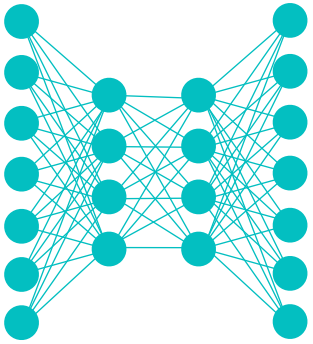
Lecture Notes for

Neural Networks and Machine Learning

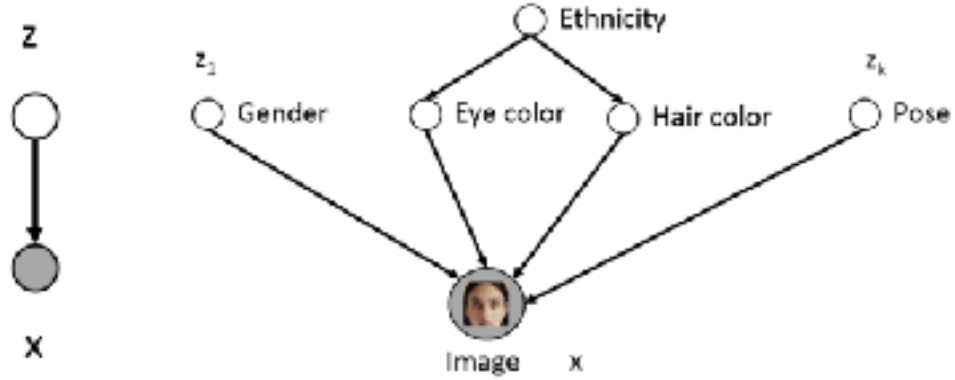
Generative Networks



Logistics and Agenda

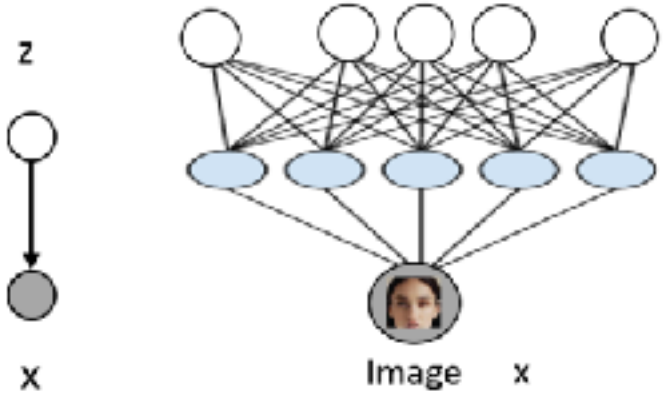
- Logistics
  - Lab Three is Posted
- Agenda
  - A historical perspective of generative Neural Networks
  - Variational Auto-Encoding
  - VAE in Keras Demo

Motivations: Latent Variables



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

Hard: **z** is expertly chosen



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

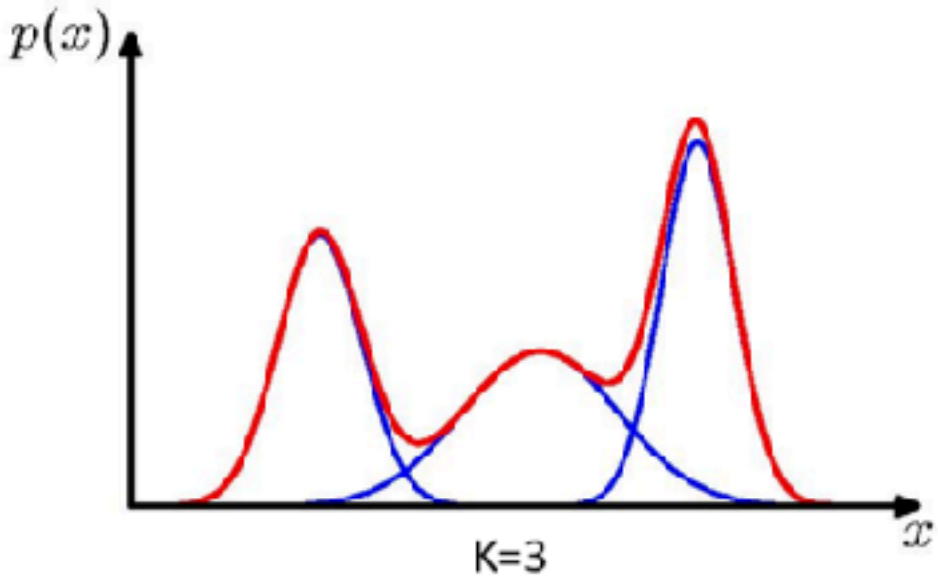
Not as Hard: **z** is trained, latent variables are uncontrolled

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

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

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

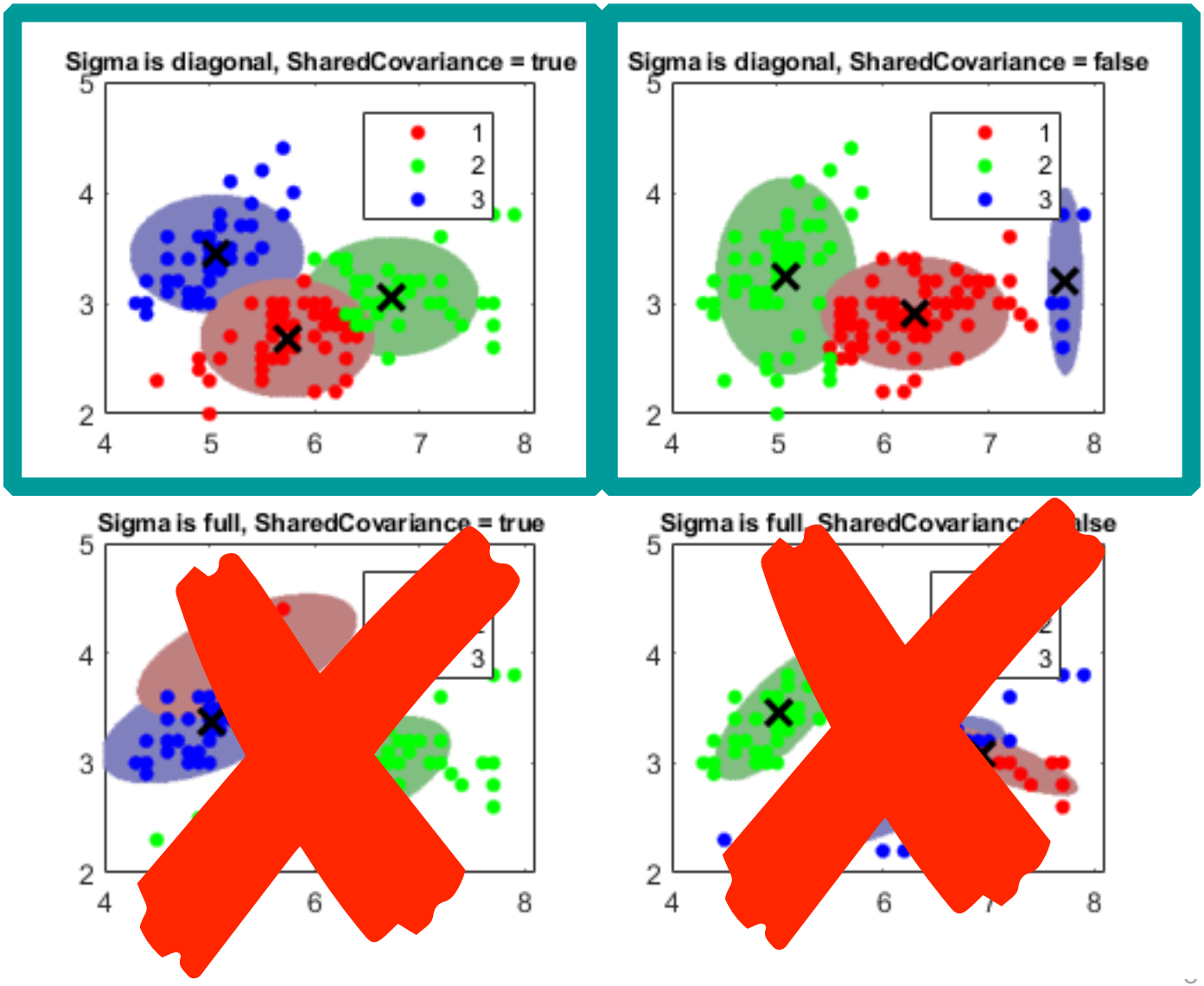
mean and covariance learned

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

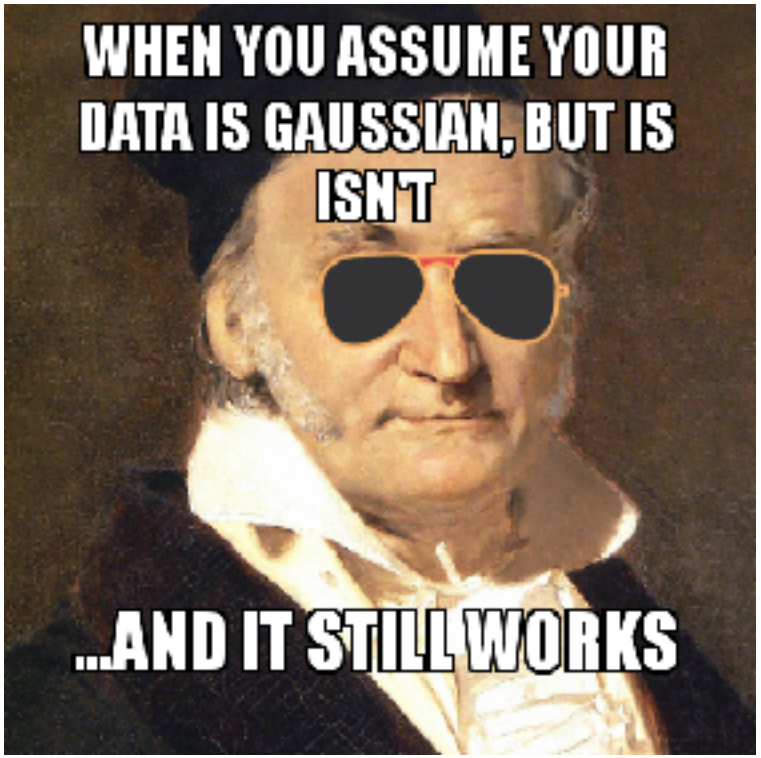
Motivation: Mixtures for Simplicity

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

mean and covariance learned



A History of Generative Networks



Taxonomy of Generative Models

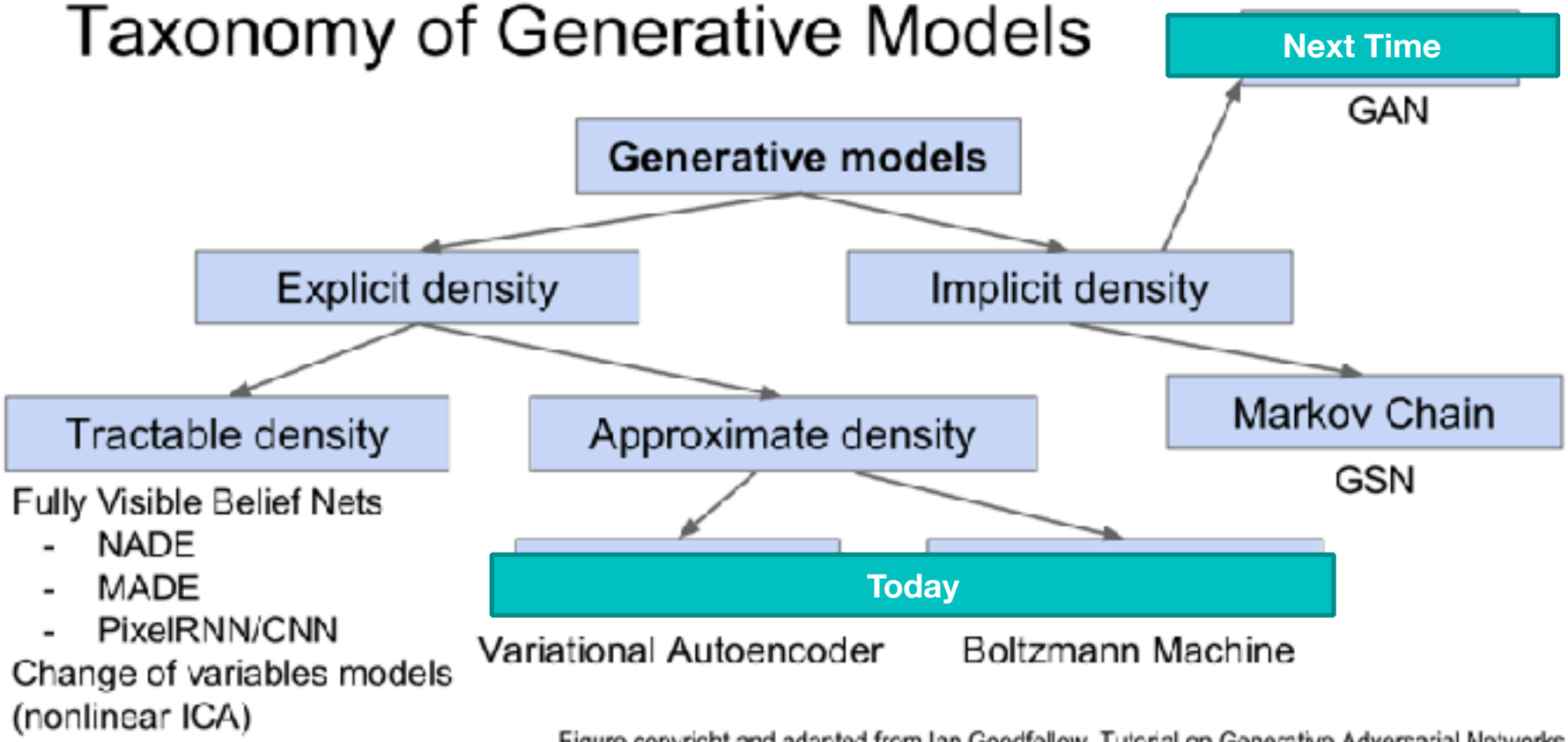


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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

Deep Generative Models, 2006

- Deep Belief Networks (DBNs)
- Iterative Layer Training (not feed forward)

energy function of the RBM

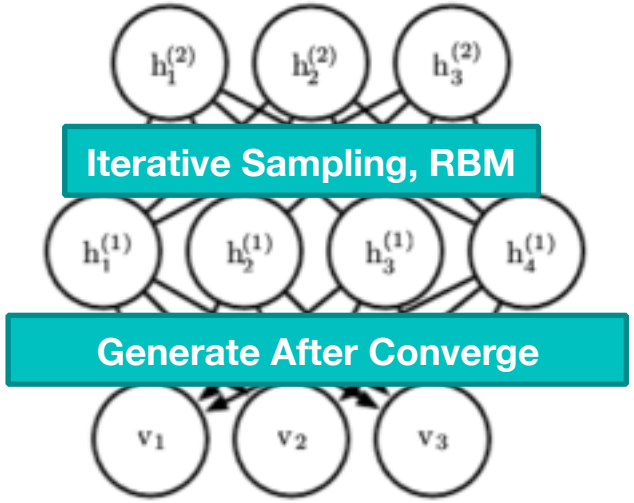
$$P(\boldsymbol{h}^{(l)}, \boldsymbol{h}^{(l-1)}) \propto \exp\left(\boldsymbol{b}^{(l)\top}\boldsymbol{h}^{(l)} + \boldsymbol{b}^{(l-1)\top}\boldsymbol{h}^{(l-1)} + \boldsymbol{h}^{(l-1)\top}\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l)}\right), \quad (20.17)$$

$$P(h_i^{(k)} = 1 \mid \boldsymbol{h}^{(k+1)}) = \sigma\left(b_i^{(k)} + \boldsymbol{W}_{:,i}^{(k-1)\top}\boldsymbol{h}^{(k+1)}\right) \forall i, \forall k \in 1, \ldots, l-2, \quad (20.18)$$

$$P(v_i = 1 \mid \boldsymbol{h}^{(1)}) = \sigma\left(b_i^{(0)} + \boldsymbol{W}_{:,i}^{(1)\top}\boldsymbol{h}^{(1)}\right) \forall i. \quad (20.19)$$

