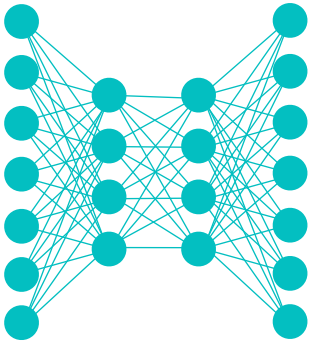
Lecture Notes for

Neural Networks and Machine Learning

Generative Networks



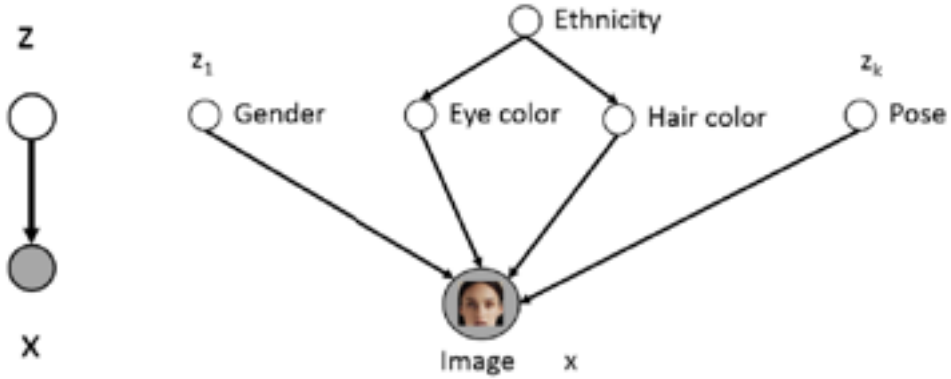
Logistics and Agenda

- Logistics
  - Welcome to online learning!
- Agenda
  - A historical perspective of generative Neural Networks
  - Variational Auto-Encoding
  - VAE in Keras Demo (next time)

Last Time

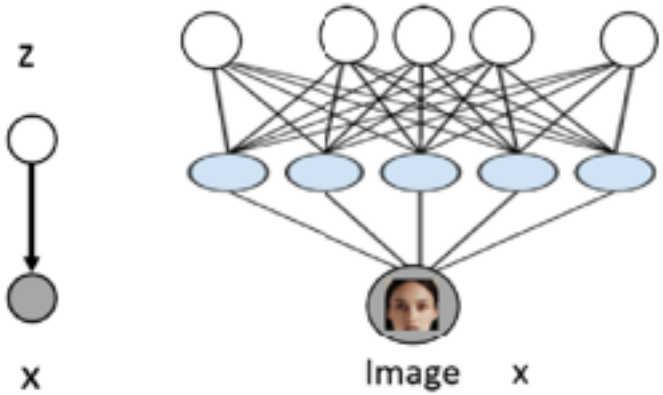
- Multi-modal and Multi-task
- Town Hall had three options for what to work on
  - ChemBL
  - DeepFake
  - Biometrics
- Now: Generative Networks

Motivations: Latent Variables



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

Hard: **z** is expertly chosen



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

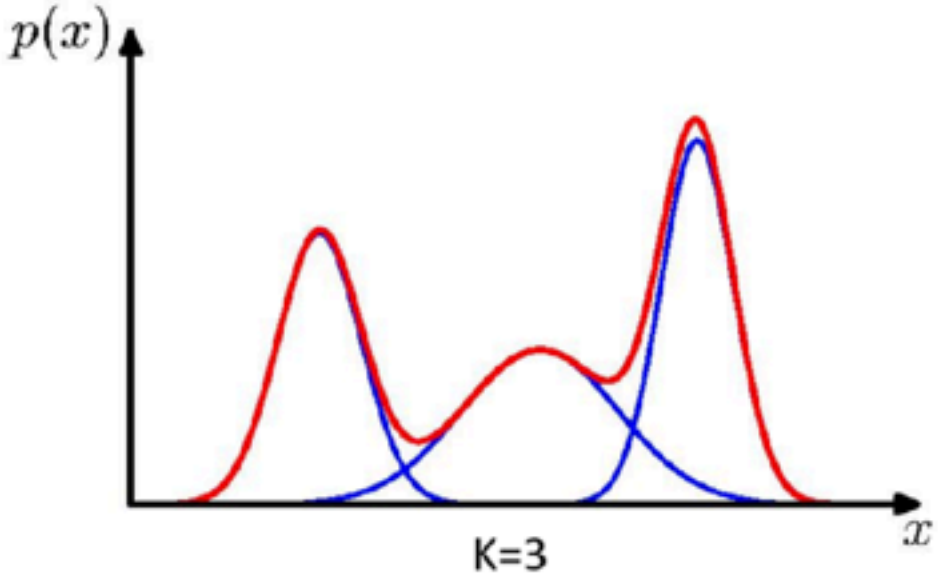
Not as Hard: **z** is trained, latent variables are uncontrolled

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 distribution
- Good choice for conditional is Normal Distribution
- Can parameterize $p(x|z)$ to be a Neural Network

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

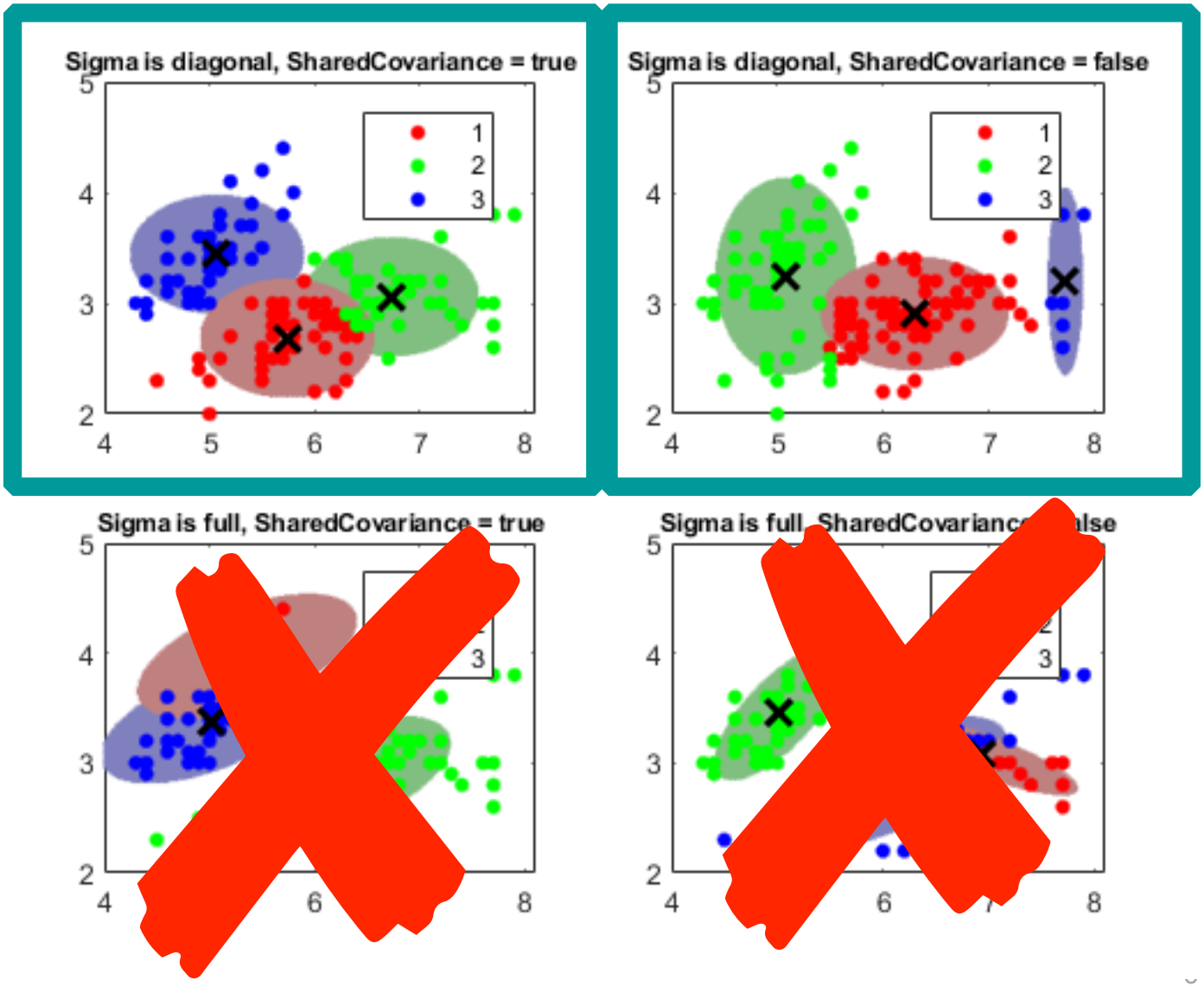
mean and covariance learned

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

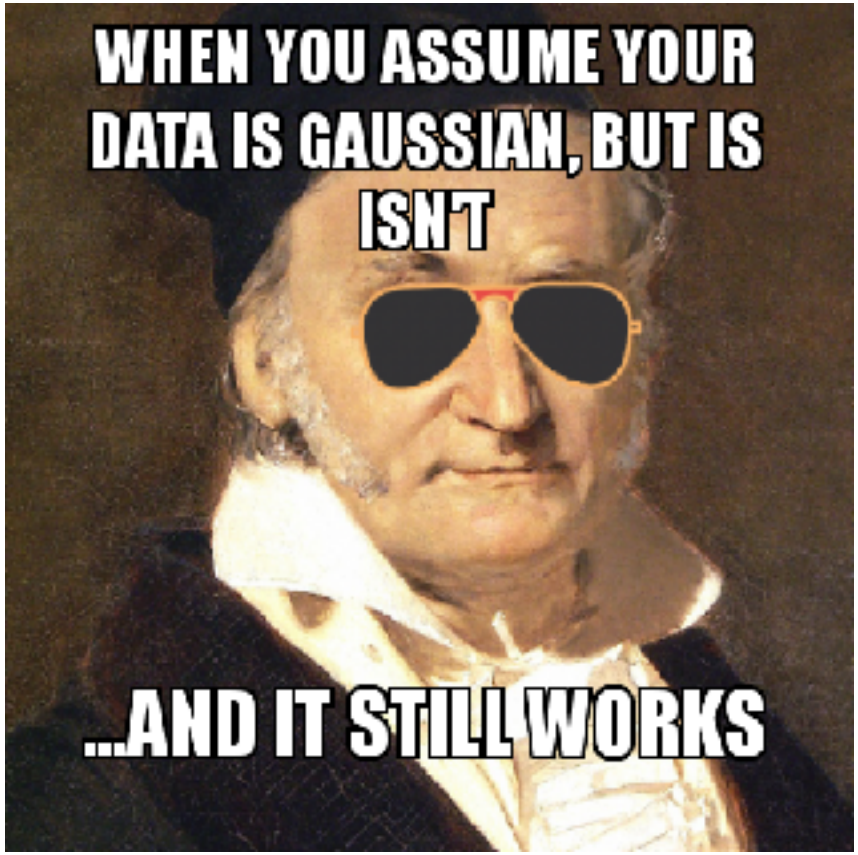
Motivation: Mixtures for Simplicity

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

mean and covariance learned



A History of Generative Networks



Aside: Sampling and expectation

some function

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

could be neural network

Expected value of f under conditional distribution, q

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

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

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:

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

Taxonomy of Generative Models

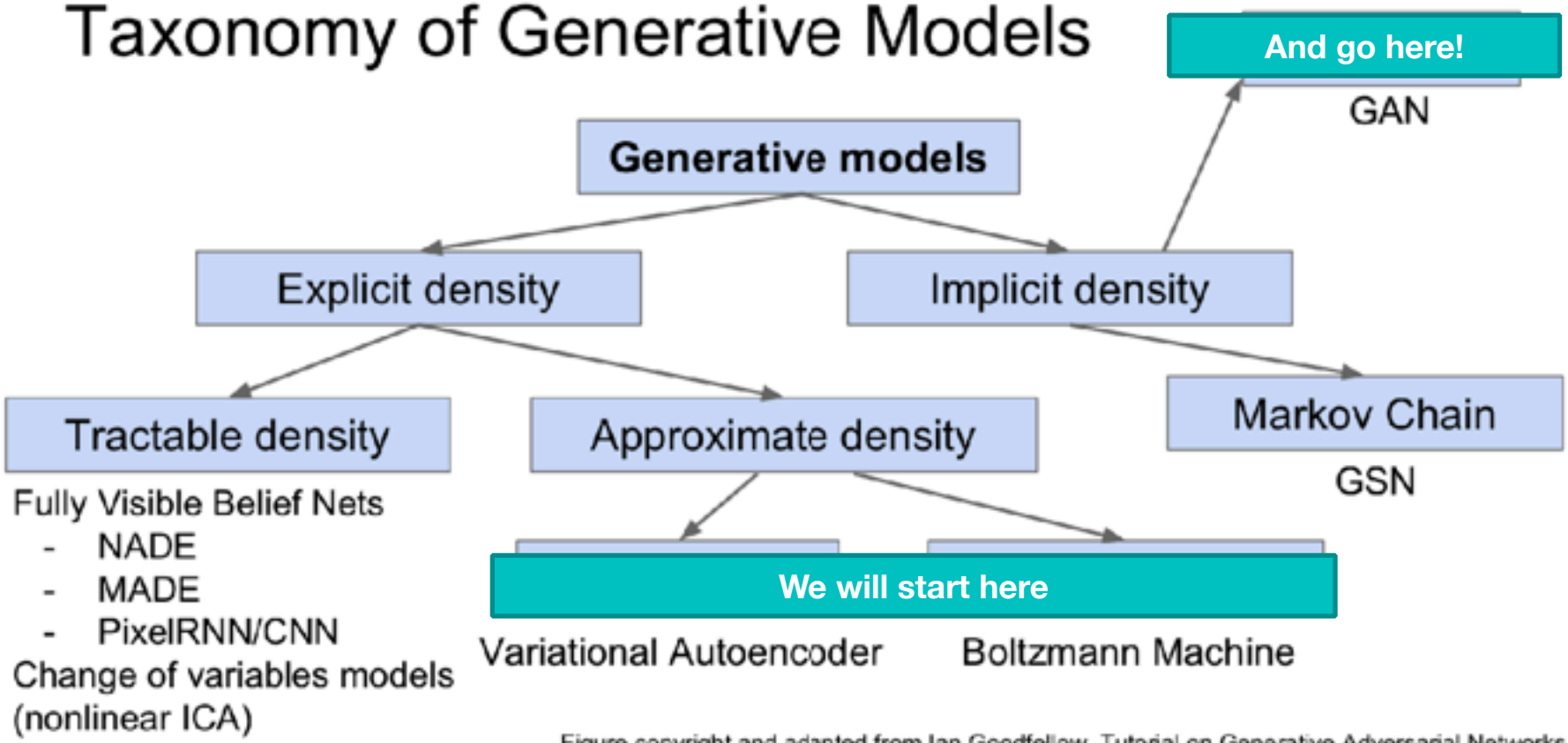
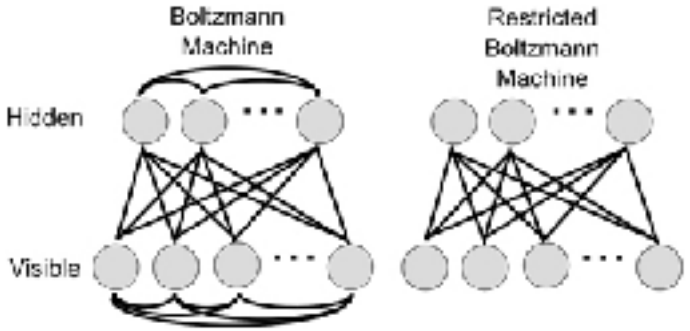


