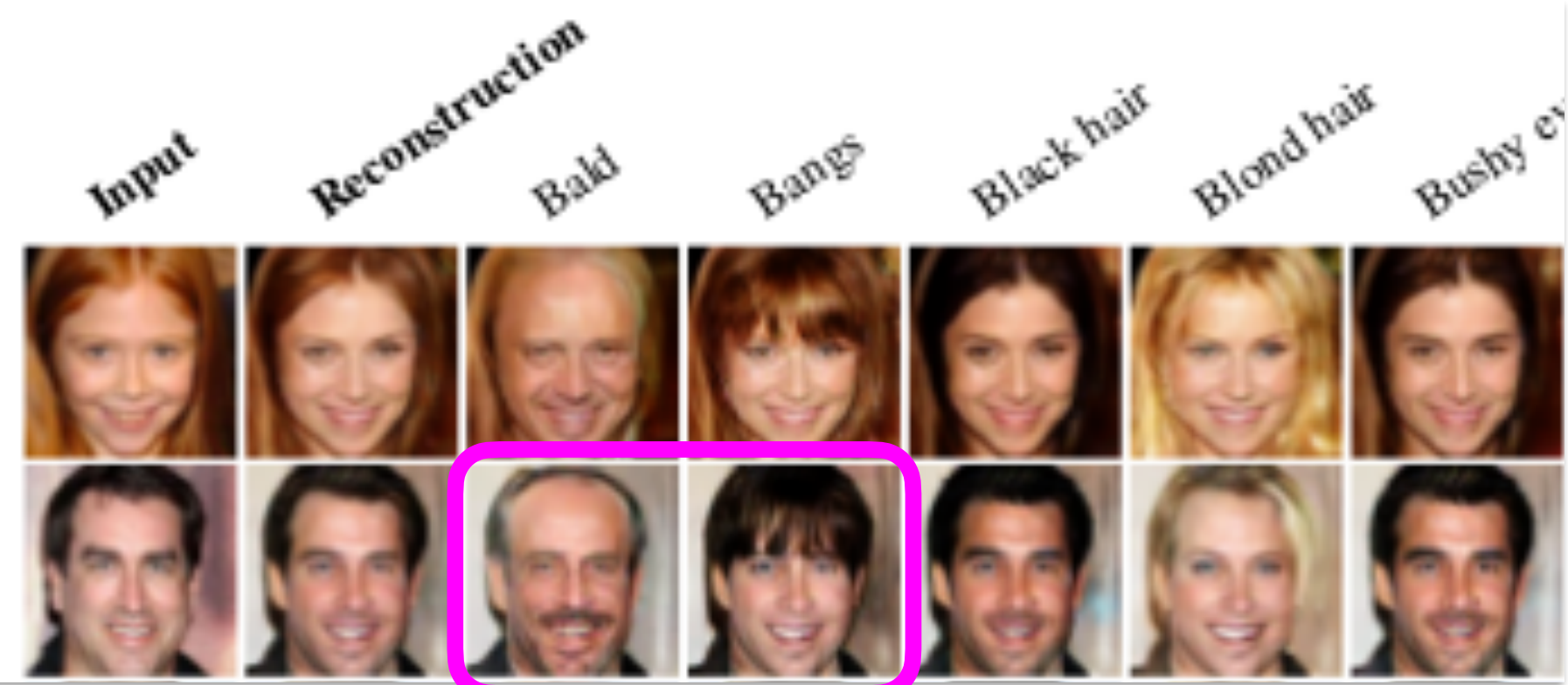


VAE Examples

Encoding faces, then adjust the “z” that relates to smiling.

VAE Examples

Different, automatically found z, latent variables



VAEs in Keras

Sampling from variational auto encoder using MNIST

Demo by Francois Chollet

In Master Repo: 07a VAEs in Keras.ipynb

Follow Along: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.4-generating-images-with-vaes.ipynb

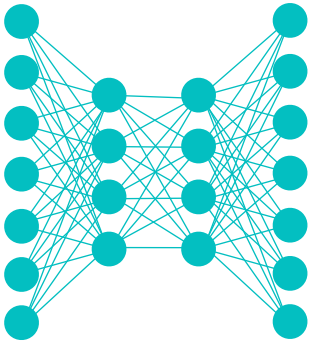
Lecture Notes for

Neural Networks and Machine Learning

Generative Networks

Next Time:
General GANs

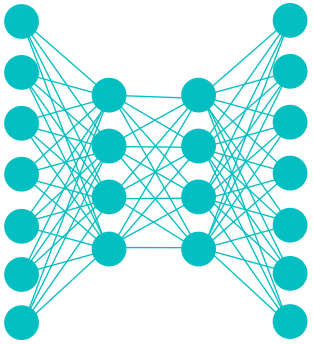
Reading: Chollet CH8



Lecture Notes for

Neural Networks and Machine Learning

Adversarial Auto Encoders



Logistics and Agenda

- Logistics
  - None
- Agenda
  - Student Paper Presentation
  - VAEs, Last time
  - AAE
  - Simple Generative Adversarial Networks
  - GANs Demo (next time)

Paper Presentation

Masked Autoencoders As Spatiotemporal Learners

Christoph Feichtenhofer* **Haoqi Fan*** **Yanghao Li** **Kaiming He**
Meta AI, FAIR

https://github.com/facebookresearch/mae_st

Last Time: VAEs

```
# Encode the input into a mean and variance parameter
z_mean, z_log_variance = encoder(input_img)
# Draw a latent point using a small random epsilon
z = z_mean + exp(z_log_variance) * epsilon
# Then decode z back to an image
reconstructed_img = decoder(z)
# Instantiate a model
model = Model(input_img, reconstructed_img)
```

$$z = \mu(x^{(i)}) + \exp(\Sigma(x^{(i)})) \cdot \mathcal{N}(0,1)$$

```
def vae_loss(self, x, z_decoded):
    x = K.flatten(x)
    z_decoded = K.flatten(z_decoded)
    xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
    kl_loss = -5e-4 * K.mean(
        1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
    return K.mean(xent_loss + kl_loss)
```

$$-\mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right]$$

$$-\sum_{z} 1 + \widehat{\Sigma(x^{(i)})} - \mu(x^{(i)})^2 - \exp(\widehat{\Sigma(x^{(i)})})$$

Note:
Flipped from maximization to minimization

$$= -\mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right] - \sum 1 + \widehat{\Sigma(x^{(i)})} - \mu(x^{(i)})^2 - \exp(\widehat{\Sigma(x^{(i)})}$$

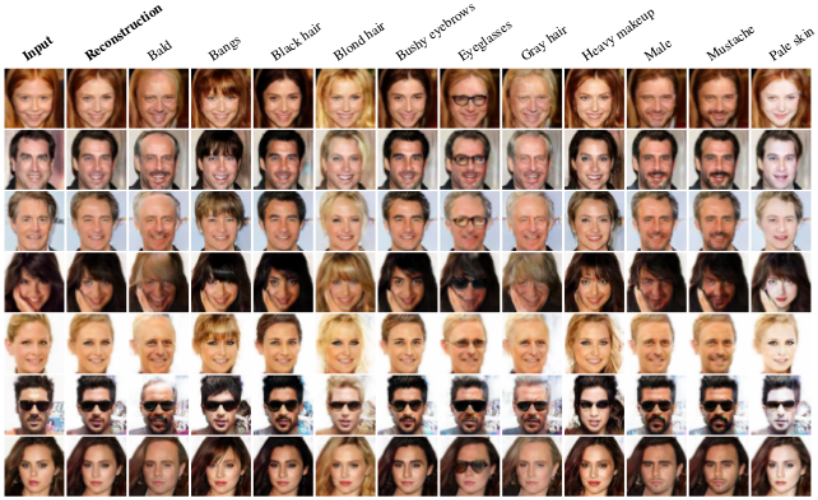
VAEs in Keras

Sampling from variational auto encoder using MNIST

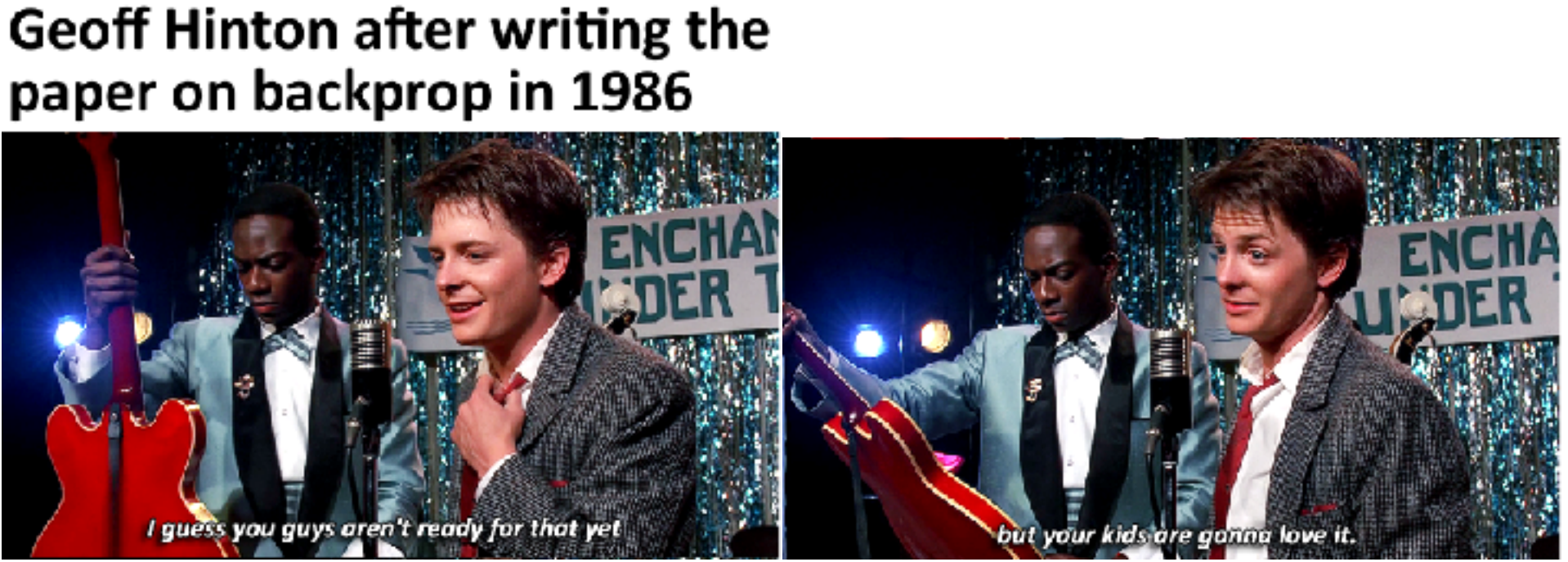
Demo by Francois Chollet

In Master Repo: 07a VAEs in Keras.ipynb

Follow Along: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.4-generating-images-with-vaes.ipynb



Adversarial Auto Encoding



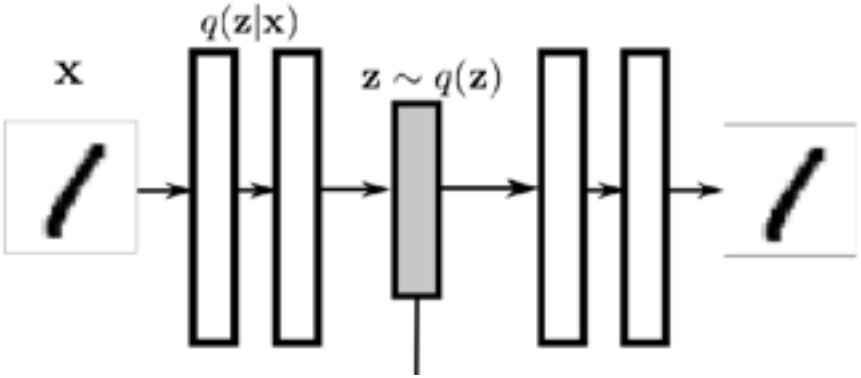
Do we need something more than VAE?

- Arguments for Yes:
  - ELBO is not global optimum! But… provides theory
  - Assumption of Normal distributions to q(z) is limiting
  - Training tends to be slower (…*so do GANs…*)
  - Manifold of distributions do not cover the latent space completely (not guaranteed)
  - We can't incorporate distributions separately for different classes without reformulating loss function
- Arguments for No:
  - It seems hard, how can we research methods that aren't low hanging fruit? Plus the VAE math was like really hard for me to understand so this is not going to be very fun, guaranteed. Ah, fine lets look at it.

The Main Idea

- How can we enforce constraints on the latent space with a pair of networks?

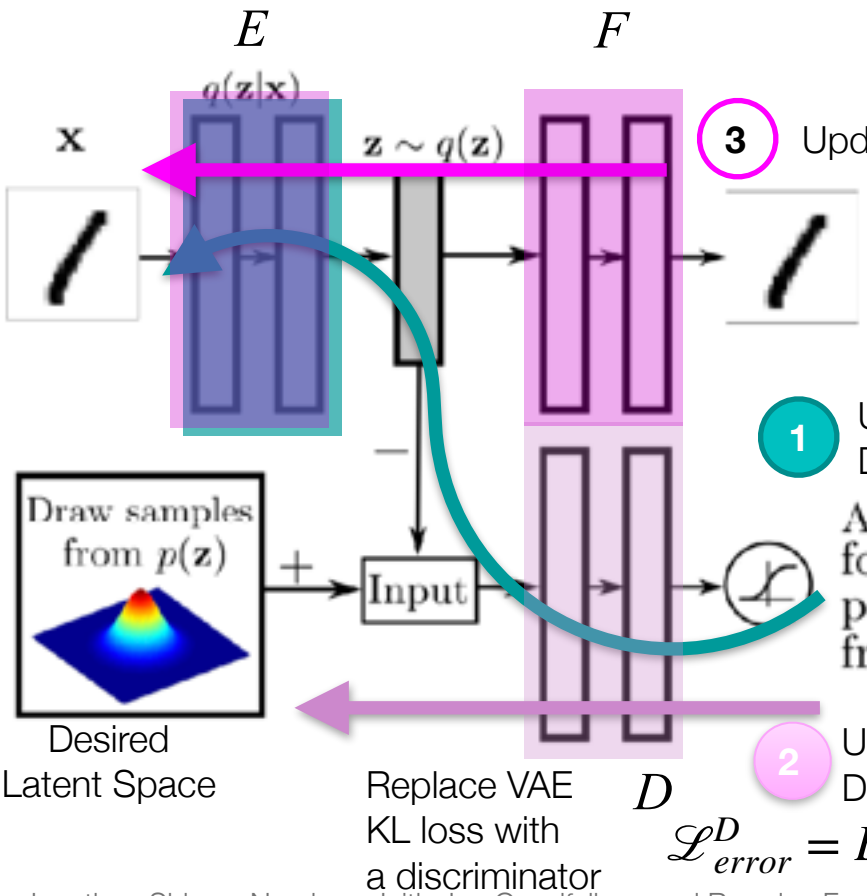
Encoder-Decoder Network



Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders." arXiv preprint arXiv:1511.05644 (2015).

The Main Idea

- How can we enforce constraints on the latent space with a pair of networks?



$$\mathscr{L}^{E,F}_{error} = BCE(x, F \circ E(x))$$

3 Update Encoder and Decoder Pair

Impose Reconstruction Loss

$$\mathscr{L}^{E}_{error} = BCE(1, D \circ E(x))$$

1 Update Encoder to fool Discriminator
Do **not** update Discriminator

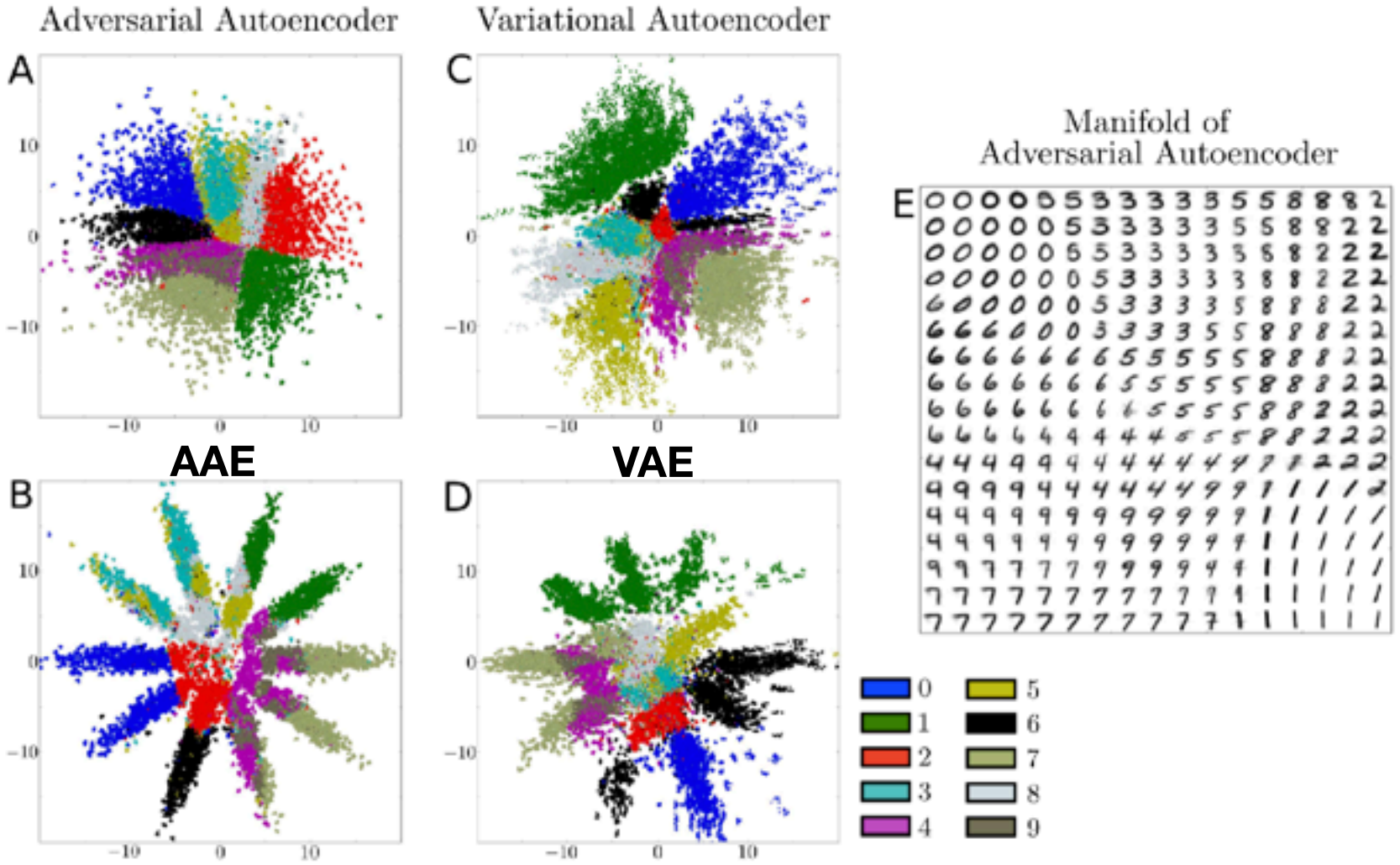
Adversarial cost for distinguishing positive samples $p(\mathbf{z})$ from negative samples $q(\mathbf{z})$