In the case of real-valued visible units, substitute

$$\mathbf{v} \sim \mathcal{N}\left(v; \boldsymbol{b}^{(0)} + \boldsymbol{W}^{(1)\top}h^{(1)}, \boldsymbol{\beta}^{-1}\right) \quad (20.20)$$



$$\mathbf{E}_{v \leftarrow p}[\log p(v)]$$

$p(v)$ is intractable

$$\mathbf{E}_{v \leftarrow p}\left[\mathbf{E}_{h^{(1)} \leftarrow p^{(1)}(h^{(1)}|v)}[\log p^{(2)}(h^{(1)})]\right]$$

optimize with constrastive divergence

i.e., approximation of EM, not Back Prop

To train a deep belief network, one begins by training an RBM to maximize $\mathbb{E}_{\mathbf{v} \sim p_{\text{data}}} \log p(\boldsymbol{v})$ using contrastive divergence or stochastic maximum likelihood. The parameters of the RBM then define the parameters of the first layer of the DBN. Next, a second RBM is trained to approximately maximize

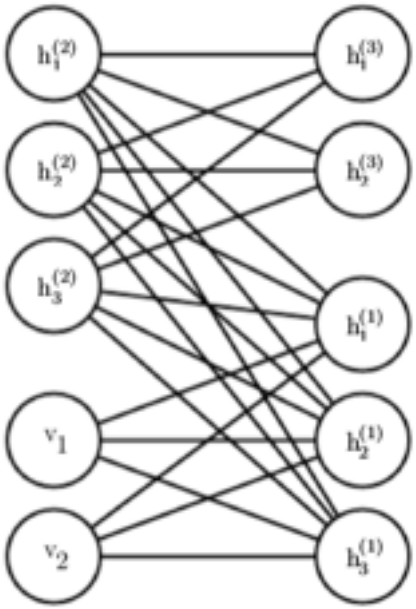
$$\mathbb{E}_{\mathbf{v} \sim p_{\text{data}}} \mathbb{E}_{\mathbf{h}^{(1)} \sim p^{(1)}(\boldsymbol{h}^{(1)}|\boldsymbol{v})} \log p^{(2)}(\boldsymbol{h}^{(1)}) \quad (20.21)$$

where $p^{(1)}$ is the probability distribution represented by the first RBM and $p^{(2)}$ is the probability distribution represented by the second RBM. In other words, the second RBM is trained to model the distribution defined by sampling the hidden units of the first RBM, when the first RBM is driven by the data. This

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Generative Models, 2009

- Deep Boltzmann Machine



$$P\left(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}\right) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta})\right). \tag{20.24}$$

To simplify our presentation, we omit the bias parameters below. The DBM energy function is then defined as follows:

$$E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta}) = -\boldsymbol{v}^\top \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)\top} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)} - \boldsymbol{h}^{(2)\top} \boldsymbol{W}^{(3)} \boldsymbol{h}^{(3)}. \tag{20.25}$$

We now develop the mean field approach for the example with two hidden layers. Let $Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})$ be the approximation of $P(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})$. The mean field assumption implies that

$$Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) = \prod_j Q(h_j^{(1)} \mid \boldsymbol{v}) \prod_k Q(h_k^{(2)} \mid \boldsymbol{v}). \tag{20.29}$$

Not tractable: Can only optimize the Evidence lower bound, ELBO

Approximate via MCMC like **Gibbs Sampling**

$$\mathrm{KL}(Q \| P) = \sum_{\boldsymbol{h}} Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) \log\left(\frac{Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})}{P(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})}\right). \tag{20.30}$$

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Image Generation from Samples

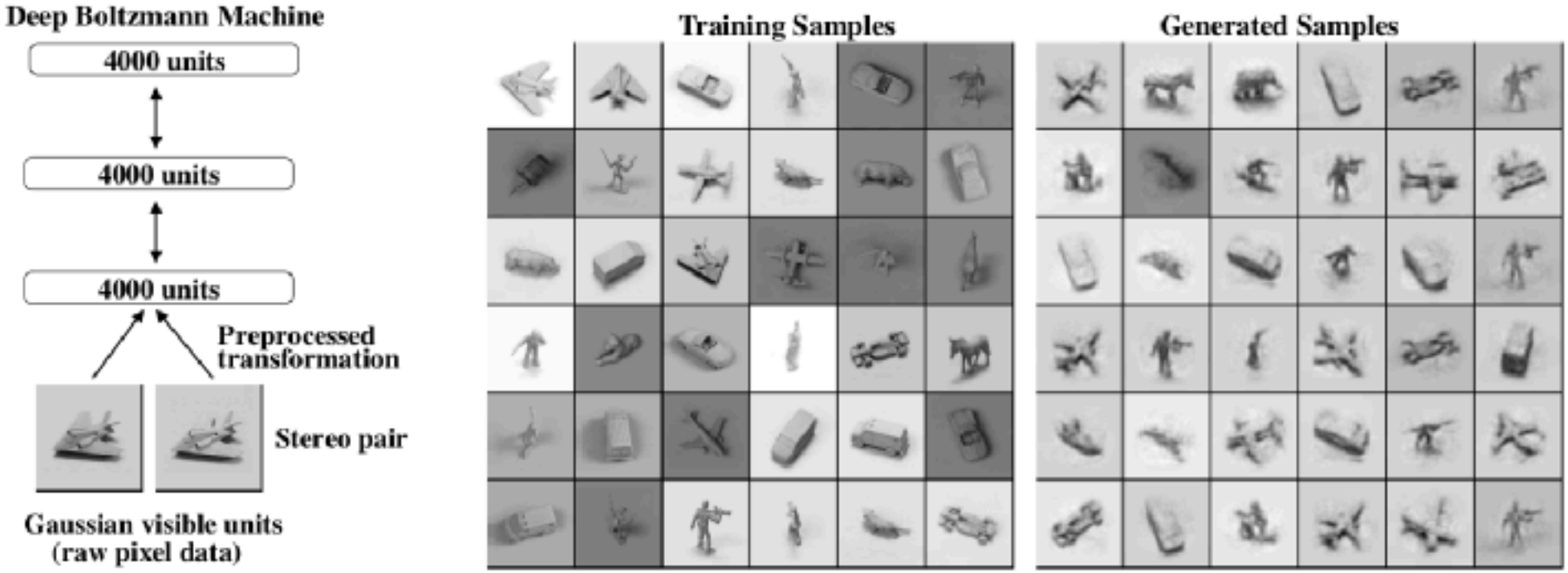


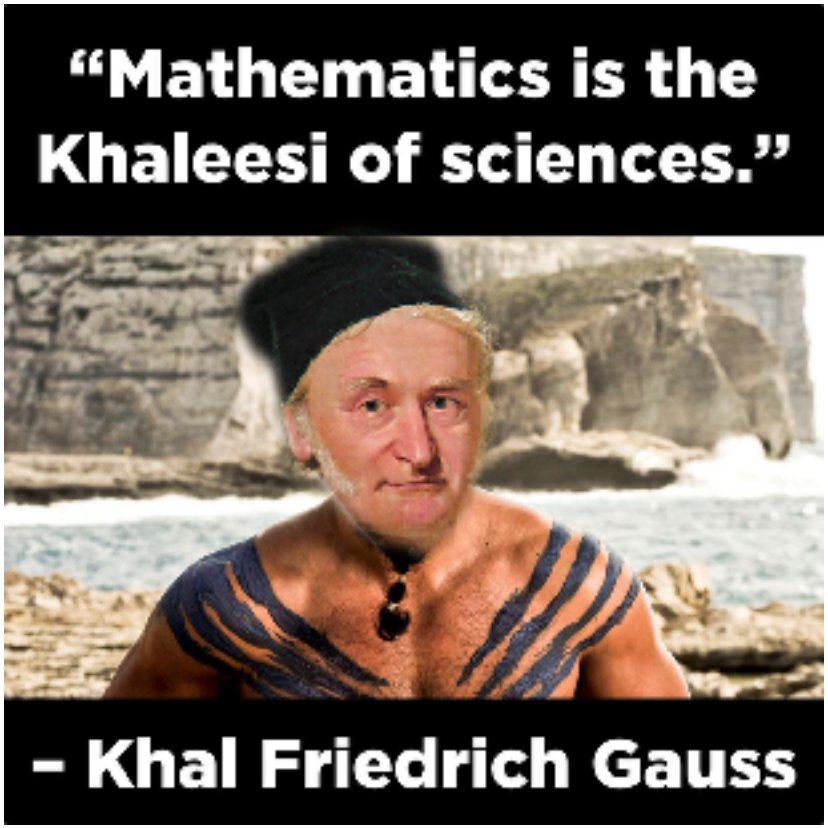
Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

Salakhutdinov, Ruslan, and Geoffrey Hinton. "Deep boltzmann machines." In Artificial intelligence and statistics, pp. 448-455. 2009.

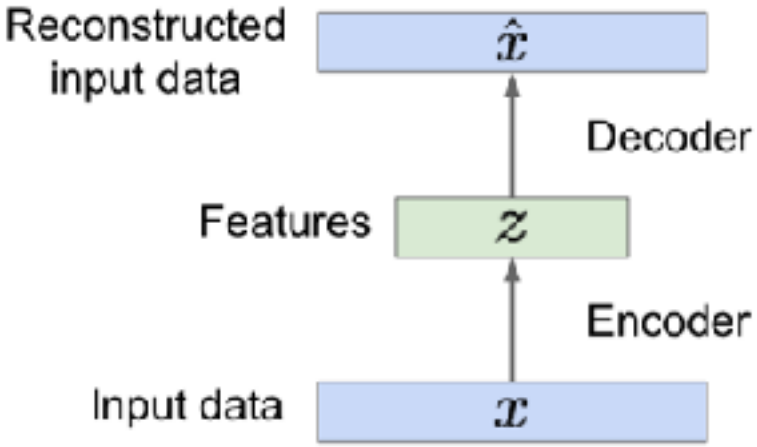
Contemporary Modeling

- DBNs and DBMs have mostly been abandoned
  - Mathematics detracts from popular understanding
  - Often methods using sampling are not scalable
  - Cannot directly use Gradients (no Back Prop) 😢
- Evidence Lower Bound (ELBO) considered "good enough" because … we can't do better computationally
- Popular method for calculating generative networks with ELBO approximation:
  - Variational Auto Encoding
    - Guaranteed NOT to find global minimum
    - But scalable and will converge in finite time

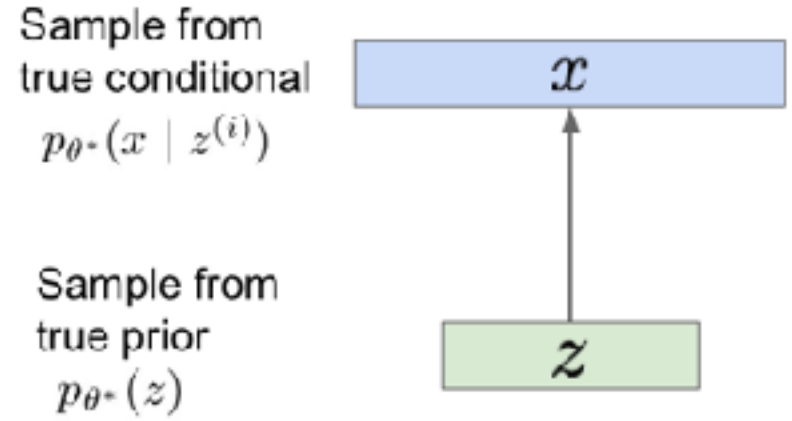
Variational Auto Encoding



Aside: Auto Encoding



Once trained, is it possible to generate data?



- Does this work for simple auto encoding?
  - Nope.
- Learned space is not continuous
- How to sample from the latent space?
- Features could be highly correlated
- Need to define some constraints on the latent space…

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

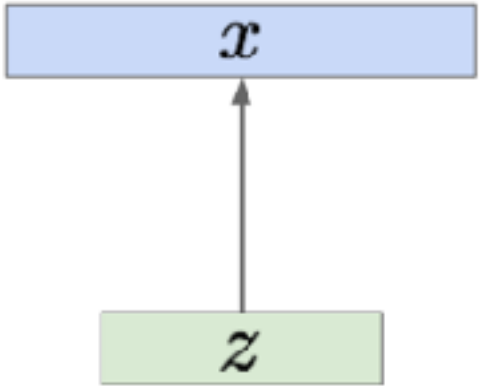
Reasonable constraints for p(z)?

Sample from true conditional

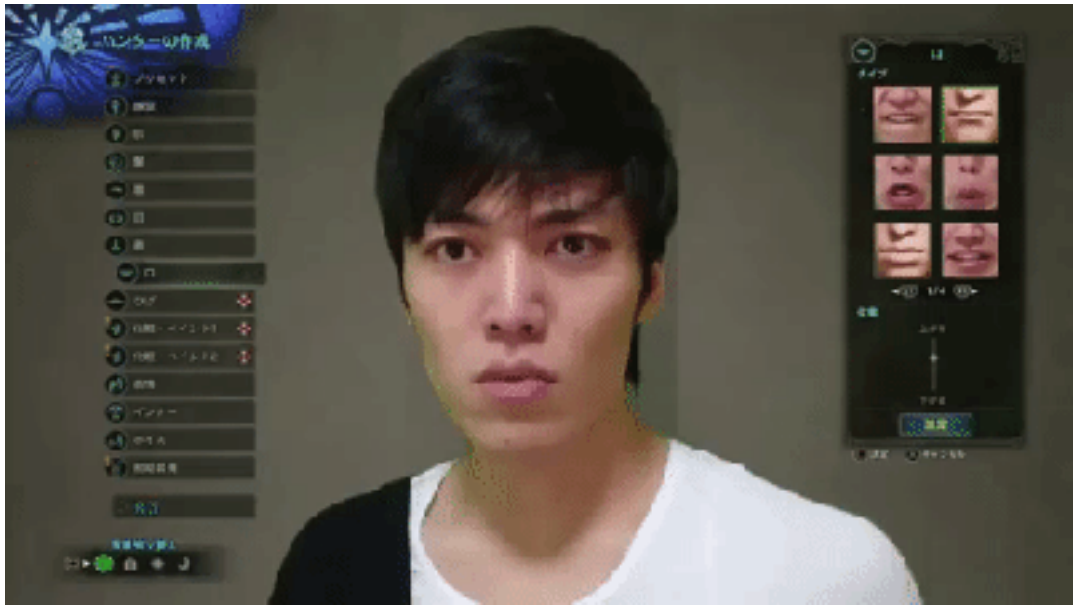
$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$p_{\theta^*}(z)$



- Should be simple, easily to sample from: Normal
- Each component should be independent: Covariance
  - Encourages features that are semantic



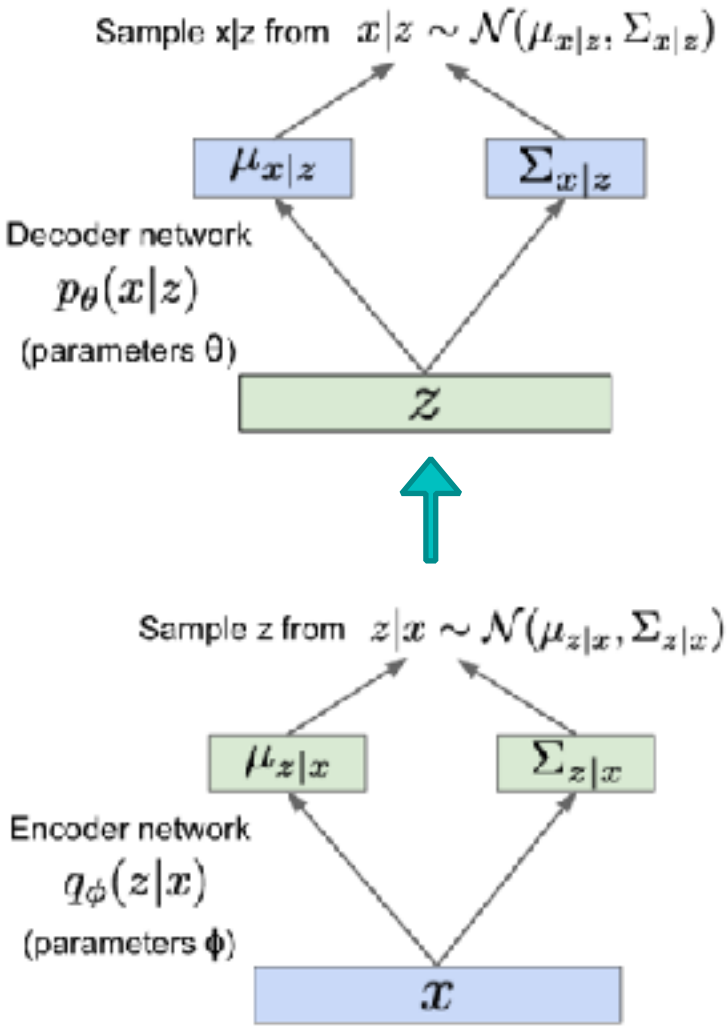
Optimizing

- We want generated samples from latent space to be as close as possible to the true p(x)…
- How can we optimize this?