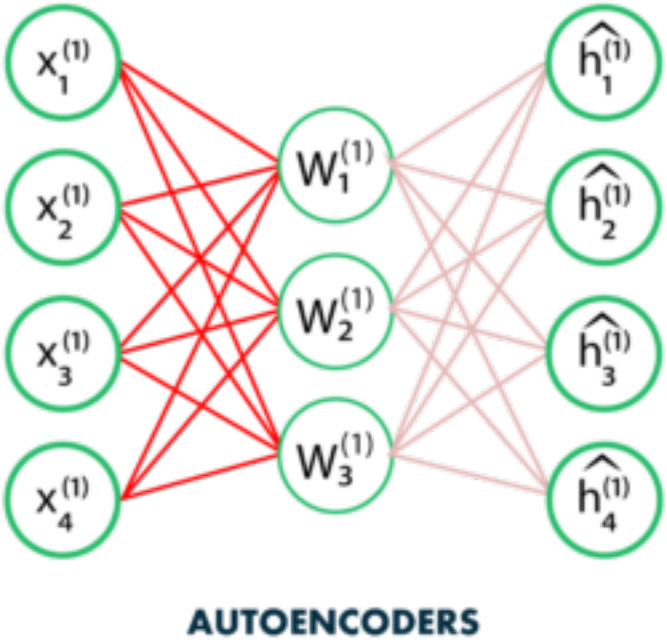
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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

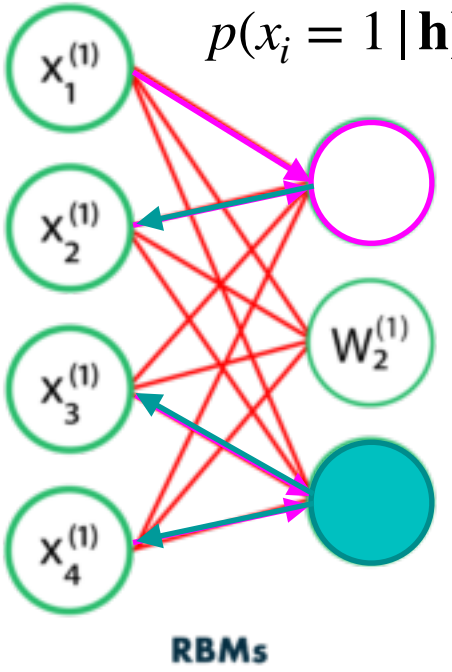
Deep Generative Models, 2006

- Restricted Boltzmann Machine
  - Forward pass (visible to latent)
  - Backward pass (latent to visible)
  - Similar to an auto encoder





$$p(x_i = 1 \mid \mathbf{h}) = \sigma(b + \mathbf{W} \cdot \mathbf{h})$$

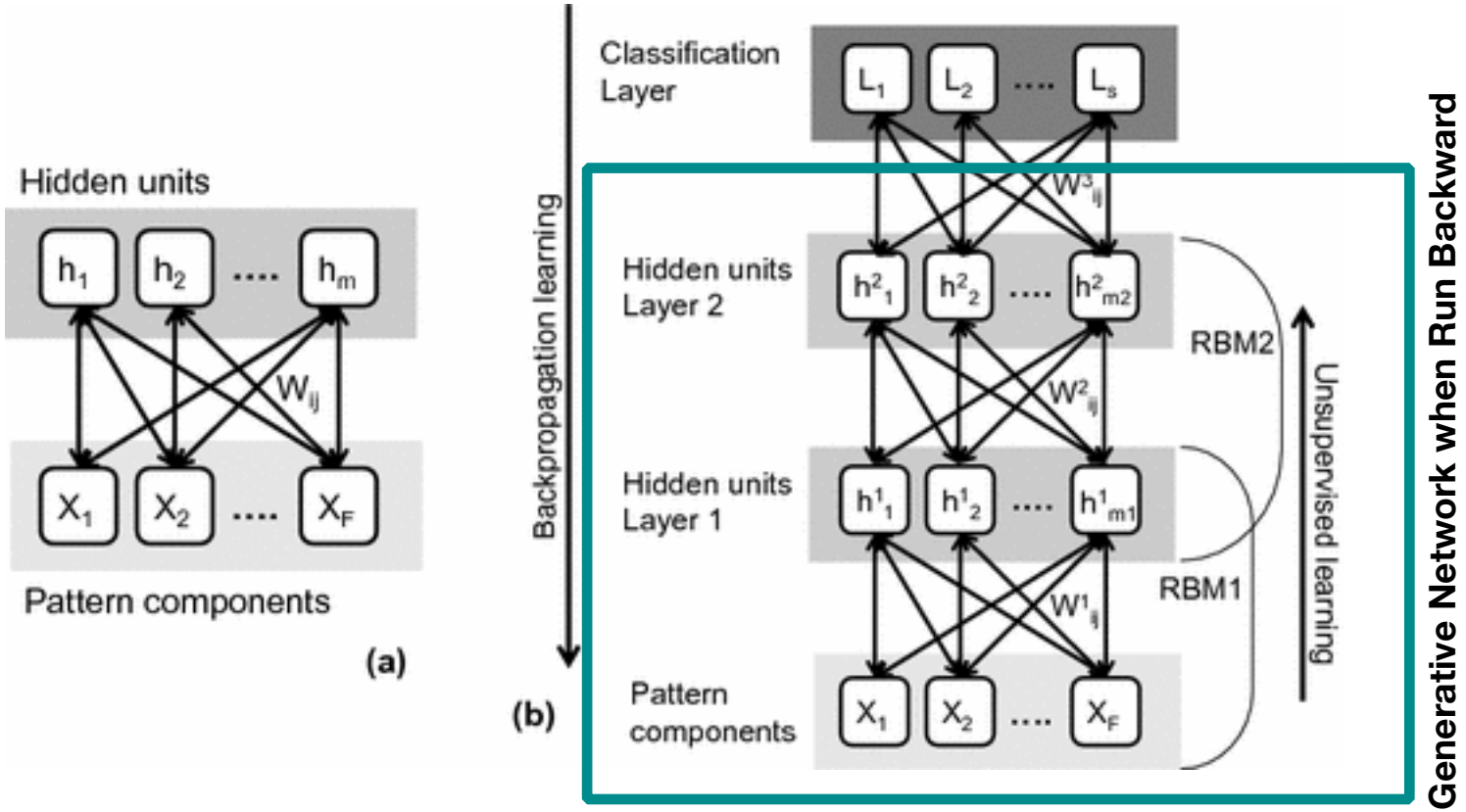


Each weight defines a sigmoidal probability of observation that is summed.

https://www.edureka.co/blog/restricted-boltzmann-machine-tutorial/

Deep Generative Models, 2006

- Deep Belief Network
  - Many RBM blocks together!



https://link.springer.com/chapter/10.1007/978-3-319-59153-7_3

Deep Beleif Networks, 2006

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

Definitions for Restricted Boltzman Machine

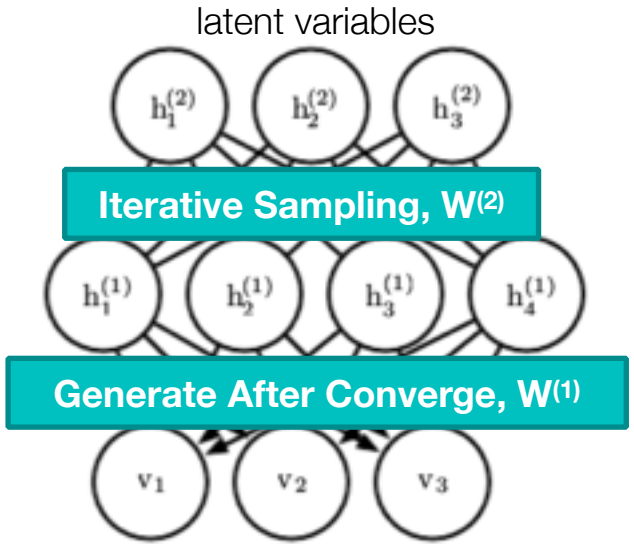
$$P(\boldsymbol{h}^{(l)}, \boldsymbol{h}^{(l-1)}) \propto \exp\left(\boldsymbol{b}^{(l)\top}\boldsymbol{h}^{(l)} + \boldsymbol{b}^{(l-1)\top}\boldsymbol{h}^{(l-1)} + \boldsymbol{h}^{(l-1)\top}\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l)}\right), \quad (20.17)$$

Energy Function, relates each latent variable (binary)

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

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

Sigmoids relate conditional distributions (like always)



$$\mathbf{E}_{v \leftarrow p_{obs}}[\log p(v)]$$ Maximize!! want samples from p to match observed v

$p(v)$ is intractable

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

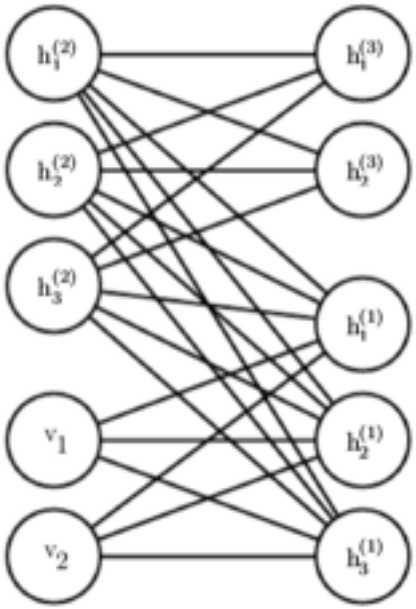
optimize with constrastive divergence

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

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

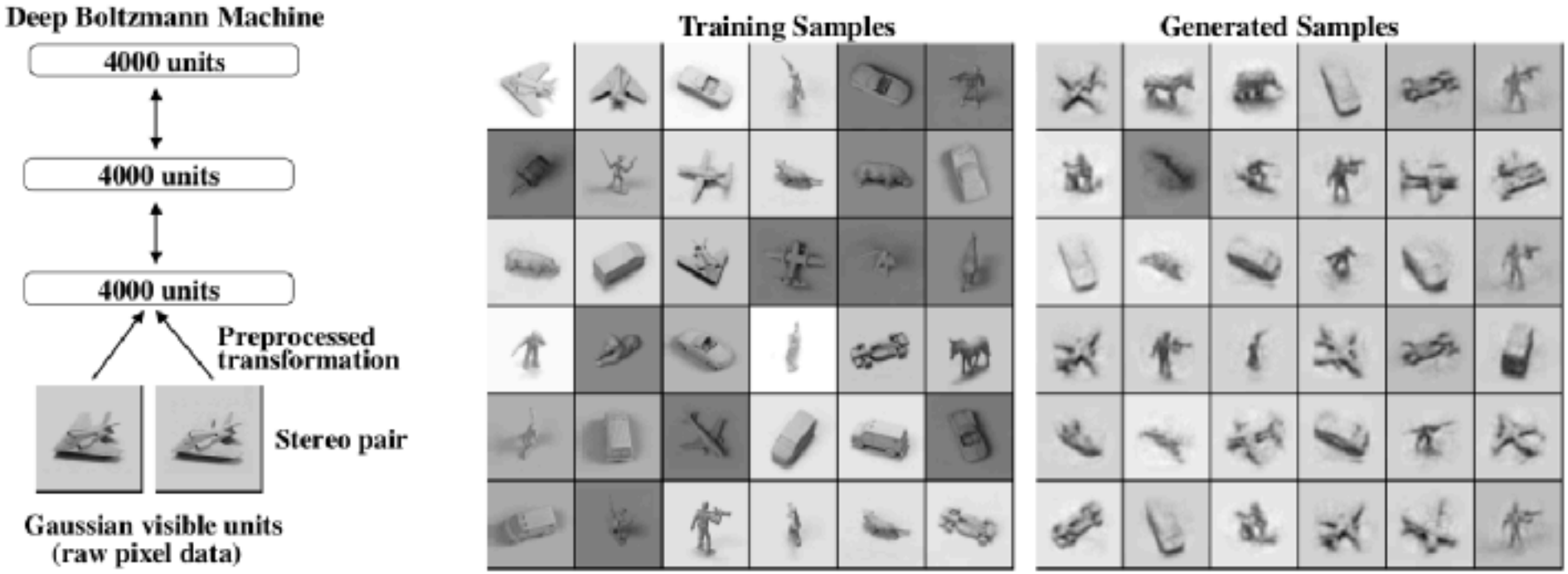


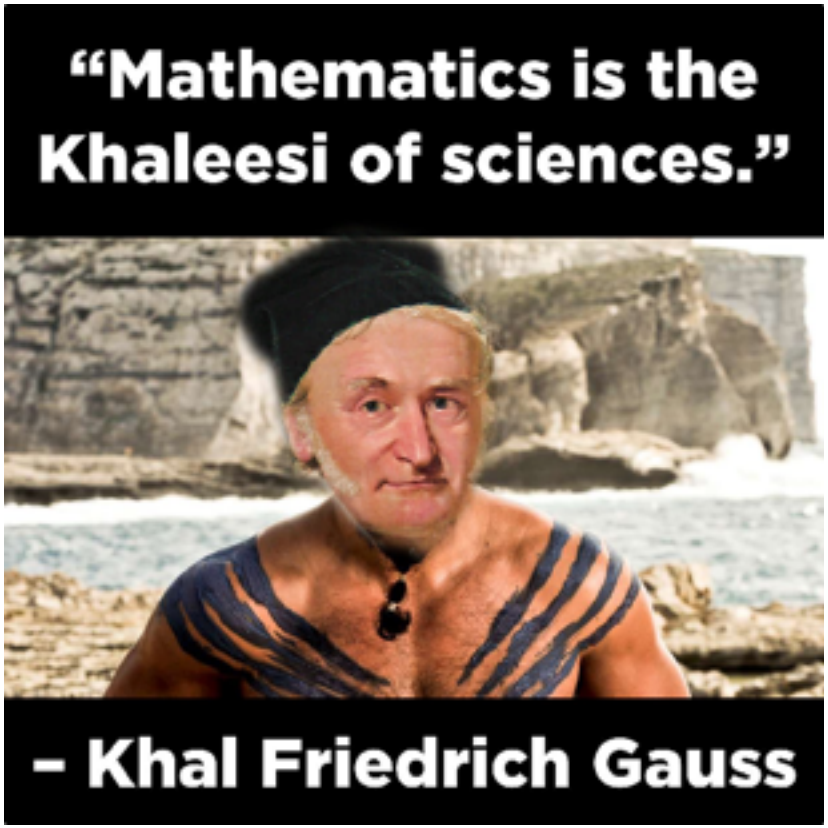
Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