2 Update Discriminator to Detect Fake z
Do **not** update Encoder

$$\mathscr{L}^{D}_{error} = BCE(0, D \circ E(x)) + BCE(1, D(z))$$

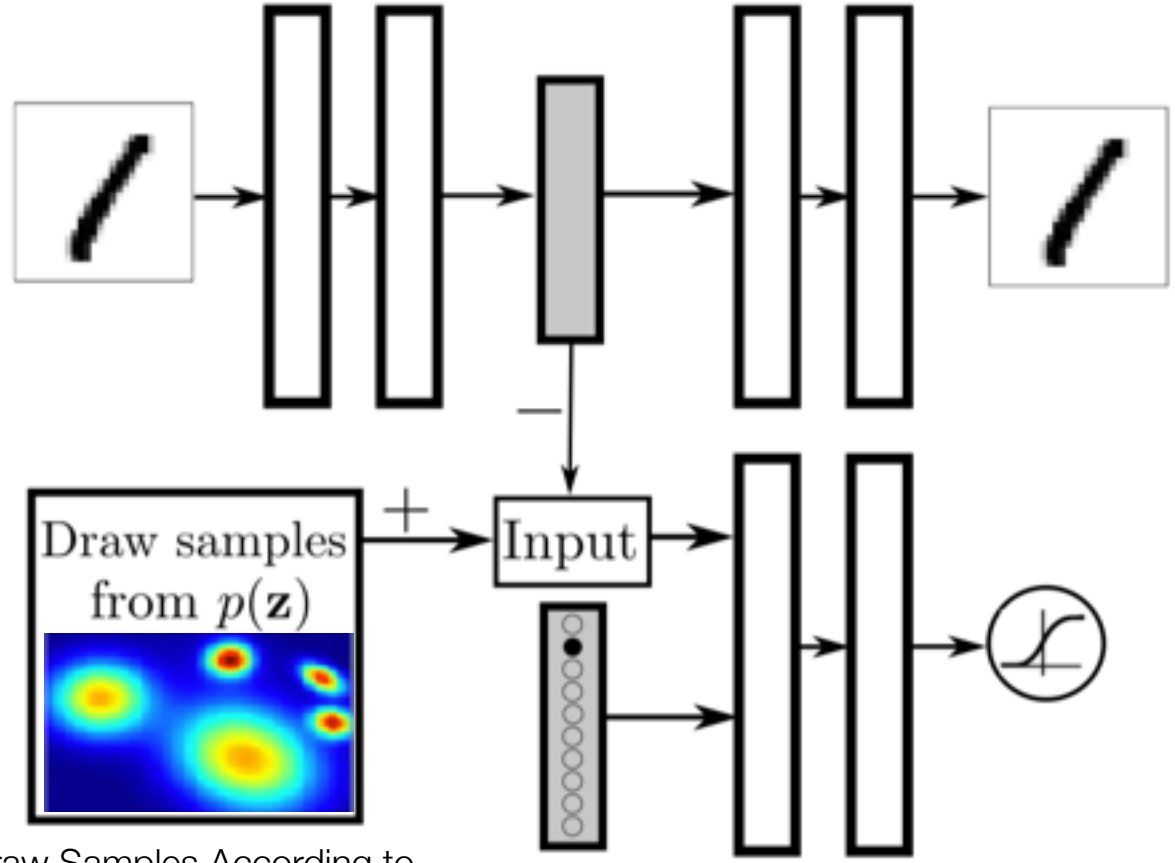
Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders." arXiv preprint arXiv:1511.05644 (2015).

Arbitrary Prior Distributions



Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders." arXiv preprint arXiv:1511.05644 (2015).

Sampling From Classes

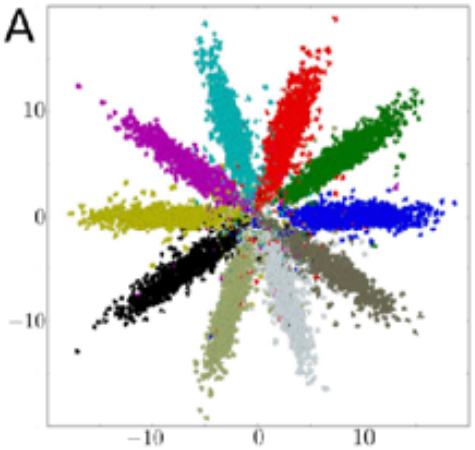


Draw Samples According to
Specific Class Prior Distributions

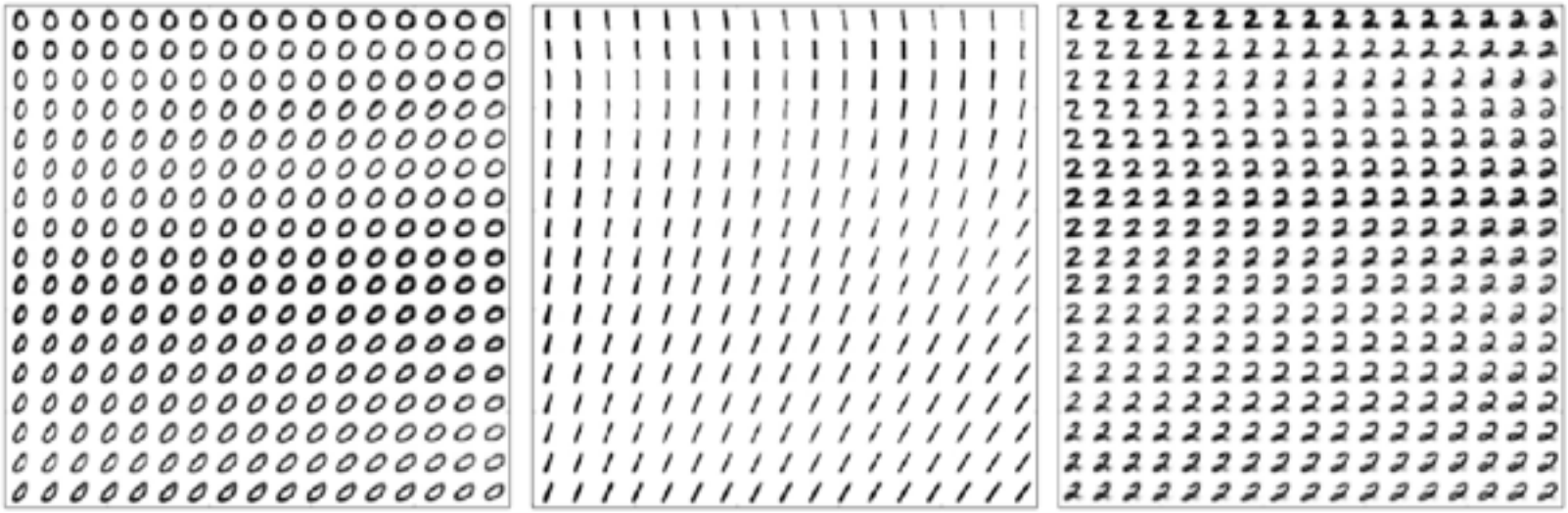
Provide Class Chosen
as One Hot Vector in Discriminator

Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders." arXiv preprint arXiv:1511.05644 (2015).

Conditional Class Latent Spaces

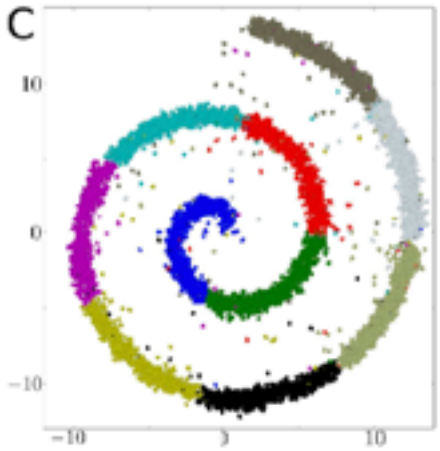


Sample Along Main Axis of the Gaussian Component for Each Digit

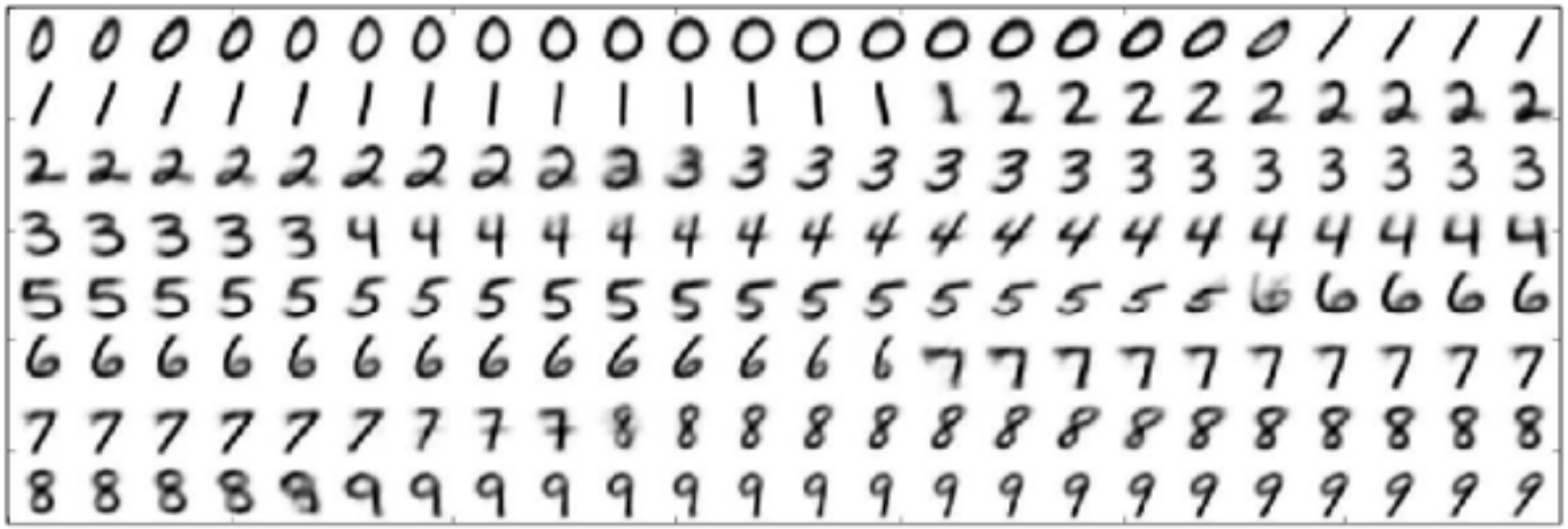


Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, **Ian Goodfellow**, and Brendan Frey. "Adversarial autoencoders." *arXiv preprint arXiv:1511.05644* (2015).

Conditional Class Latent Spaces

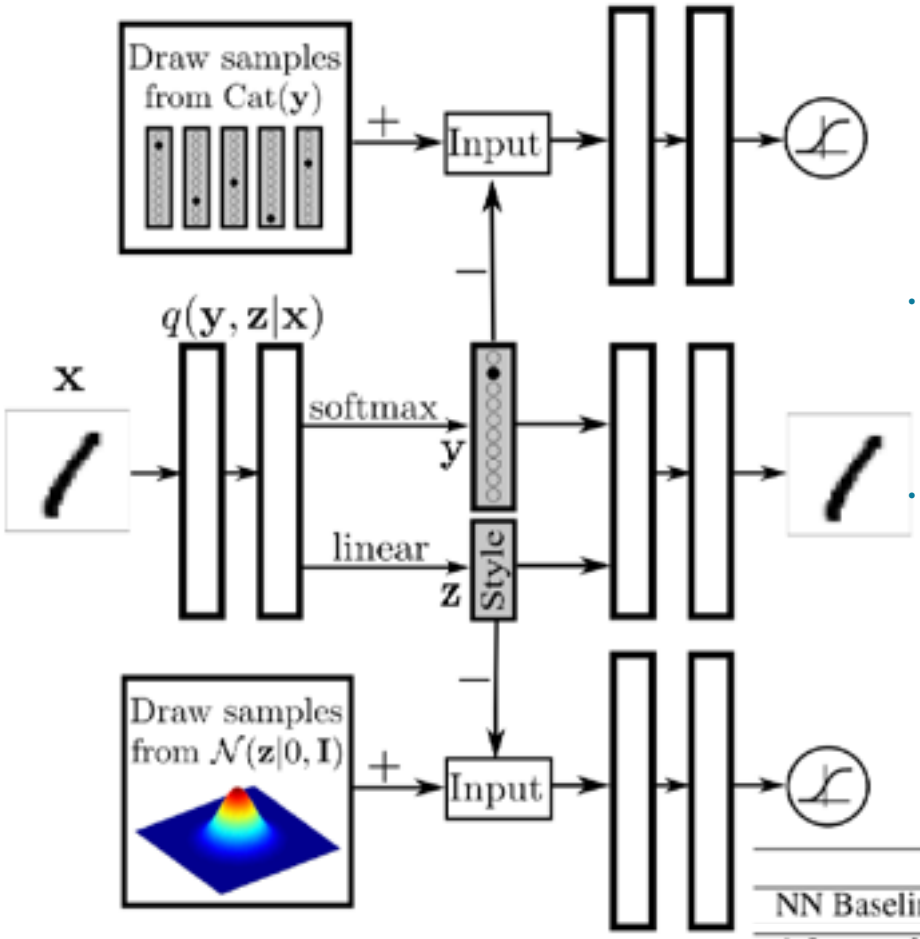


Sample Along Swiss Roll Axis



Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, **Ian Goodfellow**, and Brendan Frey. "Adversarial autoencoders." *arXiv preprint arXiv:1511.05644* (2015).

Semi-Supervised Classification



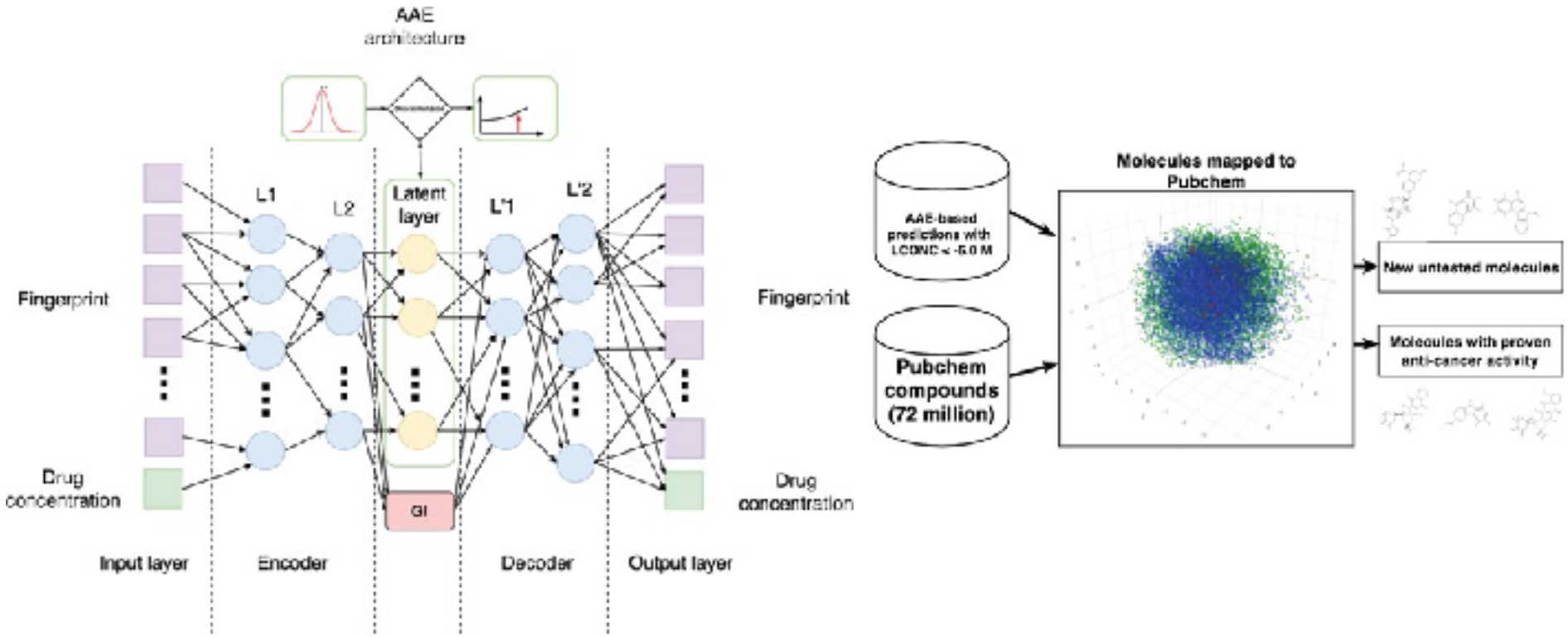
- Top Discriminator
  - Draw samples from one hot encoded labels, Cat(y)
  - Ensures created vector is categorical, softmax(z_{prev}) → "one hot"
- Bottom Discriminator
  - Same as previous
  - Constrains latent representation
- Supervised Training (disentangled?)
  - Update AAE networks with a few batches
  - Use small labeled mini-batches to update $q(y|x)$ with binary cross entropy
  - Repeat

	MNIST (100)	MNIST (1000)	MNIST (All)
NN Baseline	25.80	8.73	1.25
Adversarial Autoencoders	1.90 (±0.10)	1.60 (±0.08)	0.85 (±0.02)

Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, **Ian Goodfellow**, and Brendan Frey. "Adversarial autoencoders." *arXiv preprint arXiv:1511.05644* (2015).

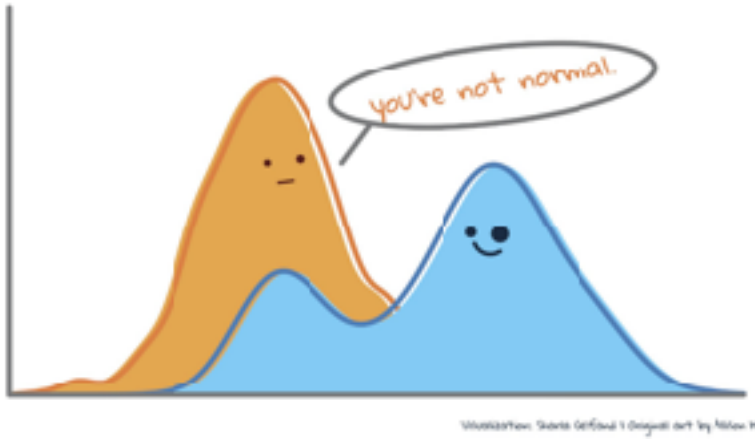
Other Uses For AAE

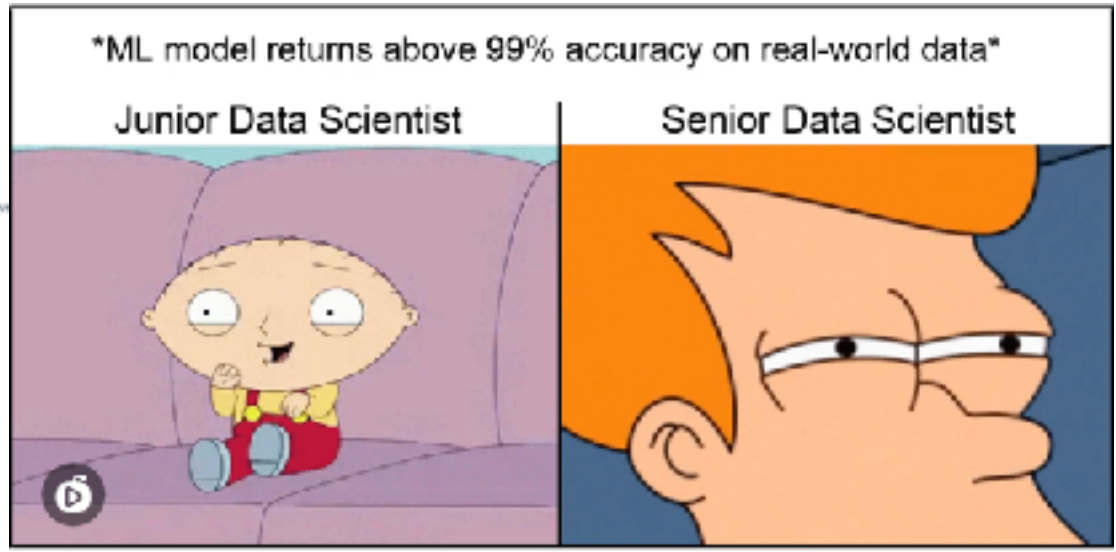
- Molecular Fingerprinting



Kadurin, Artur, Alexander Aliper, Andrey Kazennov, Polina Mamoshina, Quentin Vanhaelen, Kuzma Khrabrov, and Alex Zhavoronkov. "The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology." *Oncotarget* 8, no. 7 (2017): 10883.