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- We can't! Intractable computation for every "z"
- So let's define this with variational inference:
  - Only needs to work for z with observed $x^{(i)}$
  - Encode observed $x^{(i)}$ using network q,
    - so $q = p(z \mid x^{(i)})$
  - Optimize decoder to minimize reconstruction

Need a new formulation



if we optimize **z**, this is MLE

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

$$= \sum_{z \in \mathcal{Z}|x^{(i)}} q(z \,|\, x^{(i)}) \cdot \log p(x^{(i)})$$

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

Need a new formulation

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

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

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

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

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

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

What have we really done here? Is this still MLE?

The Loss Function

Maximize through
Error of Reconstruction
This is just negative cross entropy

Here we
assume p(z) is Normal
because we like Normal

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\mathrm{tr}(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log\det(\Sigma(X))\right)$$

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \log\prod_k \Sigma(X)\right)$$

$$= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \sum_k \log\Sigma(X)\right)$$

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

$$= \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

The Loss Function

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

Maximize through
Error of Reconstruction
This is just negative cross entropy

Here we
assume p(z) is Normal
because we like Normal

$$= \frac{1}{2}\sum_{k}\left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

covariance is not numerically stable because of underflow

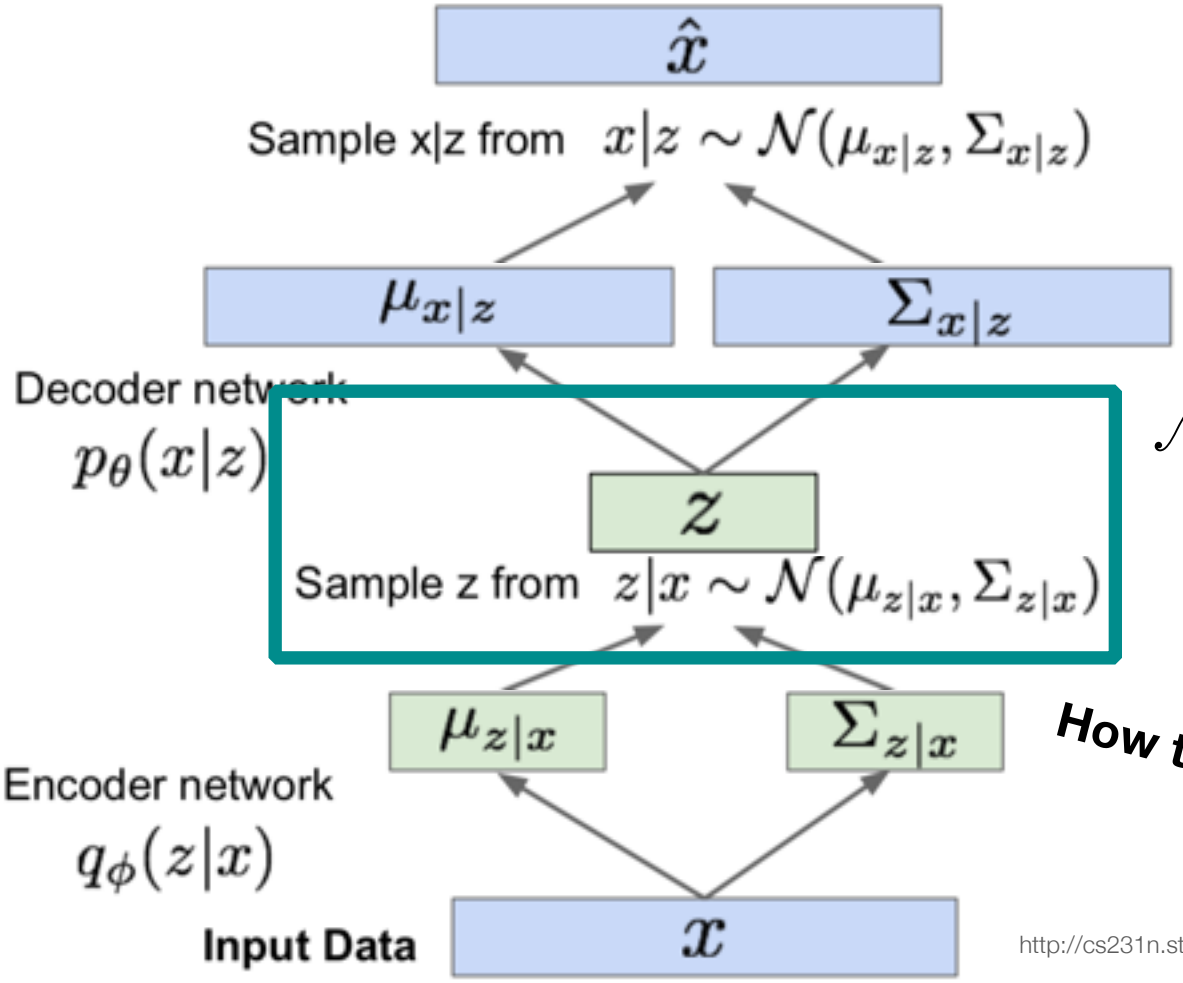
$$D_{KL}[N(\mu(X), \Sigma(X)) \| N(0,1)] = \frac{1}{2}\sum_{k}\left(\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X)\right)$$

so we actually optimize log variance
(remember we assume diagonal covariance)

https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

Back Propagating

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



This is partially differentiable by chain rule…

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

How to back propagate?

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

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)
```

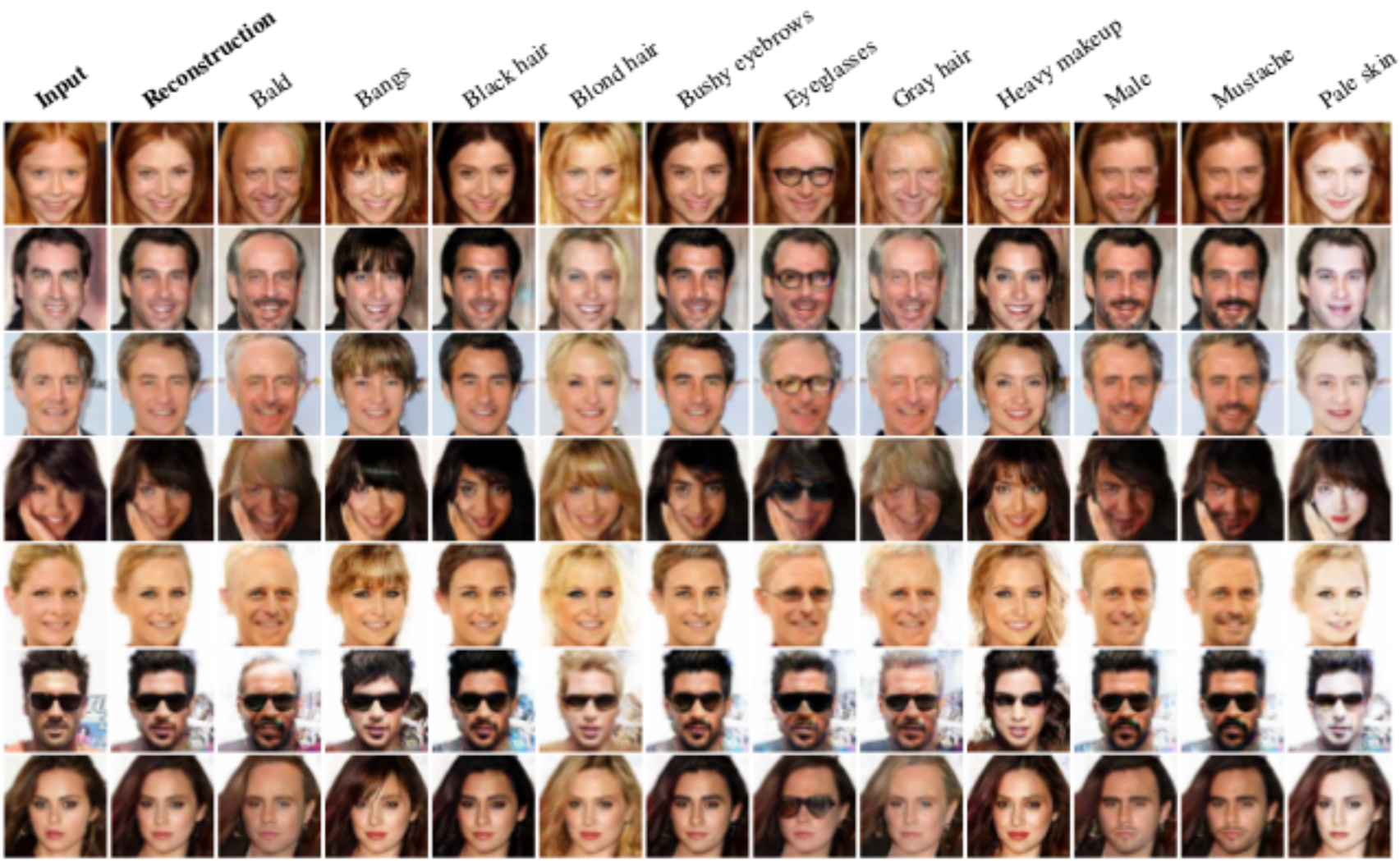
$$\mathcal{N}(\mu, \Sigma) = z$$
$$= \mu(x^{(i)}) + \Sigma(x^{(i)}) \cdot \mathcal{N}(0,1)$$

```
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)
```

$$\mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)} \mid z)\right] - D_{KL}\left[q(z \mid x^{(i)}) \mid\mid p(z)\right]$$
$$= \mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)} \mid z)\right] - \sum_{k} \exp(\Sigma(x^{(i)}) + \mu(x^{(i)})^2 - 1 - \Sigma(x^{(i)})$$

VAE Examples

VAE Examples



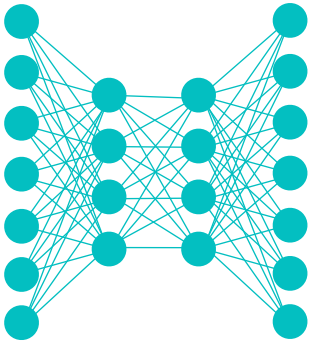
Lecture Notes for

Neural Networks and Machine Learning

Generative Networks

Next Time:
GANs

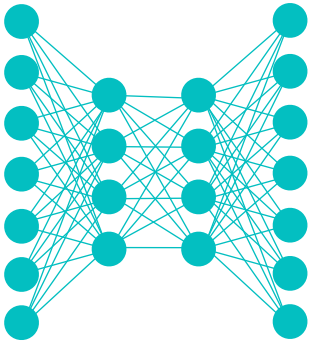
Reading: Chollet CH8



Lecture Notes for

Neural Networks and Machine Learning

Generative Adversarial Networks



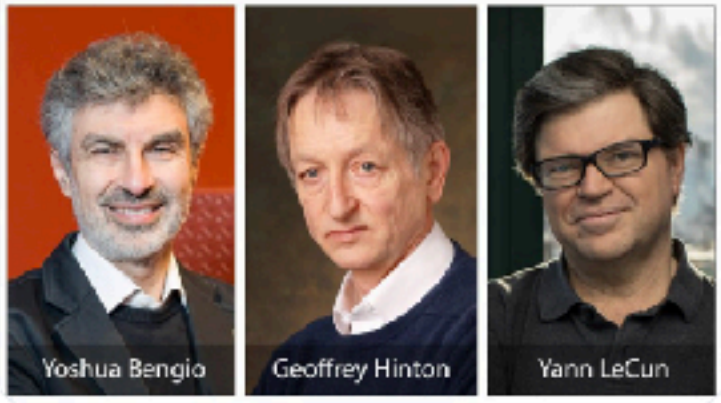
Logistics and Agenda

- Logistics
  - None!
  - How is the ChEMBL filtering going?
- Agenda
  - Working with ChEMBL
  - VAE Demo
  - Simple Generative Adversarial Networks

Working with ChEMBL

Official ACM @TheOfficialACM

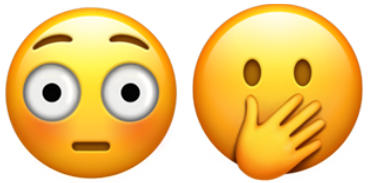
Yoshua Bengio, Geoffrey Hinton and Yann LeCun, the fathers of #DeepLearning, receive the 2018 #ACMTuringAward for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing today. bit.ly/2HVJtdV



Olivier Grisel liked

Volodymyr Kuleshov @volkuleshov · 11h

Congratulations! I hope Jurgen Schmidhuber is not too upset that he wasn't included. Could lead to some awkward questions at the award ceremony ;)



Working with ChEMBL Tables

- **Assays**: need assay type to be binding affinity
- **Activities**:
  - Has **IC50** column, **molregno**, and target (**record_id**)
  - Should calculate “top” records here
  - Do filtering here to find needed compounds
- **Compound_Structures**
  - Has **molregno** and **canonical_smiles** representation
- Building final dataset is simply a join operation on calculated fingerprints and binary IC50 results

Working with ChEMBL

activities

activity_id	assay_id	doc_id	record_id	molregno	star	standard_value	standard_units	stan	standard_type
31863	54505	6424	206172	180094	>	100000	nM	1	IC50
31864	83907	6432	208970	182268	=	2500	nM	1	IC50
31865	88152	6432	208970	182268	>	50000	nM	1	IC50
31866	83907	6432	208987	182855	=	9000	nM	1	IC50

assays

assay_id	doc_id	description	assay_type
1	11087	The compou	B
2	684	Compound w	F
3	15453		B
4	17841	Binding affin	B

compound_structures

molregno	mo	sta	sta	canonical_smiles
1		InC	OV	Cc1cc(ccc1C(=O)c2ccccc2Cl)N3N=CC(=O)NC3=O
2		InC	ZJ	Cc1cc(ccc1C(=O)c2ccc(cc2)C#N)N3N=CC(=O)NC3=O
3		InC	YO	Cc1cc(cc(C)c1C(O)c2ccc(Cl)cc2)N3N=CC(=O)NC3=O
4		InC	PS	Cc1ccc(cc1)C(=O)c2ccc(cc2)N3N=CC(=O)NC3=O
5		InC	KE	Cc1cc(ccc1C(=O)c2ccc(Cl)cc2)N3N=CC(=O)NC3=O

Back to VAEs

Geoffrey Hinton @geoffreyhinton · 23h

Replying to @JeffDean @ylecun and @TheOfficialACM

Thanks to my graduate students and postdocs whose work won a Turing award. Thanks to my visionary mentors Inman Harvey, David Rumelhart and Terry Sejnowski. And thanks to Jeff Dean for creating the brain team that turns basic research in neural nets into game-changing products.

40 313 2,208

Yann LeCun @ylecun · 20h

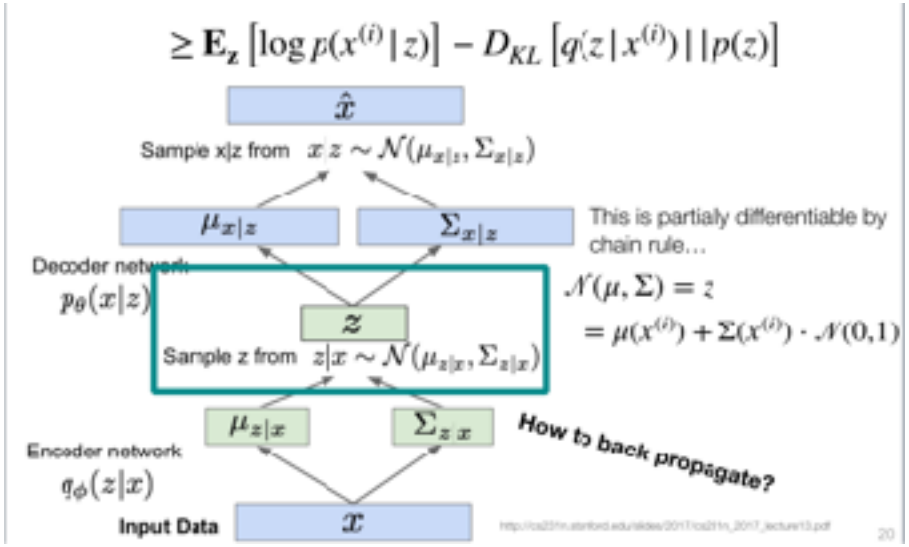
Congratulations Geoff, from one of your former postdocs ;-)

3 11 312

This is why we like Hinton.

Last Time

- Variational auto encoding

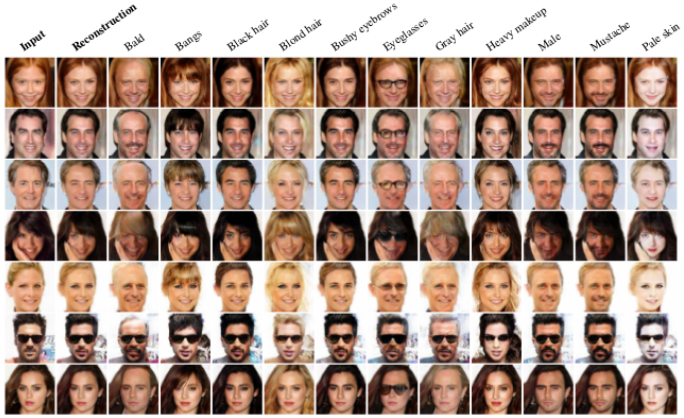