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

Contemporary Modeling

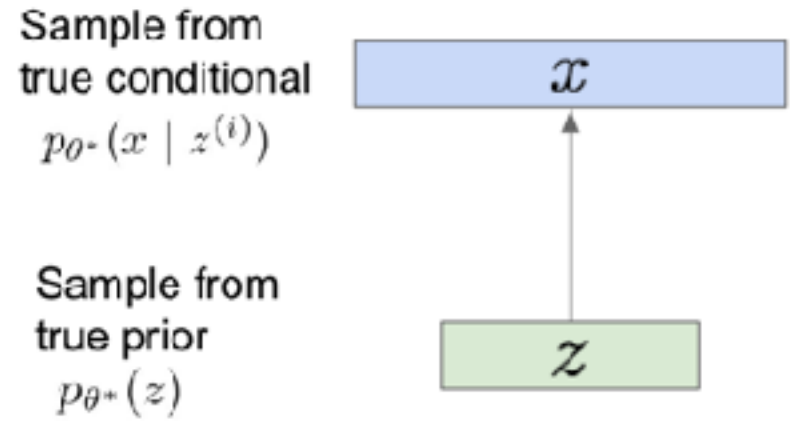
- DBNs and DBMs did not become very popular
  - Mathematics detracts from popular understanding
  - Often methods using sampling are not scalable
  - Cannot directly use Gradients (no Back Prop) 😢
- Evidence Lower Bound (ELBO) considered "good enough" because … we can't do better computationally
- Popular 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

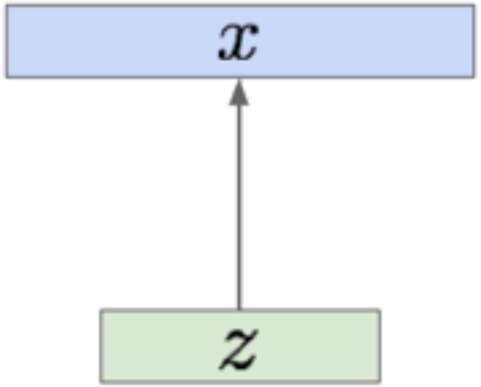
Reasonable constraints for p(z)?

Sample from true conditional

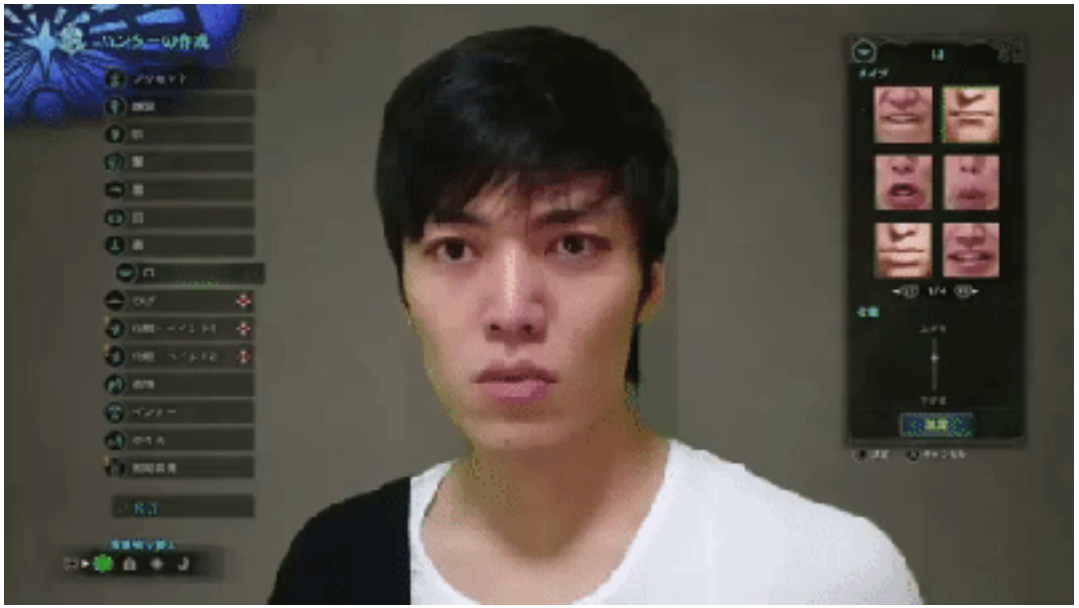
$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

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



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



Optimizing

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

We need this inference in order to compute latent variable

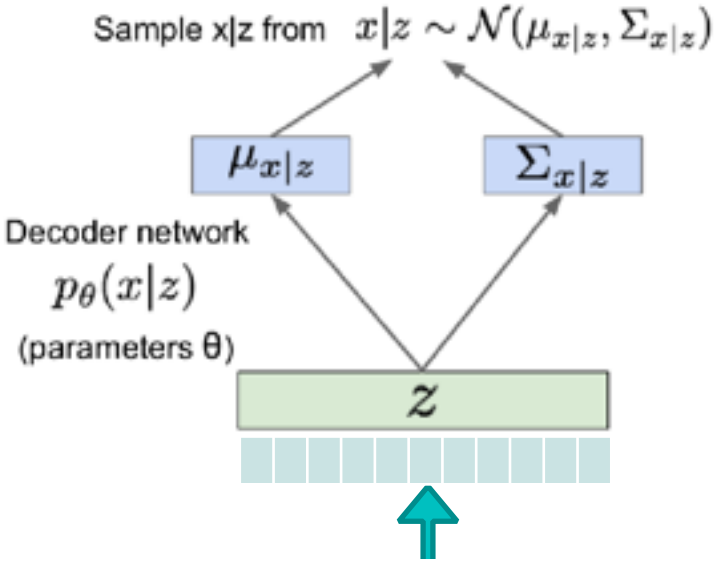
$$p(x) = \int p(x \mid z)p(z)dz$$

Denominator is of this form

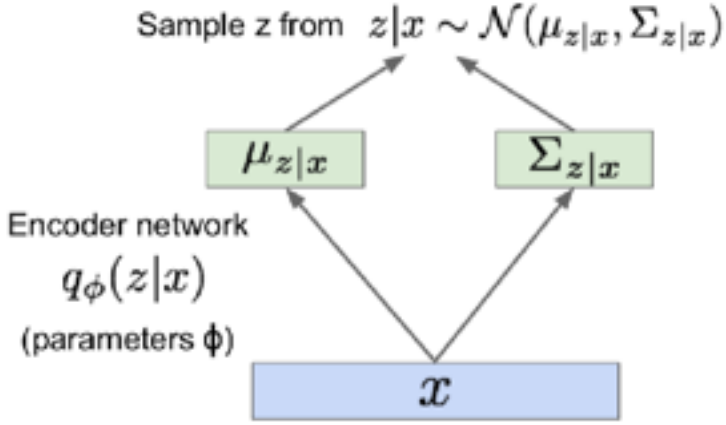
- We can't compute! **Intractable computation** for all "z"
- So let's define this with **variational inference**:
  - Only needs to work for z **with observed** $x^{(i)}$
  - 1. **Encode** observed $x^{(i)}$ using network $q(z \mid x^{(i)})$,
  - 2. Use $q(z \mid x^{(i)})$ to sample z appropriately, then **decode** with another neural network
  - 3. Make $q(z \mid x^{(i)})$ as close as possible to $p(z \mid x)$

Need a new formulation

Step Two: Decode



Step One: Encode



Step Three: Make p and q Similar

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

we can manipulate this equation to yield the following:

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

intuition MLE: maximize probability of observed $x^{(i)}$

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

$$\mu(x^{(i)}) \text{ and } \Sigma(x^{(i)})$$

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

Need a new formulation

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

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

Bayes rule + multiply by one

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

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

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

always negative

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

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

assume p(z) is Normal,
zero mean and std=1
because we like Normal

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

outputs of q network

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

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

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

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

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

The 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
This is just negative cross entropy

assume p(z) is Normal,
zero mean and std=1
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

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

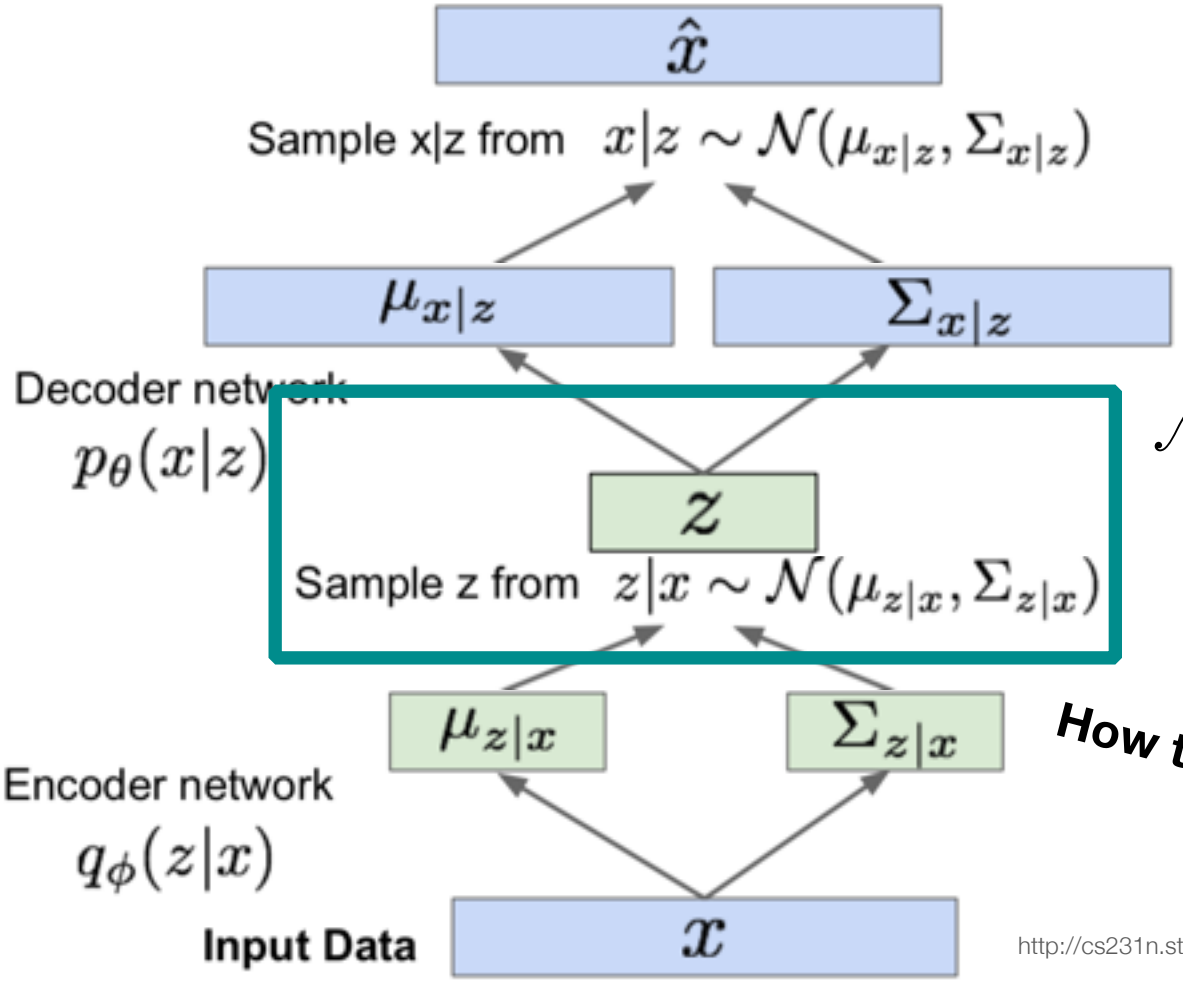
so we will have the neural network output log variance