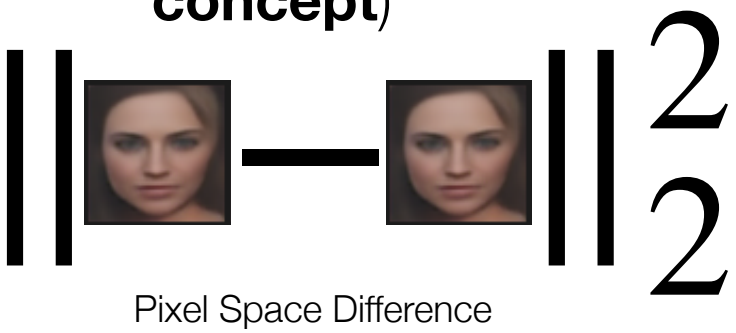
Adversarial Latent Auto-Encoders



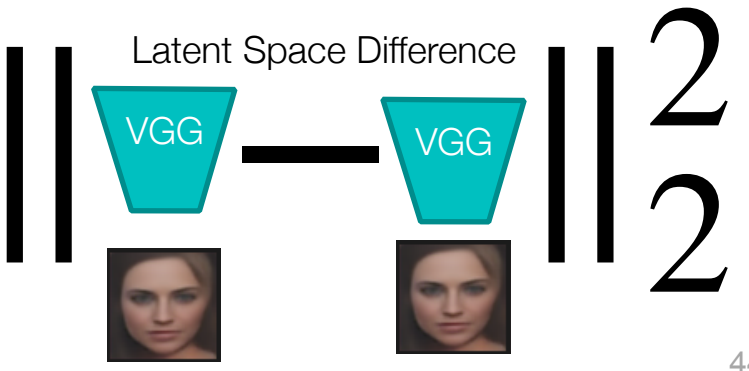


Drawbacks of AAEs

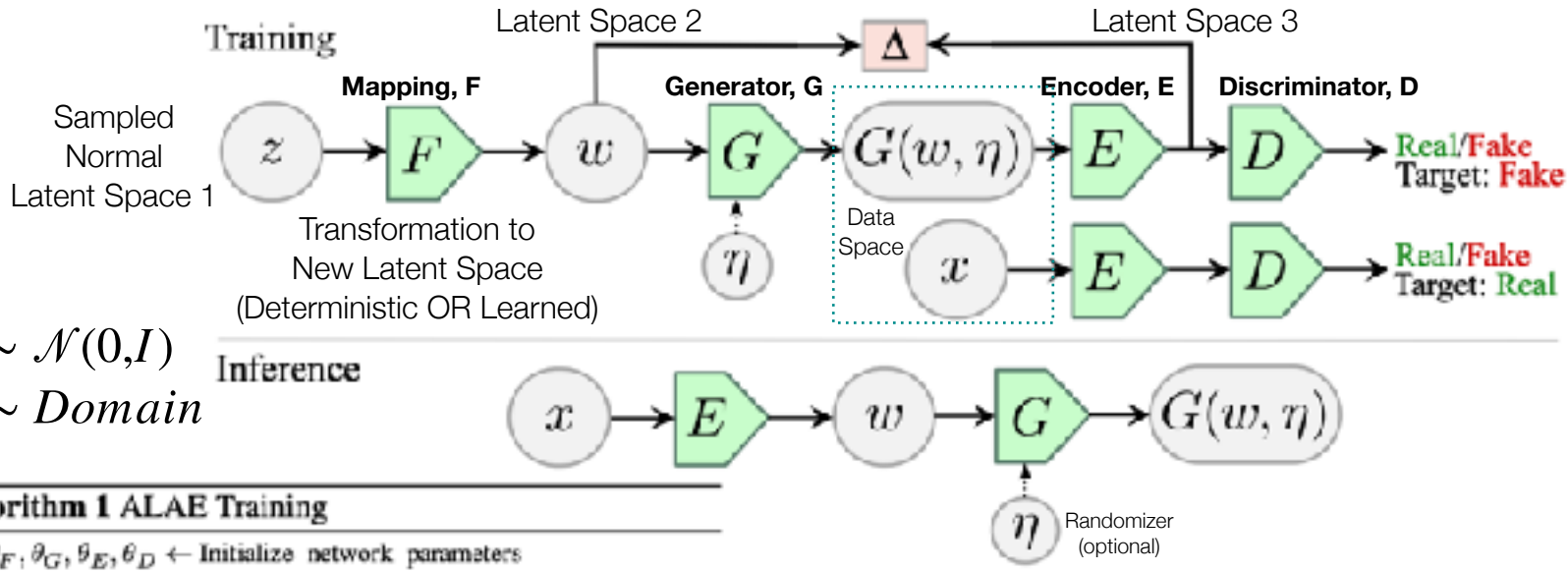
- AAEs capture representational properties of an encoder-decoder pair, with latent space following a specified structure
- Drawbacks:
  - Not entirely generative because we need examples
  - Not guaranteed to create **disentangled** representations (*VAE is better for this*…)
  - Requires that latent space distribution be selected apriori
  - Reconstruction loss calculated in data space, which may not be informative for optimizing latent space (**important concept**)



Pixel Space Difference



Adversarial Latent Auto-Encoders, ALAE



$z \sim \mathcal{N}(0,I)$

$x \sim Domain$

Algorithm 1 ALAE Training

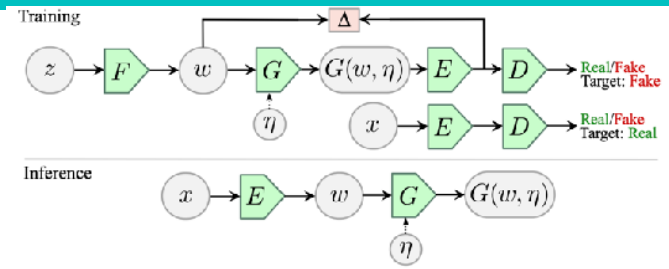
1: $\theta_F, \theta_G, \theta_E, \theta_D \leftarrow$ Initialize network parameters
2: **while** not converged **do**
3: Step I. Update E, and D
4: $x \leftarrow$ Random mini-batch from dataset
5: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
6: $L_{adv}^{E,D} \leftarrow \text{softplus}(D \circ E \circ G \circ F(z))) + \text{softplus}(-D \circ E(x)) + \frac{\gamma}{2} \mathbf{E}_{p_{\mathcal{D}}(x)} \left[\|\nabla D \circ E(x)\|^2\right]$
7: $\theta_E, \theta_D \leftarrow \text{ADAM}(\nabla_{\theta_D, \theta_E} L_{adv}^{E,D}, \theta_D, \theta_E, \alpha, \beta_1, \beta_2)$
8: Step II. Update F, and G
9: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
10: $L_{adv}^{F,G} \leftarrow \text{softplus}(-D \circ E \circ G \circ F(z)))$
11: $\theta_F, \theta_G \leftarrow \text{ADAM}(\nabla_{\theta_F, \theta_G} L_{adv}^{F,G}, \theta_F, \theta_G, \alpha, \beta_1, \beta_2)$
12: Step III. Update E, and G
13: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
14: $L_{error}^{E,G} \leftarrow \|F(z) - E \circ G \circ F(z)\|_2^2$
15: $\theta_E, \theta_G \leftarrow \text{ADAM}(\nabla_{\theta_E, \theta_G} L_{error}^{E,G}, \theta_E, \theta_G, \alpha, \beta_1, \beta_2)$
16: **end while**

- Train Mapping/Generator to fool discriminator
- Train Encoder/Discriminator to find fake
- Minimize latent space encoder & mapper
- Reconstruction loss: x and G(E(x)), *never actually calculated*, just inferred
- Therefore, encoder and mapper are adversaries

Pidhorskyi, Stanislav, Donald A. Adjeroh, and Gianfranco Doretto. "Adversarial latent autoencoders." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14104-14113. 2020.

Adversarial Latent Auto-Encoders, ALAE

$$\sigma(x) = softplus = \log(1 + \exp(x))$$ smoothed ReLU $[0,\infty)$



E,D: Detect fake samples
minimize D for fake samples

E,D: Detect real samples
min -D (max D) for real samples

$$6 : \mathscr{L}^{E,D}_{disc} = \sigma(D \circ E \circ G \circ F(z)) + \sigma(-D \circ E(x))$$

$$+\frac{\gamma}{2} \cdot \mathbf{E}[\|\nabla D \circ E(x)\|]$$

F,G: Try to fool discriminator,
min -D (max D) for fake samples

E,D: Gradient Penalty
Keep Gradient Magnitude Small
Keep in mind for later

Algorithm 1 ALAE Training

$\theta_F, \theta_G, \theta_E, \theta_D \leftarrow$ Initialize network parameters
while not converged **do**
  Step I. Update E, and D
  $x \leftarrow$ Random mini-batch from dataset
  $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
  $L^{E,D}_{adv} \leftarrow \text{softplus}(D \circ E \circ G \circ F(z))) + \text{softplus}(-D \circ E(x)) + \frac{\gamma}{2} \mathrm{E}_{p_{\mathcal{D}}(x)} [\|\nabla D \circ E(x)\|^2]$
  $\theta_E, \theta_D \leftarrow \text{Adam}(\nabla_{\theta_D, \theta_E} L^{E,D}_{adv}, \theta_D, \theta_E, \alpha, \beta_1, \beta_2)$
  Step II. Update F, and G
  $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
  $L^{F,G}_{adv} \leftarrow \text{softplus}(-D \circ E \circ G \circ F(z)))$
  $\theta_F, \theta_G \leftarrow \text{Adam}(\nabla_{\theta_F, \theta_G} L^{F,G}_{adv}, \theta_F, \theta_G, \alpha, \beta_1, \beta_2)$
  Step III. Update E, and G
  $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
  $L^{E,G}_{error} \leftarrow \|F(z) - E \circ G \circ F(z)\|^2_2$
  $\theta_E, \theta_G \leftarrow \text{Adam}(\nabla_{\theta_E, \theta_G} L^{E,G}_{error}, \theta_E, \theta_G, \alpha, \beta_1, \beta_2)$
end while

$$10 : \mathscr{L}^{F,G}_{gen} = \sigma(-D \circ E \circ G \circ F(z))$$

$$14 : \mathscr{L}^{E,G}_{latent} = \|F(z) - E \circ G \circ F(z)\|^2$$

E,G: Keep latent spaces similar

6: 10: Based On Wasserstein Distance, … which is really helpful…

Pidhorskyi, Stanislav, Donald A. Adjeroh, and Gianfranco Doretto. "Adversarial latent autoencoders." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14104-14113. 2020.

Do we really need to learn this?

Reconstructions

Generated Images (Fake)

Pidhorskyi, Stanislav, Donald A. Adjeroh, and Gianfranco Doretto. "Adversarial latent autoencoders." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14104-14113. 2020.

Yep, We need to study GANs

- The ALAE used some notation and processes that we need to study in order to understand:
  - why these are advantageous
  - how to do them properly
  - the tradeoffs and computational cost
- Therefore, the remaining topics:
  - Nash Equilibrium (Vanilla GAN)
  - GAN Training Tricks (condensed)
  - PyTorch (skipping)
  - Least Squares GAN (skipping)
  - Wasserstein GAN
  - BigGAN, StyleGAN
  - Stable Diffusion

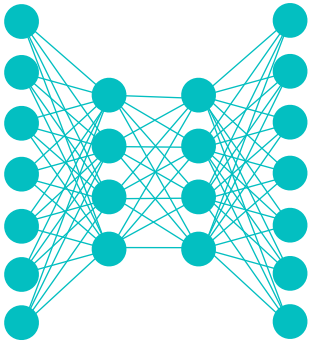
Lecture Notes for

Neural Networks and Machine Learning

ALAEs

Next Time:
GANs

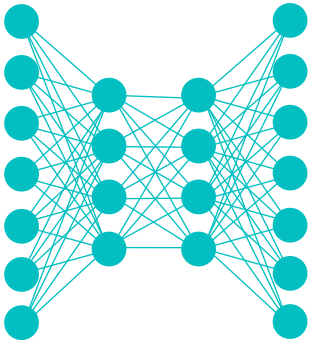
Reading: Chollet CH8



Lecture Notes for

Neural Networks and Machine Learning

Simple GAN and DCGAN



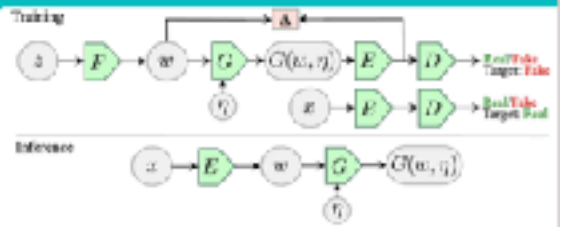
Logistics and Agenda

- Logistics
  - Student Presentations Today
  - Grading…
- Agenda
  - Review from Last Time
  - **Student Presentation**: Representational Learning with Deep Convolutional GANs, Radford
  - GAN Ticks and GAN Demo
  - Practical GANs
  - LS-GAN (Next Time)
  - Wasserstein GAN (Next time)
  - WGAN-GP (Next Time)
  - Big GAN (Next Next Time)

Last Time

Adversarial Latent Auto-Encoders, ALAE

$\sigma(x) = softplus = \log(1 + \exp(x))$ smoothed ReLU [0,∞)



E,D: Detect fake samples
minimize D for fake samples

E,D: Detect real samples
min -D (max D) for real samples

$$6: \mathcal{L}^{E,D}_{disc} = \sigma(D \circ E \circ G \circ F(z)) + \sigma(-D \circ E(x)) + \frac{\gamma}{2} \cdot \mathbf{E}[\|\nabla D \circ E(x)\|]$$

F,G: Try to fool discriminator,
min -D (max D) for fake samples

E,D: Gradient Penalty
Keep Gradient Magnitude Small
Keep in mind for later

$$10: \mathcal{L}^{F,G}_{gen} = \sigma(-D \circ E \circ G \circ F(z))$$

$$14: \mathcal{L}^{E,G}_{latent} = \|F(z) - E \circ G \circ F(z)\|^2$$

E,G: Keep latent spaces similar

Algorithm 1 ALAE Training

1: $\theta_F, \theta_G, \theta_E, \theta_D \leftarrow$ Initialize network parameters
2: **while** not converged **do**
3: Step I. Update E, and D
4: $x \leftarrow$ Random mini-batch from dataset
5: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
6: $L^{E,D}_{adv} \leftarrow \text{softplus}(D \circ E \circ G \circ F(z))) + \text{softplus}(-D \circ E(x)) + \frac{\gamma}{2} \mathbf{E}_{p_L(x)}[\|\nabla D \circ E(x)\|^2]$
7: $\theta_E, \theta_D \leftarrow \text{ADAM}(\nabla_{\theta_E, \theta_D} L^{E,D}_{adv}, \theta_D, \theta_E, \alpha, \beta_1, \beta_2)$
8: Step II. Update F, and G
9: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
10: $L^{F,G}_{adv} \leftarrow \text{softplus}(-D \circ E \circ G \circ F(z)))$
11: $\theta_F, \theta_G \leftarrow \text{ADAM}(\nabla_{\theta_F, \theta_G} L^{F,G}_{adv}, \theta_F, \theta_G, \alpha, \beta_1, \beta_2)$
12: Step III. Update E, and G
13: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
14: $L^{E,G}_{error} \leftarrow \|F(z) - E \circ G \circ F(z)\|^2_2$
15: $\theta_E, \theta_G \leftarrow \text{ADAM}(\nabla_{\theta_E, \theta_G} L^{E,G}_{error}, \theta_E, \theta_G, \alpha, \beta_1, \beta_2)$
16: **end while**

6: 10: Based On Wasserstein Distance, … which is really helpful…

Pidhorskyi, Stanislav, Donald A. Adjeroh, and Gianfranco Doretto. "Adversarial latent autoencoders." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14104-14113. 2020.