```
# Encode the input into a mean and variance parameter
z_mean, z_log_variance = encoder(input_img)

# 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)
```

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

```
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)
```

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



VAEs in Keras

Implementation from Book on MNIST

Demo by Francois Chollet

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

The Simple GAN

Geoff Hinton after writing the paper on backprop in 1986

Latent Variable Approximation with GANS

- **Idea**: transform random input data into target of interest
  - Like an image!
- Rather than training an encoder with variational inference, we need a transformation protocol
  - Transformer will be a…
  - Neural Network (of course!)
- Transformer is a “complex” sampling method
- How to optimize and provide training examples?
  - Use two Neural Networks!
  - … Deep Learning is like the chiropractory of the machine learning…

Generative Adversarial Network

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

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

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

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

only portion dependent on g
generator minimizes

everything dependent on d
discriminator maximizes

Taking turns

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

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

- If only the problem was convex! This would be easy to solve…
- But v(g,d) is not convex, not even close
  - Saddle points, saddle points everywhere
- Taking turns on the gradient descent will probably never reach equilibrium
  - There is no convergence guarantee
  - But practically we like when discriminator == chance

Practical Objectives

- Discriminator objective, gradient ascent:

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

- Generator objective, gradient descent:

$$\min_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x))\right]$$

- But **practically** generator is really hard to train, amplified by a decreasing gradient
- 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:

$$\min_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x)) \right]$$

- 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 its wrong?
    - not trying to win, just make the other person lose

$$\max_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(d(x)) \right]$$ **No longer a zero sum game?!**

Final Loss Functions

- Discriminator:

$$\max_{d} \mathbf{E}_{x \leftarrow p_{data}} \left[\log d(x)\right] + \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x))\right]$$

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

- Generator:

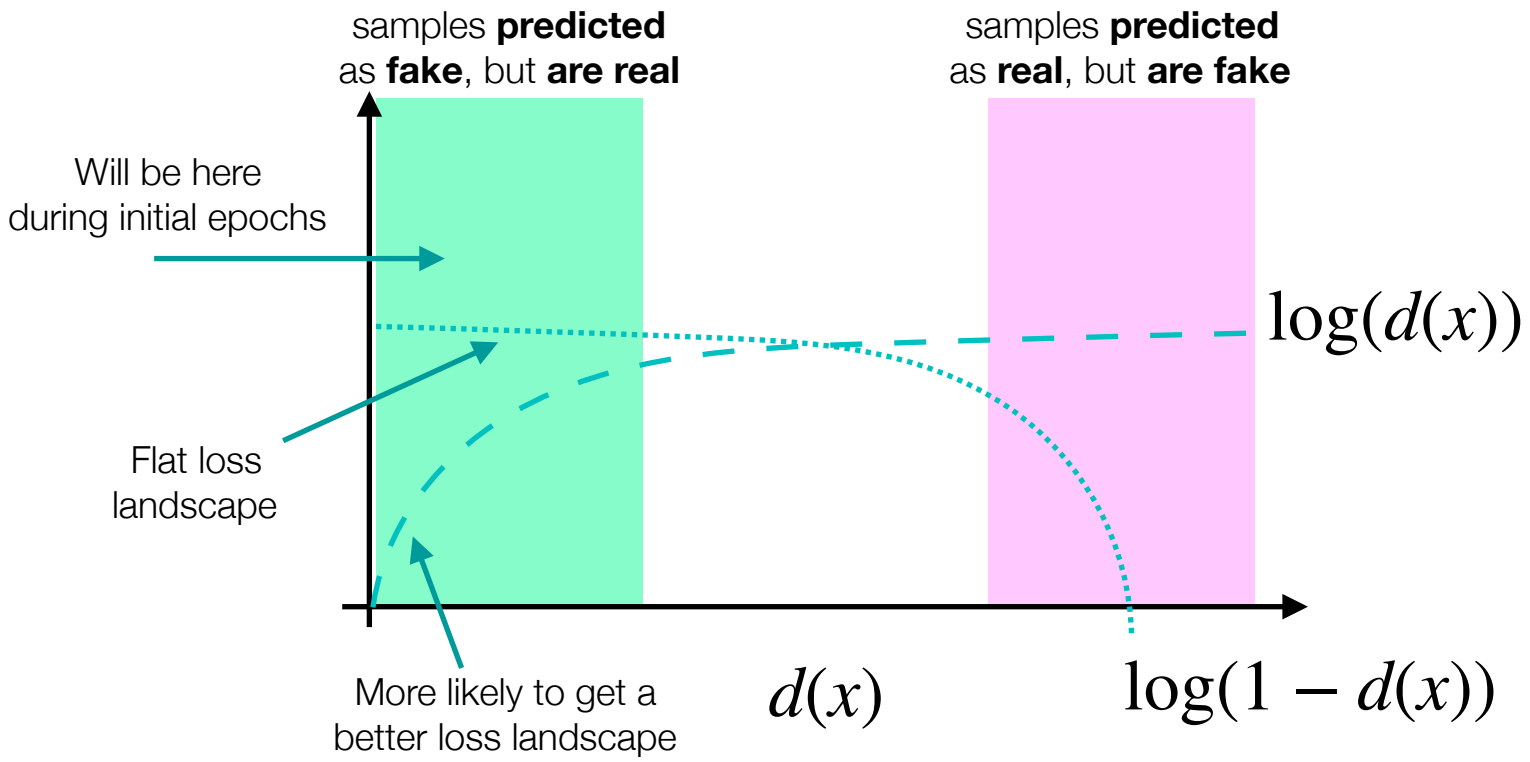
$$\max_{g} \mathbf{E}_{x \leftarrow g(z)} \left[\log(d(x))\right]$$

Freeze Discriminator Weights

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

Why maximize incorrect?

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



How to Train your (dra) GAN

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

"GAN-splaining with Ian"

—Geoffrey Hinton, Probably

Simple Training Approach

- Some caveats on why training will be difficult:
  - The latent space discovered by the Generator has almost no guarantees regarding continuity and structure (different than VAEs)
  - Convergence of GAN is not possible, only equilibrium
    - even saying discriminator is chance is not enough!
  - Each iteration through, the entire loss function changes because generator changes
    - Also we are sampling different points from generator…

Simple Training Approach

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 $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - 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

Does this work?! **Not really! Why?**

Goodfellow et al. “Generative Adversarial Networks” (NeurIPS 2014)