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

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

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…

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

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)
```

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

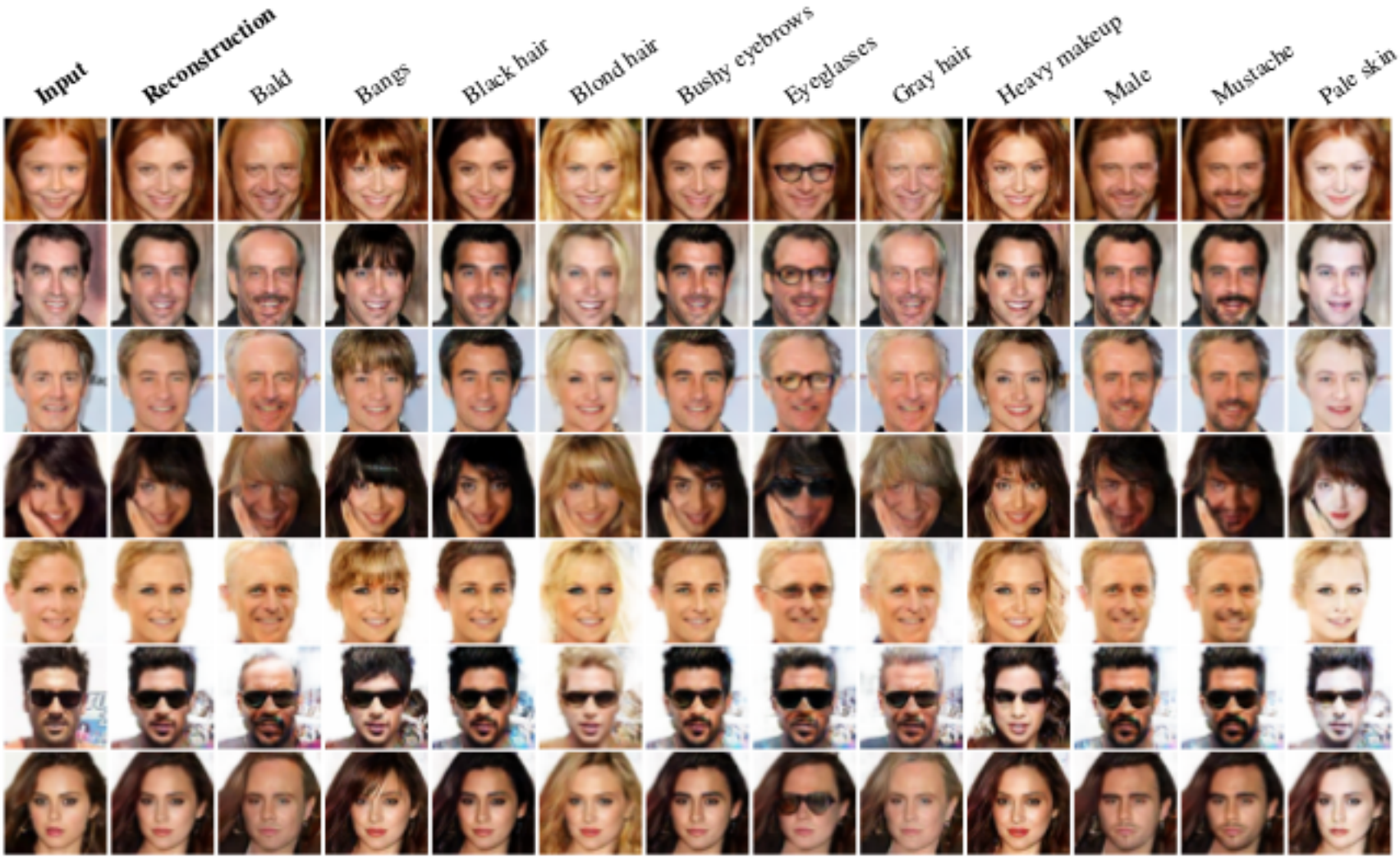
$$= \mathbf{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right] - \sum_{k} \exp(\Sigma(x^{(i)})) + \mu(x^{(i)})^2 - 1 - \Sigma(x^{(i)})$$

VAE Examples

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

VAE Examples

Different, automatically found z, latent variables



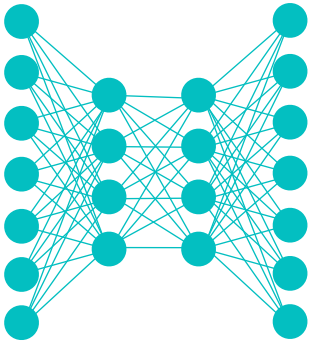
Lecture Notes for

Neural Networks and Machine Learning

Generative Networks

Next Time:
GANs

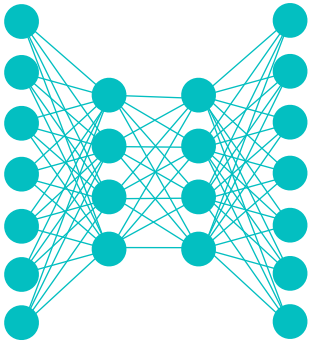
Reading: Chollet CH8



Lecture Notes for

Neural Networks and Machine Learning

VAE Demo
Generative Adversarial Networks



Logistics and Agenda

- Logistics
  - Online Learning Reminders
  - How is the format?
  - How is lab three going (discuss deadline)?
- Agenda
  - VAE Demo
  - Simple Generative Adversarial Networks

Back to VAEs

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

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

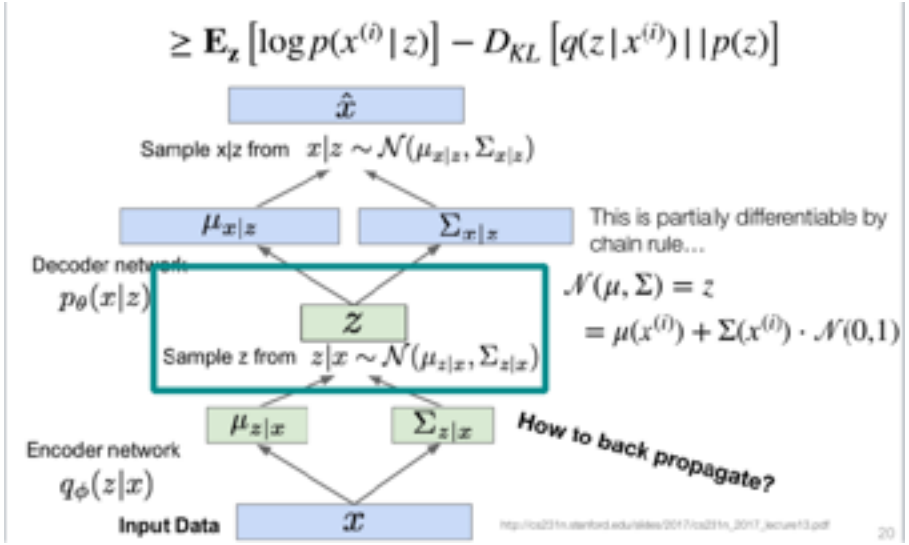
Yann LeCun @ylecun · 20h

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

3 11 312

Last Time

- Variational auto encoding

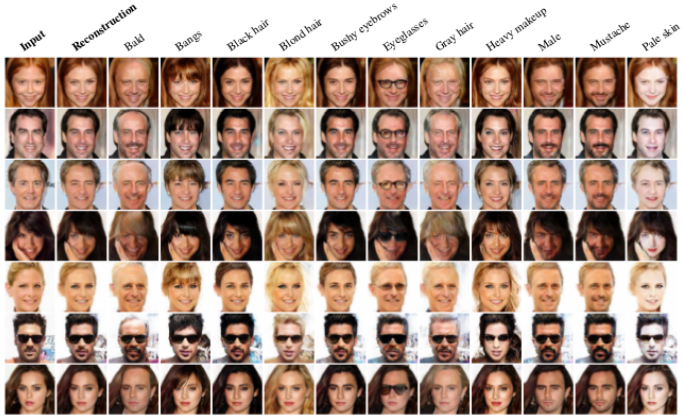



```
# Encode the input into a mean and variance parameter
z_mean, z_log_variance = encoder(input_img)

# 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

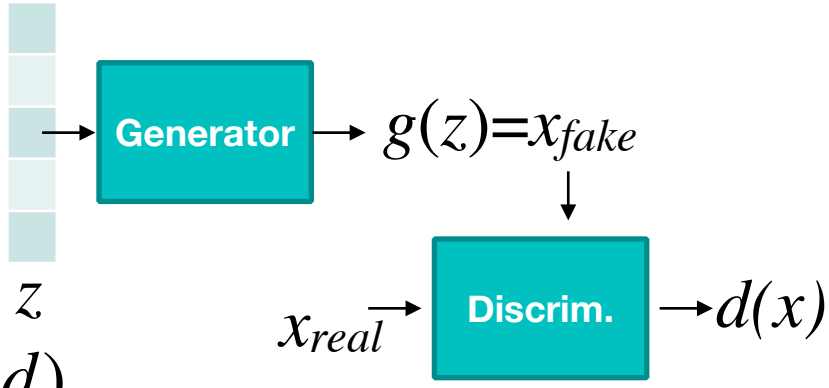
Latent Variable Approximation with GANS

- **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!
  - … Neural Networks are like the chiropractory of the machine learning…

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):

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

only portion dependent on g generator minimizes

everything dependent on d discriminator maximizes

Taking turns

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

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

- 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
  - Like optimizing in an Ocean
- 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:

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

- 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:

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

- 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

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

Final Loss Functions

- Discriminator:

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

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

- Generator:

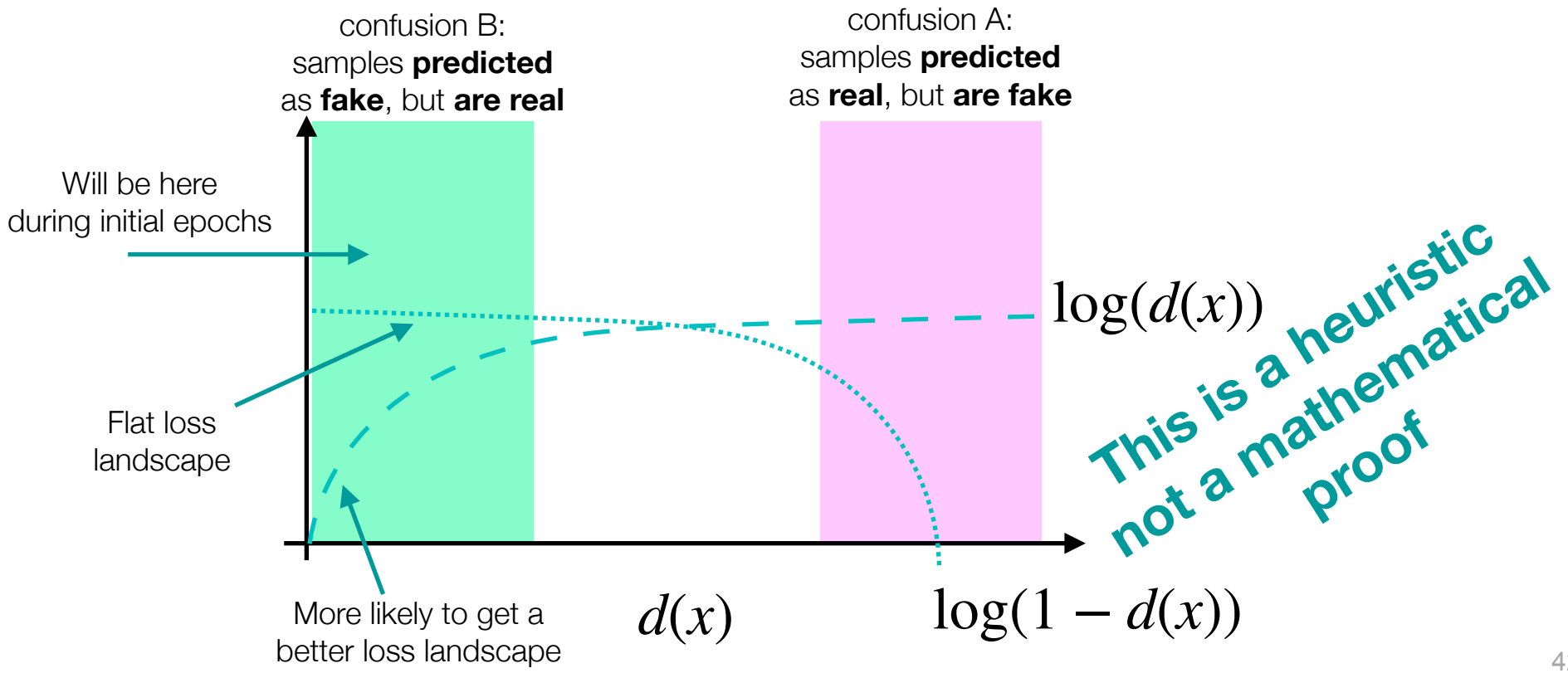
$$\max_{g} \mathbf{E}_{x \leftarrow g(z)} \left[\log(d(x_{fake})) \right]$$

Freeze Discriminator Weights

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

Why was minimizing 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

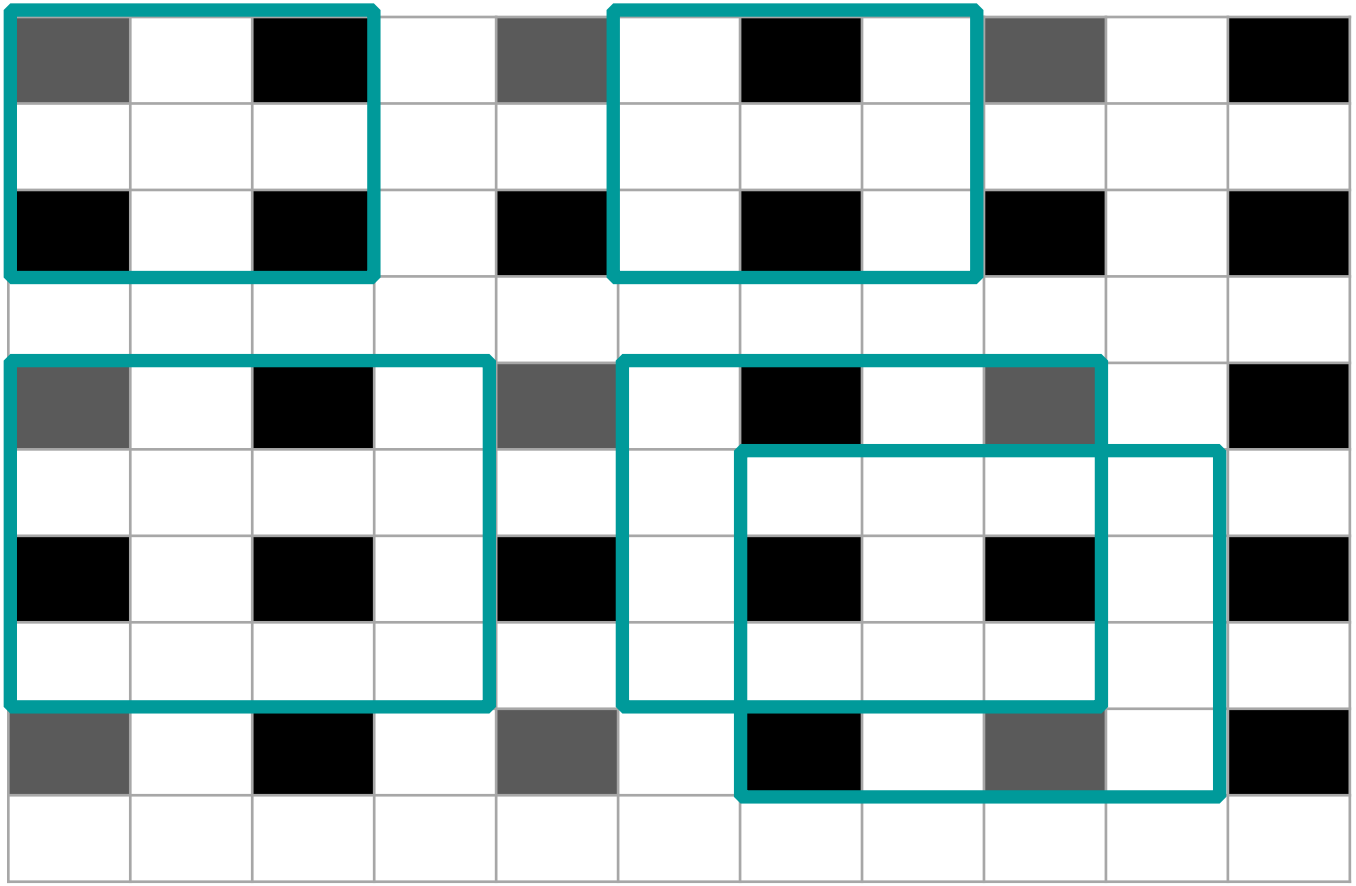
The process of training GANs and tuning GAN implementations is notoriously difficult. There are a number of known tricks you should keep in mind. Like most things in deep learning, it's **more alchemy than science**: **these tricks are heuristics, not theory-backed guidelines**. They're supported by 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 in optimizer and labels
  - 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)

What do you mean checkerboard patterns?