46

The Simple GAN

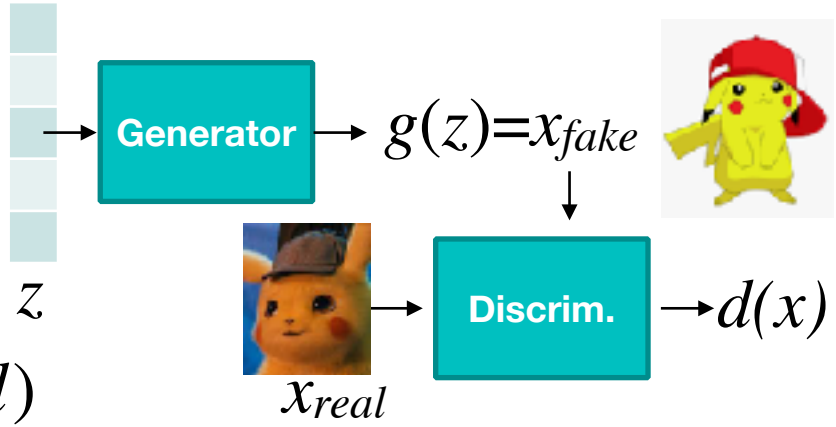
Every time Ian Goodfellow has a tutorial, It should be called:

"GAN-splaining with Ian"

—Geoffrey Hinton, Probably

Generative Adversarial Network

- Generator: $x = g(z)$
- Discriminator: $\{0,1\} = d(x)$
- Mini-max, turn-based game:



$$\hat{w} = \arg\min_{g} \max_{d} v(g,d)$$

- Nice differentiable choice for v (zero sum game):

$$v(g,d) = \mathbf{E}\left[\log d(x)\right] + \mathbf{E}\left[\log(1 - d \circ g(z))\right]$$

only portion dependent on g generator minimizes

everything dependent on d discriminator maximizes

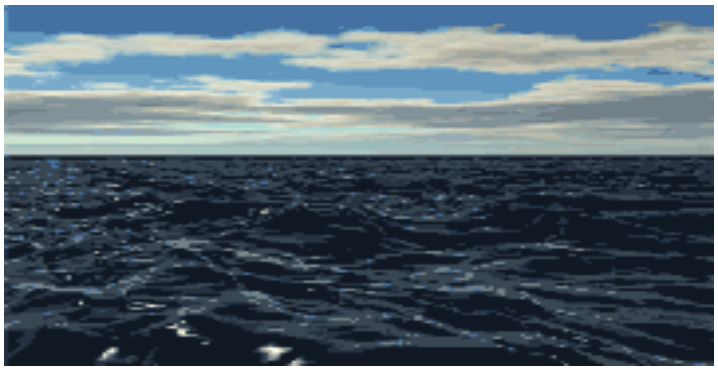
Nash Equilibrium:
Each player wins and loses repeatedly, thus its a zero sum game when they tradeoff performances

Taking turns: Nash Equilibrium

$$v(g,d) = \mathbf{E}\left[\log d(x)\right] + \mathbf{E}\left[\log(1 - d \circ g(z))\right]$$

$$\hat{w} = \arg\min_{g}\max_{d} v(g,d)$$

- If only the problem was convex! This would be easy to solve…
- But $v(g,d)$ is not convex
  - Saddle points everywhere
  - Like optimizing in an Ocean



- Taking turns on the gradient descent may never converge (Nash equilibrium is not convergence)
  - **So we have no convergence guarantee**
  - But practically we like when discriminator == chance

Practical Objectives

- Discriminator objective, gradient **ascent**:

$$\max_{d} \left(\mathscr{L}^{D}_{disc} = \mathbf{E}\left[\log d(x)\right] + \mathbf{E}\left[\log(1 - d \circ g(z))\right] \right)$$

- Generator objective, gradient **descent**:

$$\min_{g} \left(\mathscr{L}^{G}_{gen} = \mathbf{E}\left[\log(1 - d \circ g(z))\right] \right)$$

- But **practically** generator is really hard to train, amplified by a decreasing gradient
- Ian Goodfellow tried to solve this mathematically through a number of different formulations
  - Nothing seemed to work, and then…

Practical Objectives

- Original Generator objective, gradient descent:

$$\min_g \left(\mathscr{L}^G_{gen} = \mathbf{E}\left[\log(1 - d \circ g(z))\right] \right)$$

- Goodfellow *et al.* gave up on using rigorous mathematics (intractable) and relied on heuristic:
  - Instead of minimizing when discriminator is right
  - Why not maximize when it is wrong?
    - not trying to win, just make the other person lose

$$\max_g \left(\mathscr{L}^G_{gen,new} = \mathbf{E}\left[\log(d \circ g(z))\right] \right)$$

No longer a fair zero sum game?!

Now we are not guaranteed convergence or equilibrium when training the GAN…
But it trains…

Final Loss Functions

- Discriminator:

Do not Update Generator Weights

$$\max_{d} \left(\mathscr{L}^{D}_{disc} = \mathbf{E}\left[\log d(x)\right] + \mathbf{E}\left[\log(1 - d \circ g(z))\right] \right)$$

Same as minimizing the binary cross entropy of the model with: (1) real data=1 and (2) generated data=0!

- Generator:

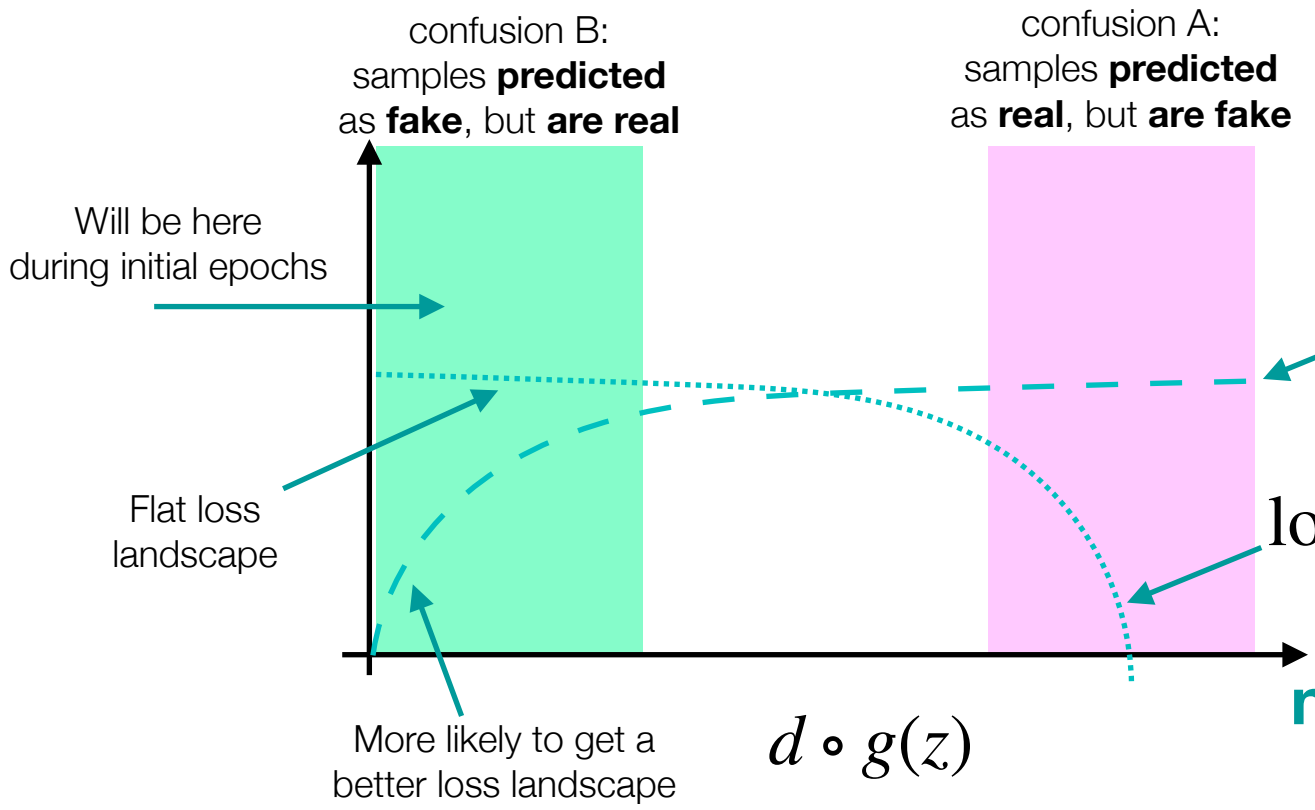
$$\max_{g} \left(\mathscr{L}^{G}_{gen} = \mathbf{E}\left[\log(d \circ g(z))\right] \right)$$

Do Not Update Discriminator Weights

Same as minimizing the binary cross entropy of "mislabeled" generated data=1!

Intuition for why this really helps…

- Training two networks is inherently unstable, need heuristic
- Let's assume generator is not any good for initial epochs!



This is a heuristic not a mathematical proof

DC-GAN

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

Alec Radford & Luke Metz
indico Research
Boston, MA
[alec,luke]@indico.io

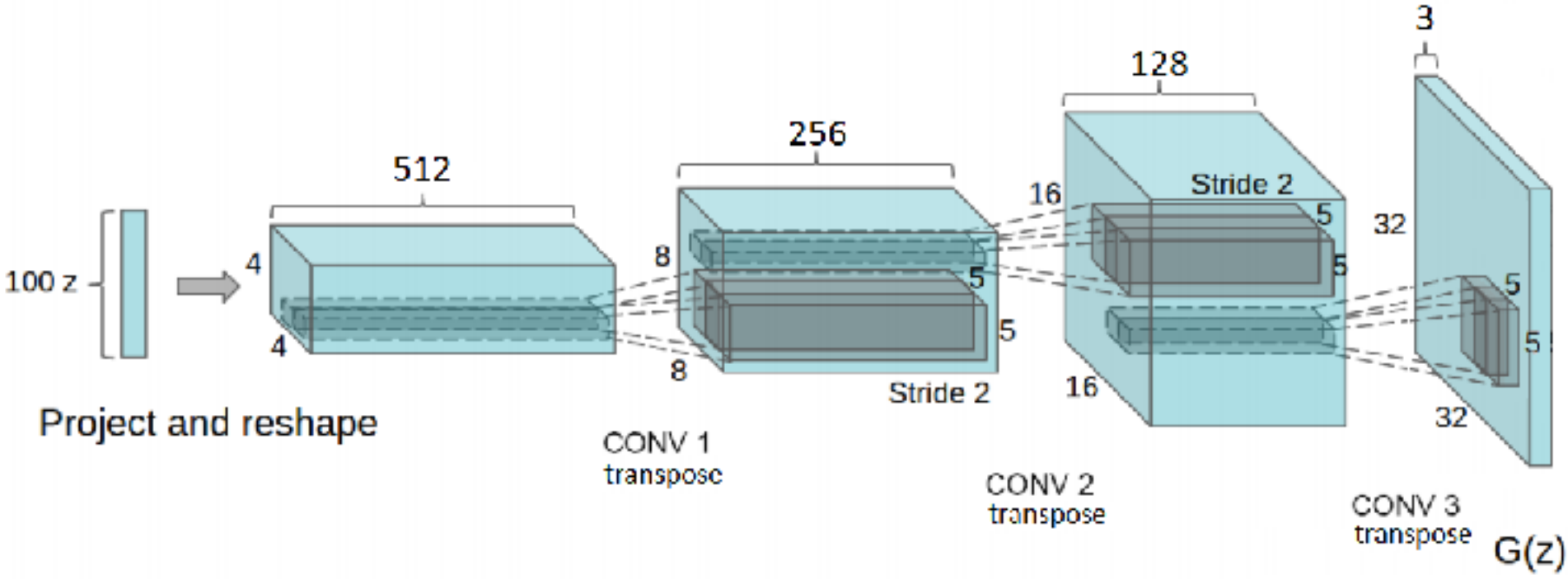
Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

Abstract

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Deep Convolutional GANs

- Just need to reshape and use upsampling



https://sthalles.github.io/intro-to-gans/

Some Results of Generation (not conv)

Goodfellow et al. "Generative Adversarial Networks" (NeurIPS 2014)

Highlight: Nearest Training Sample

So

Radford, Ale ersarial
networks." *a*

A bag of tricks from F. Chollet

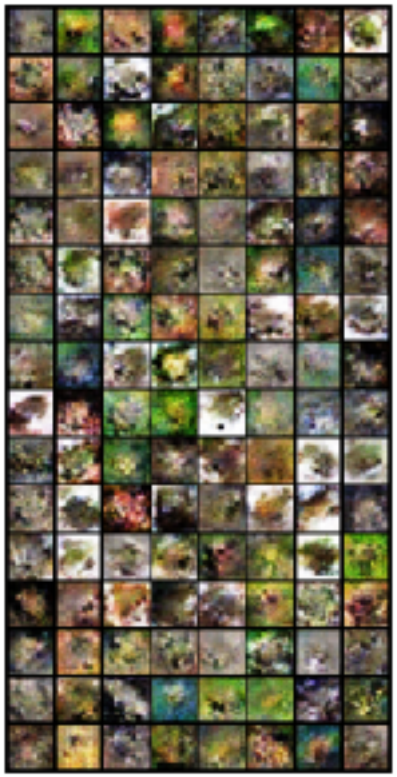
The process of training GANs and tuning GAN implementations is notoriously difficult. There are a number of known tricks you should keep in mind. Like most things in deep learning, it's **more alchemy than science**: **these tricks are heuristics, not theory-backed guidelines**. They're supported by a level of intuitive understanding of the phenomenon at hand, and they're known to work well empirically, although not necessarily in every context.

—Francois Chollet, DL with Python

- Use tanh for generator output, not a sigmoid
  - *We typically normalize inputs to a NN to be from [-1, 1], generator needs to mirror this squashing*
- Sample from Normal Distribution
  - *Everyone else is doing it (practically whatever we do here should help to create a latent space easy to sample from)*
- Random is more robust in optimizer and labels
  - *GANs get stuck a lot, label noise can help move optimization*
- Sparse gradients are **not** your friend here (discriminator should be more transparent, so generator can backprop through it)
  - *No max pooling, no ReLU* 😳
- Make decoder upsampling multiple of stride…
  - *Unequal pixel coverage (checkerboards)*

GANs Demo

Master Repository:
`07c GANsWithKeras.ipynb`



GANs in Keras (optional)

Implementation from Book on
"Frogs from CIFAR"

Demo by Francois Chollet

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.5-introduction-to-gans.ipynb

GANs Loss

Meta-meta-learning for Neural Architecture Search through arXiv Descent

Antreas Antoniou
MetaMind
aa@mm.ai

Nick Pawlowski
Google x^2
nick@x.x

Jack Turner
slow.ai
jack@slow.ai

James Owers
Facebook AI Research Team
jim@fart.org

Joseph Mellor
Institute of Yellow Jumpers
joe@anditwasall.yellow

Elliot J. Crowley
ClosedAI
elliot@closed.ai

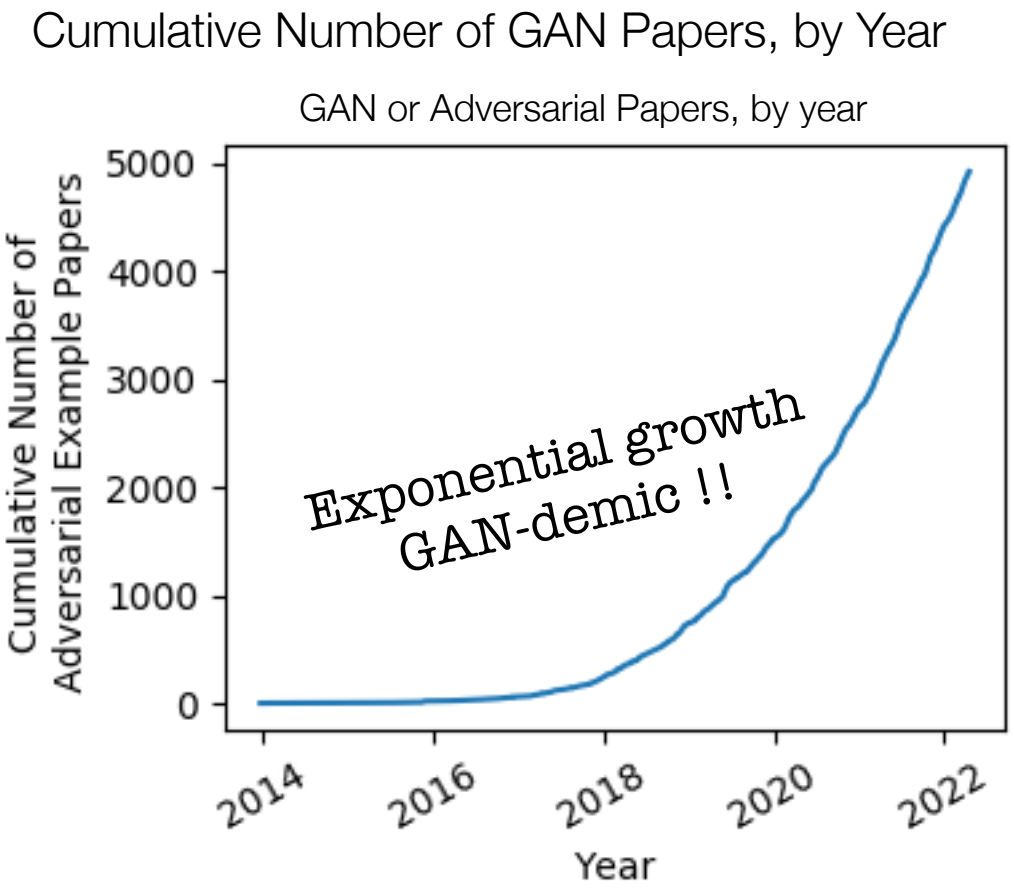
Abstract

Recent work in meta-learning has set the deep learning community alight. From minute gains on few-shot learning tasks, to discovering architectures that are slightly better than chance, to solving intelligence itself[1], meta-learning is proving a popular solution to every conceivable problem ever conceivably conceived ever.

In this paper we venture deeper into the computational insanity that is meta-learning, and potentially risk exiting the simulation of reality itself, by attempting to meta-learn at a third learning level. We showcase the resulting approach—which we call *meta-meta-learning*—for neural architecture search. Crucially, instead of *meta-learning* a neural architecture differentiably as in DARTS (Liu et al., 2018) we *meta-meta-learn* an architecture by searching through arXiv. This *arXiv descent* is GPU-free and only requires a handful of graduate students. Further, we introduce a regulariser, called *college-dropout*, which works by randomly removing a single graduate student from our system. As a consequence, procrastination levels decrease significantly, due to the increased workload and sense of responsibility each student attains.

The code for our experiments is publicly available at [redacted].
Edit: we have decided not to release our code as we are concerned that it may be used for malicious purposes.

There were a bunch go GANs proposed

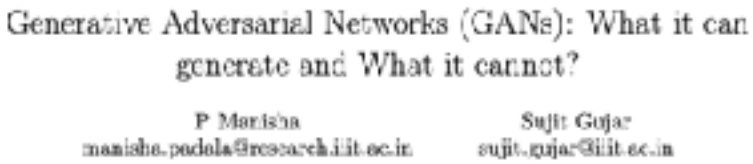


https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html

ABC-GAN — ABC-GAN: Adap
ABC-GAN — GANs for LIFE:
AC-GAN — Conditional Image
acGAN — Face Aging With C
ACGAN — Coverless Informa
acGAN — On-line Adaptative
ACtuAL — ACtuAL: Actor-Criti
AdaGAN — AdaGAN: Boostin
Adaptive GAN — Customizing
AdvEntuRe — AdvEntuRe: Ad
AdvGAN — Generating adver
AE-GAN — AE-GAN: adversa
AE-OT — Latent Space Optim
AEGAN — Learning Inverse M
AF-DCGAN — AF-DCGAN: A
AffGAN — Amortised MAP Inf
AIM — Generating Informative
AL-CGAN — Learning to Gen
ALI — Adversarially Learned I
AlignGAN — AlignGAN: Learn
AlphaGAN — AlphaGAN: Gen
AM-GAN — Activation Maximi
AmbientGAN — AmbientGAN
AMC-GAN — Video Predictio
AnoGAN — Unsupervised An
APD — Adversarial Distillation
APE-GAN — APE-GAN: Adve
ARAE — Adversarially Regula
ARDA — Adversarial Represe
ARIGAN — ARIGAN: Syntheti
ArtGAN — ArtGAN: Artwork S
ASDL-GAN — Automatic Steg
ATA-GAN — Attention-Aware
Attention-GAN — Attention-G
AttGAN — Arbitrary Facial Att
AttnGAN — AttnGAN: Fine-Gr
AVID — AVID: Adversarial Vis

Why are GANs so difficult to train?

- Why did we need to add noise to labels?
- Why were sparse gradients needed?
- Does using least squares really solve the problem?
- Formalization:
  - GANs will not converge
  - GAN outputs all might be similar (**mode collapse**)
  - Slow training: gradient vanishes, sometimes not recoverable


Generative Adversarial Networks (GANs): What it can generate and What it cannot?