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 (even though no assumption of Gaussian in GAN formulation)
- Random is more robust
  - GANs get stuck a lot
- Sparse gradients are not your friend here
  - No max pooling, no ReLU 😳
- Make encoder and decoder symmetric
  - Unequal pixel coverage (checkerboards)

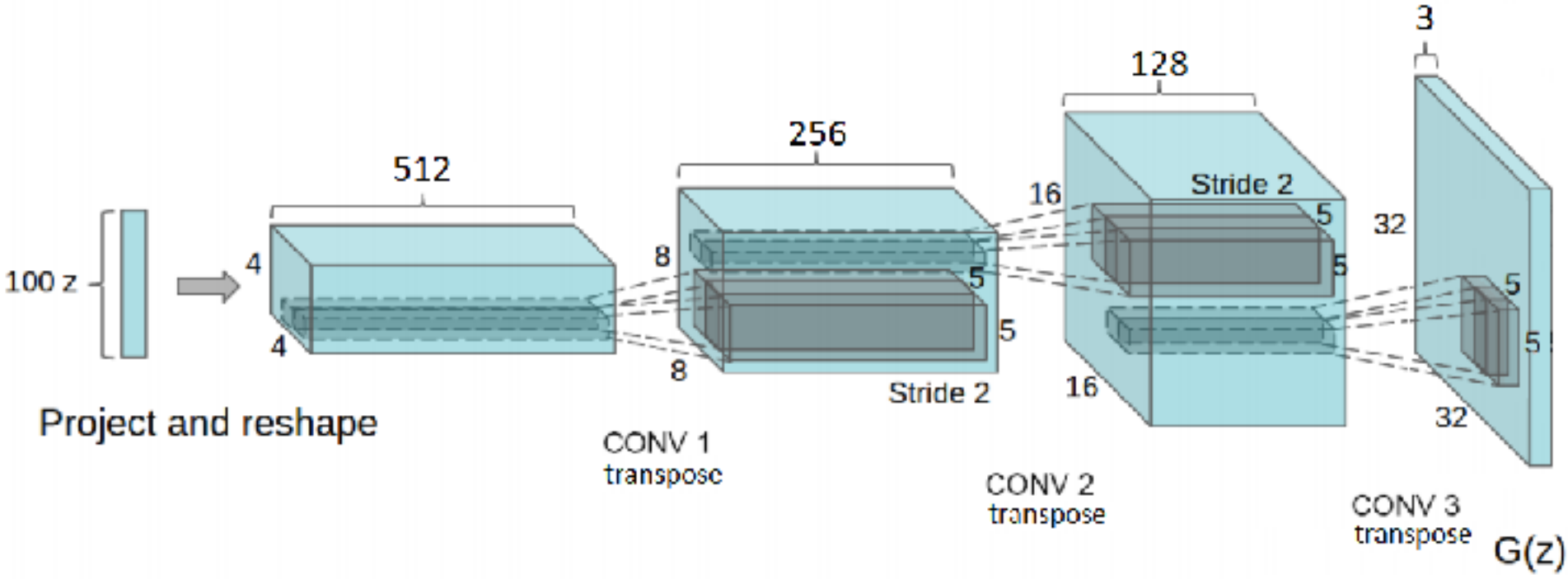
Some Results of Generation

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

Highlight: Nearest Training Sample

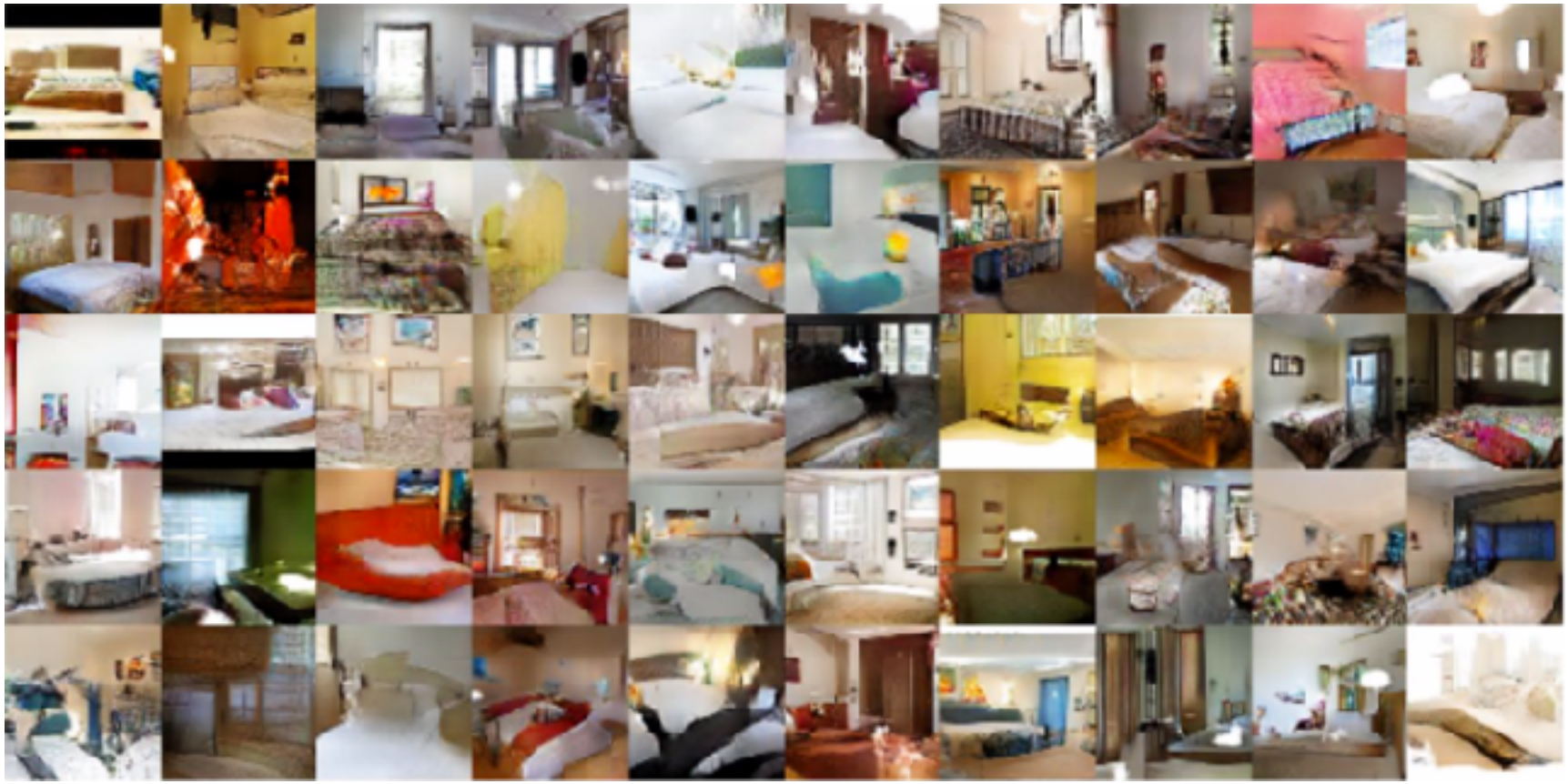
Deep Convolutional GANs

- Just need to reshape and use upsampling



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

Some Results of Generation



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

GANs in Keras

Implementation from Book on "Frogs from CIFAR"

Demo by Francois Chollet

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

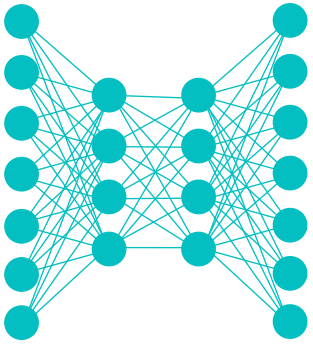
Lecture Notes for

Neural Networks and Machine Learning

GANs

Next Time:
Practical GANs

Reading: Chollet CH8

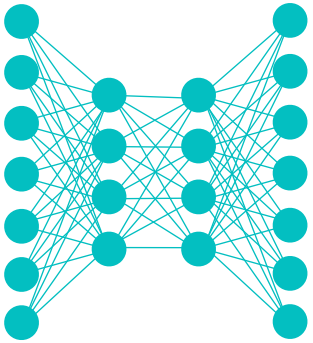


Lecture Notes for

Neural Networks and Machine Learning

Practical

GAN Training



Logistics and Agenda

- Logistics
  - Paper Presentation, Next Lecture: GANs
  - Which paper?
- Agenda
  - Review from Last Time, GAN Demo
  - Training GANs in different loss landscapes
  - Wasserstein GAN
  - GANs for increased classification
  - Adversarial Auto Encoding

Last Time

- Simple GANS

- Discriminator:

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

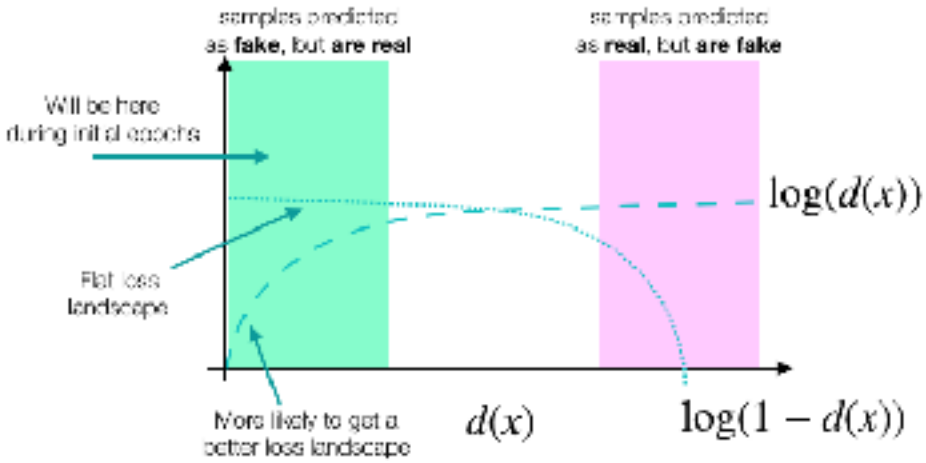
Same as minimizing the binary cross entropy of the model with: (1) labeled data and (2) generated data!

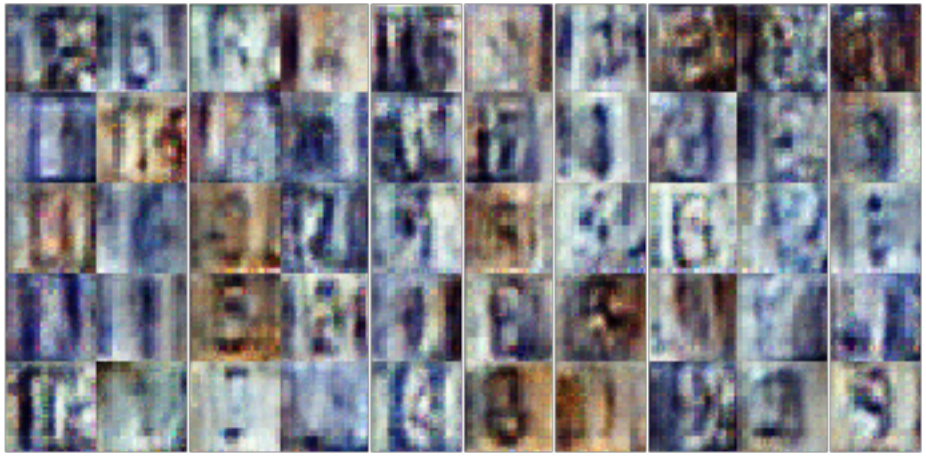
- Generator:

$$\max_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(d(x))\right]$$

Freeze Discriminator Weights

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





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 (even though no assumption of Gaussian in GAN formulation)
- Random is more robust
  - GANs get stuck a lot
- Sparse gradients are not your friend here
  - No max pooling, no ReLU 😳
- Make encoder and decoder symmetric
  - Unequal pixel coverage (checkerboards)

GANs in Keras

Implementation from Book on "Frogs from CIFAR"

Demo by Francois Chollet

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

GANs Loss Landscape

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.

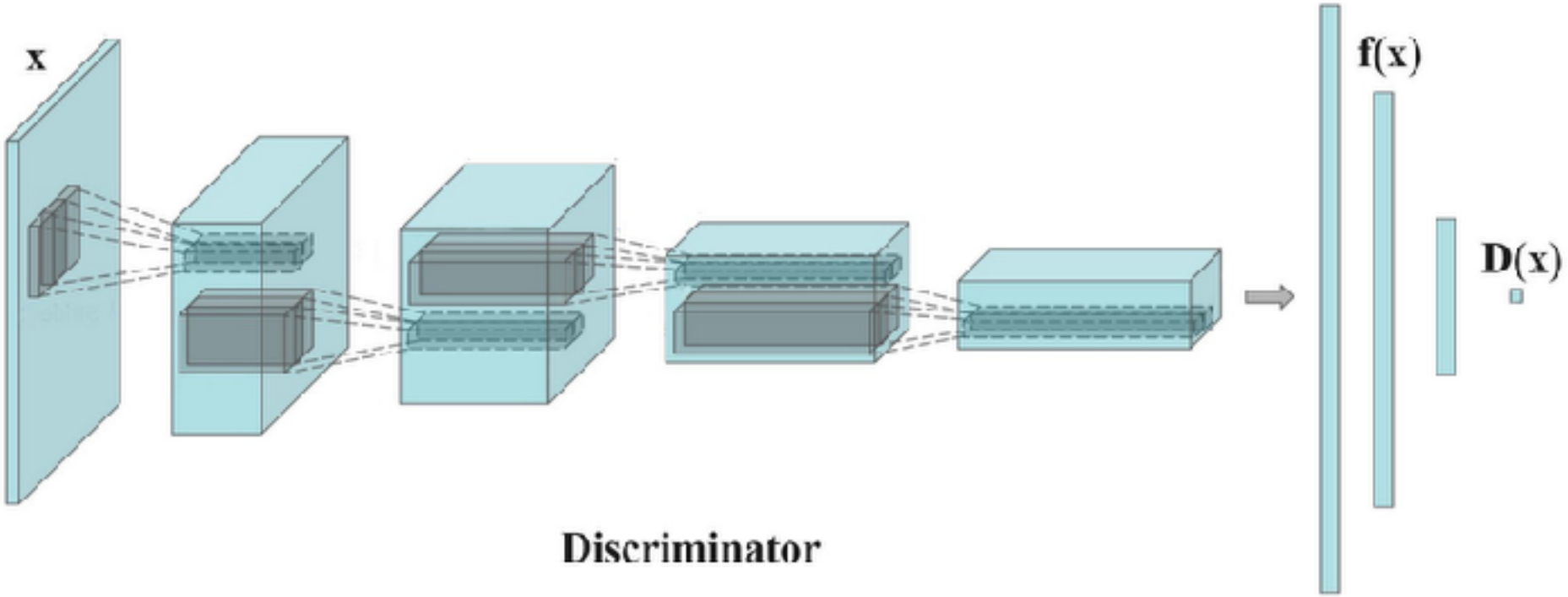
The GAN Loss Landscape

- A collection of Heuristics and Observations that is more Alchemy than Science
  - Feature Matching loss
  - Mini-batch discrimination
  - Historical averaging
  - Virtual Batch normalization
  - Experience Replay
  - Wasserstein Loss (W-GANs…)

There are no GuAraNtees

- From Salimans, Goodfellow, et al., :
  - "*In this work, we introduce several techniques intended to encourage convergence of the GANs game. These techniques are motivated by a heuristic understanding of the non-convergence problem. They lead to improved semi-supervised learning peformance and improved sample generation. We hope that some of them may form the basis for future work, providing formal guarantees of convergence.*"

Before we go too far



Feature Matching

- Loss function for generator is really difficult and unstable
  - hard to optimize based on feedback only from discriminator output!
- Since generator is trying to statistically match features inside the discriminator (to fool it)
  - try to make generator match statistics of discriminator activations

$$\|\mathbf{E}_{x \leftarrow data}[\mathbf{f}(x)] - \mathbf{E}_{x \leftarrow q(z)}[\mathbf{f}(G(z))]\|_2^2$$

mean discriminator activations from **real** data

mean discriminator activations from **fake** data

Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

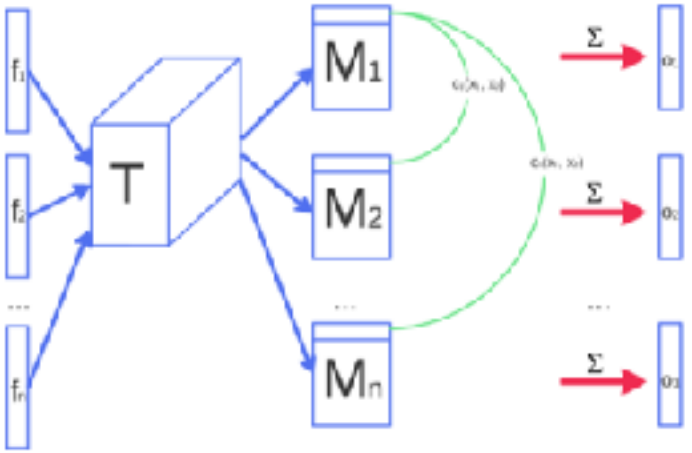
Mini-Batch Discrimination