- Kernel size should be a symmetric multiple of the stride



Bias needs to account for both when 2 pixels and four pixels are in the kernel space

Multiple of stride ensures that same number of active pixels are there.

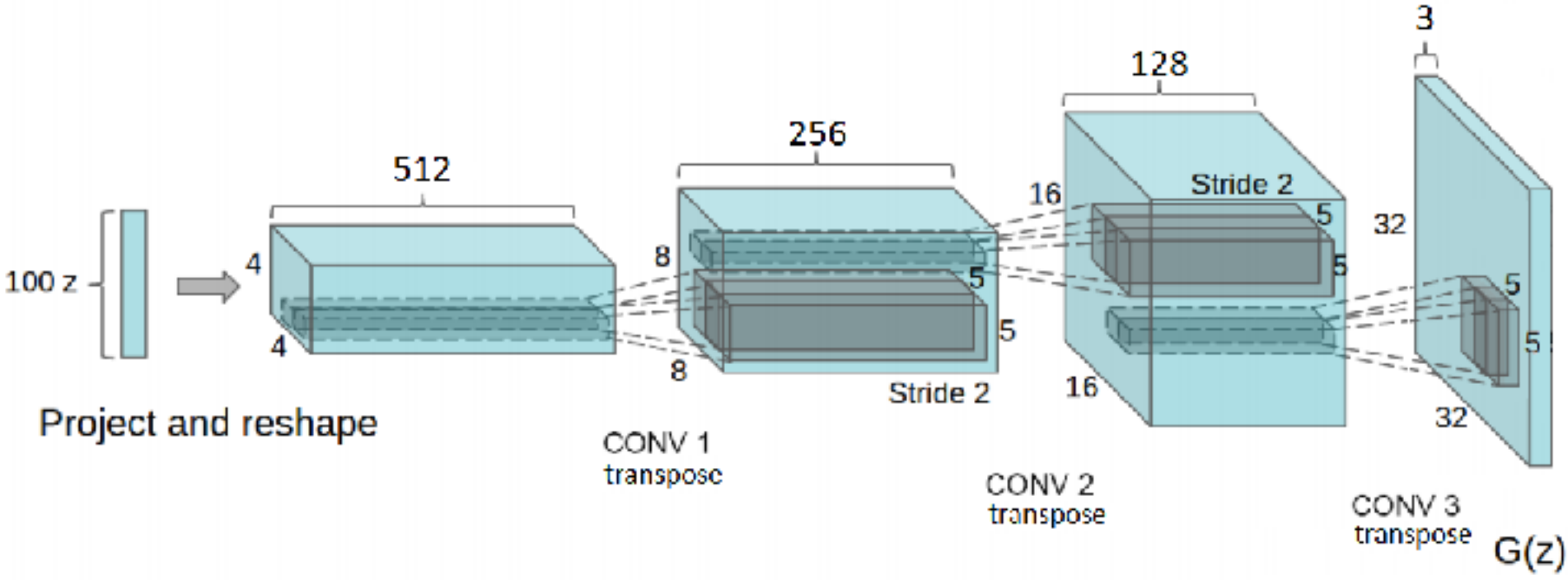
Some Results of Generation

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

Highlight: Nearest Training Sample

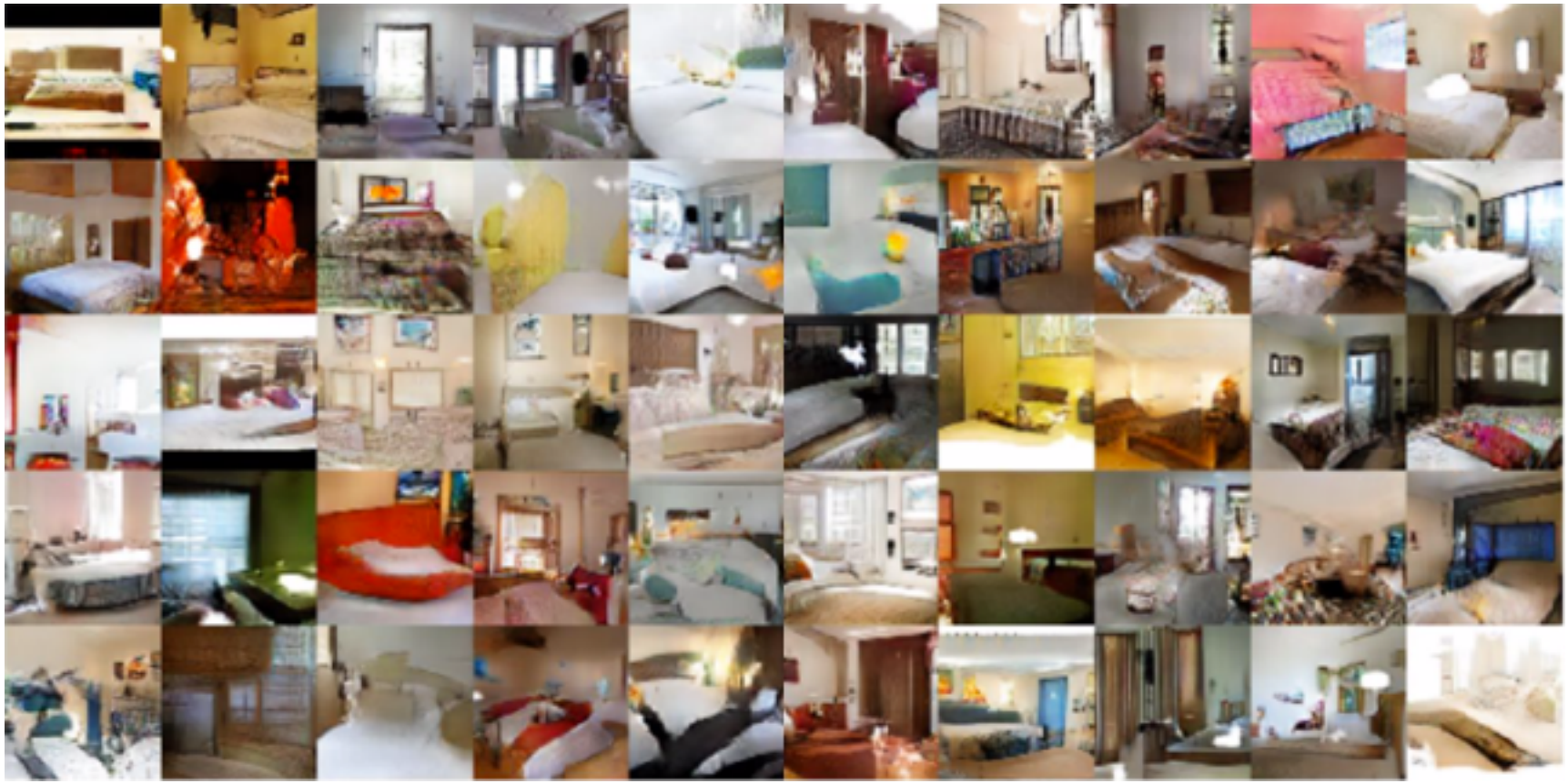
Deep Convolutional GANs

- 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

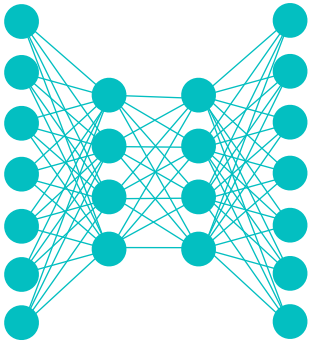
Lecture Notes for

Neural Networks and Machine Learning

GANs

Next Time:
Practical GANs

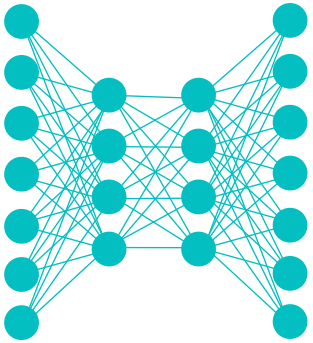
Reading: Chollet CH8



Lecture Notes for

Neural Networks and Machine Learning

Practical GANs



Logistics and Agenda

- Logistics
  - Student Presentations Today and Next Lecture
- Agenda
  - Review from Last Time, GAN Demo
  - Training GANs in different loss landscapes
  - **Student Presentation**: Representational Learning with Deep Convolutional GANs, Radford
  - Wasserstein GAN

Last Time

- Simple GANS

- Discriminator:

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

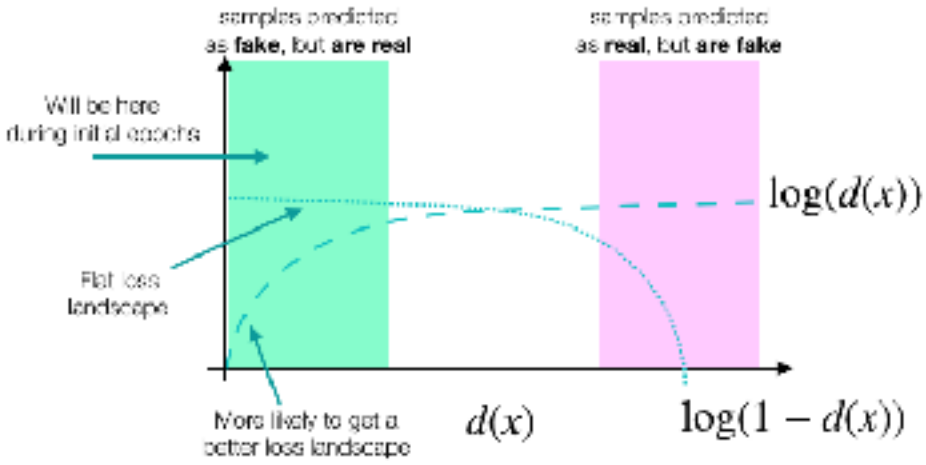
Same as minimizing the binary cross entropy of the model with: (1) labeled data and (2) generated data!

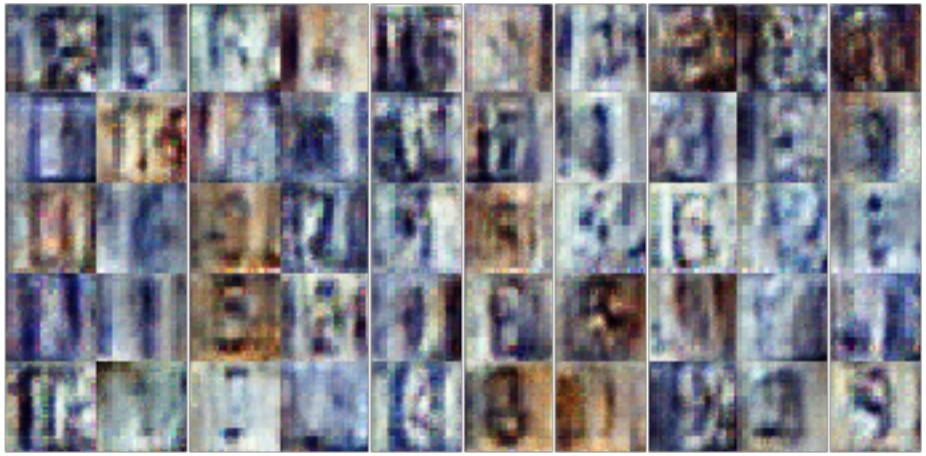
- Generator:

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

Freeze Discriminator Weights

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





GANs in Keras

Implementation from Book on
"Frogs from CIFAR"

Game:
Can you catch any errors!?

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
Googel 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.

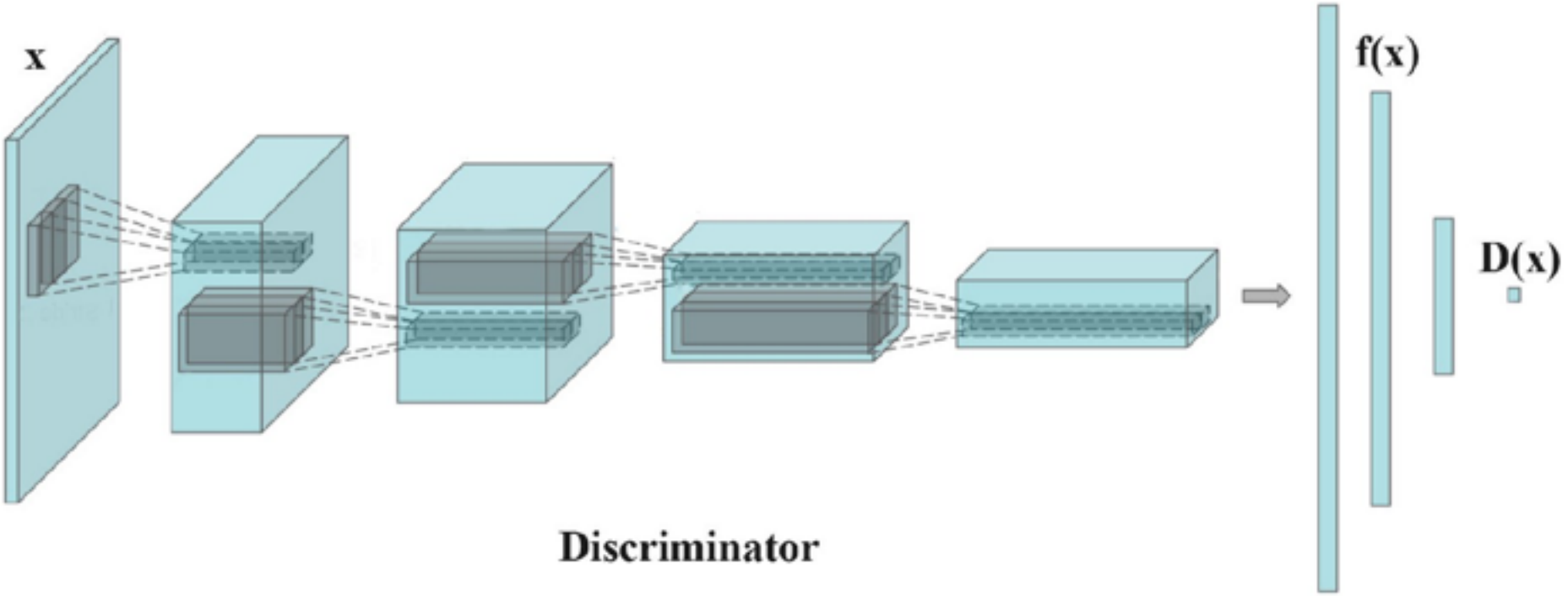
The GAN Loss Landscape

- Problems:
  - GANs might not converge
  - GAN output might be the same sample (mode collapse)
  - Slow training: gradient vanishes
- 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
  - Manipulating the Loss Function