P Manisha
manisha.padala@research.iiit.ac.in

Sujit Gujar
sujit.gujar@iiit.ac.in


TOWARDS PRINCIPLED METHODS FOR TRAINING GENERATIVE ADVERSARIAL NETWORKS

Martin Arjovsky
Courant Institute of Mathematical Sciences
martinarjovsky@gmail.com

Léon Bottou
Facebook AI Research
leonb@fb.com

The Optimal Discriminator

- Discriminator is maximizing:

$$\max_d \mathbf{E}\left[\log d(x)\right] + \mathbf{E}\left[\log(1 - d \circ g(z))\right]$$

… math …

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

-**Conclusion**: Discriminator is optimal when this condition occurs

In principle, it would seem like **training the discriminator fully** for each update of the generator. Then the generator could simply find a good update to itself based on the optimal discriminator…
…but that is not what we observe. Subsequent updates to the generator (as discriminator is better) do not seem to keep up…

As the discriminator gets more and more confident (i.e., more optimal), it pushes the real distribution of the data away from the generator latent space… gradients vanish

What this means for the Generator

- It can be shown that the generator objective function (according to the Goodfellow loss) is optimal when it maximizes this:

$$C(G) = -\log(4) + KL\left(p_{data} \parallel \frac{p_{data} + p_g}{2}\right) + KL\left(p_g \parallel \frac{p_{data} + p_g}{2}\right)$$

$$C(G) = -\log(4) + 2.JSD(p_{data} \parallel p_g)$$ Jenson-Shannon Divergence

- Which helps explain the **Vanishing gradients**: if $p_{data} \| p_g \approx 0$ then JSD saturates and the gradient is basically zero. Optimization has no idea what direction to move in…
  - **Indirect Solution:** Perhaps we can just use tips/tricks to get around the vanishing gradient problem?
    - Not covering these methods this semester …
  - **Direct Solution**: get rid of the loss function from Goodfellow
    - **Use a better objective: Wasserstein Distance**

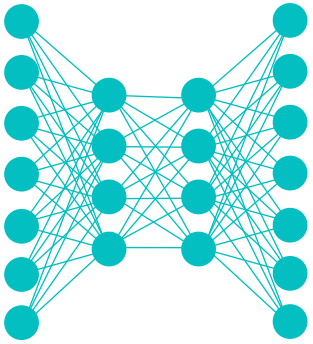
Lecture Notes for

Neural Networks and Machine Learning

Simple GANs

Next Time:
Wasserstein GANs

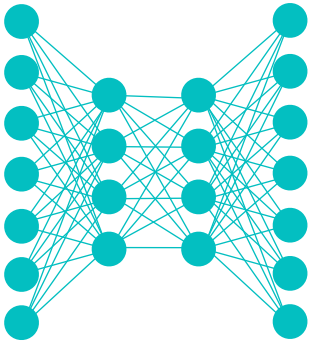
Reading: WGAN Paper



Lecture Notes for

Neural Networks and Machine Learning

Wasserstein GANs



Logistics and Agenda

- Logistics
  - **Student Presentation**: None
- Agenda
  - Wasserstein GAN
  - WGAN-GP
  - Demo
  - Town Hall

Last Time:

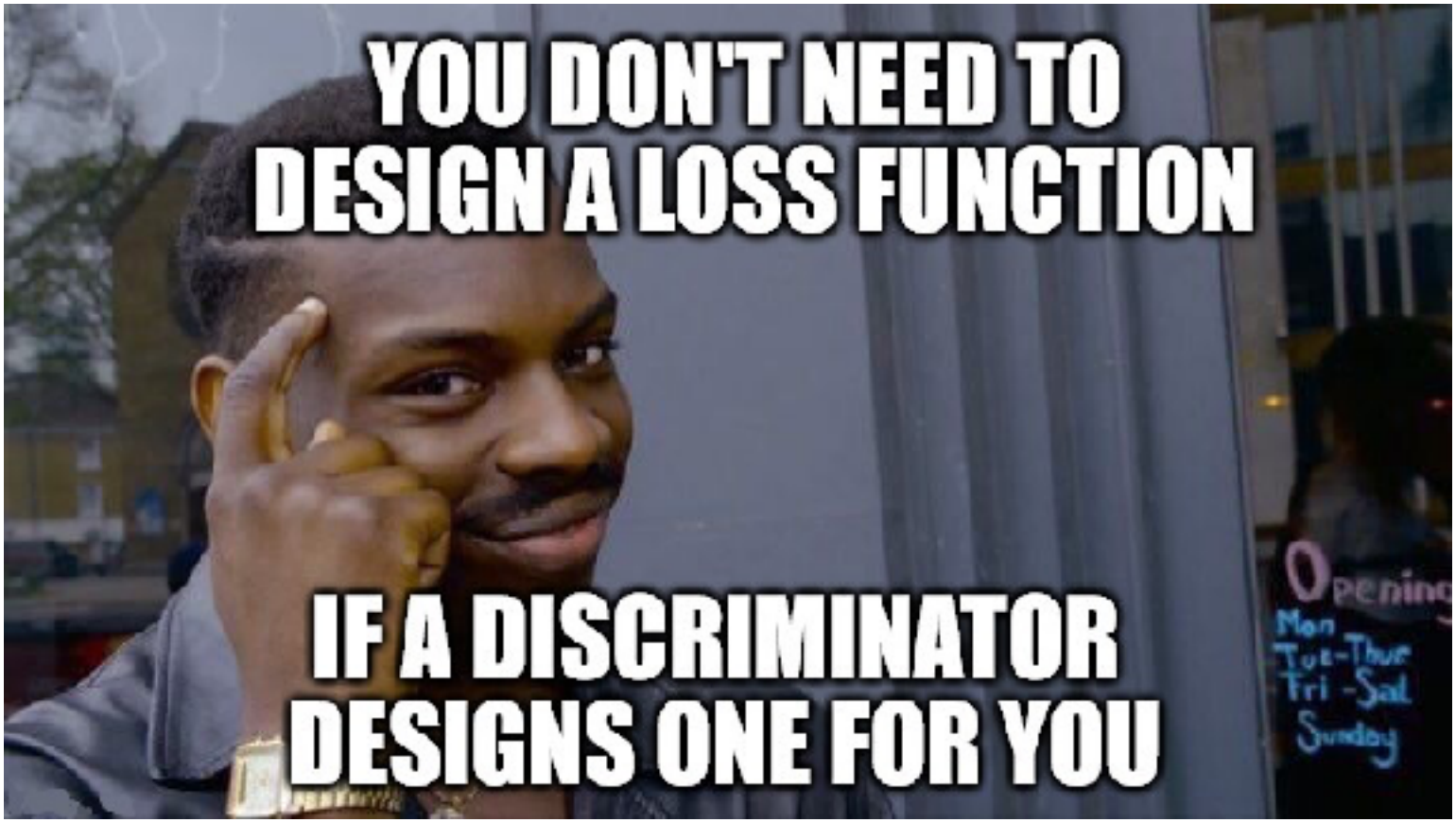
- It can be shown that the generator objective function (according to the Goodfellow loss) is optimal when it maximizes this:

$$C(G) = -\log(4) + KL\left(p_{data} \parallel \frac{p_{data} + p_g}{2}\right) + KL\left(p_g \parallel \frac{p_{data} + p_g}{2}\right)$$

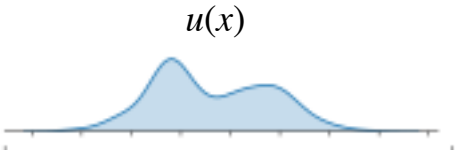
$$C(G) = -\log(4) + 2.JSD(p_{data} \parallel p_g)$$ Jenson-Shannon Divergence

- Which helps explain the **Vanishing gradients**: if $p_{data} \| p_g \approx 0$ then JSD saturates and the gradient is basically zero. Optimization has no idea what direction to move in…
  - **Indirect Solution:** Perhaps we can just use tips/tricks to get around the vanishing gradient problem?
    - Not covering these methods this semester …
  - **Direct Solution**: get rid of the loss function from Goodfellow
    - **Use a better objective: Wasserstein Distance**

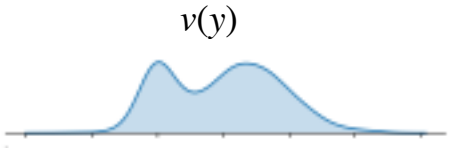
Wasserstein GANs

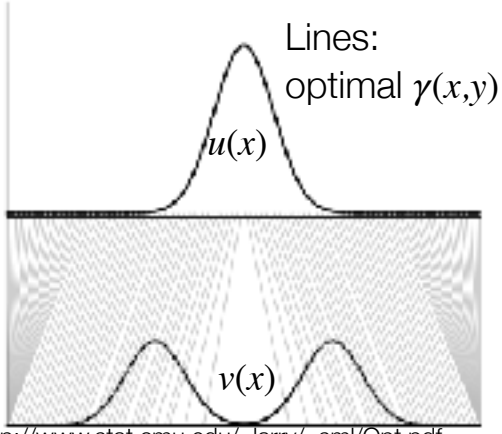


Wasserstein Distance (Earth Mover)



how to move dirt from u to v ?

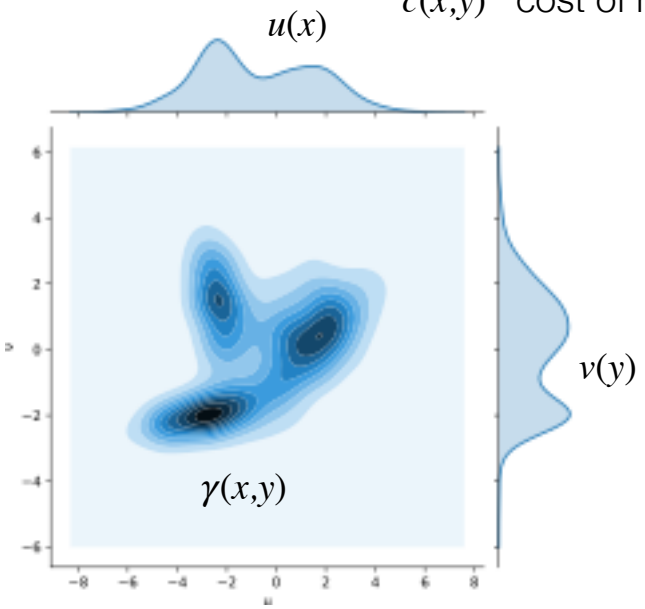




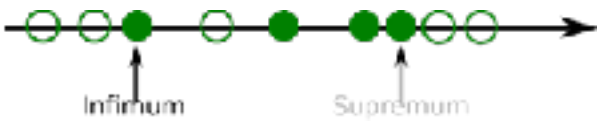
http://www.stat.cmu.edu/~larry/=sml/Opt.pdf

$\gamma(x,y)$ one plan to move u to v

$c(x,y)$ cost of moving u to v



$$\int\int c(x,y)\cdot\gamma(x,y)\,\partial x\partial y$$ Total cost of plan γ



$$\inf_{\gamma\in\Pi}\int\int c(x,y)\cdot\gamma(x,y)\,\partial x\partial y$$

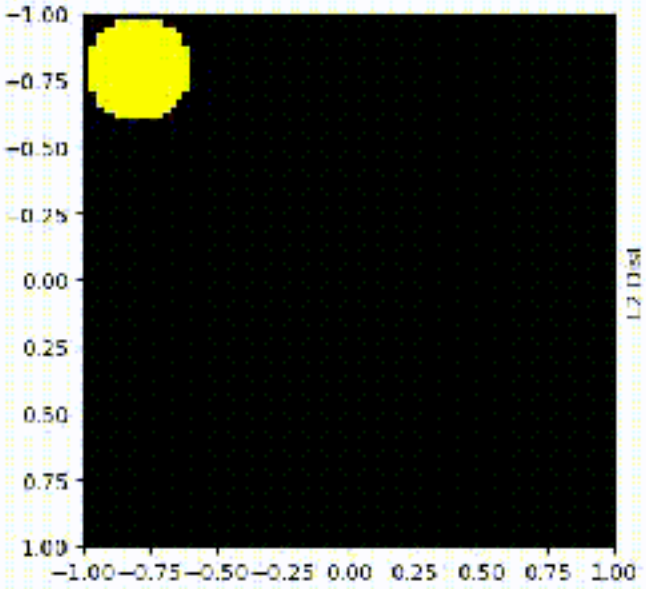
Optimal plan
has greatest lower bound
(minimum cost in set Π)

$$\inf \mathbf{E}_{x,y\leftarrow\gamma}\left[|x-y|\right] = \inf_{\gamma\in\Pi}\int\int |x-y|\cdot\gamma(x,y)\,\partial x\partial y$$

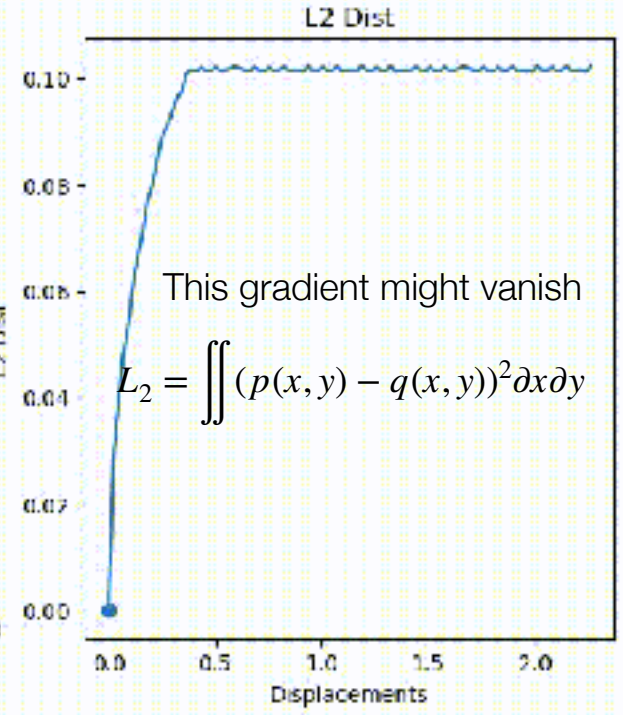
If cost is difference
to move from x to y

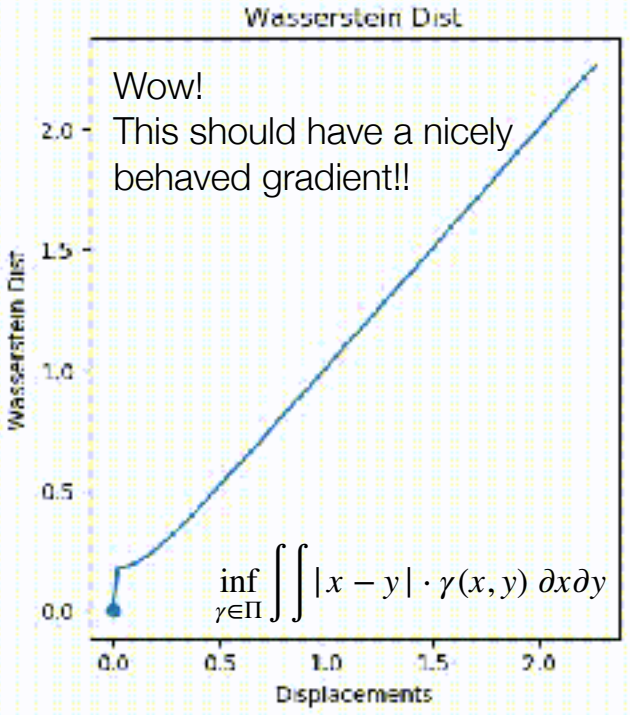
image: wikipedia

Wasserstein Distance



Two 2D Normal Distributions





- *Wasserstein GAN adds few tricks to allow the Discriminator to approximate Wasserstein (aka Earth Mover's) distance between real and model distributions. Wasserstein distance roughly tells "how much work is needed to be done for one distribution to be adjusted to match another" and is remarkable in a way that it is defined even for non-overlapping distributions.*

https://gist.github.com/ctralie/66352ae6ab06c009f02c705385a446f3

https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

Wasserstein Distance

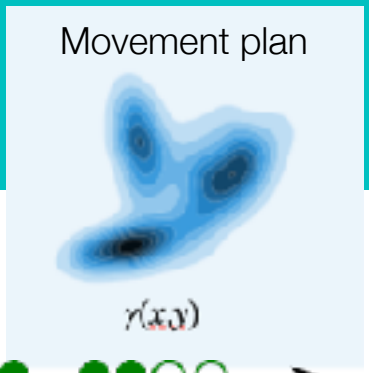
- How much do I need to change one distribution to get it to match another?
- Also, we will only have **samples** from the distributions (not the actual equations)
- **Wasserstein Distance** for continuous probabilities:

$$W(p_{data}, p_g) = \inf_{\gamma \in \prod(p_{data}, p_g)} \mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| \right]$$

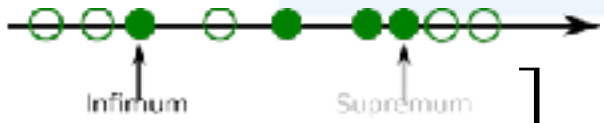
- $\inf$ is greatest lower bound (min of a set)
- γ is completely and utterly

intractable to **compute**

Wasserstein Duality, Critic



$$W(p_{data}, p_g) = \inf_{\gamma \in \prod(p_{data}, p_g)} \mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| \right]$$



$$= \inf_{\gamma} \mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| + \underbrace{\sup_{f} \mathbf{E}_{x \leftarrow p_{data}}[f(x)] - \mathbf{E}_{x \leftarrow p_g}[f(x)] - \big(f(x) - f(y)\big)}_{\text{enforce constraint = finite, if } \gamma \in \Pi, \text{ else } +\infty} \right]$$

$$= \inf_{\gamma} \sup_{f} \mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| + \mathbf{E}_{x \leftarrow p_{data}}[f(x)] - \mathbf{E}_{x \leftarrow p_g}[f(x)] - \big(f(x) - f(y)\big) \right]$$

$$= \sup_{f} \mathbf{E}_{x \leftarrow p_{data}}[f(x)] - \mathbf{E}_{x \leftarrow p_g}[f(x)] - \inf_{\gamma} \underbrace{\mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| + \big(f(x) - f(y)\big) \right]}_{\text{new constraint = finite, if } |f| \leq 1, \text{ else } -\infty}$$

$$= \sup_{|f| \leq 1} \mathbf{E}_{x \leftarrow p_{data}}[f(x)] - \mathbf{E}_{x \leftarrow p_g}[f(x)]$$

what have we done here????

read this: https://vincentherrmann.github.io/blog/wasserstein/

Wasserstein (Kantorovich-Rubinstein) Duality

- 1-Lipschitz constraint, critic: $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$
- Wasserstein Duality Formula (approx. for samples):

$$\mathscr{L}_{wass} = \sup_{|f| \leq 1} \mathbf{E}[f(x_{real})] - \mathbf{E}[f(x_{fake})] \approx \frac{1}{m}\sum_{i=1}^{m} f(x_{real}^{(i)}) - \frac{1}{n}\sum_{j=1}^{n} f(g(z^{(j)}))$$

- where sup is least upper bound (max within set $|f|<1$, same as valid movement plans set)
- f will be the critic, learned as a neural network, $m=n$

$$\mathscr{L}_{wass}^{D} = \frac{1}{m}\sum_{i=1}^{m} f(x_{real}^{(i)}) - f(g(z^{(i)}))$$