- Mode collapse of generator happens when parameters emit the same point $G(z)$
- Solution: Incentivize dissimilarity of generated samples by letting discriminator see batches (detect mode collapse early, incentivize generator to prevent it)
- Use discriminator activation $\mathbf{f}(\ G(z)\ )$

$$o(x_i) = \sum_{j \in Batch} \exp\left(-\|\mathbf{f}(x_i) \cdot \mathbf{T} - \mathbf{f}(x_j) \cdot \mathbf{T}\|\right)$$

i^{th} sample in batch

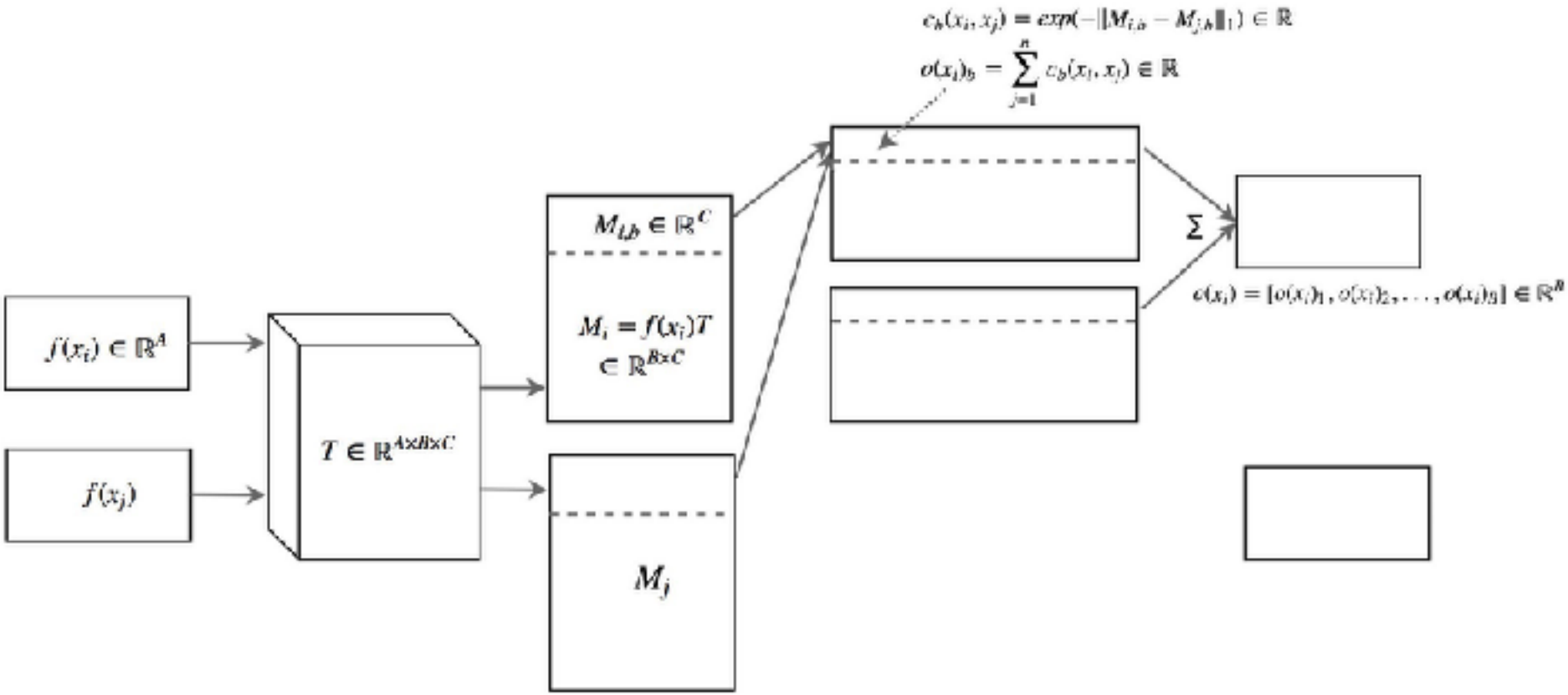
j^{th} sample in batch



- Concatenate and Feed $o(x_{1:n})$ as a feature into next discriminator layer

Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

Mini-Batch Discrimination



Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

Historical Averaging

- Because we cannot converge, parameters should constantly be exploring different solutions
- But mode collapse and other issues might cause parameters to converge
- Solution: Incentivize changes directly in loss functions

$$\left\| w[T] - \frac{1}{t} \sum_{i=T-n}^{T-1} w[t] \right\|^2$$

Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

Virtual Batch Normalization

- Batch normalization is really swell
- But for GANs, it makes current x's dependent on other generated x's (and it was already hard to train…)
- Why not use a reference batch for statistics? And just use statistics of the reference batch activations for normalizing?
  - Every time we update W's in the network, we need to recompute the statistics for the reference batch
  - That's it!

$$\mu^{ref} = \frac{1}{M}\sum_i a_i^{ref} \quad \sigma^{ref\,2} = \frac{1}{M}\sum_i (a_i^{ref} - \mu^{ref})^2 \quad z_i = \gamma \cdot \frac{(a_i - \mu^{ref})}{\sqrt{\sigma^{ref\,2} + \epsilon}} + \beta$$

Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

Experience Replay

- The discriminator can overpower the generator quickly and it over-learns to a particular epoch
- Solution: feed it more than just the current epoch
- Save previous epochs and randomly grab from them when updating the generator!
- Grab a coffee and chill…

Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

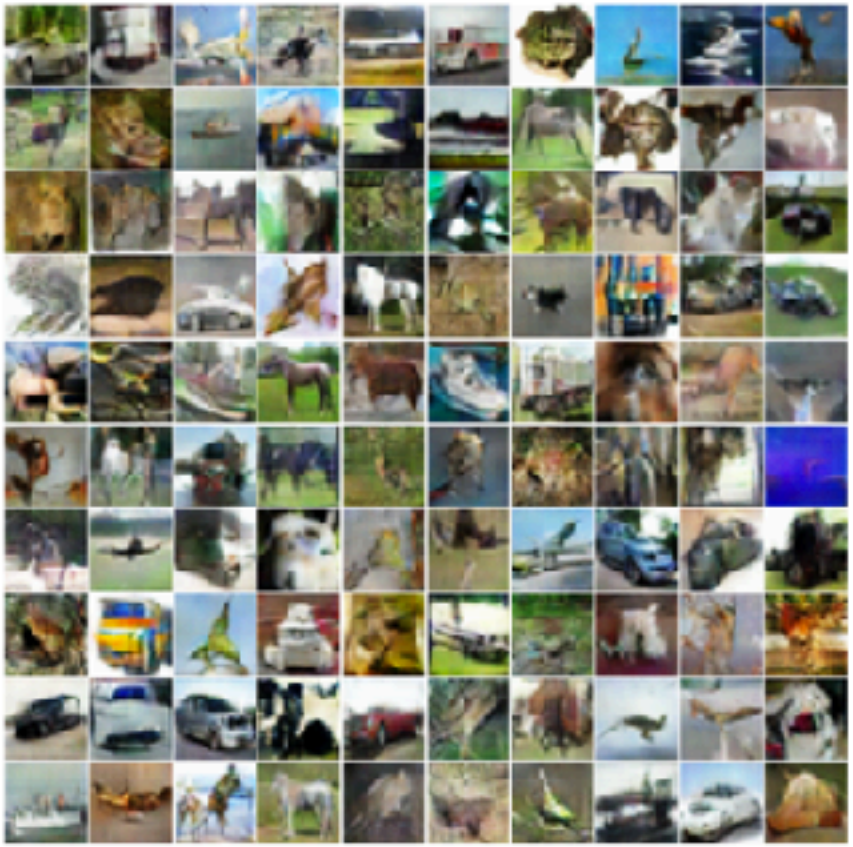


Figure 4: Samples generated during semi-supervised training on CIFAR-10 with feature matching (Section 3.1, *left*) and minibatch discrimination (Section 3.2, *right*).

Change the loss function entirely

Name	Value Function
GAN	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_n \leftarrow clip\_by\_value(W_n, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\|\nabla D(\alpha x - (1-\alpha G(z)))\| - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\|\nabla D(\alpha x - (1-\alpha x_p))\| - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	$L_D^{CGAN} = E[\log(D(x,c))] + E[\log(1 - D(G(z),c))]$ $L_G^{CGAN} = E[\log(D(G(z),c))]$
InfoGAN	$L_{D+Q}^{InfoGAN} = L_D^{GAN} - \lambda L_1(c,c')$ $L_G^{InfoGAN} = L_G^{GAN} - \lambda L_1(c,c')$
ACGAN	$L_{D+Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\|x)] + E[P(class = c\|G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\|G(z))]$
EBGAN	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) = \lambda \cdot PT$
BEGAN	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

The best loss function to use:

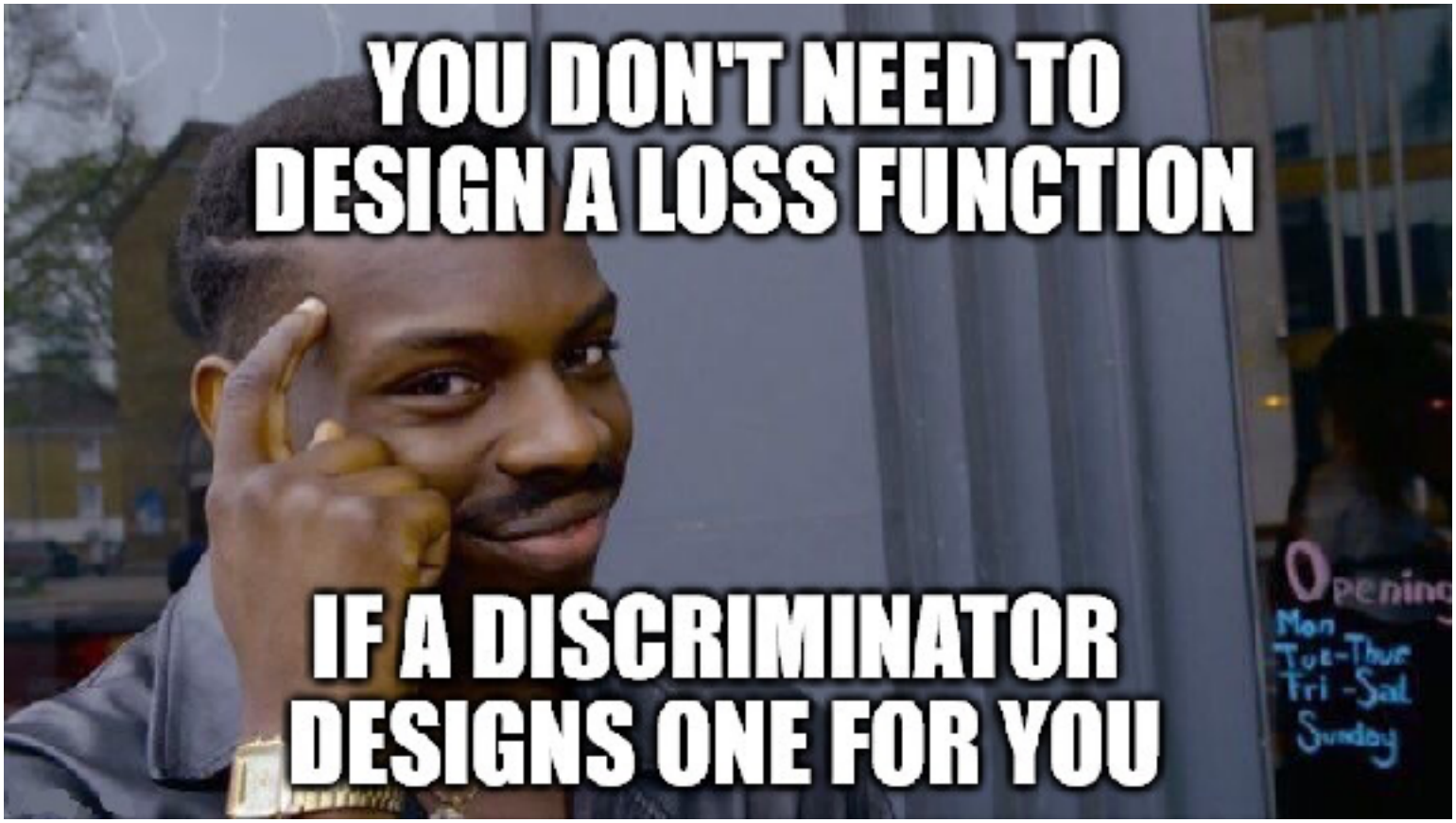
¯_(ツ)_/¯

	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	[illegible]	[illegible]	[illegible]	[illegible]
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	[illegible]	[illegible]	[illegible]	[illegible]
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

Academic blasphemy:
Maybe it doesn't really matter

https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

Wasserstein GANs



Caveat

- Wasserstein Loss is actually very useful
- But the mathematics pinning it down are based on approximations of approximations
- So its not really true to say its actually working for the motivated reasons
- But we cannot say that isn't why it working either