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.*"

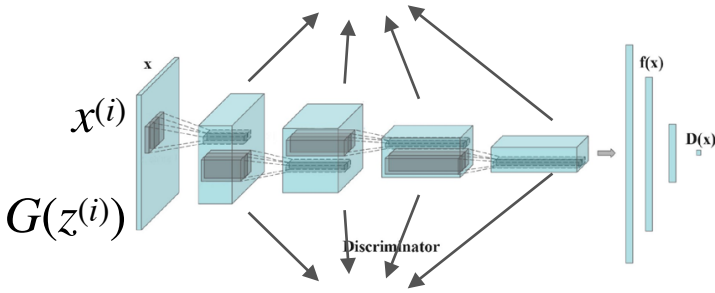
Before we go too far



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

$$\mathbf{f}(x^{(1)}), \mathbf{f}(x^{(2)}), \ldots \mathbf{f}(x^{(N)}),$$



$$\mathbf{f}(G(z^{(1)})), \mathbf{f}(G(z^{(2)})), \ldots \mathbf{f}(G(z^{(M)}))$$

1 … N are 1 … M
one turn of real/fake data
usually, N==M

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

mean discriminator activations from **real** data

mean discriminator activations from **fake** data

```
real_mean  = K.reduce_mean(real_features, axis = 0)
fake_mean  = K.reduce_mean(fake_features, axis = 0)
feat_match = K.reduce_mean(tf.square(real_mean-fake_mean))
```

Salimans, Tim, I… *…dvances in neural information processing syste…*

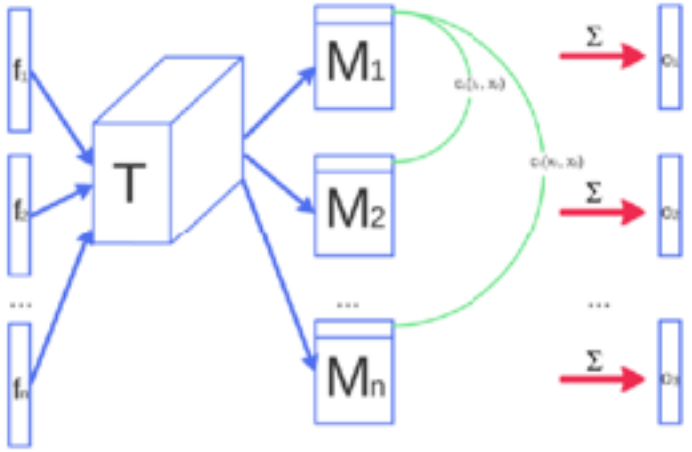
Mini-Batch Discrimination

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

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

i^{th} sample in batch

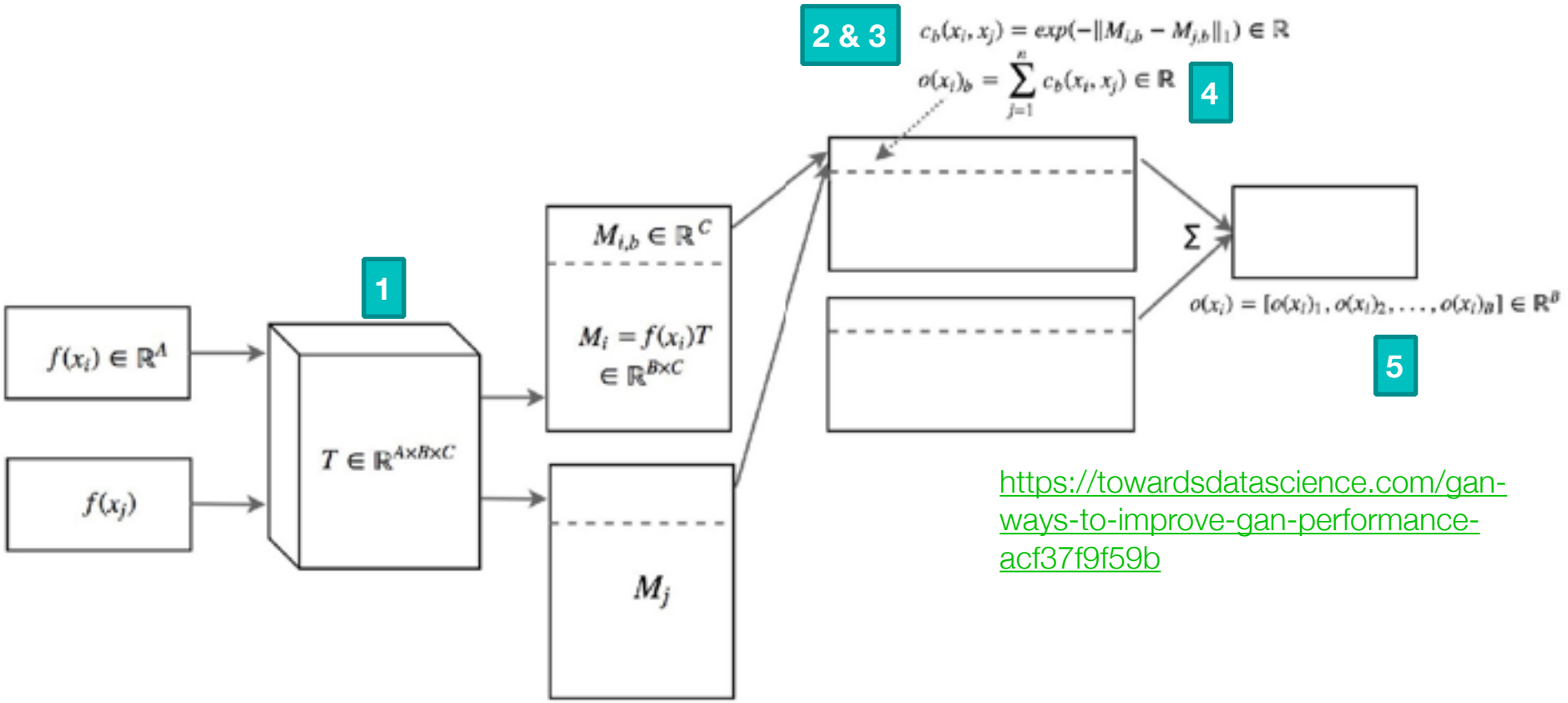
j^{th} sample in batch



- 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



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

```
M = K.reshape(K.dot(x, self.T), (-1, self.num_kernels, self.kernel_dim))          1
L1 = K.abs(K.expand_dims(M, 3) - K.expand_dims(K.permute_dimensions(M, [1, 2, 0]), 0))   2
c = K.exp(-K.sum(L1, axis=2))                                                      3
o = K.sum(c, axis=2)                                                               4
return K.concatenate([x, o], 1)                                                    5
```

This broadcast is tricky, but powerful!

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 and exploiting current found solutions
- Solution: Add explore/exploit tradeoff directly in loss function

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

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…

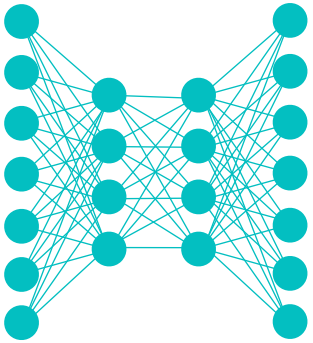
Salimans, Tim, **Ian Goodfellow**, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in neural information processing systems*, pp. 2234-2242. 2016.

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*).

Lecture Notes for

Neural Networks and Machine Learning

Wasserstein GANs



Logistics and Agenda

- Logistics
  - Student Presentations Today
- Agenda
  - **Student Presentation**: Representational Learning with Deep Convolutional GANs, Radford
  - **Student Presentation**: Towards Principled Methods for Training GANs, Arjovsky and Bottou 2017
  - Wasserstein GAN
  - Demo

Paper Presentation

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.

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	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1-\alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1-\alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
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	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c,c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c,c')$
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)_/¯

There has to be a more principled way of doing this!!!

	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

Paper Presentation

TOWARDS PRINCIPLED METHODS FOR TRAINING GENERATIVE ADVERSARIAL NETWORKS

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

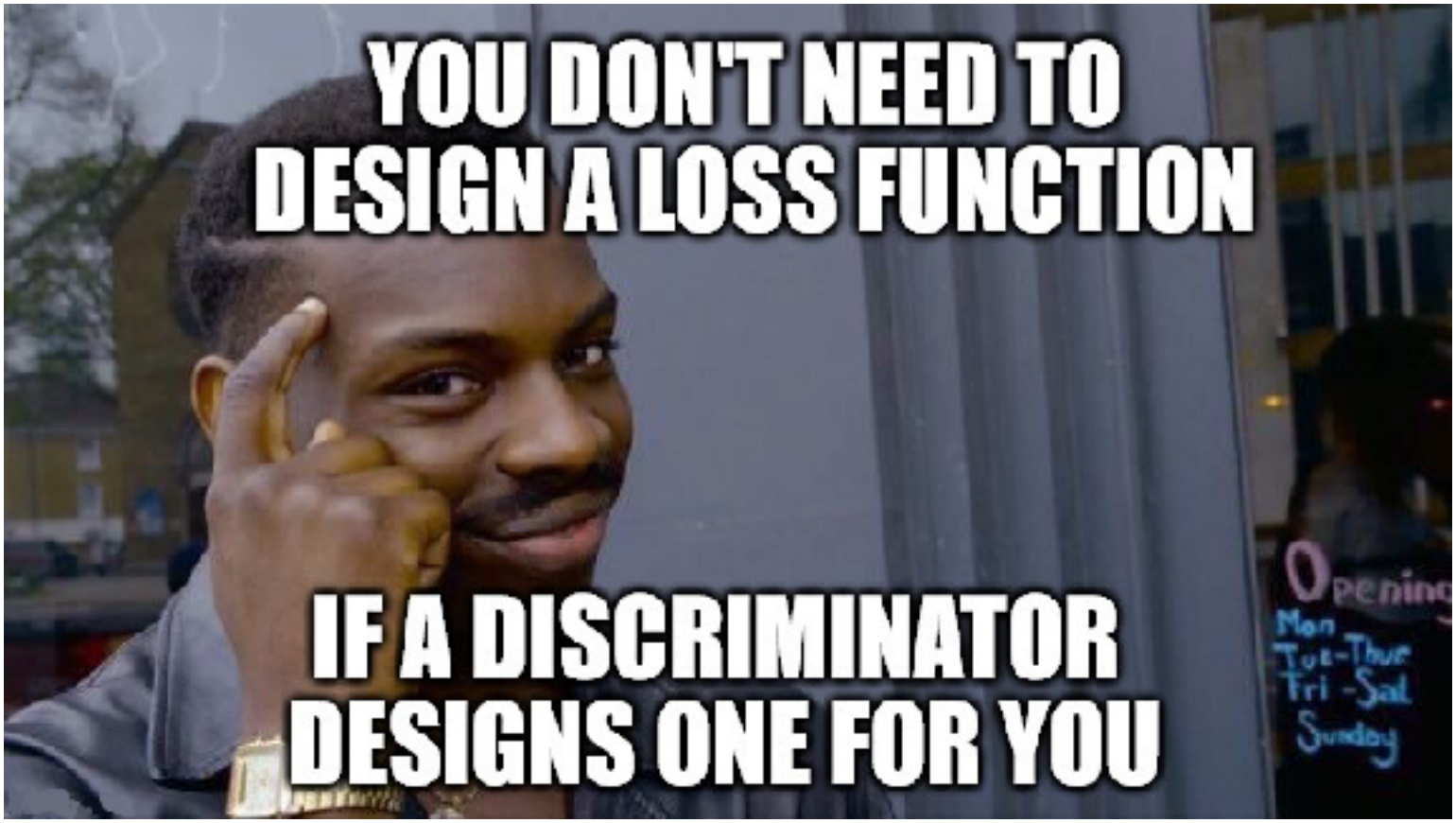
Léon Bottou
Facebook AI Research
leonb@fb.com

ABSTRACT

The goal of this paper is not to introduce a single algorithm or method, but to make theoretical steps towards fully understanding the training dynamics of generative adversarial networks. In order to substantiate our theoretical analysis, we perform targeted experiments to verify our assumptions, illustrate our claims, and quantify the phenomena. This paper is divided into three sections. The first section introduces the problem at hand. The second section is dedicated to studying and proving rigorously the problems including instability and saturation that arize when training generative adversarial networks. The third section examines a practical and theoretically grounded direction towards solving these problems, while introducing new tools to study them.

This is still a good paper, its just not going to give you what you want: **answers**

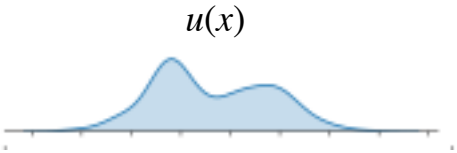
Wasserstein GANs



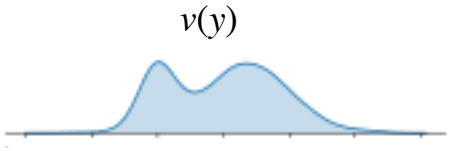
Caveat

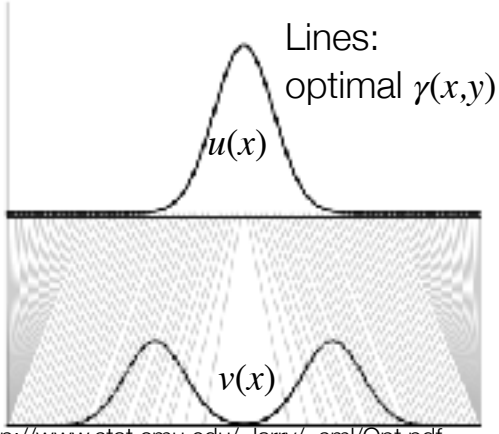
- Wasserstein Loss is actually very useful
  - But the mathematics pinning it down are based on approximations (like always)
  - So its **not really true** to say its actually working for the motivated reasons
  - But **it might be**
- So until we have a better theory, **lets assume it works** because it truly is a great distance measure

Wasserstein Distance (Earth Mover)



how to move dirt from u to v ?