Max with f (called critic), freeze generator weights

$$\mathscr{L}_{wass}^{G} = \frac{1}{m}\sum_{i=1}^{m} f(g(z^{(i)}))$$

Max with g, freeze weights of critic

Sometimes we use this, minimizes hinge loss:

$$\frac{1}{m}\sum_{i=1}^{m} f(x_{real}^{(i)}) \cdot f(g(z^{(i)}))$$

when *values* are -1 to 1

f( )	*f*(*g*( ))	*f*-*f*(*g*)	*f* x *f*(*g*)
-1	-1	0	1
-1	1	-2	-1
1	-1	2	-1
1	1	0	1
		max	min

https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

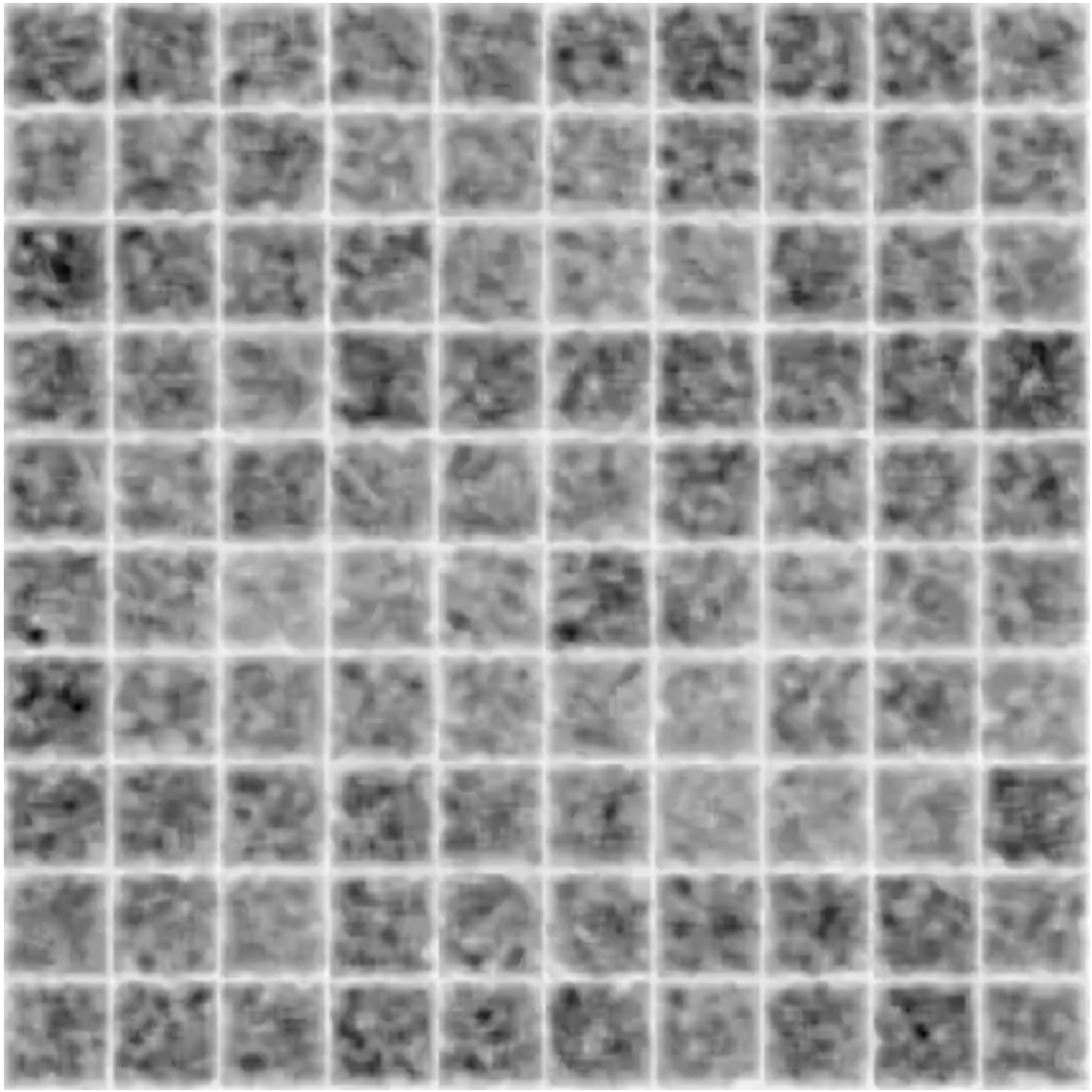
Practical Wasserstein Loss

- Discriminator (now critic) becomes **f**, and its output can be Lipschitz if all weights are small (squashes inputs)
  - so we clip them to (-c to c) (where $c \sim 0.01$)
  - critic output should be linear

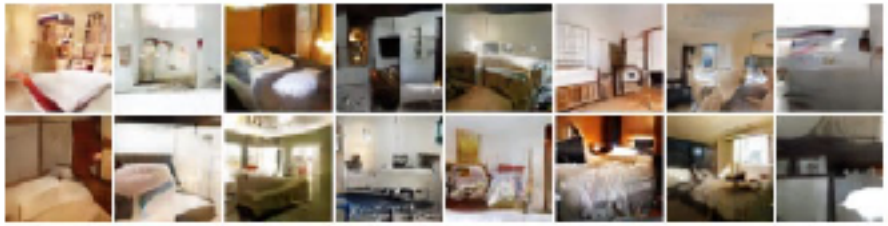
```
# define the constraint
const = ClipConstraint(0.01)
...
# use the constraint in a layer
model.add(Conv2D(..., kernel_constraint=const))
```

https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/

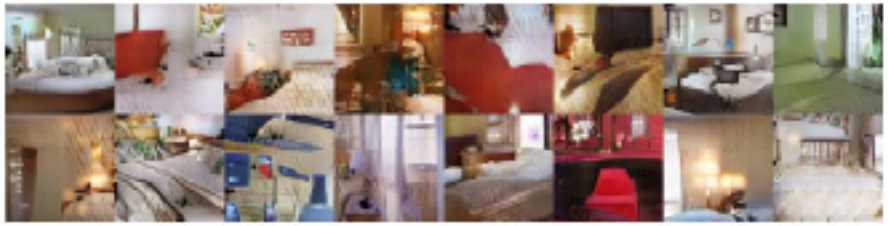
WGAN Example from Keras (MNIST)



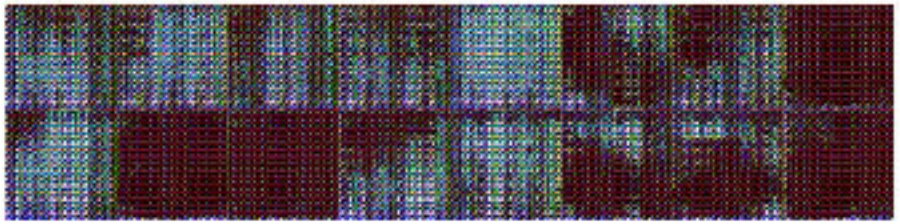
- In their empirical findings, no mode collapse problems
- And training longer resulted in good quality images (motivates that distributions overlap)
- Still some of the most competitive GAN results



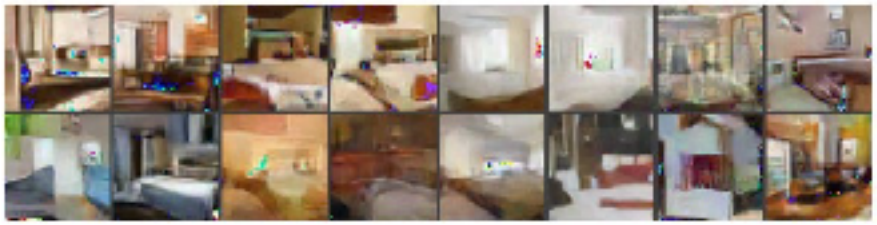
WGAN with DCGAN generator

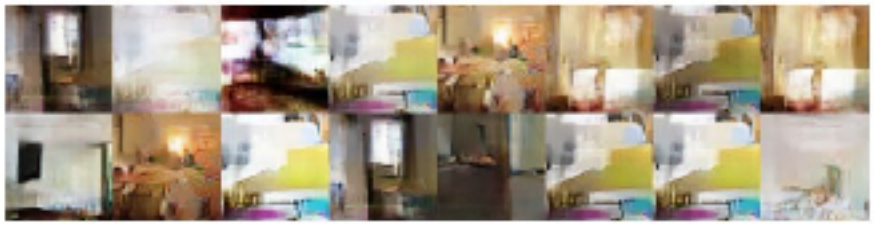


GAN with DCGAN generator



Without batch normalization &
constant number of filters at each layer





Using a MLP as the generator

So we should always use Wasserstein!

- **Actually not really**
- Its really, really,…. really slow
- Clipping (from WGAN paper):

Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs).

- Others have made improvements by incorporating gradient constraints
- New critic: WGAN with gradient penalty

WGAN-GP

WGAN-GP (Gradient Penalty)

$$\frac{1}{m}\sum_{i=1}^{m} f(g(z^{(i)}))$$

Maximize g,
freeze weights of critic
No change

- Theorem: a function is 1-Lipschitz if and only if its gradient norm is $\|\nabla f(\hat{x})\|_2 \leq 1$, but that is very constraining

Maximize f
freeze generator weights,

incentivize critic gradient norm to be one
Choose large lambda (e.g., 10)

$$\frac{1}{m}\sum_{i=1}^{m} f(x_{real}^{(i)}) - f(g(z^{(i)})) - \lambda \frac{1}{W}\sum_{i=1}^{W} \left(\|\nabla f(\hat{x}^{(i)})\|_2 - 1\right)^2$$

where for ϵ in U[0,1] $\hat{x}^{(i)} = \epsilon \cdot x_{real}^{(i)} + (1-\epsilon)\cdot g(z^{(i)})$

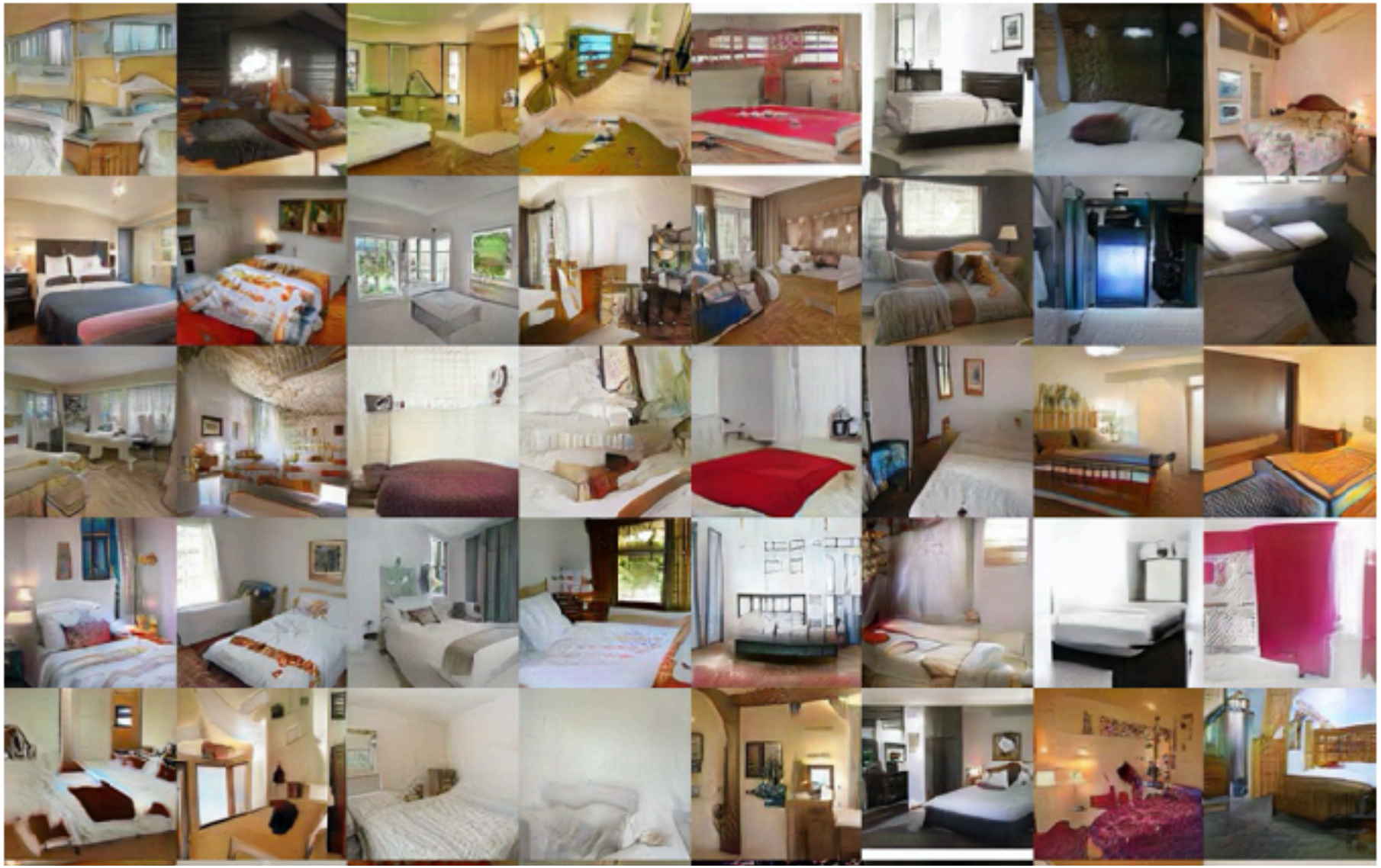
randomly mix together real and fake images, for gradient estimation

WGAN-GP Heuristics

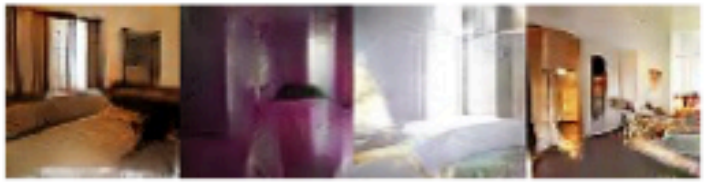
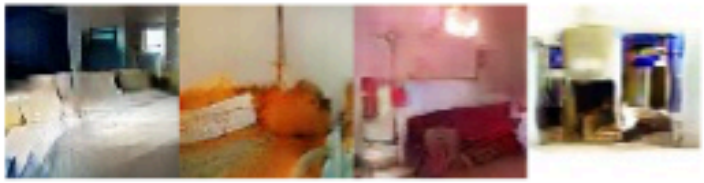
- Choose large penalty coefficient
- Don't use batch normalization (in critic)
  - it forces the gradient norm to be dependent on batches
  - but layer norm seems to be okay
- Enforce gradient norm to be close to one, rather than less than one

Two-sided penalty We encourage the norm of the gradient to go towards 1 (two-sided penalty) instead of just staying below 1 (one-sided penalty). Empirically this seems not to constrain the critic too much, likely because the optimal WGAN critic anyway has gradients with norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$ and in large portions of the region in between (see subsection 2.3). In our early observations we found this to perform slightly better, but we don't investigate this fully. We describe experiments on the one-sided penalty in the appendix.

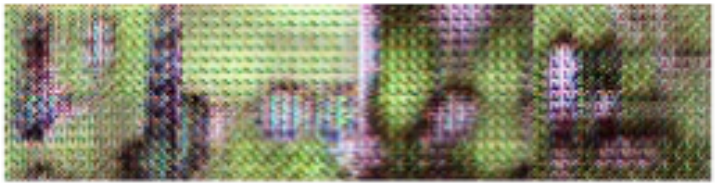
WGAN-GP Results



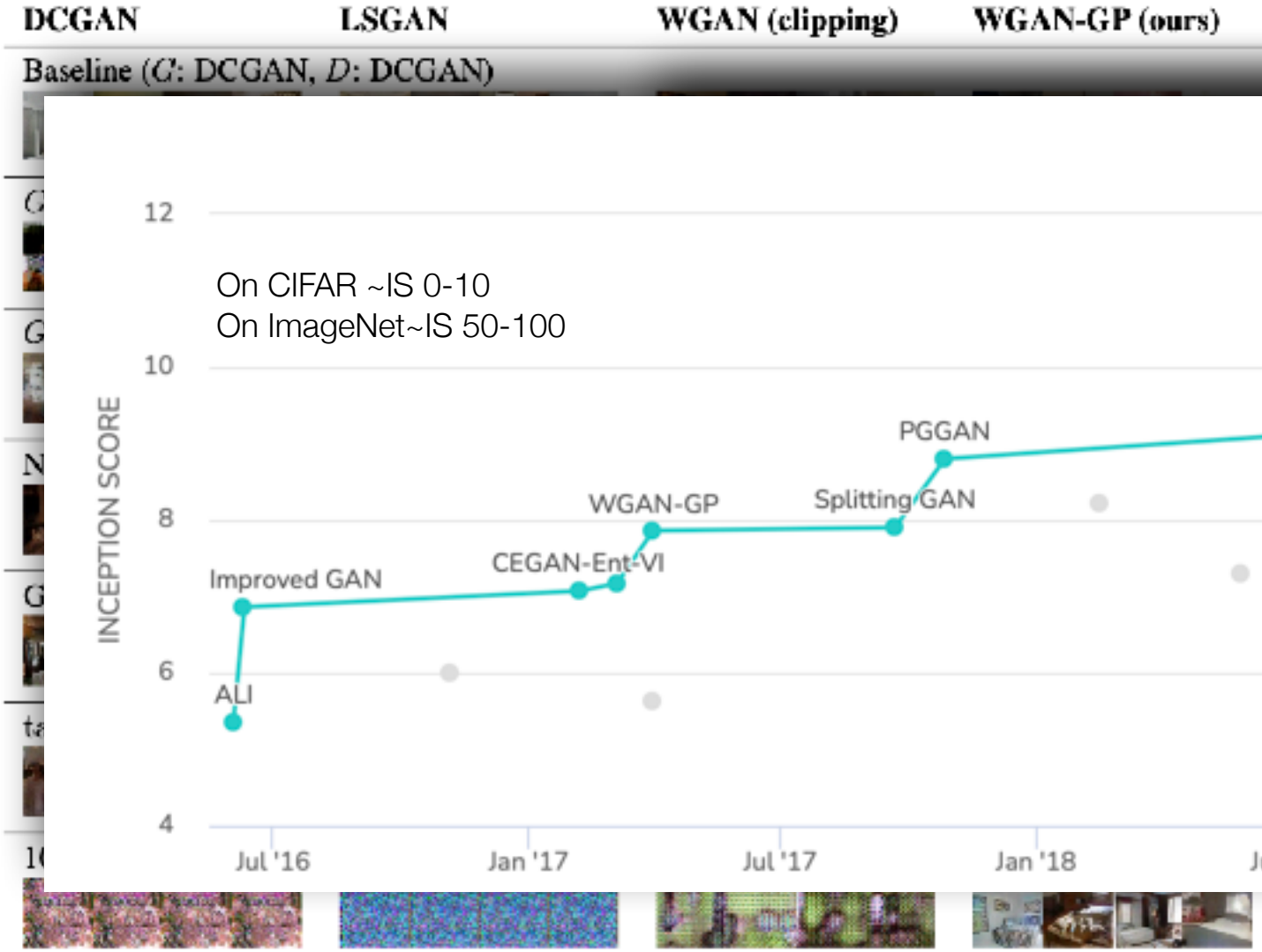
WGAN-GP Comparative Results

WGAN (clipping)	WGAN-GP (ours)
Baseline: G: DCGAN	Crit: DCGAN
	
G: DCGAN no BN	Crit: DCGAN
	
G: MLP	Crit: DCGAN
	

WGAN-GP Comparative Results

WGAN (clipping)	WGAN-GP (ours)
G/Crit: Tanh Non-linearities	
	
G/Crit: 101 Layer ResNet	
	

WGAN-GP Comparative Results



WGAN-GP

Main Repository:
07c GANsWithKeras.ipynb

Other Examples

Keras Example: https://github.com/keras-team/keras-contrib/blob/master/examples/improved_wgan.py DCGAN