- So until we have a better understanding of the mechanisms, we need to go through the math motivated in the paper

Wasserstein

- *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.*
- That should help training even when the generator and discriminator are not matching
- …but Wasserstein Distance is not tractable to calculate so its time to start approximating!

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

The Optimal Discriminator

- Discriminator is maximizing:

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

$$\max_d \int_{x \in [P_{data}, P_g]} \left(p_{data}(x) \cdot \log d(x) + p_g(x) \cdot \log(1 - d(x)) \right) dx$$

$$\nabla_f = 0 = \frac{1}{\partial d(x)} \left(p_{data}(x) \cdot \log d(x) + p_g(x) \cdot \log(1 - d(x)) \right)$$

... math ...

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

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

Generator is optimal when

p_g and p_{data} are close together.

So we want to optimize the overlap of these distributions

Wasserstein Distance

- How much do I need to change one distribution to get it to match another?
- Could use KL divergence, but we already know that has many problems associated with vanishing gradients
- We will only have samples from the distributions (not the actual equations)
- **Wasserstein Distance** for continuous probabilities:

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

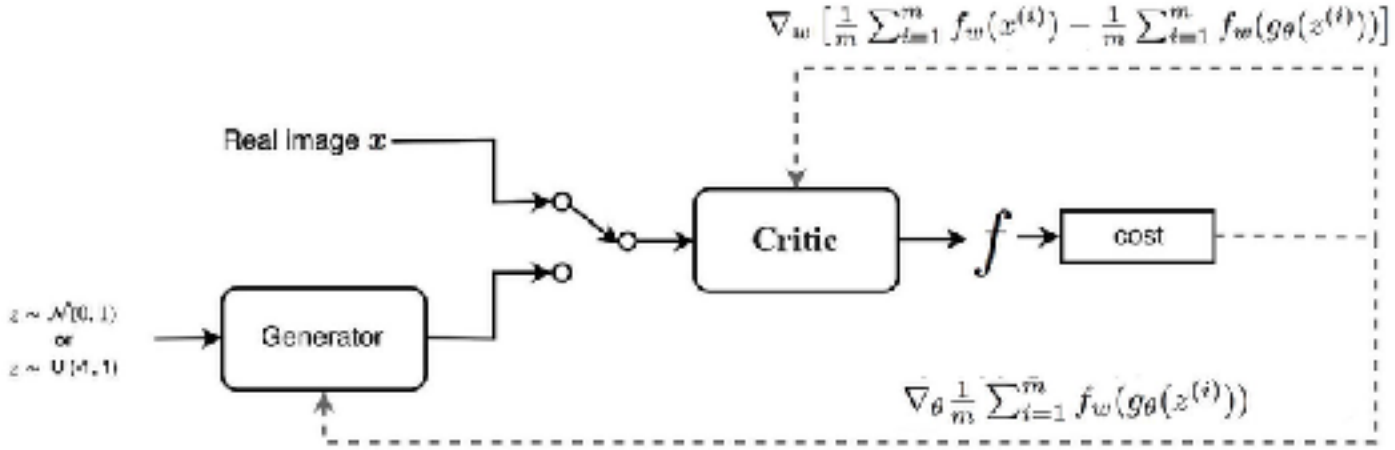
- $\inf$ is greatest lower bound
- gamma is completely and utterly intractable to compute

Wasserstein Duality

- 1-Lipschitz constraint: $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$.
- Wasserstein Duality Formula:

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

- where $\sup$ is the least upper bound (which we cannot find directly, so we assume its a maximization)



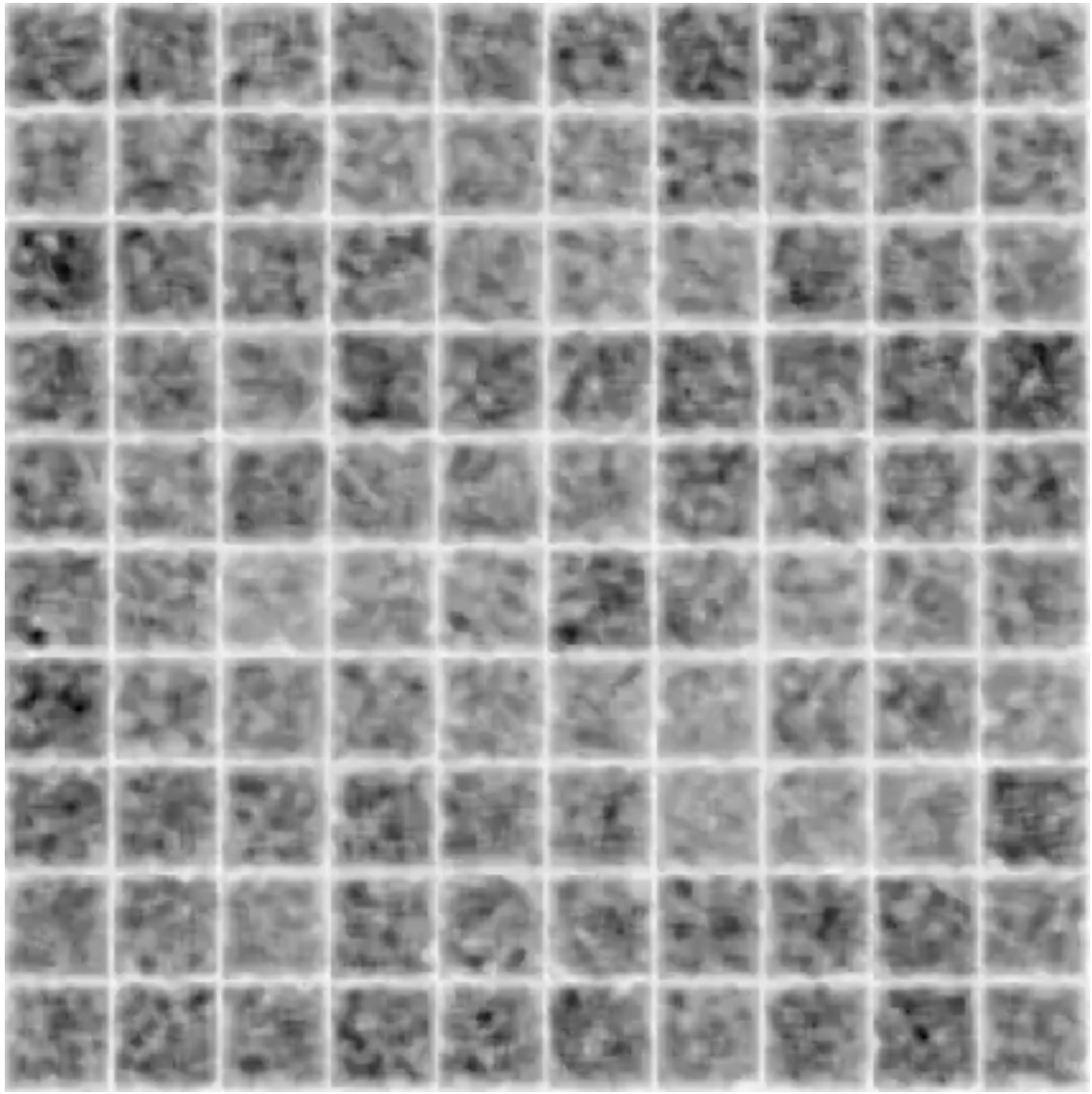
https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

Practical Wasserstein Loss

- Discriminator (critic) becomes **f**, and its output can be Lipschitz if all weights are small (squashes inputs)
  - so we clip them to -c to c (c~0.01)
  - critic output will be linear (and is now a measure of Wasserstein distance from samples input)
  - and maximize for discriminator: $\nabla_w \frac{1}{m}\sum_{i=1}^{m}\left[f(x^{(i)}) - f(G(z^{(i)}))\right]$
  - and maximize for generator: $\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f(G(z^{(i)}))$
- In their empirical findings, no mode collapse problems
- And training longer resulted in good quality images (motivates that distributions overlap)
- Still some some of the most competitive GAN results

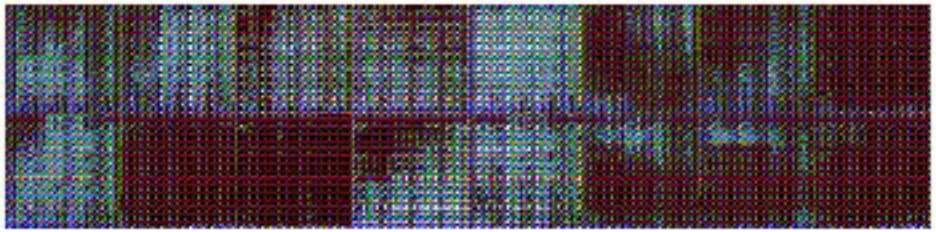
https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

WGAN Example from Keras (MNIST)

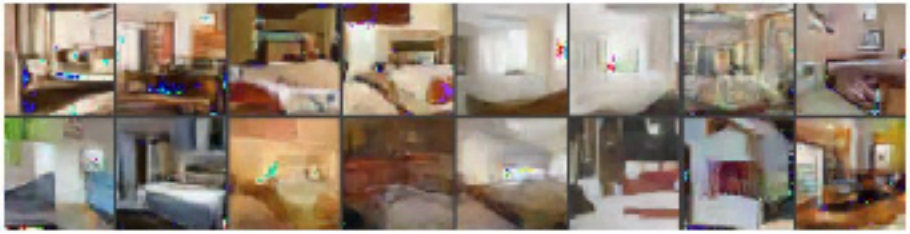


WGAN with DCGAN generator

GAN with DCGAN generator



Without batch normalization &
constant number of filters at each layer



Using a MLP as the generator

All critics and discriminators follow the same discriminator design in DCGAN

So we should always use Wasserstein!

- **Actually not really**
- Its really, really slow
- Clipping:

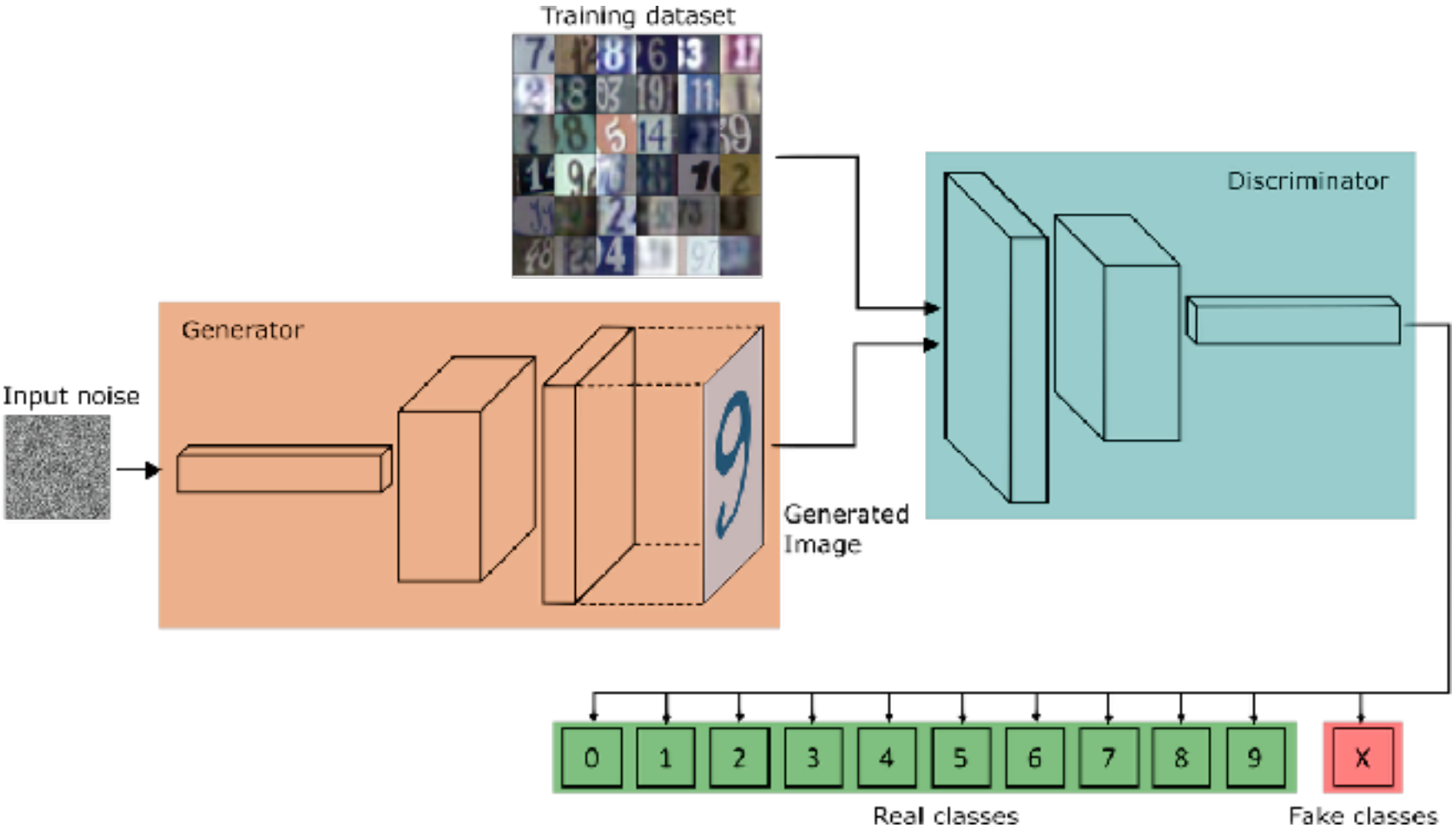
Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing 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, which also adds training time (lots of time)

GANs for more than Generating Images

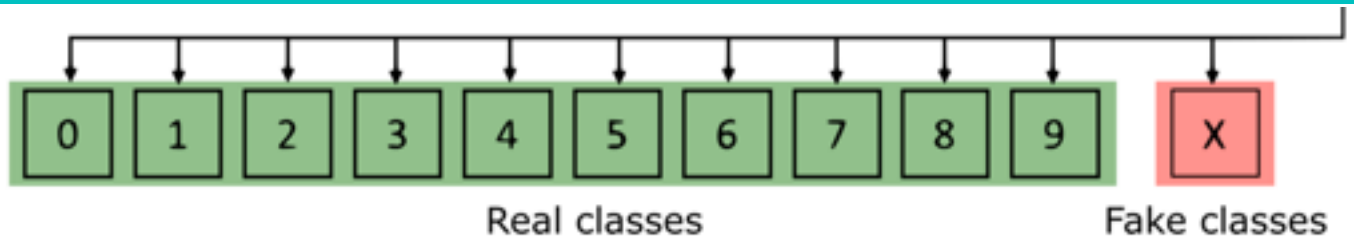
hadou-GAN

Fewer Labels, Still Lots of Data



https://sthalles.github.io/semi-supervised-learning-with-gans/

Minimizing Labels

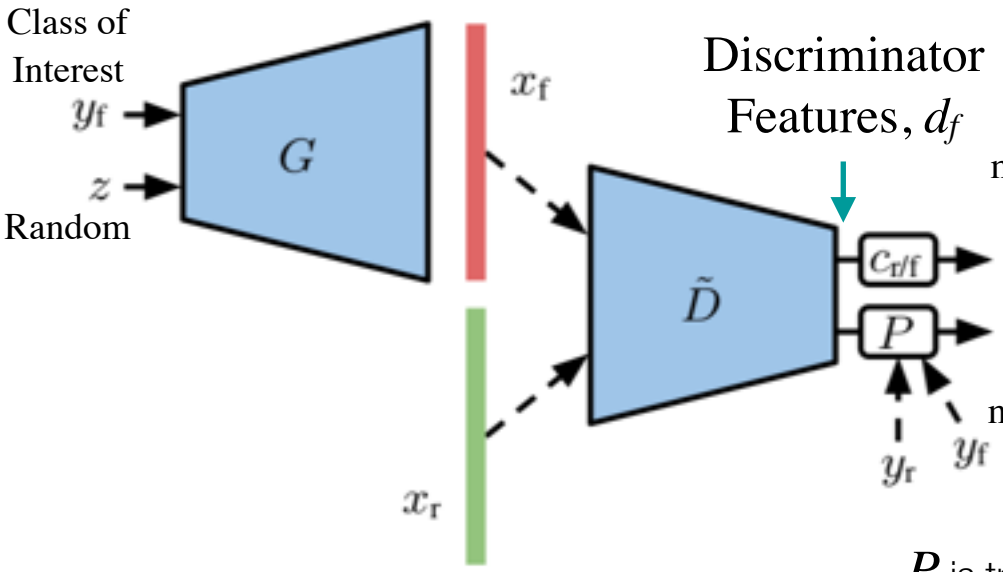


- MNIST Performance
  - 60,000 labels → 1,000 labels
  - 0.6% Error (~SOA)
- SVHN Performance
  - 73,000 labels → 1,000 labels
  - 1.3% Error (~SOA)
- ImageNet
  - 20% Labels (of 1.3M)
  - 10.3% Top-5 Error (slightly worse than SOA)

Need a slightly more complicated loss function