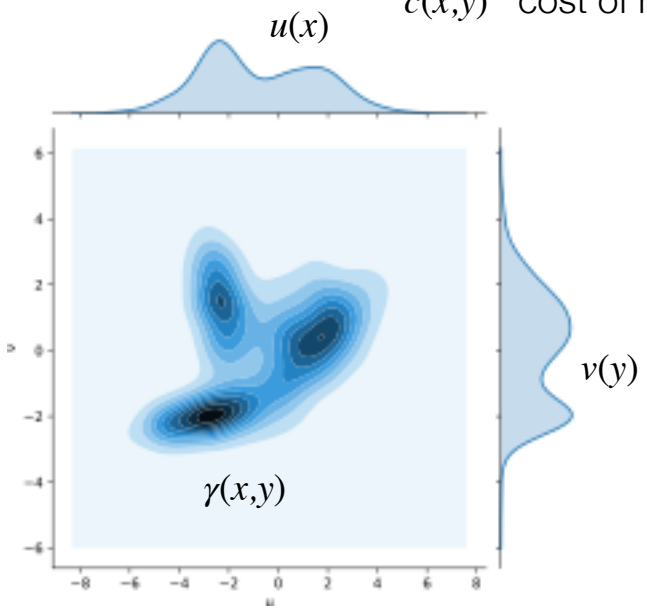




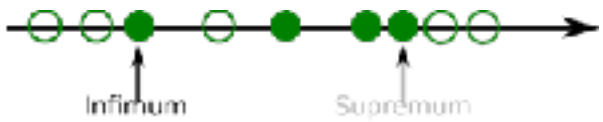
http://www.stat.cmu.edu/~larry/=sml/Opt.pdf

$\gamma(x,y)$ 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



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

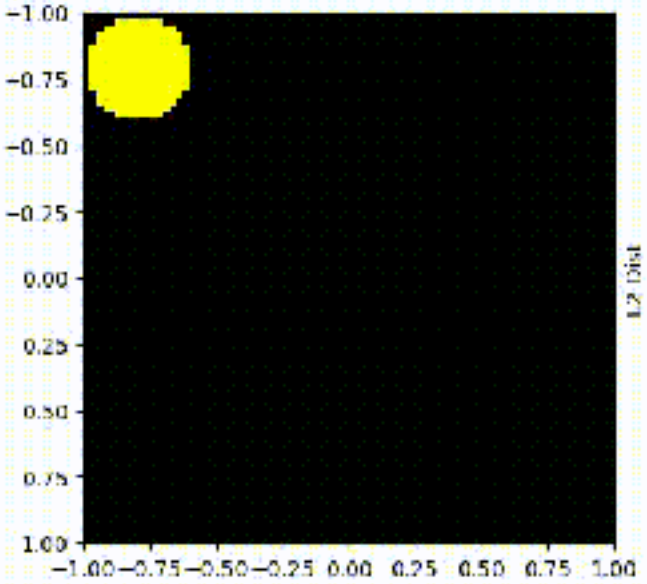
Optimal plan
has greatest lower bound
(minimum cost in set Π)

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

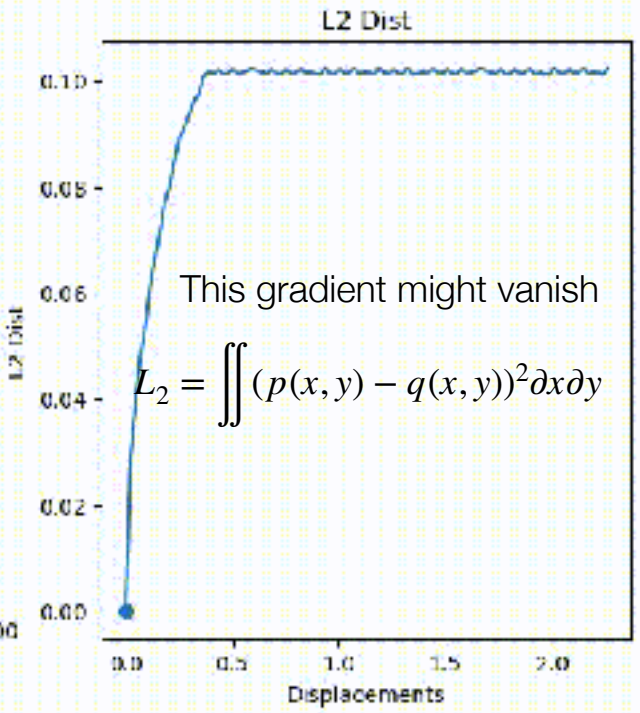
If cost is difference
to move from x to y

image: wikipedia

Wasserstein Distance

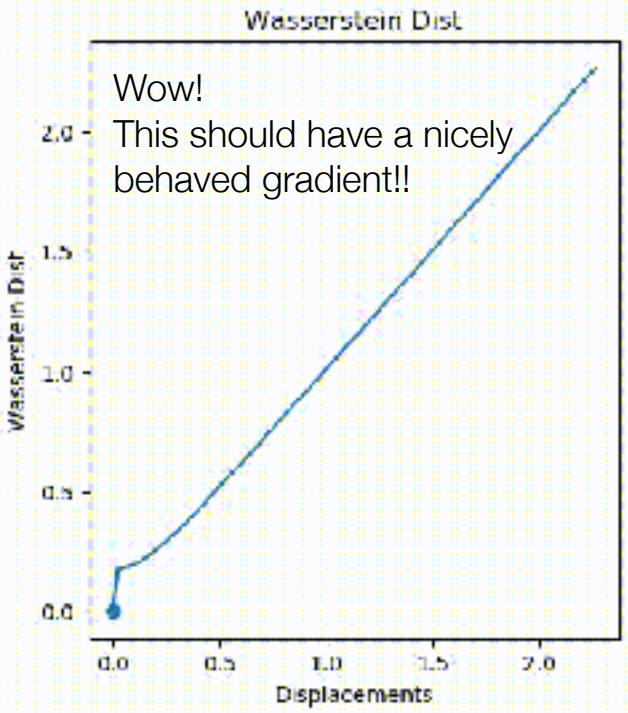


Two 2D Normal Distributions



This gradient might vanish

$$L_2 = \iint (p(x,y) - q(x,y))^2 \partial x \partial y$$



Wow!
This should have a nicely behaved gradient!!

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

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 for generative distributions so its time to start approximating!

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? Moving dirt…
- Could use KL divergence, but we already know that has many problems associated with vanishing gradients
- 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

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

$$= \inf_{\gamma} \mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| + \underbrace{\sup_{f} \mathbf{E}_{x \leftarrow p_{data}}[f(x)] - \mathbf{E}_{x \leftarrow p_g}[f(x)] - \left(f(x) - f(y)\right)}_{\text{enforce constraint } =0, \text{ if } \gamma \in \Pi, \text{ else } +\infty} \right]$$

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

$$= \sup_{f} \mathbf{E}_{x \leftarrow p_{data}}[f(x)] - \mathbf{E}_{x \leftarrow p_g}[f(x)] - \inf_{\gamma} \underbrace{\mathbf{E}_{x,y \leftarrow \gamma} \left[\|x - y\| + \left(f(x) - f(y)\right) \right]}_{\text{new constraint } =0, \text{ if } |f| \leq 1, \text{ else } -\infty}$$

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

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

Wasserstein Duality

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

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

- where sup is least upper bound (maximization within set)
- f will be the critic, learned as a neural network, m=n

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

Maximize f (called critic), freeze generator weights

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

Maximize g, freeze weights of critic

Sometimes this for critic minimize:

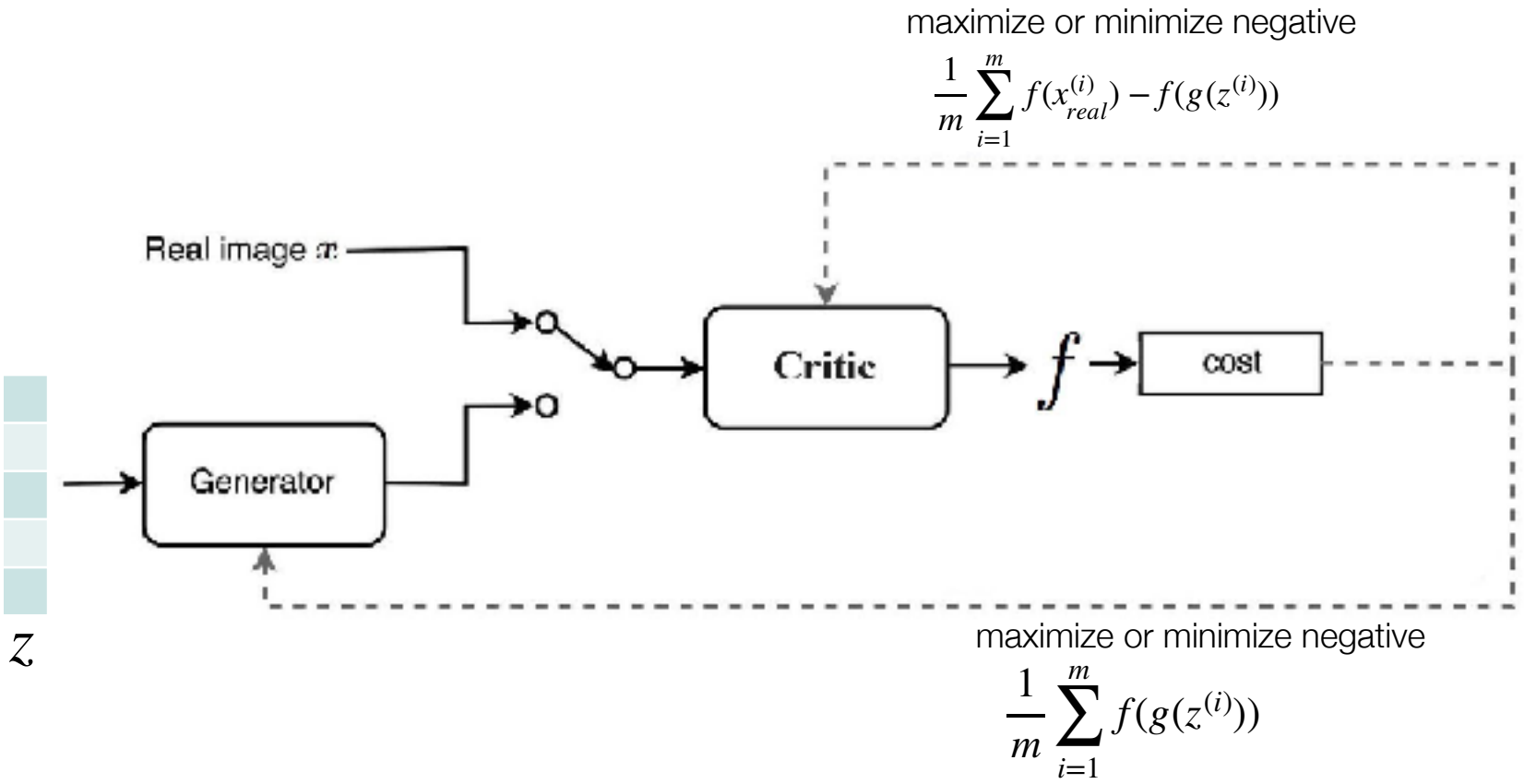
$$\frac{1}{m}\sum_{i=1}^{m} f(x_{real}^{(i)}) \cdot f(g(z^{(i)}))$$

when *label* is -1 and 1

real	fake	f(x)-g(x)	f(x)*g(x)
-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

Wasserstein Architecture



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

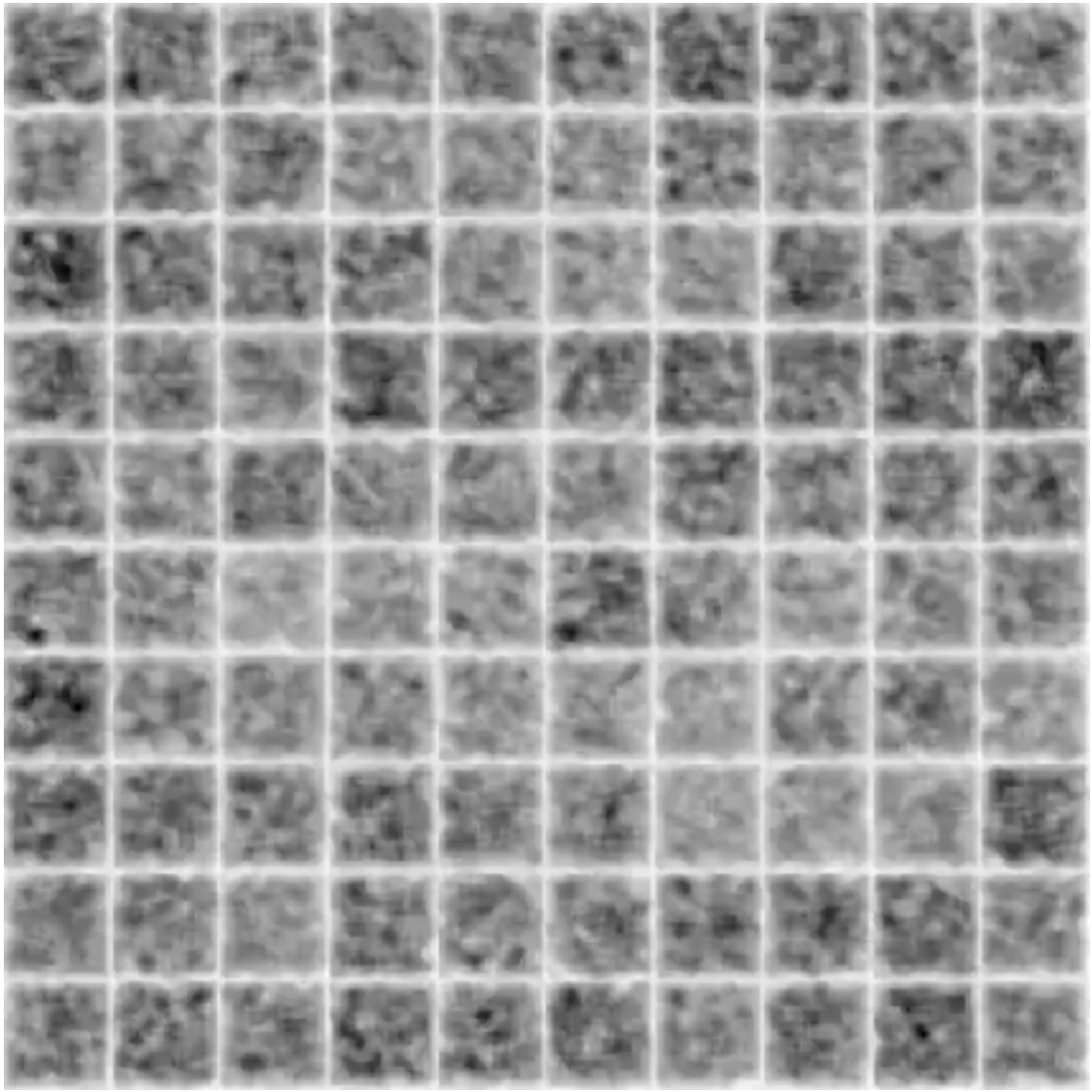
Practical Wasserstein Loss

- 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~0.01)
  - critic output should be linear

```
# define the standalone critic model
def define_critic(in_shape=(28,28,1)):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # weight constraint
    const = ClipConstraint(0.01)
    # define model
    model = Sequential()
    # downsample to 14x14
    model.add(Conv2D(64, (4,4), strides=(2,2), padding='same'
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.2))
    # downsample to 7x7
    model.add(Conv2D(64, (4,4), strides=(2,2), padding='same'
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.2))
    # scoring, linear activation
    model.add(Flatten())
    model.add(Dense(1))
```

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

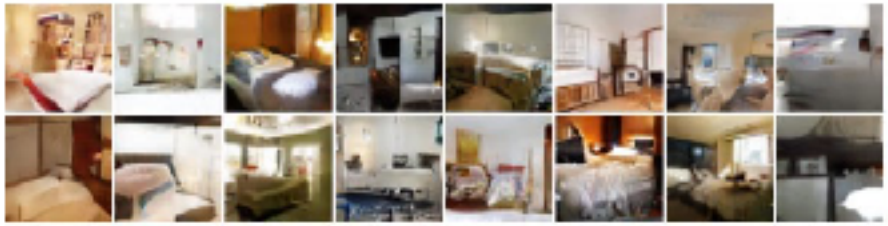
WGAN Example from Keras (MNIST)



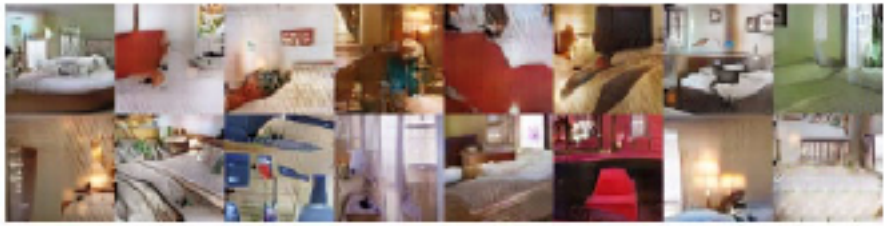
This is a conditional GAN, which we have not yet discussed.