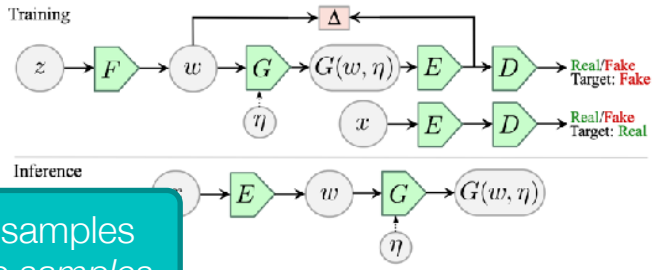
Demo by Keras Contrib Community

Other Example: https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/wgan_gp/wgan_gp.py MLP-GAN

Demo by Erik Linder-Norén

Aside: Back to ALAEs

Adversarial Latent Auto-Encoders, ALAE



Algorithm 1 ALAE Training

1: $\theta_F, \theta_G, \theta_E, \theta_D \leftarrow$ Initialize network parameters
2: **while** not converged **do**
3: Step I. Update E, and D
4: $x \leftarrow$ Random mini-batch from dataset
5: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
6: $L_{adv}^{E,D} \leftarrow \text{softplus}(D \circ E \circ G \circ F(z))) + \text{softplus}(-D \circ E(x)) +$
$\frac{\gamma}{2} \mathrm{E}_{p_{\mathcal{D}}(x)} \left[\|\nabla D \circ E(x)\|^2 \right]$
7: $\theta_E, \theta_D \leftarrow \text{ADAM}(\nabla_{\theta_D, \theta_E} L_{adv}^{E,D}, \theta_D, \theta_E, \alpha, \beta_1, \beta_2)$
8: Step II. Update F, and G
9: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
10: $L_{adv}^{F,G} \leftarrow \text{softplus}(-D \circ E \circ G \circ F(z)))$
11: $\theta_F, \theta_G \leftarrow \text{ADAM}(\nabla_{\theta_F, \theta_G} L_{adv}^{F,G}, \theta_F, \theta_G, \alpha, \beta_1, \beta_2)$
12: Step III. Update E, and G
13: $z \leftarrow$ Samples from prior $\mathcal{N}(0, I)$
14: $L_{error}^{E,G} \leftarrow \|F(z) - E \circ G \circ F(z)\|_2^2$
15: $\theta_E, \theta_G \leftarrow \text{ADAM}(\nabla_{\theta_E, \theta_G} L_{error}^{E,G}, \theta_E, \theta_G, \alpha, \beta_1, \beta_2)$
16: **end while**

E,D: Detect fake samples
minimize D for fake samples

E,D: Detect real samples
minimize -D for real samples

E,D: Gradient Penalty
Keep Gradient Magnitude Small
Now we know why!!

F,G: Try to fool discriminator,
minimize -D for fake samples

E,G: Keep latent spaces similar

Pidhorskyi, Stanislav, Donald A. Adjeroh, and Gianfranco Doretto. "Adversarial latent autoencoders." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14104-14113. 2020.

GAN Town Hall

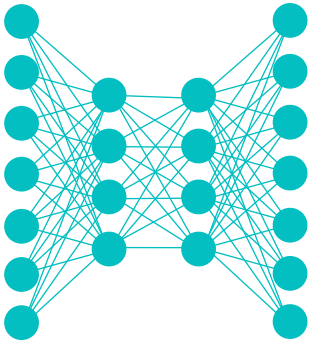
Lecture Notes for

Neural Networks and Machine Learning

Wasserstein GANs

Next Time:
BigGAN and StyleGAN

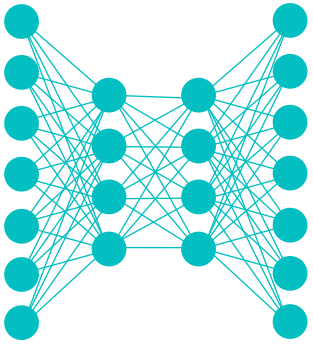
Reading: Those Papers!



Lecture Notes for

Neural Networks and Machine Learning

BigGAN and StyleGAN



Logistics and Agenda

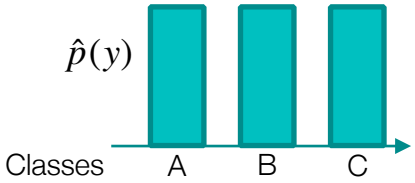
- Logistics
  - **Student Presentation**: None
- Agenda
  - BigGAN
    - Big GAN? or Biggin?
  - Next Time: Stable Diffusion

An Accepted Measure: Inception Score

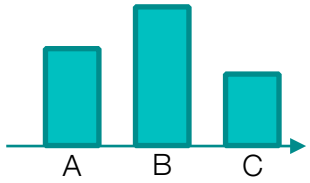
$$\hat{p}(y) = \frac{1}{N}\sum_i p(y \mid \mathbf{x}^{(i)}_{fake})$$

marginal

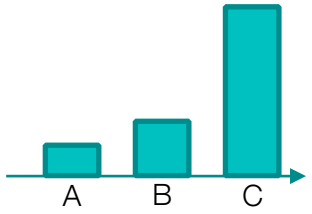
Expected class distribution through a trained CNN, like VGG should be **nearly uniform** in ideal case



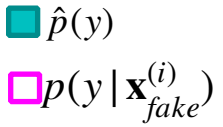
Ideal: Generator has diverse classes



Good: Generator is somewhat diverse



Bad: Generator not diverse

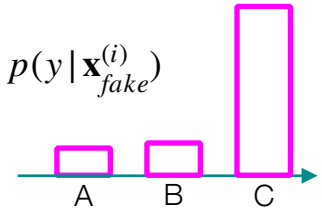


$$IS(G) \approx \exp\left(\frac{1}{N}\sum_i D_{KL}\left(p(y \mid \mathbf{x}^{(i)}_{fake}) \| \hat{p}(y)\right)\right)$$

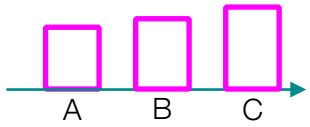
one example generated

typical generation

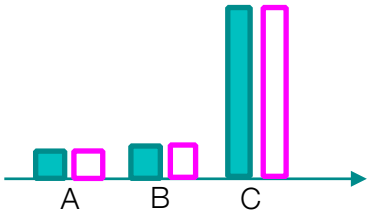
average KL Divergence of marginal of generated images with $\hat{p}$, ideally **differ dramatically**



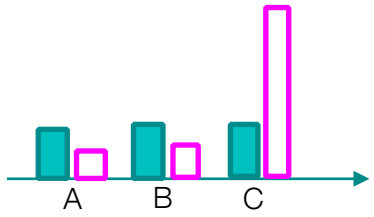
Ideal: Single Example is distinct class



Bad: Single Example is not really distinct



Bad: Distinct because not diverse



Ideal: Diverse and Distinct

Other Explanation: https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a

BigGAN

In a field with 1000's of competing papers,
BigGAN is here to use the most meaningful
Portions of each paper and put them into
One BIG paper.

LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS

Andrew Brock[*†]
Heriot-Watt University
ajb5@hw.ac.uk

Jeff Donahue[†]
DeepMind
jeffdonahue@google.com

Karen Simonyan[†]
DeepMind
simonyan@google.com

BigGAN Overview

- This is an agglomeration of GAN Knowledge from *2013-2020*
- Training is hard, so use heuristics
  - large batches, feature matching
  - use hinge loss (max margin)
- Use attention, conditional classes, spectral normalization, moving average of weights, orthogonal weight initialization, skip connections, orthogonal regularizers
- **Truncation trick:** sample a wide range during training $\sigma = \lambda$, then truncate for evaluation $\sigma = \frac{\lambda}{2}$

- Large Scale GAN Training for High Fidelity Natural Image Synthesis, 2018.
- Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019. 2019
- Self-Attention Generative Adversarial Networks, 2018.
- A Learned Representation For Artistic Style, 2016.
- Spectral Normalization for Generative Adversarial Networks, 2018.
- Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017.
- Exact Solutions To The Nonlinear Dynamics Of Learning In Deep Linear Neural Networks, 2013.
- Neural Photo Editing with Introspective Adversarial Networks, 2016.

https://machinelearningmastery.com/a-gentle-introduction-to-the-biggan/

BigGAN Part One: Spectral Normalization

- After updating weights (in critic), use the spectral norm, such that the network satisfies the Lipschitz constraint $\sigma(W) \approx 1$ for each layer
- Which makes the critic a valid Wasserstein estimate, but infinitely easier to compute!

$$\mathbf{W} \leftarrow \frac{\mathbf{W}}{\sigma(\mathbf{W})}$$ ← which is largest singular value of $\mathbf{W}$

Our spectral normalization controls the Lipschitz constant of the discriminator function f by literally constraining the spectral norm of each layer $g : \boldsymbol{h}_{in} \mapsto \boldsymbol{h}_{out}$. By definition, Lipschitz norm $\|g\|_{\text{Lip}}$ is equal to $\sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h}))$, where $\sigma(A)$ is the spectral norm of the matrix A (L_2 matrix norm of A)

$$\sigma(A) := \max_{\boldsymbol{h}:\boldsymbol{h}\neq 0} \frac{\|A\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = \max_{\|\boldsymbol{h}\|_2 \leq 1} \|A\boldsymbol{h}\|_2, \tag{6}$$

which is equivalent to the largest singular value of A.

Paraphrasing from paper:
Most layers in the generator have well-behaved spectra, but without constraints (like in WGAN-GP) a small subset grow throughout training and explode, resulting in a collapse of training. This was solved by monitoring for collapse and loading the best model before the collapse. An attempt was also made to integrate the WGAN-GP constraint in the loss function with BigGAN. While this did make the results more stable, the IS score dropped by 45%.

https://arxiv.org/pdf/1802.05957.pdf Spectral Normalization for Generative Adversarial Networks, 2018.

BigGAN Part One: Spectral Normalization

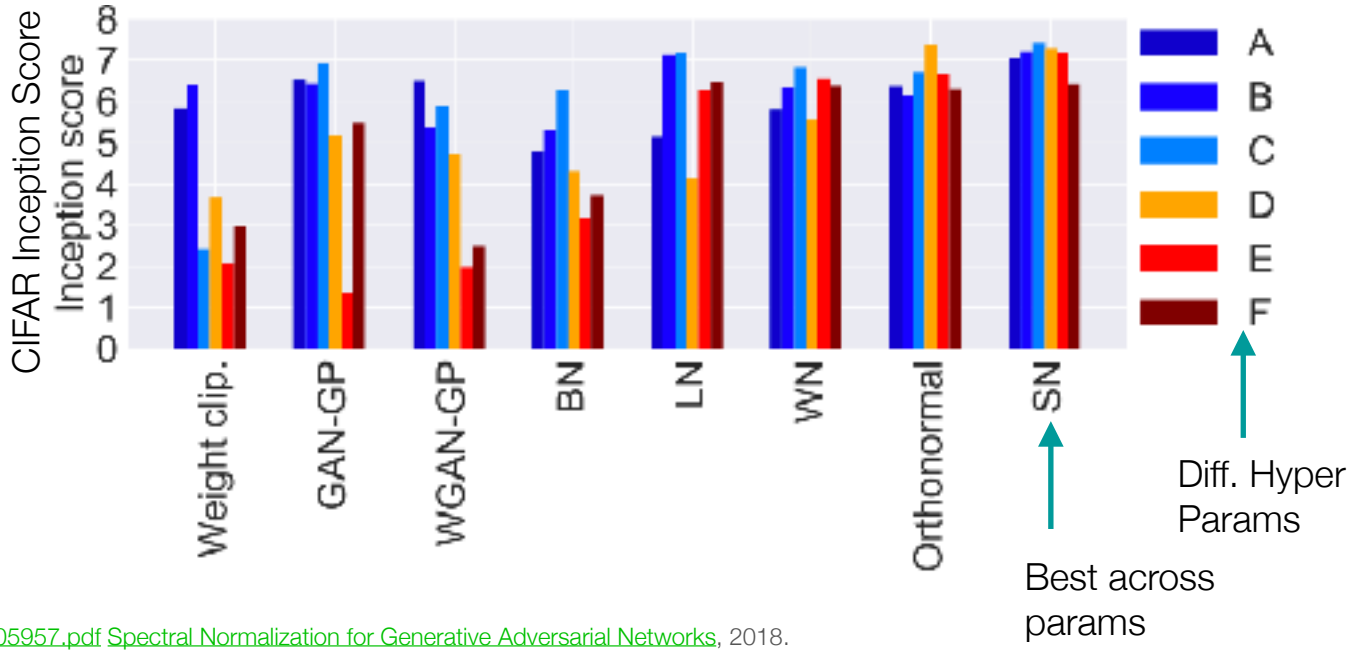
Our *spectral normalization* normalizes the spectral norm of the weight matrix W so that it satisfies the Lipschitz constraint $\sigma(W) = 1$:

$$\bar{W}_{\text{SN}}(W) := W/\sigma(W). \tag{8}$$

$$\frac{\partial \bar{W}_{\text{SN}}(W)}{\partial W_{ij}} = \frac{1}{\sigma(W)} E_{ij} - \frac{1}{\sigma(W)^2} \frac{\partial \sigma(W)}{\partial W_{ij}} W = \frac{1}{\sigma(W)} E_{ij} - \frac{[\boldsymbol{u}_1 \boldsymbol{v}_1^{\mathrm{T}}]_{ij}}{\sigma(W)^2} W \tag{9}$$

$$= \frac{1}{\sigma(W)} \left(E_{ij} - [\boldsymbol{u}_1 \boldsymbol{v}_1^{\mathrm{T}}]_{ij} \bar{W}_{\text{SN}} \right), \tag{10}$$

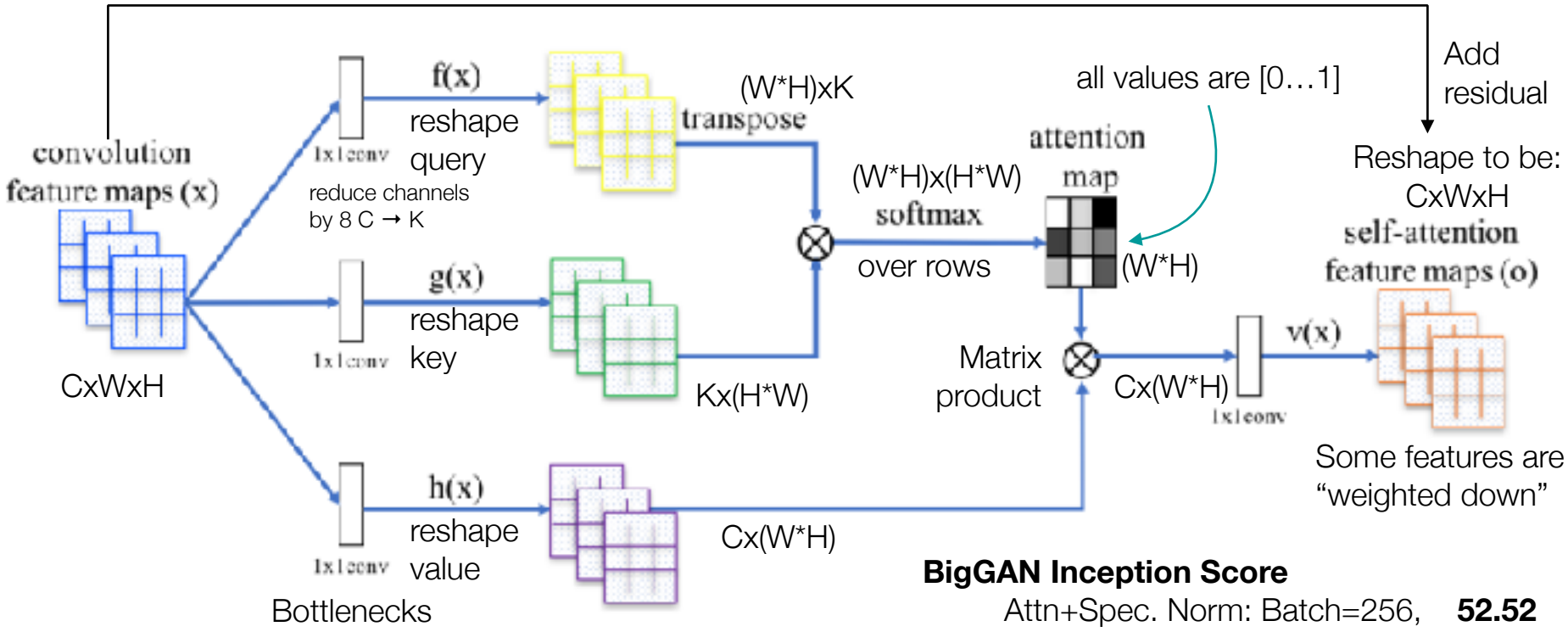
And we can back propagate through the calculation!



https://arxiv.org/pdf/1802.05957.pdf Spectral Normalization for Generative Adversarial Networks, 2018.

BigGAN Part Two: Self Attention

Layer used in both generator and discriminator (towards end)



Model	Inception Score
AC-GAN (Odena et al., 2017)	28.5
SNGAN-projection (Miyato & Koyama, 2018)	36.8
SAGAN	**52.52**

BigGAN Inception Score

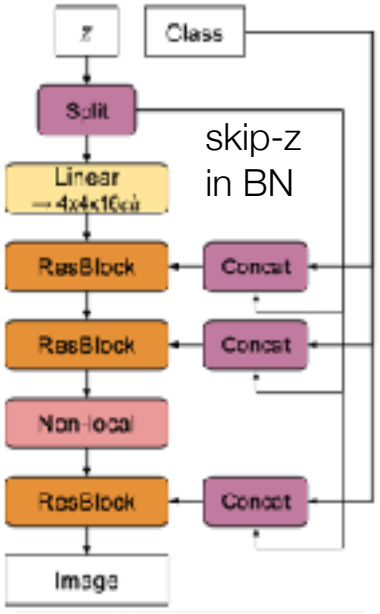
Attn+Spec. Norm: Batch=256,	**52.52**
Attn+Spec. Norm: Batch=512,	**58.77**
Attn+Spec. Norm: Batch=1024,	**63.03**
Attn+Spec. Norm: Batch=2048,	**76.85**
Attn+Spec. Norm: Batch=2048, and More filters (64 to 96):	**92.98**

IS 52.52 → 92.98

- Zhang, Goodfellow, Metaxas, Odena. Self-Attention Generative Adversarial Networks, 2018.