https://arxiv.org/pdf/1903.02271.pdf

Conditional GANs and Self-Supervision



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

Traditional Discriminator Loss, Goodfellow

Class Conditioned Discriminator Loss

$$\max_d \mathbf{E}_{x \leftarrow p_{data}} \left[\log \tilde{d}(x_r, y_r)\right] + \mathbf{E}_{x \leftarrow g(z, y_f)} \left[\log(1 - \tilde{d}(x_f, y_f))\right]$$

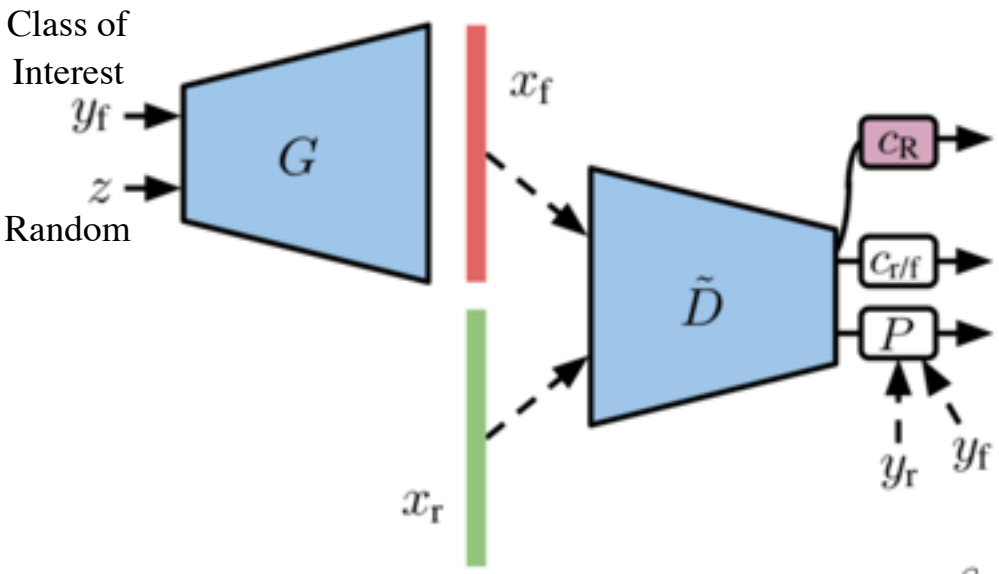
P is trained using outer product

$$\tilde{d} = d_f^T W y_{enc}$$

Lucic, Mario, Michael Tschannen, Marvin Ritter, Xiaohua Zhai, Olivier Bachem, and Sylvain Gelly. "High-Fidelity Image Generation With Fewer Labels." *arXiv preprint arXiv:1903.02271* (2019).

Conditional GANs and Self-Supervision

Lucic, Mario, Michael Tschannen, Marvin Ritter, Xiaohua Zhai, Olivier Bachem, and Sylvain Gelly. "High-Fidelity Image Generation With Fewer Labels." *arXiv preprint arXiv:1903.02271* (2019).



New task, detect rotations in image

All these equations are saying is that they classify rotations from real and fake images.

And… nothing in these equations is meaningful except α, β

$$-\frac{\beta}{|\mathcal{R}|}\sum_{r\in\mathcal{R}}\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log p(c_R(\tilde{D}(x^r)=r)]$$

and

$$-\frac{\alpha}{|\mathcal{R}|}\mathbb{E}_{(z,y)\sim p(z,y)}[\log p(c_R(\tilde{D}(G(z,y)^r)=r)],$$

Summary

- When we care about the discriminator for classification:
  - More unsupervised tasks helps feature extraction
  - Conditioning on classes with outer product helps
  - Many other attempts have been tried…
    - But not many work…
  - Open research topic!

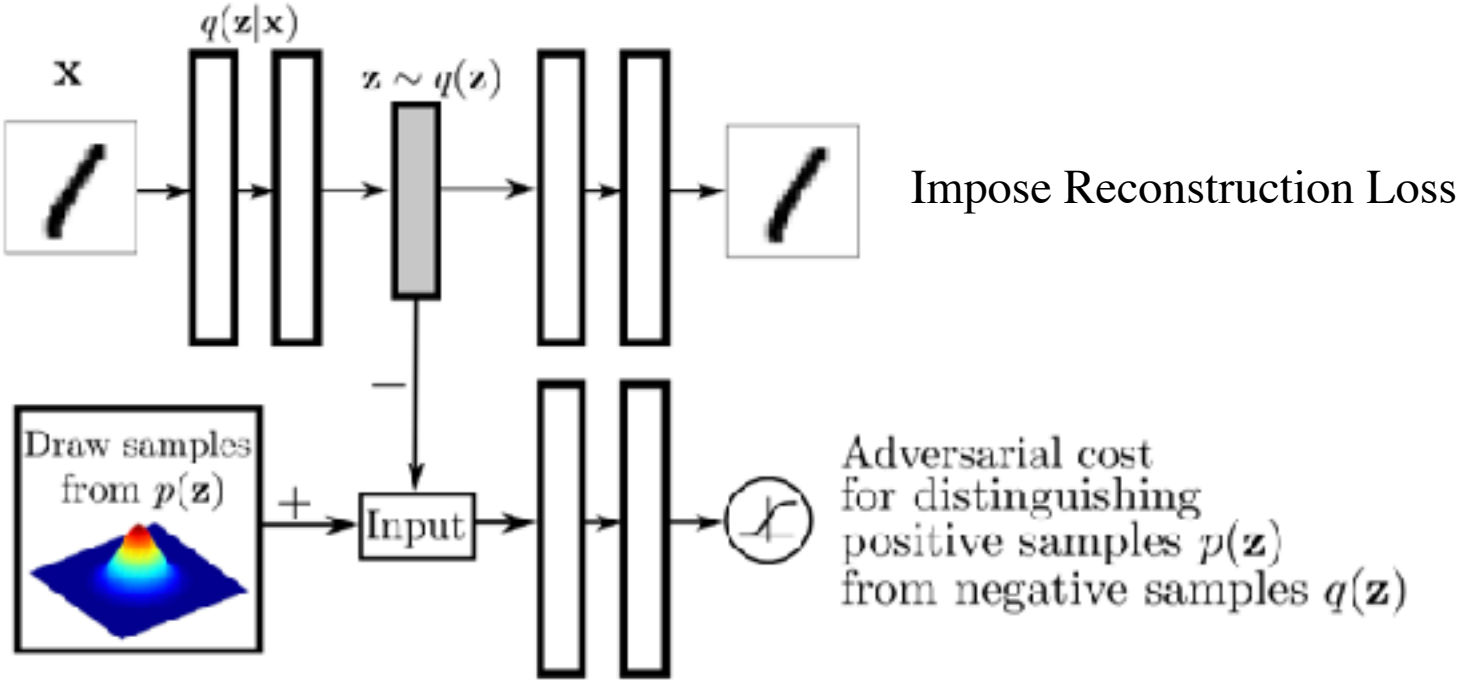
Adversarial Auto Encoding

Do we need something more than VAE?

- Arguments for Yes:
  - ELBO is flawed!
  - Assumption of Normal distributions to q(z) is limiting
  - Training tends to be slow
  - Manifold of distributions do not cover the latent space completely
  - 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.

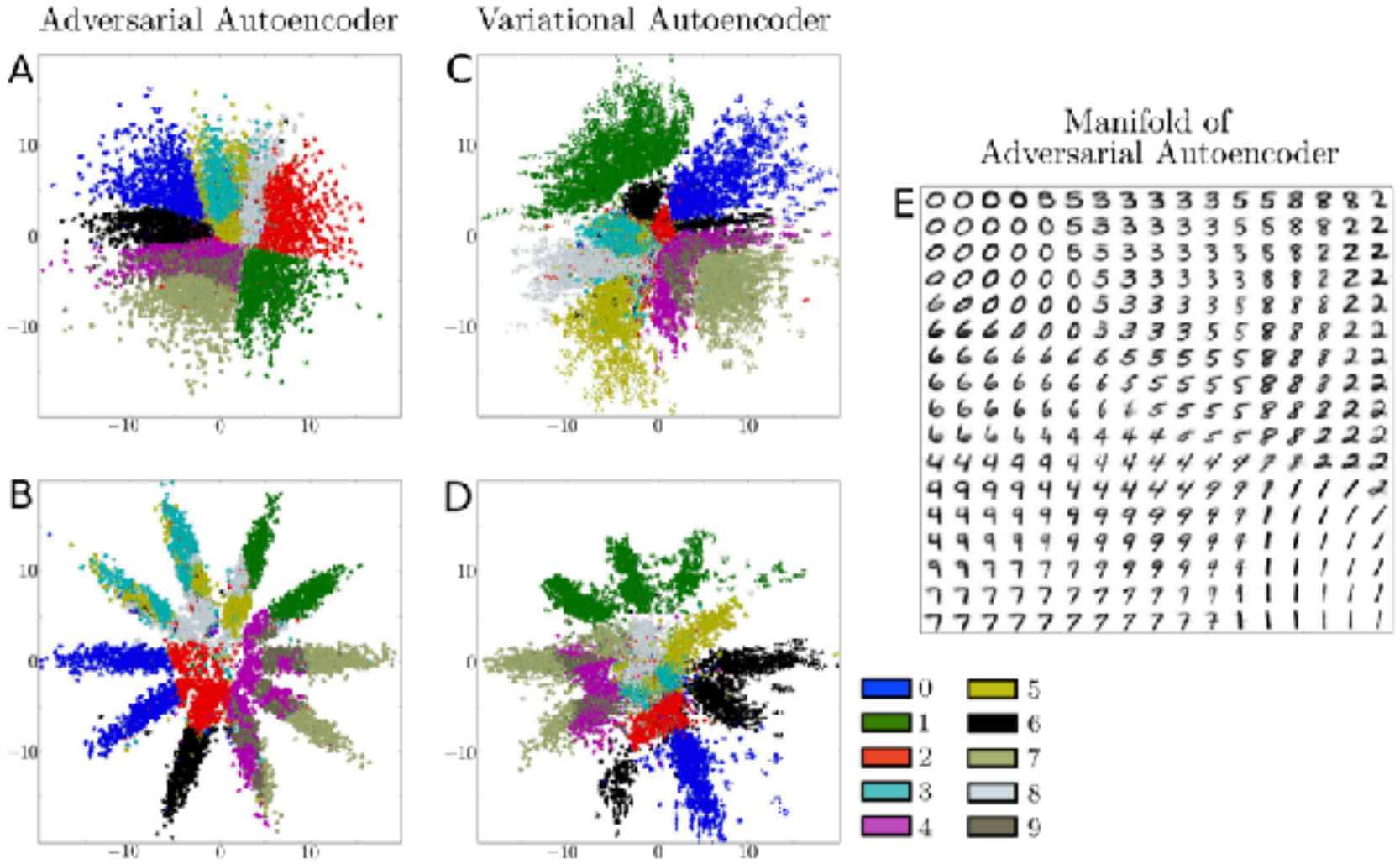
The Main Idea

- How can we enforce constraints on the latent space with GANs?



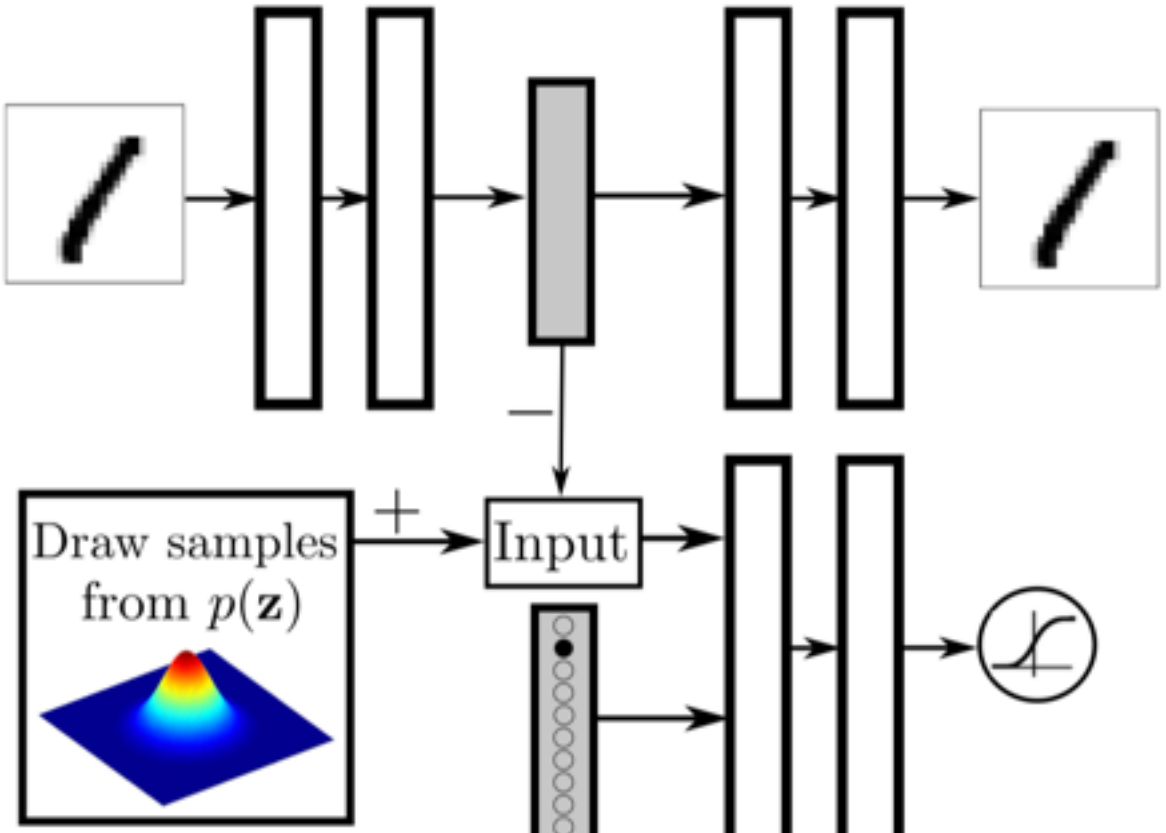
Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, **Ian Goodfellow**, and Brendan Frey. "Adversarial autoencoders." *arXiv preprint arXiv: 1511.05644* (2015).

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

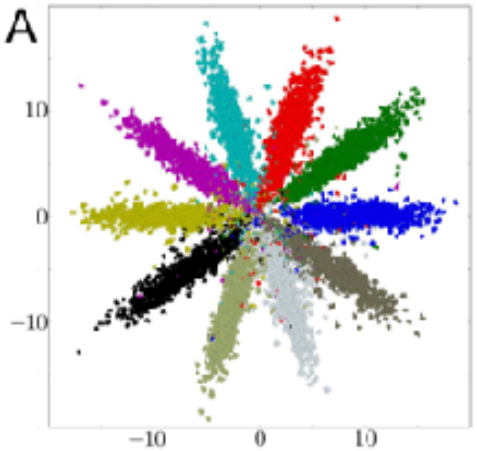


Draw Samples According to Specific Class Prior Distributions

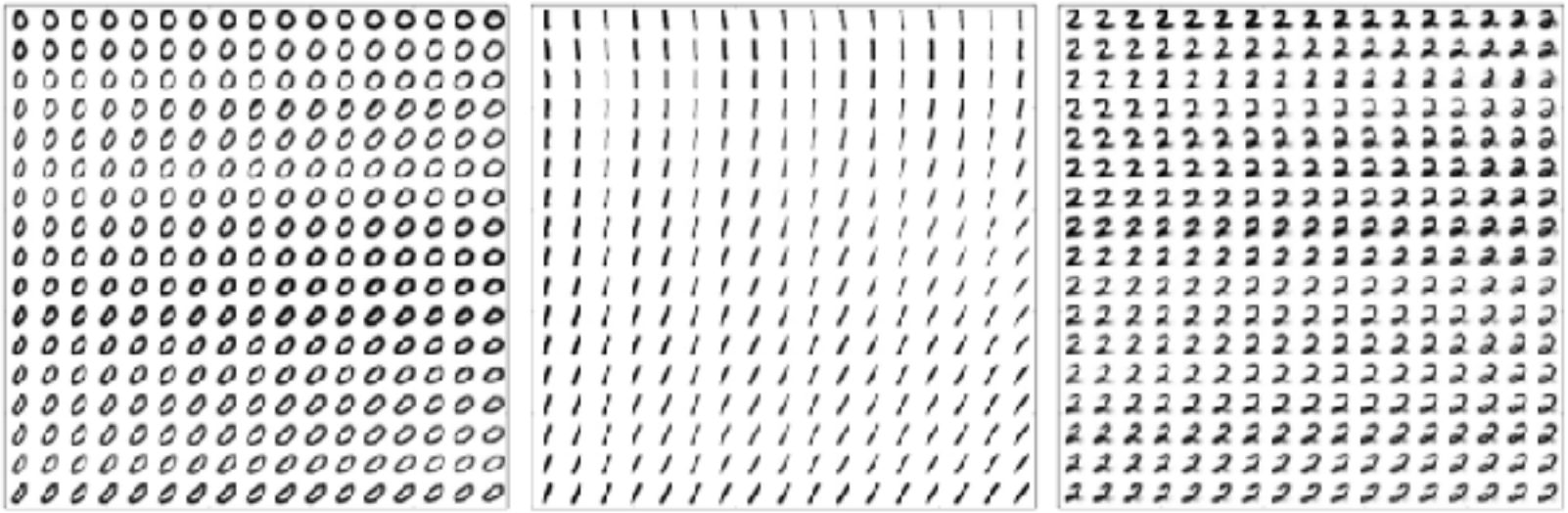
Provide Class Chosen as One Hot Vector in Discriminator

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

Conditional Class Latent Spaces

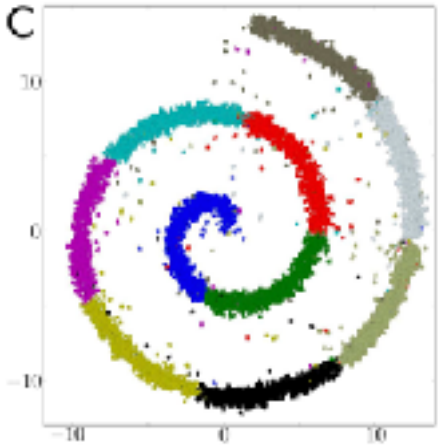


Sample Along Main Axis of the Gaussian Component for Each Digit

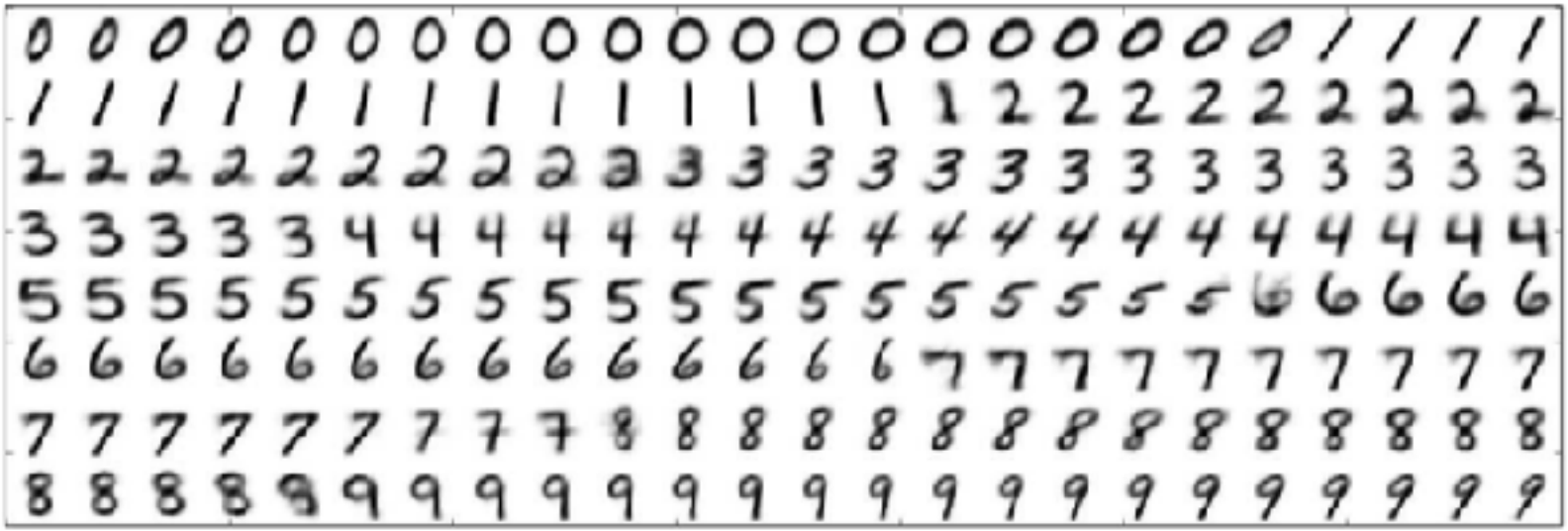


Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, **Ian Goodfellow**, and Brendan Frey. "Adversarial autoencoders." *arXiv preprint arXiv:1511.05644* (2015).

Conditional Class Latent Spaces

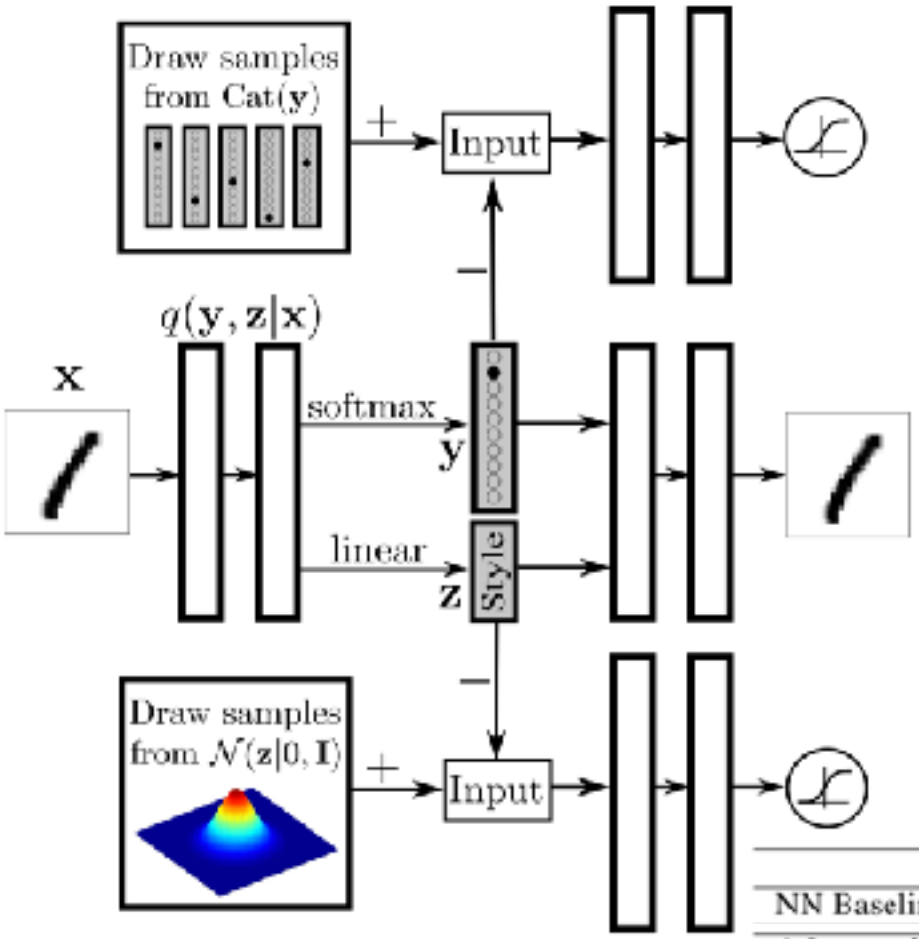


Sample Along Swiss Roll Axis



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

Semi-Supervised Classification



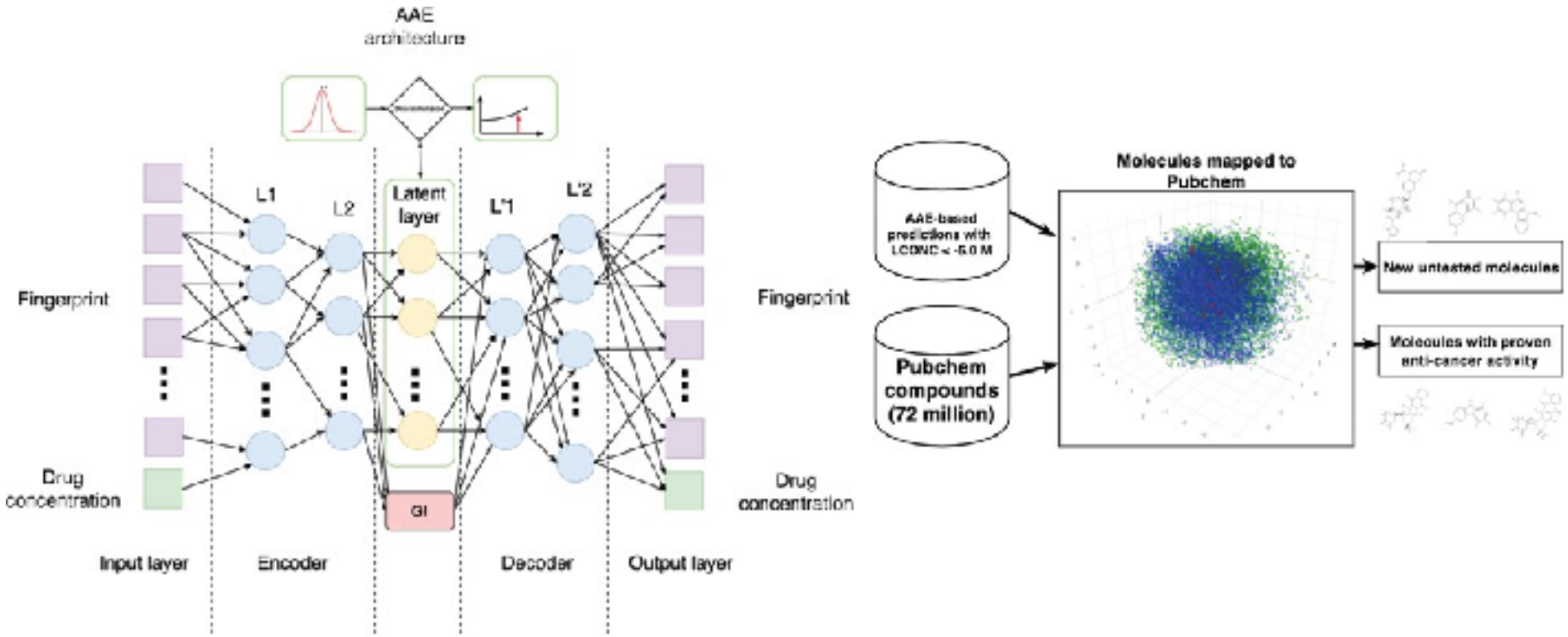
- Top Discriminator
  - Draw samples from one hot encoded labels
  - Ensures aggregated posterior matches class distributions
- Bottom Discriminator
  - Same as previous
  - Constrains latent representation
- Supervised Training
  - After GAN training:
  - Use small mini-batches on $q(y|z)$

	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.

Other Uses For AAE

- Chaos

The Newest AI-Enabled Weapon: 'Deep-Faking' Photos of the Earth

Step 1: Use AI to make undetectable changes to outdoor photos. Step 2: release them into the open-source world and enjoy the chaos.

Patrick Tucker • March 29, 2019

Lecture Notes for

Neural Networks and Machine Learning

Practical GANs

Next Time:
GAN Examples

Reading: Chollet CH8