Intuitively it adds some control over the class being generated.

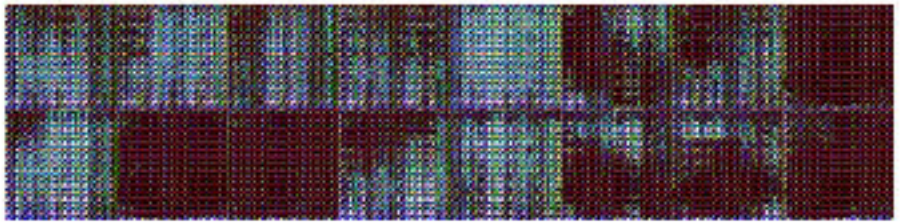
- In their empirical findings, no mode collapse problems
- And training longer resulted in good quality images (motivates that distributions overlap)
- Still some of the most competitive GAN results



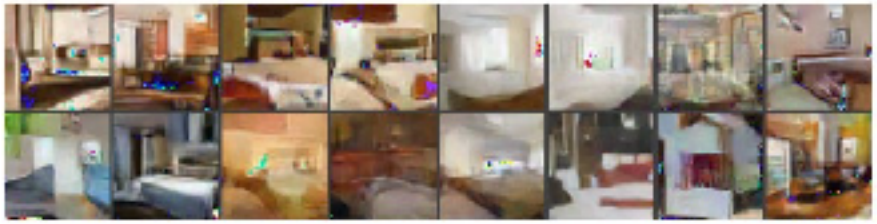
WGAN with DCGAN generator

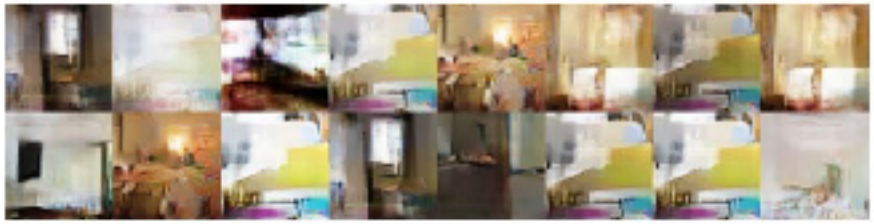


GAN with DCGAN generator



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 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
- New critic: WGAN with gradient penalty
- But… its still really slow compared to others

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 an only if its gradient has a norm less than or equal to one.

Maximize f
freeze generator weights,
incentivize critic gradient norm to be one

Choose large lambda
like lambda = 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 $\mathrm{U}[0,1]$ $\quad \hat{x}^{(i)} = \epsilon \cdot x_{real}^{(i)} + (1-\epsilon)\cdot g(z^{(i)})$

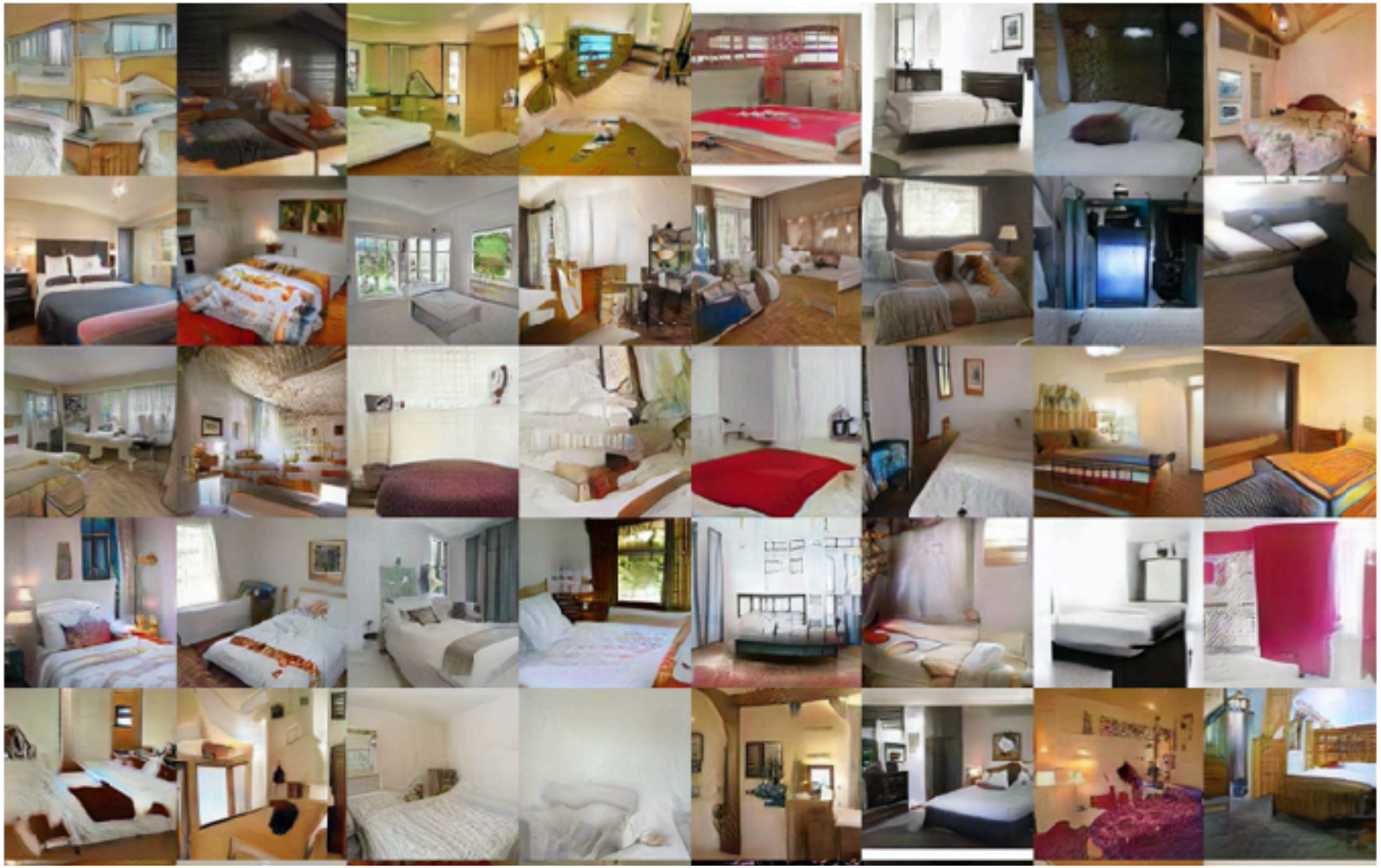
randomly mix together real and fake images

WGAN-GP Heuristics

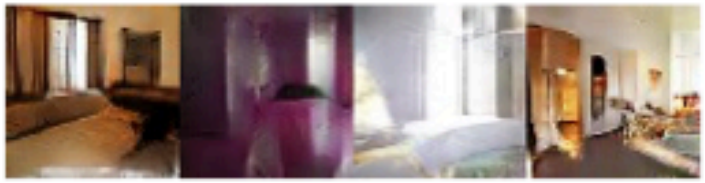
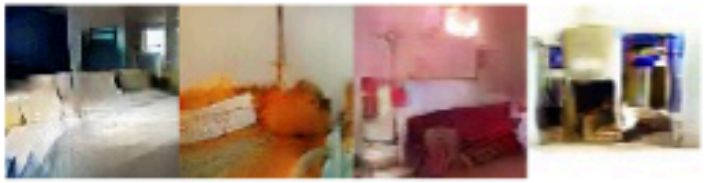
- Choose large penalty coefficient
- Don't use batch normalization (in critic)
- Enforce gradient norm to be close to one, rather than less than one

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.

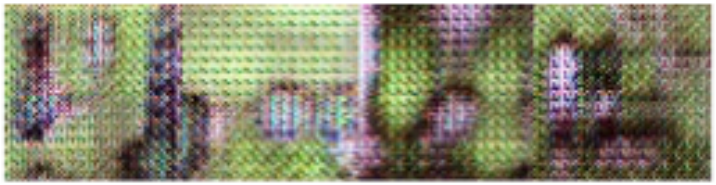
WGAN-GP Results



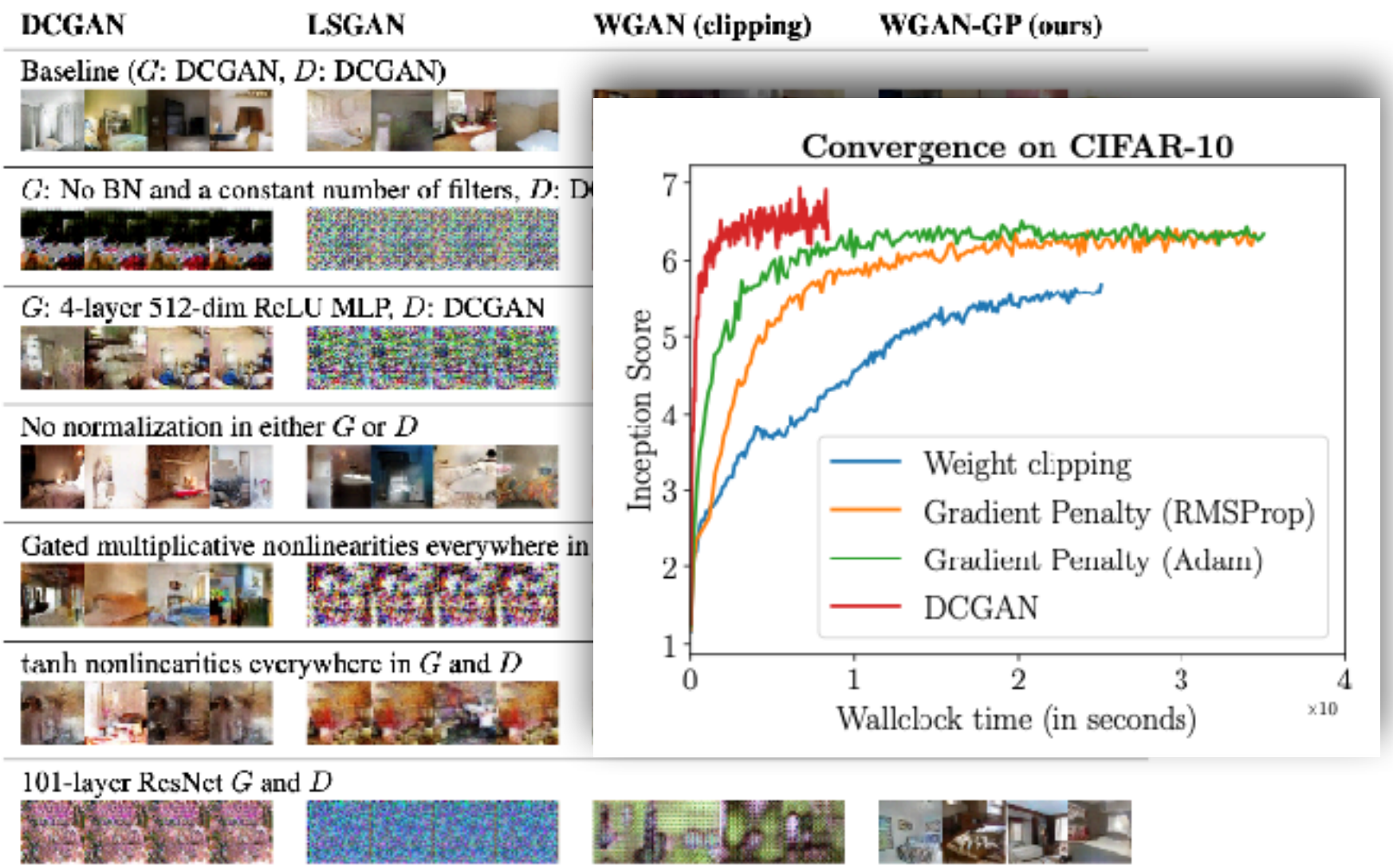
WGAN-GP Comparative Results

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-GPs in Keras/PyTorch

Implementation from Github on "MNIST"

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

Demo by Keras Contrib Community

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

Demo by Erik Linder-Norén

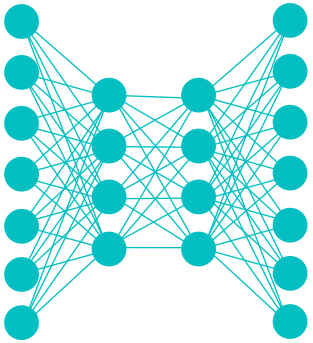
Lecture Notes for

Neural Networks and Machine Learning

Wasserstein GANs

Next Time:
Beyond Generation

Reading: Chollet CH8



Backup slides

Title Between Topics

Example Slide

Title

Subtitle

`Follow Along: Notebook Name`