IS 52.52 → 92.98

BigGAN Part Three: Class Info + Skip-z



(a) Generator

RGB image $x \in \mathbb{R}^{128\times128\times3}$
ResBlock down 64
ResBlock down 128
ResBlock down 256
ResBlock down 512
ResBlock down 1024
ResBlock 1024
ReLU
Global sum pooling
dense → 1

(b) Discriminator for unconditional GANs.

RGB image $x \in \mathbb{R}^{128\times128\times3}$
ResBlock down 64
ResBlock down 128
ResBlock down 256
Concat(Embed(y), h)
ResBlock down 512
ResBlock down 1024
ResBlock 1024
ReLU
Global sum pooling
dense → 1

(c) Discriminator for conditional GANs. For computational ease, we embedded the integer label $y \in \{0, \ldots, 1000\}$ into 128 dimension before concatenating the vector to the output of the intermediate layer.

Categorical, One Hot Class

Embedded via Batch Norm

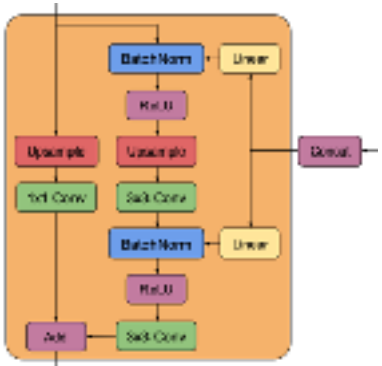
$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$x_{bn} = \gamma \cdot \hat{x} \cdot g(y_{OHE}) - \beta - b(y_{OHE})$$

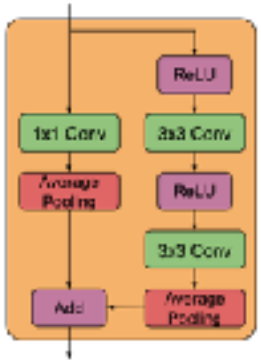
Can help to learn **class specific** properties in generator and discriminator

Shared Class Info

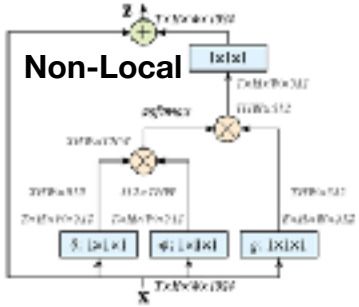
IS 92.98 → 94.94



ResBlock Up



ResBlock Down



Skip z- multiple copies

IS 94.94 → 98.76

Non-local is Self Attention with a residual connection

IS 94.94 → 98.76

BigGAN Part Four: Orthogonality

- **Start Orthogonal**: Initialize with orthogonal weights per layer
  - $W \cdot W^T = I$
- **Stay that way**: Add orthogonal regularization to loss:

$$\mathscr{L}_{orthogonal} = \alpha_{orth} \sum \| W \cdot W^T - \mathbf{I} \|$$

Usually too restrictive…

applied across channels of filters

$$\mathscr{L}_{orthogonal} = \alpha_{orth} \sum \| W \cdot W^T \odot (\mathbf{1} - \mathbf{I}) \|$$

penalize non zero diagonals

IS 98.76 → 99.31

Brock, Lim, Ritchie, Wetson, Neural Photo Editing with Introspective Adversarial Networks. 2017 https://arxiv.org/abs/1609.07093

BigGAN: Miscellaneous

IS 98.76 → 99.31

- Update discriminator twice as often as generator
- Sample from censored Normal: $\max(N(0, I), 0)$
- Use skip connections in model architecture, starting from z
- Use LOTS of filters: 150% more filters than related work
- **Truncation trick**: during training, use wider sampling than during evaluation
- Use hinge loss: $\frac{1}{m}\sum_{i=1}^{m} f(x_{real}^{(i)}) \cdot f(g(z^{(i)}))$
- Use moving average in Generator: $W_k = \sum_{i \in Epoch} \gamma^i W_{k-i}$
- **Note**: "Stabilizes" in fewer iterations!! But then the training collapses… (they ran for 150,000 iterations on ImageNet)

BigGAN Results Summary




Batch	Ch.	Param (M)	Shared	Skip-z	Ortho.	IS
256	64	81.5	SA-GAN Baseline			52.52
512	64	81.5	✗	✗	✗	58.77(±1.18)
1024	64	81.5	✗	✗	✗	63.03(±1.42)
2048	64	81.5	✗	✗	✗	76.85(±3.83)
2048	96	173.5	✗	✗	✗	92.98(±4.27)
2048	96	160.6	✓	✗	✗	94.94(±1.32)
2048	96	158.3	✓	✓	✗	98.76(±2.84)
2048	96	158.3	✓	✓	✓	99.31(±2.10)
2048	64	71.3	✓	✓	✓	86.90(±0.61)

ImageNet IS ~ 50

BigGAN Results

256 x 256

BigGAN Results

512 x 512

BigGAN Results: Linear Interpolation

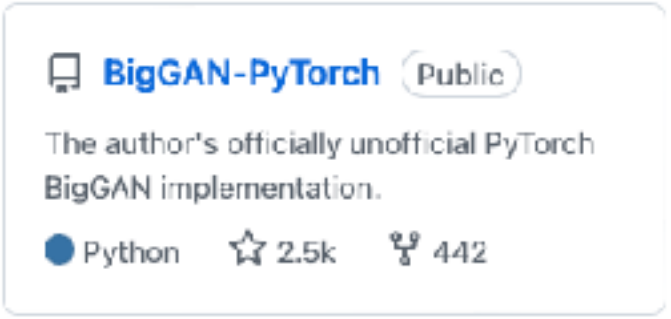
BigGAN-Torch

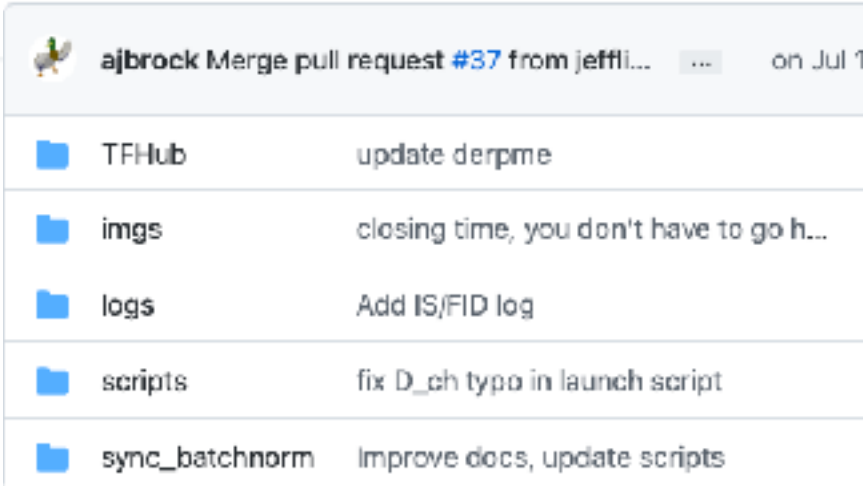
Main Repository:

`07d BigGANTorch.ipynb`

Large Scale GAN Training for High Fidelity Natural Image Synthesis

Modified from Andy Brock Implementation

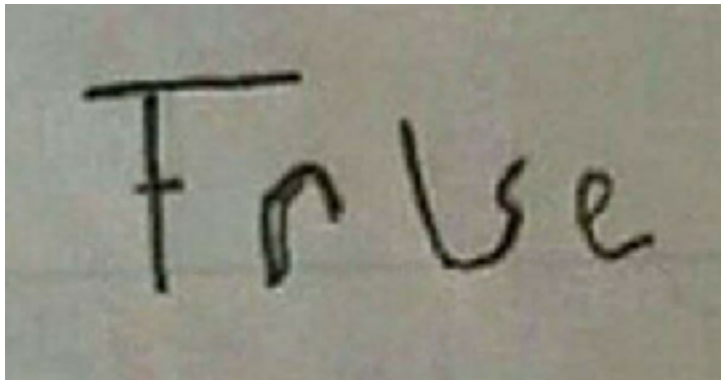




Andrew Brock[*†]
Heriot-Watt University
ajb5@hw.ac.uk

StyleGAN 3.0 and StyleCLIP

When binary classification == 0.5

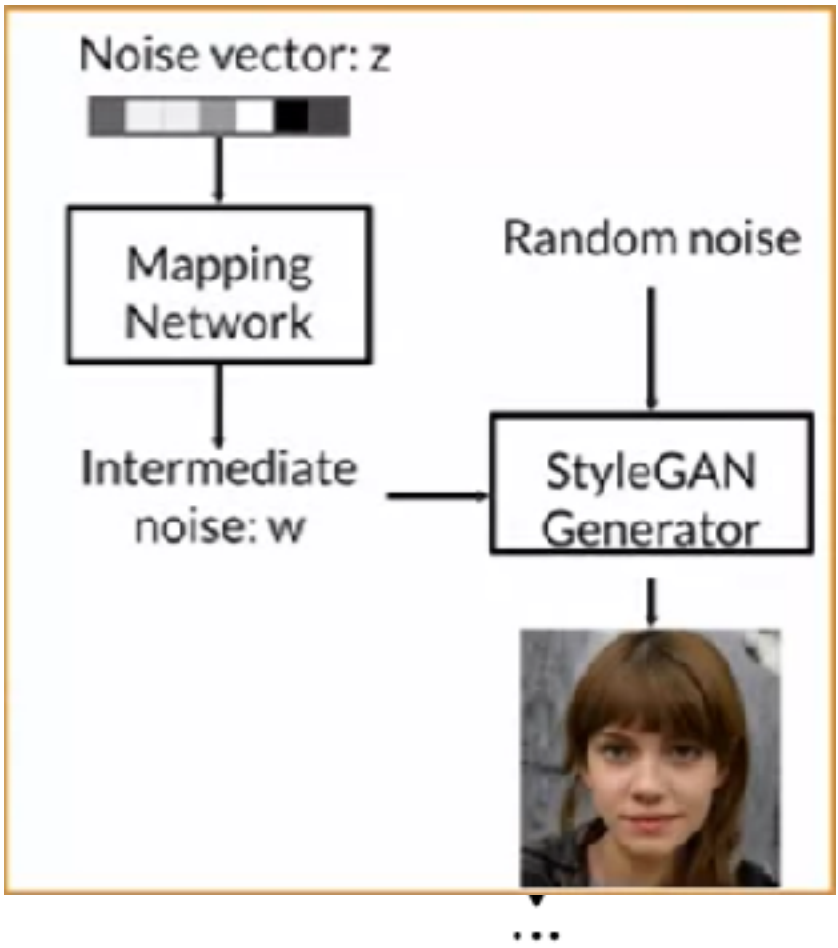


https://arxiv.org/abs/2103.17249

StyleGAN

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

- Noise added everywhere!
  - But start with constant 4x4x512
- Adaptive Input Normalization
- Bilinear Upsampling
- Progressive Growing



A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras
NVIDIA
tkarras@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

Timo Aila
NVIDIA
taila@nvidia.com

https://arxiv.org/pdf/1812.04948.pdf

Figure 10. Uncurated set of images produced by our style-based generator (config F) with the LSUN BEDROOM dataset at 256^2. FID computed for 50K images was 2.65.

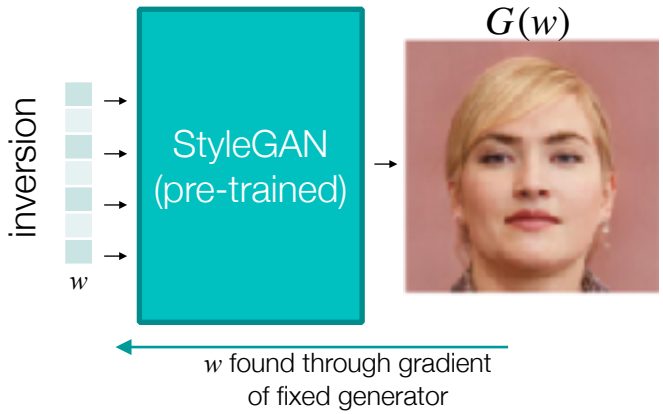
Figure 11. Uncurated set of images produced by our style-based generator (config F) with the LSUN CAR dataset at 512×384. FID computed for 50K images was 3.27.

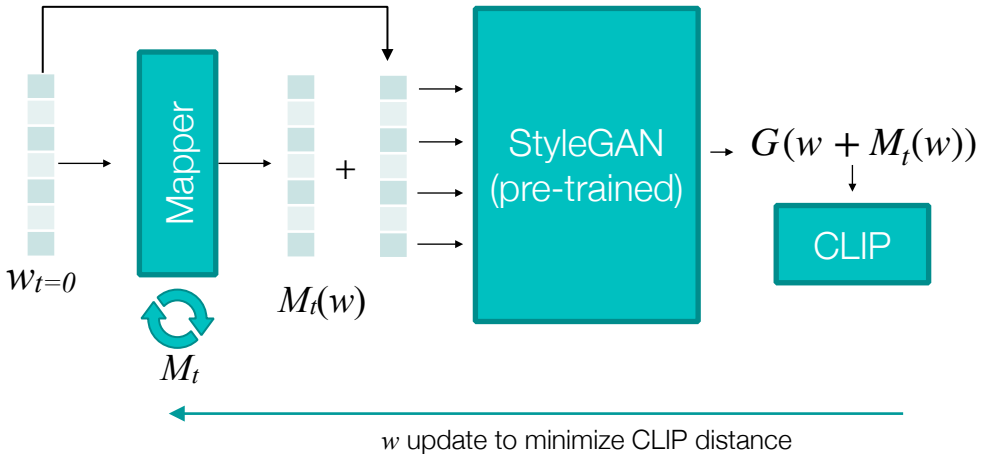
StyleCLIP

StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery

Or Patashnik[†*] Zongze Wu[†*] Eli Shechtman[§] Daniel Cohen-Or[†] Dani Lischinski[‡]
[†]Hebrew University of Jerusalem [‡]Tel-Aviv University [§]Adobe Research

https://arxiv.org/abs/2103.17249

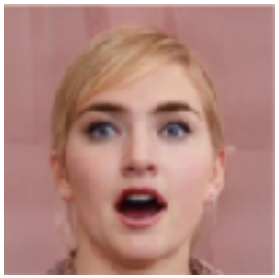




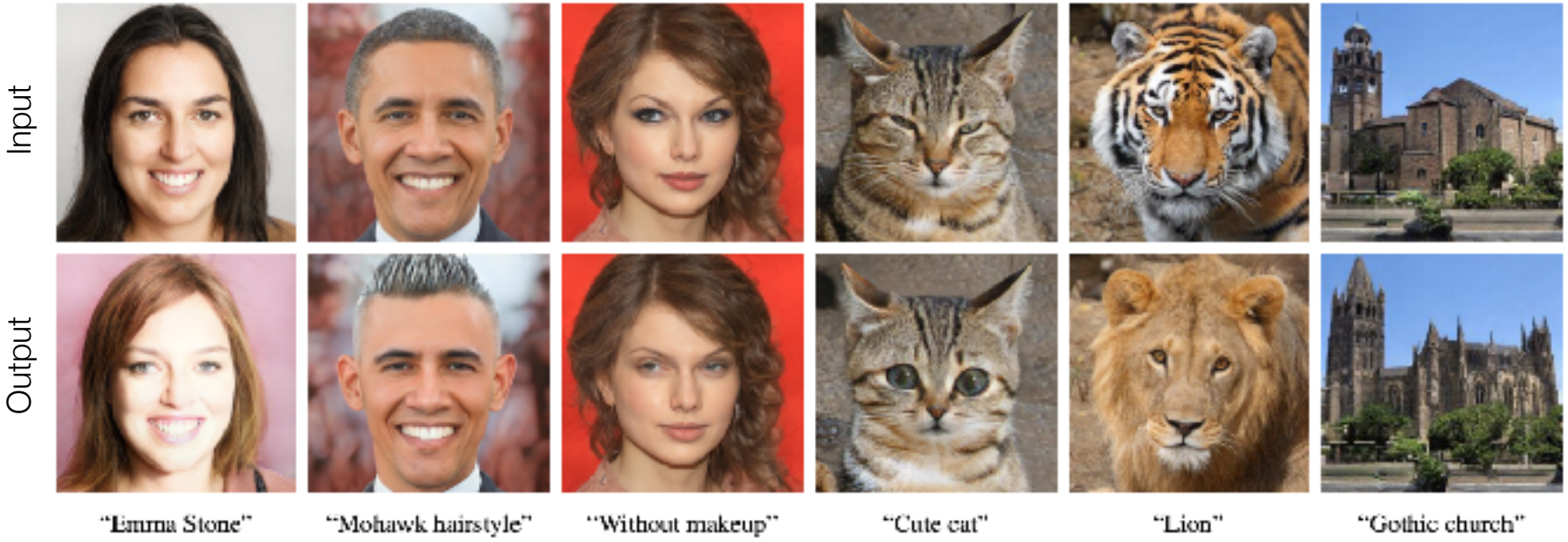
$$\mathcal{L}_{CLIP}(w) = D_{CLIP}\left[G(w + M_t(w)), t\right]$$

Run until the CLIP is optimized by the input image and it is not too far away from the starting vector w

$\mathcal{L}_w$("surprised")



$$\mathcal{L}_w = \mathcal{L}_{CLIP}(w) + \lambda_{L2}\|w_0 - w_T\|^2 + \underbrace{\lambda_{ID}[R(G(w_0)) - R(G(w_T))]}_{R \rightarrow pre-trained}$$


Input
Output
"Emma Stone"
"Mohawk hairstyle"
"Without makeup"
"Cute cat"
"Lion"
"Gothic church"


Input
Happy
Big Eyes
Golden Fur
Bulldog


Input
Jeep
Sports
From Sixties
Classic

CLIP Explanation: Optional

Image Encoder Pre-trained

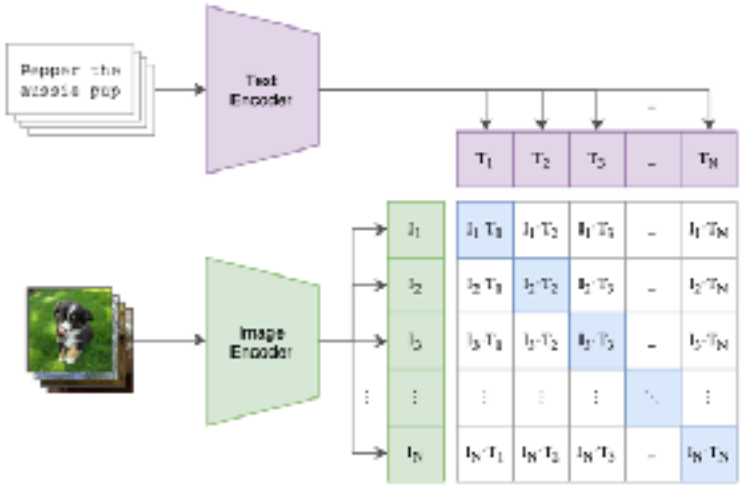
W_i

"A Narcissistic Professor"

Text Encoder Pre-trained

W_t

Positive Examples: Make similar
Negative Examples: Make dissimilar



Contrastive Language-Image Pre-training

CLIP

Radford et al. 2021, Learning Transferable Visual Models From Natural Language Supervision, https://arxiv.org/pdf/2103.00020.pdf

Lecture Notes for

Neural Networks and Machine Learning

BigGAN

Next Time:
Stable Diffusion

Reading: Chollet CH8